ANALYSIS OF OPTIMAL STATE FILTER PERFORMANCE ON APOLLO 11 FINAL DESCENT TRAJECTORY STATE ESTIMATION

Erin McNeil

Applying and decerning the performance between the Extended Kalman Filter, Unscented Kalman Filter and Particle Filter to the problem of highly nonlinear altitude sensor error in the case of Moon landings, is the subject of this paper. The three filters were first developed and tested with data derived from digitized plots of the Apollo 11 final descent trajectory then in a 6-DOF simulation with 3-DOF autopilot control in the loop. The three filters are evaluated against 11 separate nonlinear sensor conditions and their RMS estimation errors are compared. Additionally, their application and performance in the 6-DOF simulation flight-like environment is evaluated and discussed.

INTRODUCTION

NASA and others are funding missions to land equipment and personnel on the Moon's surface in the next several years. Reliable proximity sensor data will be required for mission success. However, if sensor data becomes corrupted or unreliable, use of optimal state estimators (e.g., EKF, UKF, etc.) can help mitigate the potential damage.

OPTIMAL STATE ESTIMATION

The use of an optimal state estimator to estimate the trajectory of a lunar module to the Moon's surface is advantageous over the sensor alone because it bases the believability of the sensor data from the projection of the lunar lander state vector temporally along with its associated statistics in comparison to the measurements the sensor provides and iteratively improves its estimation by weighing the statistical likelihood of both. Regardless of the initial estimates the filter will converge on a value with highest statistical likelihood. For this project, I had a particular interest in understanding how each of these filters performed when a sensor proved to have highly unexpected and nonlinear errors, and which one of the three performed best.

EKF Problem Formulation

To begin the process of building an Extended Kalman Filter (EKF) you must know the dynamics of the model you are interested in estimating. In this case, I was interested in the final descent trajectory of the Apollo 11 Lunar Module to the Moon's surface. It is additionally important to have data to compare your dynamic model to validate against. Unfortunately, the Apollo 11 data logs are no longer available. However, there are several surviving reports that have plots with trajectories both planned and flown.[1] Reference 2, utilized these surviving plots, namely the trajectories of the Lunar Module in the longitudinal plane, and used digitization software to recreate an estimation of the original Apollo 11 data logs.[2] I used the same digitization tool, GRABIT, to digitize the plots from Reference 2 to use as inputs to my dynamics state-space model and as observations from a potential proximity sensor. [3] As described in Reference 2, the longitudinal plane equations of motion that describe the path of the rigid body Lunar Module with inertial frame fixed to the surface of the moon with a unit vector for altitude *h* pointing down to the surface and a unit vector pointing out to the desired landing point *r*, are the following:

$$\begin{gathered} m\ddot{r} = -T\sin\theta \\ m\ddot{h} = mg - T\cos\theta \\ I\ddot{\theta} = M \end{gathered} \tag{1}$$

I rearranged the equations in equation 1 to achieve this thrust over mass expression (equation 2) and plugging this into equation 1 gives us the resulting equation for the Lunar Module's altitude acceleration (equation 3).

$$\frac{T}{m} = \frac{g - \ddot{h}}{\cos\theta} \tag{2}$$

$$\ddot{h} = -\frac{\ddot{r}}{\tan\theta} - g \tag{3}$$

From equation 3, to achieve a state space model, I selected the altitude *h* as my first state parameter and altitude rate $\dot{h}$ as my second state parameter.

$$\begin{gathered} x(1) = \mathrm{h} \\ x(2) = \dot{h} \\ \dot{x}(1) = x(2) \\ \dot{x}(2) = u = -\frac{\ddot{x}}{\tan\theta} - g \\ \begin{bmatrix} \dot{x}(1) \\ \dot{x}(2) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_{k-1} \end{gathered} \tag{4}$$

Additionally establishing the use of the trapezoidal method for numerical integration, led me to the following:

$$\begin{gathered} x_k(1) = x_{k-1}(1) + x_{k-1}(2) * dt + \frac{u_{k-1} * dt}{2} \\ x_k(2) = x_{k-1}(2) * dt + u_{k-1} * dt \\ y = x(1) \end{gathered} \tag{5}$$

From equation 4, you can generate a discrete state space model of the system:

$$\begin{bmatrix} x_k(1) \\ x_k(2) \end{bmatrix} = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1}(1) \\ x_{k-1}(2) \end{bmatrix} + \begin{bmatrix} dt/2 \\ dt \end{bmatrix} u_{k-1}$$

$$[y_k] = [1 \quad 0] \begin{bmatrix} x_k(1) \\ x_k(2) \end{bmatrix} + [0]u_k \tag{6}$$

Now that we have our system state and measurement model, we can incorporate the filter. The main components to the EKF are the following: [4]

The model system dynamics and measurement equations:

$$\begin{gathered} x_k = f_{k-1}(x_{k-1}, u_{k-1}, w_{k-1}) \\ y_k = h_k(x_k, v_k) \\ w_k \sim (0, Q_k) \\ v_k \sim (0, R_k) \end{gathered} \tag{7}$$

Updating equation 5 to include the state estimation and measurement errors we get:

$$\begin{bmatrix} x_k(1) \\ x_k(2) \end{bmatrix} = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1}(1) \\ x_{k-1}(2) \end{bmatrix} + \begin{bmatrix} dt/2 \\ dt \end{bmatrix} u_{k-1} + w_{k-1}$$

$$[y_k] = [1 \quad 0] \begin{bmatrix} x_k(1) \\ x_k(2) \end{bmatrix} + v_k \tag{8}$$

The values for the state dynamics and measurement model errors and variances were a point of variability in the model. There are many potential sources of errors during space flight for sensors to fault or produce errors- variability in operating conditions from what was anticipated prior to the mission, communication channel noise, changes in temperature, unanticipated magnitude, phase, or drift of periodic sensor noise, etc. In this case the statistics of the measurement noise may be entirely incongruous from what you may have anticipated prior to launch. As a result of this, the standard linear Kalman filter will provide strongly biased state estimates. Because of this, I was interested in seeing which of the 3 filters (EKF, UKF and Particle Filter) provided the best estimation of a sensor with highly nonlinear error. The assumed measurement error covariance is 1000 ft^2 (which would assume an error of sqrt(R)*randn) with true random Gaussian noise error shown in Table 1. For the "Worst Case" test at the bottom of Table 1, I also added simulated "stale" data where the values are not able to be updated for several time steps. The Process Noise covariance I chose to be equal to identity.

Table 1. Measurement Noise Variations (v_k)

Test Number	Periodic Element	Magnitude (ft)	Time Between Updates (sec)
Test 1	-	1000	0.8
Test 2	-	1000	1.5
Test 3	cos(50*t)	1000	0.8
Test 4	cos(50*t)	1000	1.5
Test 5	cos(5*t)	1000	0.8
Test 6	cos(25*t)	1000	0.8
Test 7	cos(500*t)	1000	0.8

Test 8	cos(5*t)	100	0.8
Test 9	cos(5*t)	5000	0.8
Test 10	cos(5*t)	7000	0.8
WORST CASE	cos(500*t)	7000	1.5

The initial conditions for the system dynamics and state estimation covariance matrix:

$$\begin{gathered} \hat{\boldsymbol{x}}_0^+ = \boldsymbol{E}(\boldsymbol{x}_0) = \begin{bmatrix} 400 \\ 0.5 \end{bmatrix} \\ \boldsymbol{P}_0^+ = \boldsymbol{E}[(\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0^+)(\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0^+)^{\boldsymbol{T}}] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{gathered} \tag{9}$$

Based on the best estimates from the surviving Apollo 11 plots for the final descent I chose the initial condition for the altitude state estimate to be 400 feet and altitude rate state estimate to be 0.5 ft/sec. Since I am confident in those estimates, I assigned an initial value of identity to the state estimation covariance matrix *P*.

Within the filter's discrete iterative loop (k = 1, 2…) we have the following:

The linearized state transition matrix as well as the partial derivative of the state dynamics in terms of the dynamic system noise, both linearized about the filter's nominal trajectory state estimate, are given by:

$$\begin{gathered} \boldsymbol{A} = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \\ \boldsymbol{L} = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{w}}\right|_{\hat{\boldsymbol{x}}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{gathered} \tag{10}$$

The longitudinal planar dynamics I'm modeling here are linear, so the resulting state transition matrix is equivalent to the matrix given in equation 3.

I am using a continuous model for the state-vector and measurement model and discrete versions of the EKF. The time update of the state estimate and state estimation error covariance are calculated by integrating the following over one time step:

$$\begin{aligned} \dot{\hat{\boldsymbol{x}}} &= \boldsymbol{f}(\hat{\boldsymbol{x}}, \boldsymbol{u}, \boldsymbol{0}, \boldsymbol{t}) \\ \dot{\boldsymbol{P}} &= \boldsymbol{AP} + \boldsymbol{PA^T} + \boldsymbol{LQL^T} \\ &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\boldsymbol{P} + \boldsymbol{P}\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\boldsymbol{Q}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{P} & 0 \\ 0 & 0 \end{bmatrix} + \boldsymbol{Q} \end{aligned} \tag{11}$$

And the partial derivative matrices for *H* and *M*:

$$\begin{gathered} \boldsymbol{H_k} = \left.\frac{\partial \boldsymbol{h_k}}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_k^-} = [\boldsymbol{1} \quad \boldsymbol{0}] \\ \boldsymbol{M_k} = \left.\frac{\partial \boldsymbol{h_k}}{\partial \boldsymbol{v}}\right|_{\hat{\boldsymbol{x}}_k^-} = \boldsymbol{1} \end{gathered} \tag{12}$$

And finally, to end the iterative loop, the measurement update of the state estimate and estimation error covariance:

$$\begin{aligned} K_k &= P_k^- H_k^T \left(H_k P_k^- H_k^T + M_k R_k M_k^T\right)^{-1} \\ \hat{x}_k^+ &= \hat{x}_k^- + K_k[y_k - h_k(\hat{x}_k^-, 0)] \\ P_k^+ &= (I - K_k H_k) P_k^- \end{aligned} \tag{13}$$

I implemented this in MATLAB and found the following results (see additional plots in Appendix B):

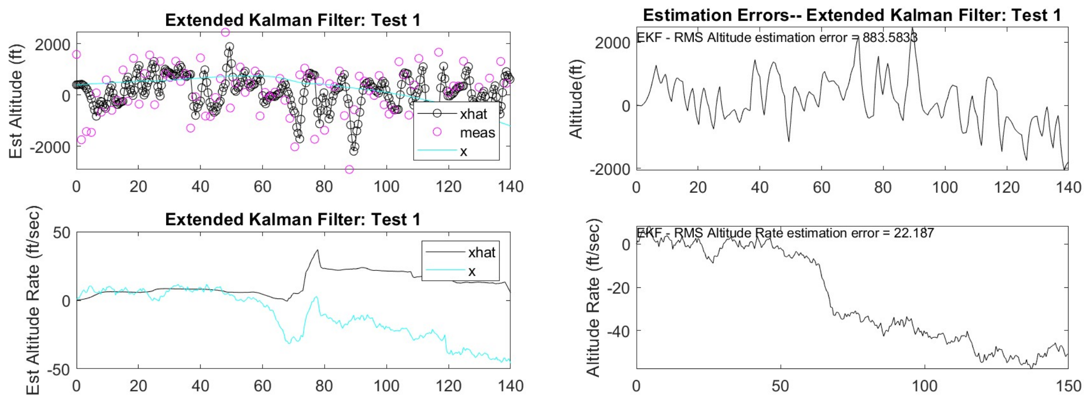


Figure 1. Extended Kalman Filter Results (Final_Project_EKF_EEM.m)

UKF Problem Formulation

The Unscented Kalman Filter (UKF) utilizes the same system dynamics and initial state conditions as outlined in the EKF section, however the filter itself is unique. Again, I will be using a sensor with highly nonlinear errors as mentioned in Table 1, with the hope that the UKF is able to, despite this complication, find a good estimate of the true state.

Unscented transformations are more accurate than linearization alone at propagating the means and covariances of the state and measurement models. To create the UKF we replace the EKF equations for state and covariance propagation with the unscented transformations.

Within the filter's discrete iterative loop (k = 1, 2…) we have the following:

$$\begin{aligned} \hat{x}_{k-1}^{(i)} &= \hat{x}_{k-1}^+ + \tilde{x}^{(i)}, \qquad i = 1, \dots, 2n \\ \tilde{x}^{(i)} &= \left(\sqrt{nP_{k-1}^+}\right)_i^T, \qquad i = 1, \dots, n \\ \tilde{x}^{(n+i)} &= -\left(\sqrt{nP_{k-1}^+}\right)_i^T, \qquad i = 1, \dots, n \end{aligned} \tag{14}$$

with *n* is the length of the state space vector. Plug in these sigma values into the longitudinal planar dynamics and then combine them to get the a priori state estimate:

$$\hat{x}_k^{(i)} = f_{k-1}(\hat{x}_{k-1}, u_k, t_k) \tag{15}$$

$$\hat{x}_k^- = \frac{1}{2n}\sum_{i=1}^{2n} \hat{x}_k^{(i)}$$

Calculate the time update of the state estimation error covariance:

$$P_k^- = \frac{1}{2n}\sum_{i=1}^{2n}\left[\left(\hat{x}_k^{(i)} - \hat{x}_k^-\right)\left(\hat{x}_k^{(i)} - \hat{x}_k^-\right)^T\right] + Q_{k-1} \tag{16}$$

Resample the sigma points with your time updated state and covariance matrices:

$$\begin{aligned} \hat{x}_k^{(i)} &= \hat{x}_k^+ + \tilde{x}^{(i)}, \qquad i = 1, \dots, 2n \\ \tilde{x}^{(i)} &= \left(\sqrt{nP_{k-1}^-}\right)_i^T, \qquad i = 1, \dots, n \\ \tilde{x}^{(n+i)} &= -\left(\sqrt{nP_{k-1}^-}\right)_i^T, \qquad i = 1, \dots, n \end{aligned} \tag{17}$$

Plug in the time updated sigma points into your measurement model and combine to get the predicted measurement value at that time step:

$$\begin{aligned} \hat{y}_k^{(i)} &= h\left(\hat{x}_k^{(i)}, t_k\right) \\ \hat{y}_k &= \frac{1}{2n}\sum_{i=1}^{2n} \hat{y}_k^{(i)} \end{aligned} \tag{18}$$

Use the predicted measurement value and a priori estimate of the state to get the covariance of the predicted measurement and the cross covariance of the a priori state with the predicted measurement:

$$\begin{aligned} P_y &= \frac{1}{2n}\sum_{i=1}^{2n}\left[\left(\hat{y}_k^{(i)} - \hat{y}_k\right)\left(\hat{y}_k^{(i)} - \hat{y}_k\right)^T\right] + R_k \\ P_{xy} &= \frac{1}{2n}\sum_{i=1}^{2n}\left[\left(\hat{x}_k^{(i)} - \hat{x}_k^-\right)\left(\hat{y}_k^{(i)} - \hat{y}_k\right)^T\right] \end{aligned} \tag{19}$$

Finally, to end the iterative loop, conduct the measurement update of the state estimate using the Kalman filter equations:

$$\begin{aligned} K_k &= P_{xy}P_y^{-1} \\ \hat{x}_k^+ &= \hat{x}_k^- + K_k[y_k - \hat{y}_k] \\ P_k^+ &= P_k^- - K_k P_y K_k^T \end{aligned} \tag{20}$$

I implemented this in MATLAB and found the following results (see additional plots in Appendix B):

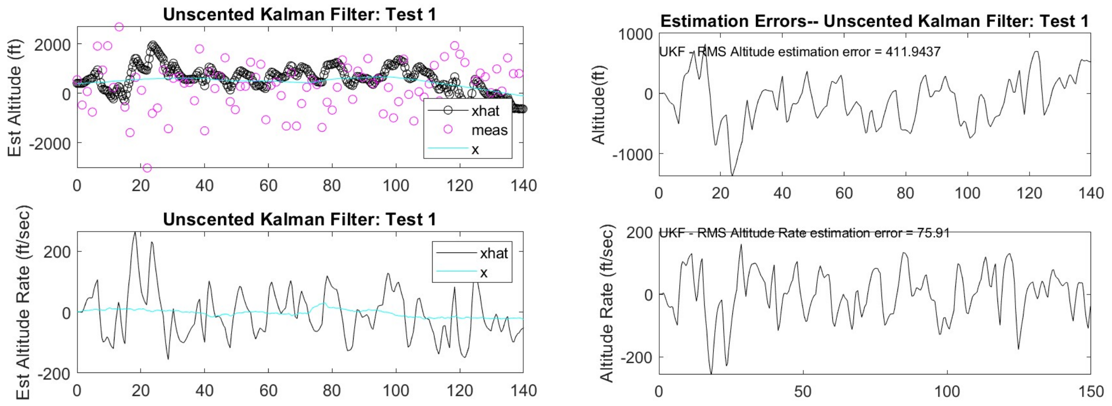


Figure 2. Unscented Kalman Filter Results (Final_Project_UKF_EEM.m)

Particle Filter Problem Formulation

Again, the Particle filter utilizes the same system and initial state dynamics and measurement model as previously discussed. The Particle Filter is advantageous for systems that do not require quick execution because of its utilization of the Bayesian estimator, which increases in accuracy as the number of particles (and therefore computational loops) increases.

Within the filter's discrete iterative loop (k = 1, 2…) we have the following:

The Particle Filter begins by performing the time-propagation step to obtain a priori particles using the known process equation and known pdf of the process noise. The total number of particles, *N*, was chosen to be 1000 for this project:

$$\hat{x}_{k,i}^{-} = f_{k-1}\left(x_{k-1,i}^{+}, w_{k-1}^{i}, t_k\right), \quad (i = 1, \dots, N) \tag{21}$$

Compute the likelihood functions, q_i, for each of the particles, where y^* is the measurement we are comparing against:

$$q_i = \frac{1}{(2\pi)^{N/2} * |R|^{1/2}} exp(\frac{-[y^* - h(x_{k,i}^{-})]^T R^{-1} [y^* - h(x_{k,i}^{-})]}{2} \tag{22}$$

Scale the likelihood functions so that the sum of all of the likelihoods equal to 1:

$$q_i = \frac{q_i}{\sum_{j=1}^{N} q_j} \tag{23}$$

And finally, resample to get the a posteriori particles by generating a random number between 0 and 1, *r,* and sum the scaled likelihoods until the value of the sum is greater than or equal to *r*. If that occurs, then set the resampled particle equal to the a priori particle:

$$\begin{gathered} r = randn \\ if, \quad \sum_{m=1}^{j} q_m \geq r, \quad then\ set\colon \quad x_{k,i}^{+} = x_{k,j}^{-}, \quad (i,j = 1, \dots, N) \end{gathered} \tag{24}$$

It is at this point, during resampling, that a phenomenon called sample impoverishment can occur. This results in only a few particles getting resampled and eventually all the samples will collapse to one value. To prevent this in my tools, I've added roughening to the resampling process. I used a roughening parameter K set to 0.4 and assumed a zero mean, Gaussian distribution for the random variable $\Delta x(m)$. For the maximum difference between particle elements, M, i and j are particle numbers, and n is the length of the state space vector.

$$x^{+}_{k,i} = x^{+}_{k,i}(m) + \Delta x(m), \quad (m = 1, \dots, n)$$

$$\Delta x(m) \sim \left(0, KM(m)N^{-\frac{1}{n}}\right) \tag{25}$$

$$M(m) = max\left|\left(x^{+}_{k,i}(m) - x^{+}_{k,j}(m)\right)\right|, \quad (m = 1, \dots, n)$$

The final step in the iterative loop is to calculate the average value of resampled and remaining particles to get our a posteriori value of the state estimate:

$$\hat{x}^{+}_{k} = \frac{1}{N}\sum_{i=1}^{N} \hat{x}^{+}_{k,i} \tag{26}$$

I implemented this in MATLAB and found the following results (see additional plots in Appendix B):

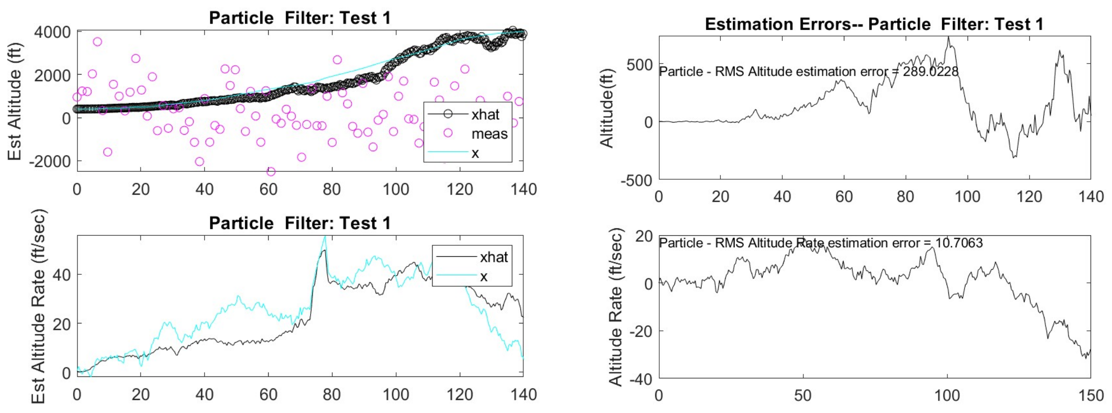


Figure 3. Particle Filter Results (Final_Project_Particle_Filter_EEM.m)

IMPLEMENTATION OF FILTERS IN 6-DOF SIMULATION

After implementing the filters with the digitized Apollo 11 trajectory data and getting the preceding results, I endeavored to test my filters in a more flight like environment. I found in the MATLAB aerospace toolbox a 6-DOF simulation of the Apollo 11 descent to the Moon (aero_dap3dof.slx) with a 3-DOF autopilot in the loop.[5] I set about adding my optimal altitude estimation tools to the Simulink environment. To include the 3 estimators into the 6-DOF simulation, changes to the filters, as described in the preceding sections, were made. The dynamics model provided by the 6-DOF simulation replaced as inputs to the estimation tools, rather than the state-space models described in the Optimal State Estimation sections above. In this use case, I chose to estimate the range as well as the altitude of the Lunar Module in relation to the target destination described in Reference 1, rather than the altitude rate. This took some rotational transformation calculations to transform the earth centered earth frame position (ECEF) values to azimuth-elevation-range (AER) inertial values. These calculations are at the top of MATLAB embedded function blocks containing the filters, shown in Figure 4 (see Appendix B for a copy of the function block contents for each filter). Additionally, the estimator embedded scripts were made robust to code generation for real time processing.

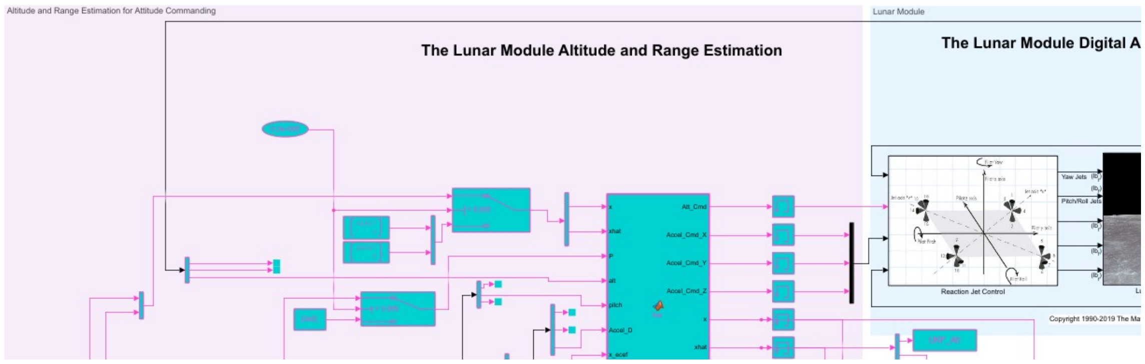

Altitude and Range Estimation for Attitude Commanding
The Lunar Module Altitude and Range Estimation
Lunar Module
The Lunar Module Digital A
x
xhat
P
alt
pitch
Accel_D
x_ecef
Att_Cmd
Accel_Cmd_X
Accel_Cmd_Y
Accel_Cmd_Z
x
xhat
Yaw Jets
Pitch/Roll Jets
Reaction Jet Control
Copyright 1990-2019 The Ma

Figure 4. Simulink Model with EKF (Final_Project_EEM_EKF_SIM.slx)

As shown in Reference 1’s Figure 7, at specified ranges and altitudes the Apollo 11 Lunar Module commanded the vehicle into a pitch angle of varying degrees to place the vehicle close to an upright position as it touched down to the lunar surface. With that functionality in mind, I added the ability to command the desired pitch angle within my estimator embedded function blocks to the 3-DOF autopilot as the state estimators register that the vehicle had hit the targeted altitude for a pitch angle update. I additionally used a constant acceleration in the z-axis of the lander’s body frame to mimic the LM descent engine.

Three separate Simulink Models and embedded MATLAB files were created for each of the filters. I demarcated my contributions to the Simulink model by formatting the color of the blocks I added to have cyan backgrounds and magenta foregrounds. The area delineated by the magenta box represents the bulk of my contribution, but some changes were made to the MATHWORKS provided 3-DOF autopilot and Plant Model, again, demarcated in cyan and pink. See Appendix A for more details.

CONCLUSION

Reviewing the results from the different use cases shown in Table 2, the Particle Filter had the lowest error in 6 or the 11 cases, the UKF had the lowest error in 6 of the 11 cases and the EKF had the lowest error in 0 of the 11 cases. Then upon reviewing each of the resulting plots shown in the attached Appendix B, the Particle filter tends to do a better job at following the system dynamics model and ignoring the measurement updates, whereas the UKF tends to do a better job at tracking both the measurements and the propagated state.

Table 2. Filter Performance Comparison

Test Number	Periodic Element	Magnitude (ft)	Time Between Updates (sec)	EKF Filter RMS Altitude estimation error (ft)	UKF Filter RMS Altitude estimation error (ft)	Particle Filter RMS Altitude estimation error (ft)
Test 1	-	1000	0.8	883.5833	411.9437	289.0228
Test 2	-	1000	1.5	923.2525	428.7222	625.5613

Test 3	cos(50*t)	1000	0.8	745.237	292.9731	557.0452
Test 4	cos(50*t)	1000	1.5	1004.0655	396.248	497.3692
Test 5	cos(5*t)	1000	0.8	1340.3404	671.1096	490.803
Test 6	cos(25*t)	1000	0.8	1355.1052	416.7657	214.3447
Test 7	cos(500*t)	1000	0.8	1398.7000	851.7116	879.3347
Test 8	cos(5*t)	100	0.8	293.3246	208.3615	1119.4923
Test 9	cos(5*t)	5000	0.8	2303.0896	1482.84	1162.6126
Test 10	cos(5*t)	7000	0.8	4588.109	1841.7592	498.781
WORST CASE	cos(500*t)	7000	1.5	2433.4123	1787.466	346.471

Within the 6-DOF flight-like environment the three filters performed well at predicting the true altitude and range despite noisy sensor data. The Particle Filter noticeably slowed down the simulation even with 100 particles, highlighting its computational resource burden. Additionally the additional Pitch angle command, based on the filter estimated altitude, was successful for all three filters. See Appendix B for more plots showing the performance of the filters in the 6-DOF environment as well as the stand-alone filters that use state-space modeled dynamics.

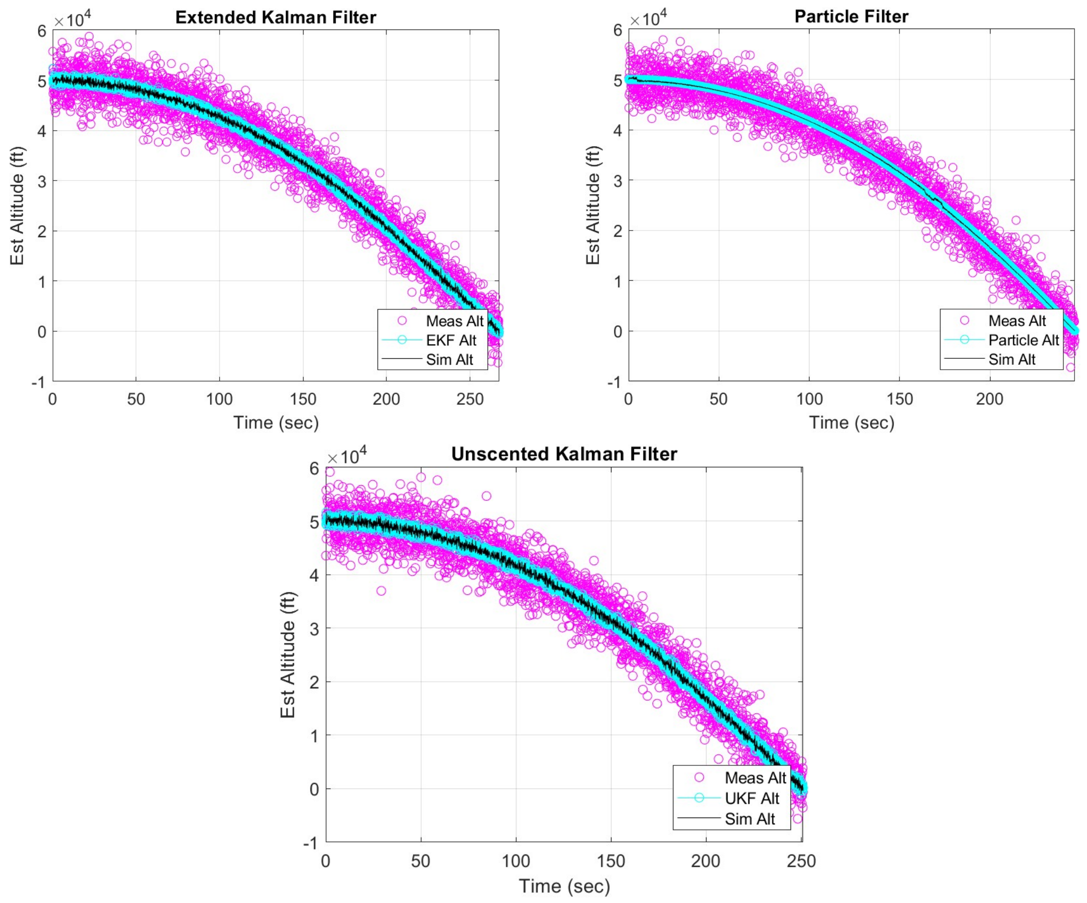
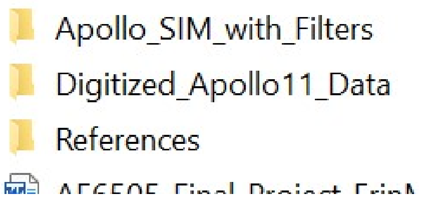


Figure 5. State Estimates of Altitude Produced by the 3 Filters in the 6-DOF Simulation Env.

APPENDIX A

To help with navigating through the MATLAB files associated with this project begin at the top level of the folders supplied:



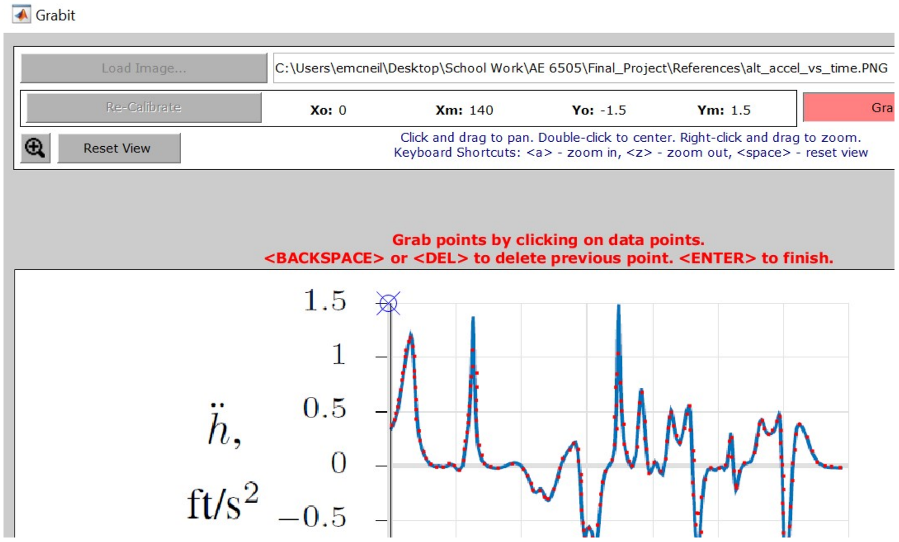
Apollo_SIM_with_Filters

Digitized_Apollo11_Data

References

Figure 6. Top Level Folders of Project Submission

The folder titled "Digitized_Apollo11_Data" contains the scripts used to create the digitized Apollo 11 data, data truncation, and the EKF, UKF and Particle Filter tools used to generate Figure 1, Figure 2, and Figure 3. The script titled "grabit.m" in Figure 8 shown below is a tool I downloaded to digitize the plots provided in Reference 2.[6] An example of one of the plots I digitized is shown in Figure 7.



Grabit

C:\Users\emcneil\Desktop\School Work\AE 6505\Final_Project\References\alt_accel_vs_time.PNG

Xo: 0 Xm: 140 Yo: -1.5 Ym: 1.5

Click and drag to pan. Double-click to center. Right-click and drag to zoom.
Keyboard Shortcuts: <a> - zoom in, <z> - zoom out, <space> - reset view

Grab points by clicking on data points.
<BACKSPACE> or <DEL> to delete previous point. <ENTER> to finish.

Figure 7. Grabit Tool Digitizing the Altitude Acceleration vs. Time Plot from the Apollo 11 Mission

The "Final_Project_Digitized_Data_Truncation.m" file I created takes the .mat files generated by the "grabit.m" tool and evens out the time steps so that they are consistent and evenly distributed. The three MATLAB scripts titled "Final_Project_EKF_EEM.m", "Final_Project_UKF_EEM.m", and "Final_Project_Particle_EEM.m" contain the Filter models I created as described in the Optimal State Estimation sections above with the measurement error test cases described in Table 1. You will also find in the "Digitized_Apollo11_Data" folder .mat data files with the digitized data generated by GRABIT as well as the truncated versions of that data generated using my truncation script.

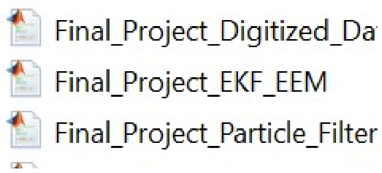

Final_Project_Digitized_Da
Final_Project_EKF_EEM
Final_Project_Particle_Filter

Figure 8. Digitization and Filter Scripts

In the folder titled "Apollo_SIM_with_Filters" you will find the Simulink models (Figure 9), like the one referenced in Figure 4, as well as their pre and post processing scripts (Figure 10). Alongside these files you will also have access to the MATHWORKS library block files for the 3-DOF lunar module autopilot, as well as the gravitational model and transformation matrix blocks as shown in Figure 11.

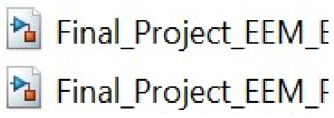

Final_Project_EEM_E
Final_Project_EEM_F

Figure 9. Apollo 11 6-DOF Simulink Models

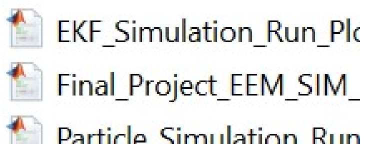

EKF_Simulation_Run_Plc
Final_Project_EEM_SIM_
Particle_Simulation_Run

Figure 10. Apollo 11 6-DOF Pre and Post Processing Scripts

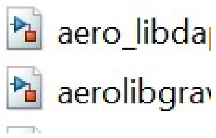

aero_libda
aerolibgra

Figure 11. Apollo 11 MATHWORKS Library Blocks

To run the Simulink models, open one of the .slxs files listed in Figure 9. The pre-processing script "Final_Project_EEM_SIM_init_params.m", I edited but originally was developed by MATHWORKS, will run as the model initializes and will populate the base workspace with the needed initial conditions and design gains, etc. Your screen should look like Figure 12. To run the model hit the green "Run" button at the center of the toolbar ribbon.

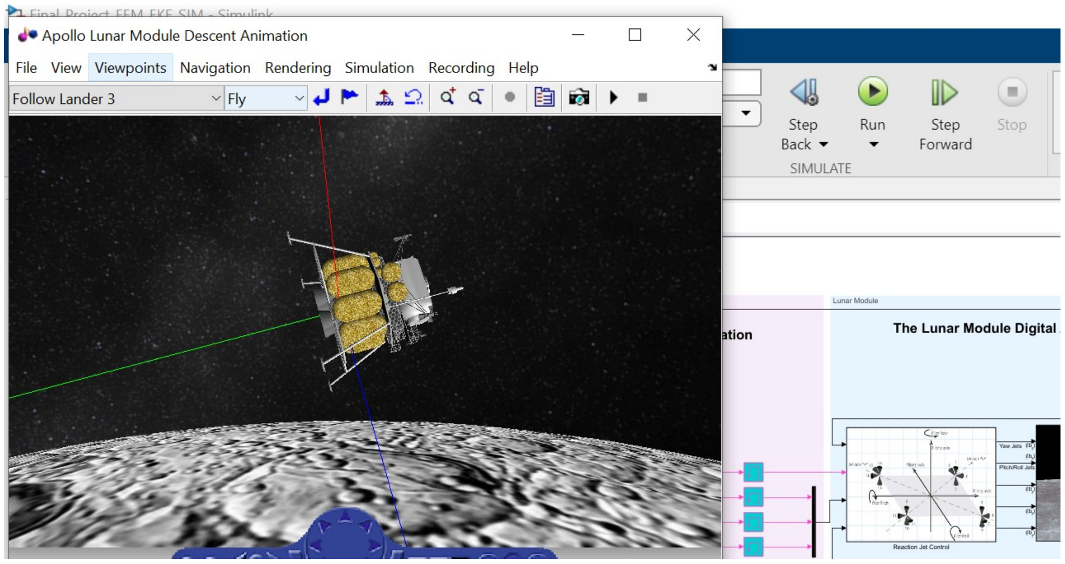

Final Project FEM EKF SIM - Simulink
Apollo Lunar Module Descent Animation
File View Viewpoints Navigation Rendering Simulation Recording Help
Follow Lander 3
Fly
Step Back
Run
Step Forward
Stop
SIMULATE
ation
Lunar Module
The Lunar Module Digital
Reaction Jet Control

Figure 12. Modified Apollo 11 6-DOF Simulink Model

You are able to watch the lunar lander Pitch to the commanded degree per my filters' command logic in the MATHWORKS developed "Apollo Lunar Module Descent Animation" tool as the simulation processes, like what is shown in Figure 13.

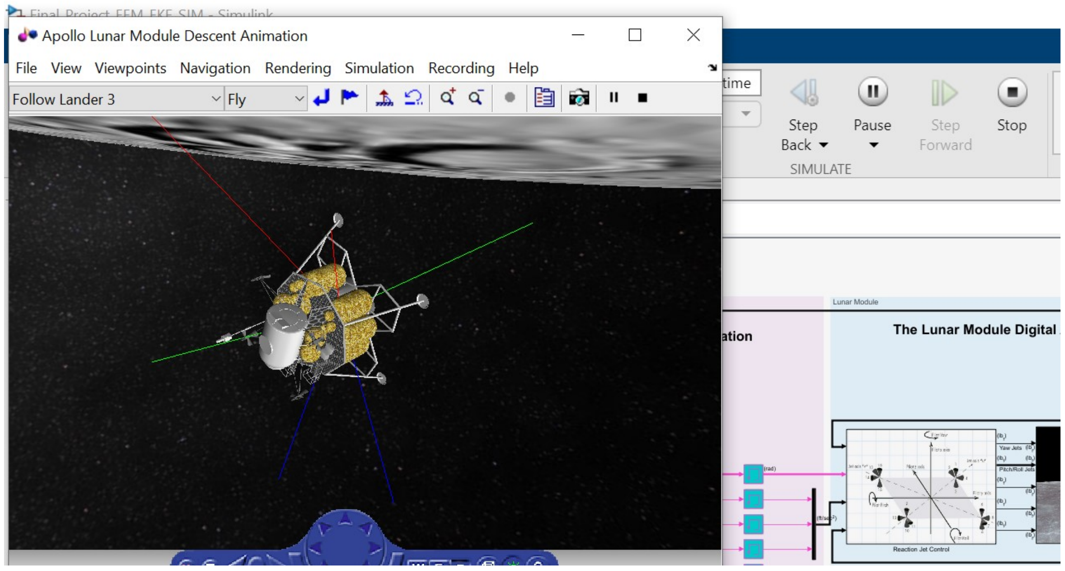

Final Project FEM EKF SIM - Simulink
Apollo Lunar Module Descent Animation
File View Viewpoints Navigation Rendering Simulation Recording Help
Follow Lander 3
Fly
time
Step Back
Pause
Step Forward
Stop
SIMULATE
ation
Lunar Module
The Lunar Module Digital
Reaction Jet Control

Figure 13. Modified Apollo 11 6-DOF Simulink Model (Running)

I built in a stop function to stop the simulation as soon as the lunar module reaches 0 ft altitude. After the animation concludes, my post processing scripts will run and will provide you with plots of the real and estimated trajectories as well as the noisy measurements I generated along with other plots of interest, like the achieved Pitch angle over the course of the trajectory, as shown in Figure 14.

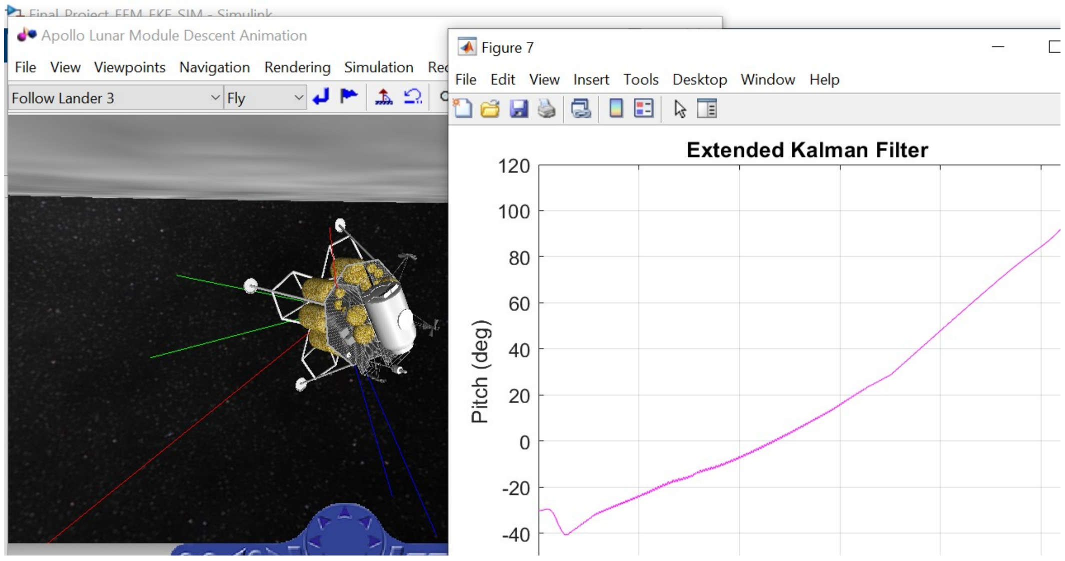

Final Project EEM EKF SIM - Simulink
Apollo Lunar Module Descent Animation
File View Viewpoints Navigation Rendering Simulation Rec
Follow Lander 3
Fly
Figure 7
File Edit View Insert Tools Desktop Window Help
Extended Kalman Filter
Pitch (deg)
120
100
80
60
40
20
0
-20
-40

Figure 14. Modified Apollo 11 6-DOF Simulink Model (Running)

Referring back to Figure 6, the "References" folder contains some of the reference papers listed in the references section below, as well as the pictures of trajectory plots used in the "grabit.m" tool to generate the trajectory data used in "Final_Project_Digitized_Data_Truncation.m" file to truncate the data to an even time step for use in in the filter files "Final_Project_EKF_EEM.m", "Final_Project_UKF_EEM.m", and "Final_Project_Particle_EEM.m."

APPENDIX B

Please see the attached PDF for Appendix B. It contains additional plots and copies of the contents of my embedded MATLAB function blocks for the EKF, UKF and Particle Filters.

REFERENCES

[1] Bennett, F.: "Apollo Lunar Descent and Ascent Trajectories". NASA TM X-58040, March 1970.

[2] Miller, L., Grauer, J., and Pei, J.: "Reconstruction of the Apollo 11 Moon Landing Final Descent Trajectory," NASA TM-20220007267, August 2022.

[3] Jiro: GRABIT. https://www.mathworks.com/matlabcentral/fileexchange/7173-grabit, accessed April 2024.

[4] Simon, D.: Optimal State Estimation. John Wiley & Sons, Inc., 1974.

[5]MathWorks Aerospace Products Team (2024). Apollo 11 Moon Landing - 50th Anniversary Model (https://github.com/mathworks/Apollo_11_Moon_Landing_-_50th_Anniversary_Model), GitHub. Retrieved April 27, 2024.