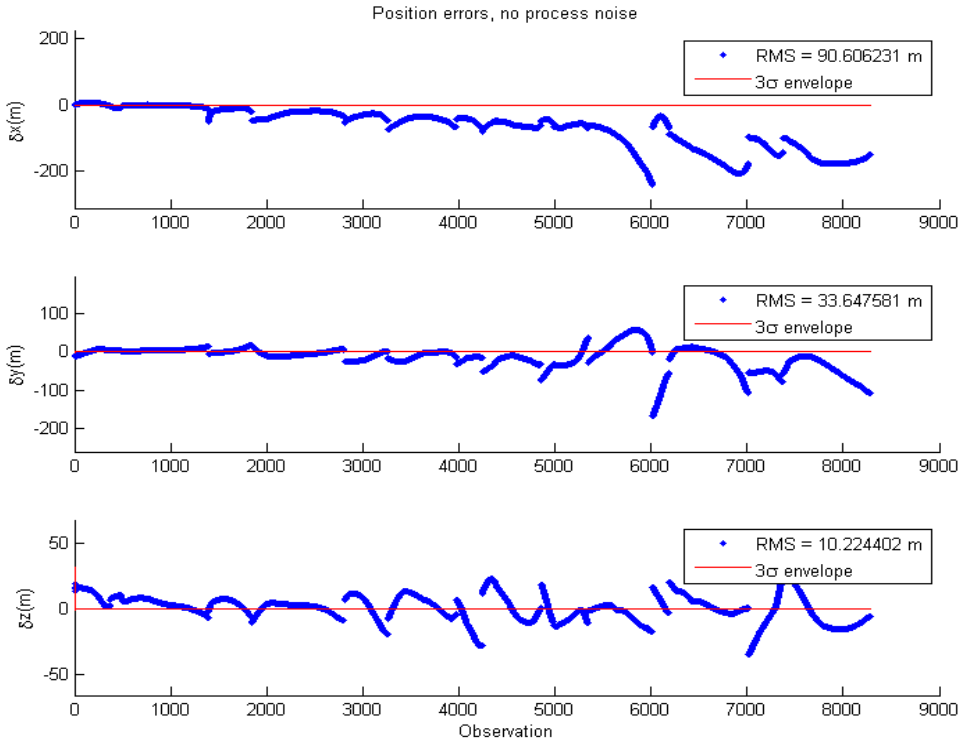
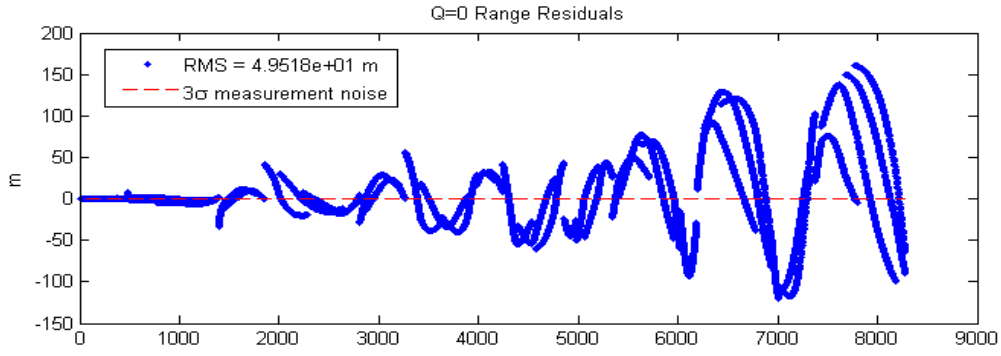
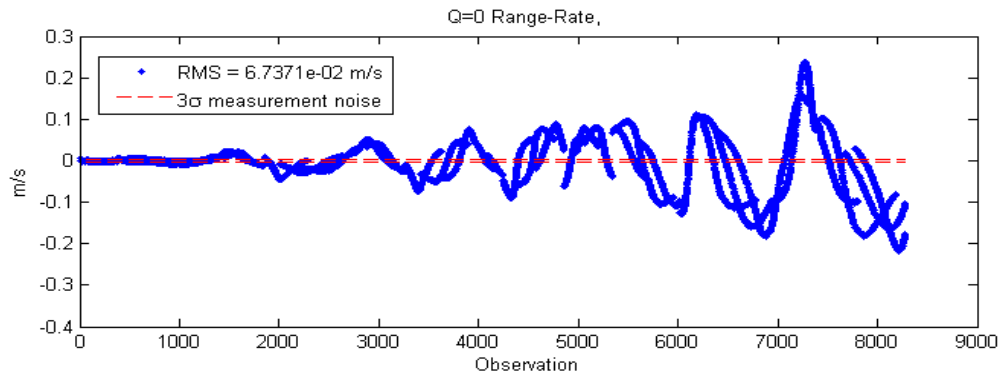
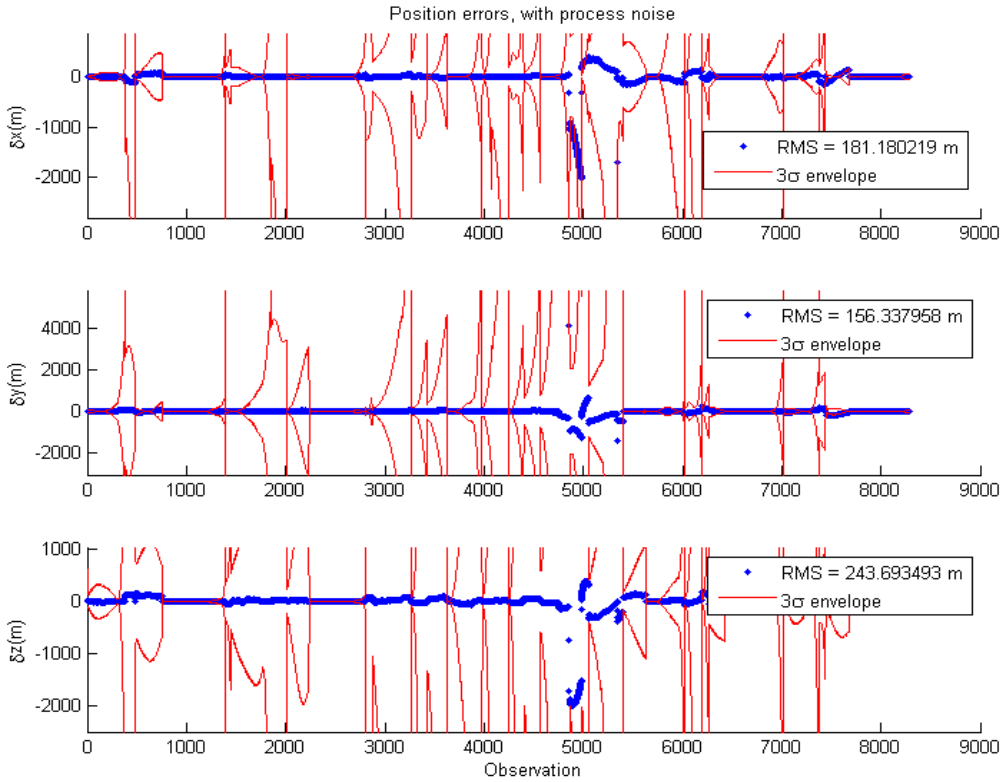
The results for $\alpha = 1.0$, $\beta = 2$ and no process noise are shown below:

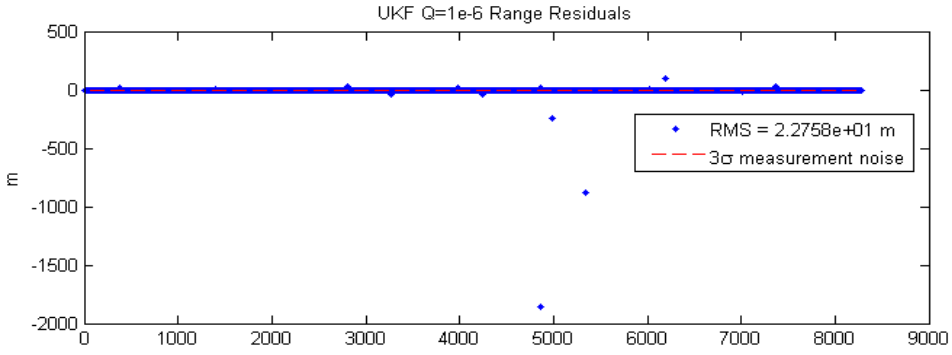


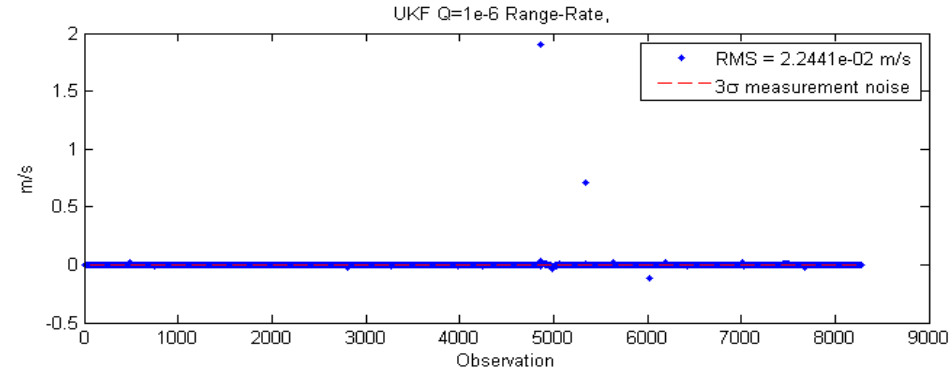




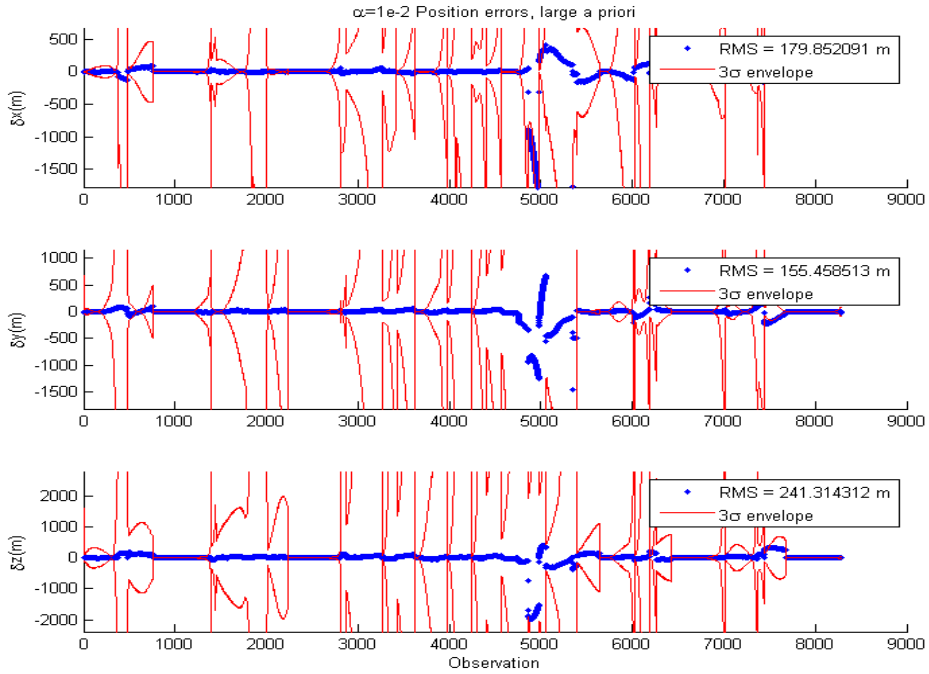
The results for $\alpha = 1.0$, $\beta = 2$ with process noise are shown below:

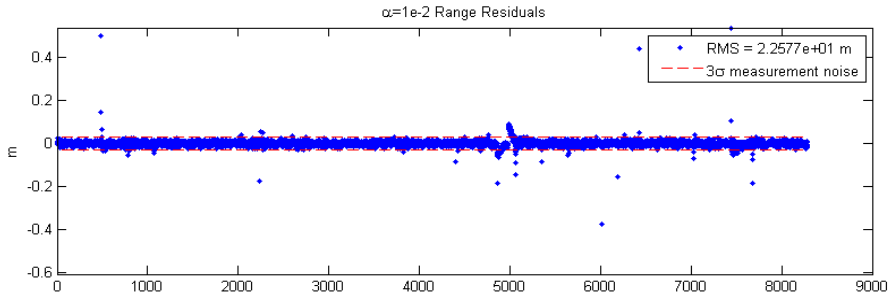


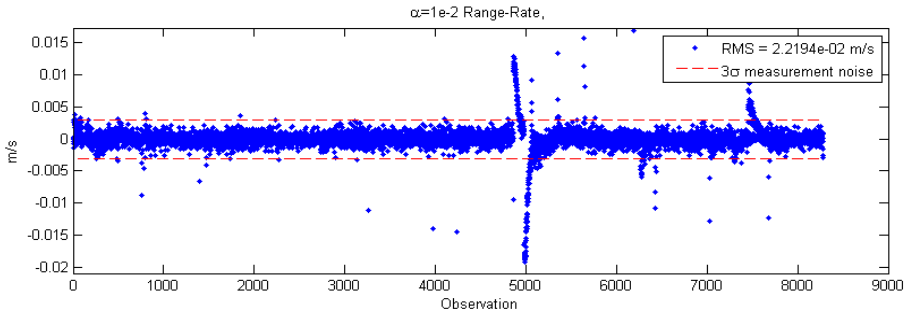




The results for α = 1e-2 are shown below (I could not get good results for 1e-4):







Unfortunately I was not able to do α = 1e-4 as requested. Sqrtm() started to introduce complex values into my filter, and I couldn't do a lower-triangular sqrt via Cholesky because P became non-positive definite. I tracked it to the calculation of Pyy and Pxy, which had large condition values. However, α =

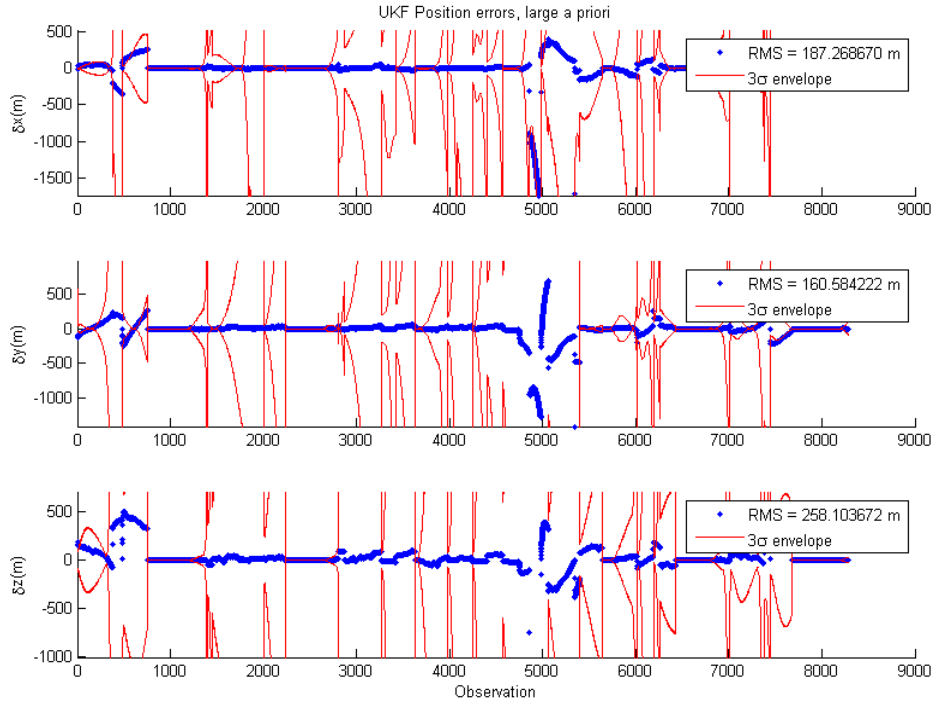
1e-2 seemed to perform quite closely to $\alpha = 1$, which surprised me given the range.

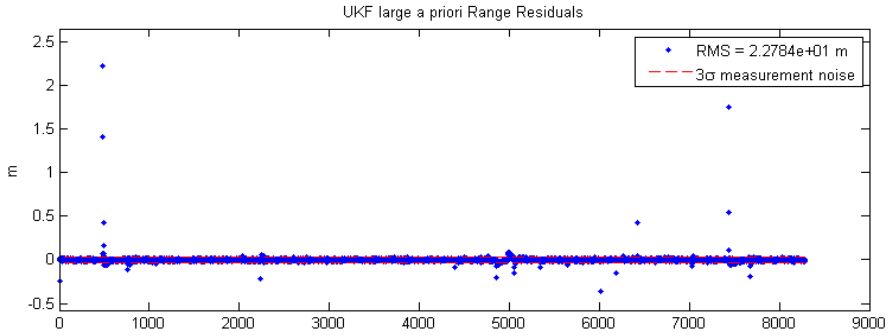
The case with the process noise performed the best for me. By inflating the covariance, the sigma points had a wider spread with which a better a priori state estimate could be found. This is despite having worse RMS error in each axis for position. These errors were bigger because there was more uncertainty – thus room for more error, compared to the no-Q situation where the covariance collapsed. The RMS was also heavily influenced by the initial state tracking near the true state (since J3 is a very small acceleration it takes time to build up the error). However, the residuals show that the measurements were listened to much better with the process noise. If these cases were ran on longer data sets, you would see the no-noise error RMS overtake the process-noise RMS.

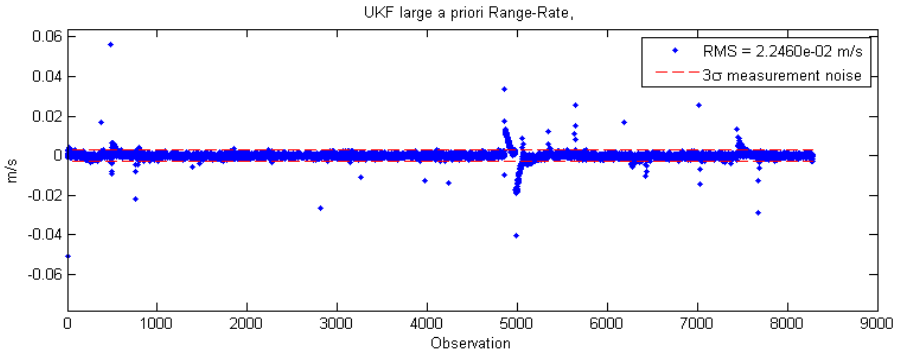
I believe the residuals on the process-noise case are a combination of those measurements being taken about an hour after the previous set, which allowed the process noise to grow a lot. Tuning should be done there. I also suspect that I should be processing measurements taken at the same time, all withing the same Y vector. That's because in the calculation of Pyy, there are bound to be cross-terms and I don't think the math will bear out to allow separate processing. Unfortunately the due date is near and I don't have time to investigate it more...

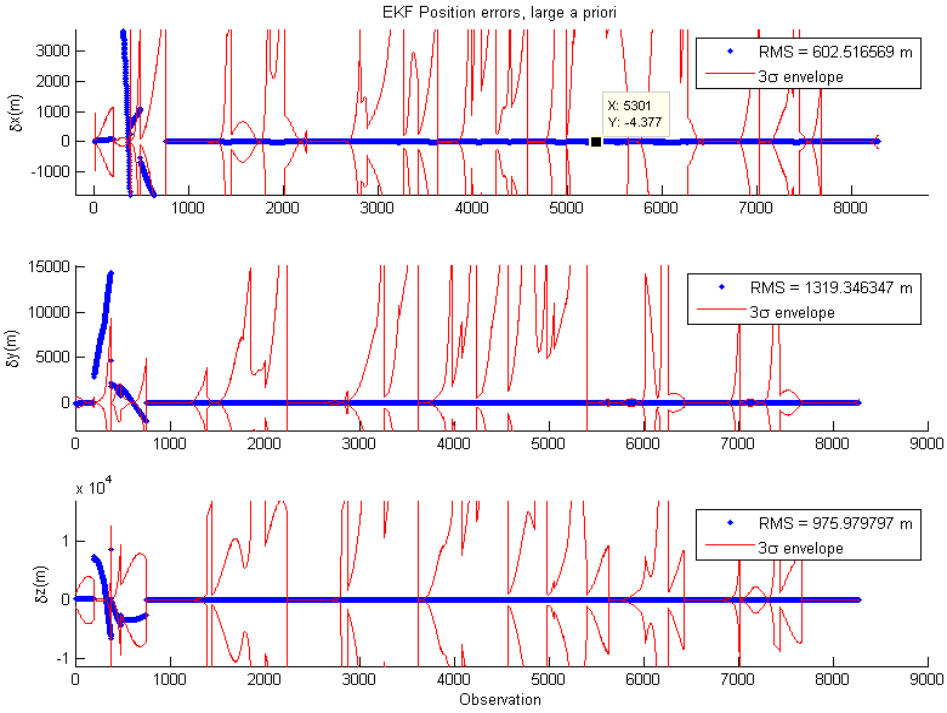
Compared to HW2, my residuals are worse with the UKF, but the UKF generally estimates closer to the true state (except around the 6k observation mark). It seems the nonlinearity of the UKF worked much better that the EKF's linearization method. I'd like to investigate if I can really bring down that large UKF error around the 5k observation mark.

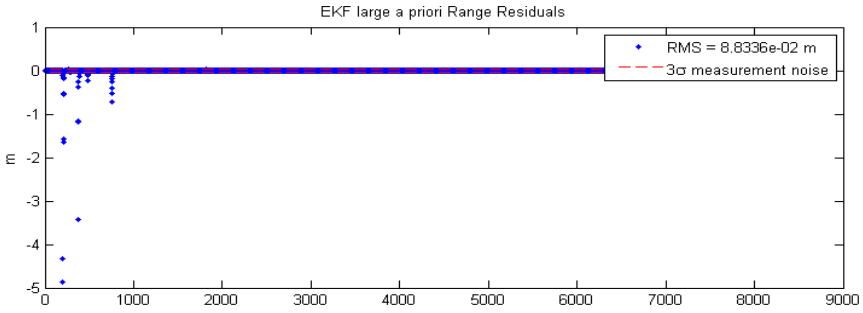
I tested UKF and EKF with a large a priori error (`state_i + [500;500;500;10;10;10], P = eye(6)*500;`). Results are below:

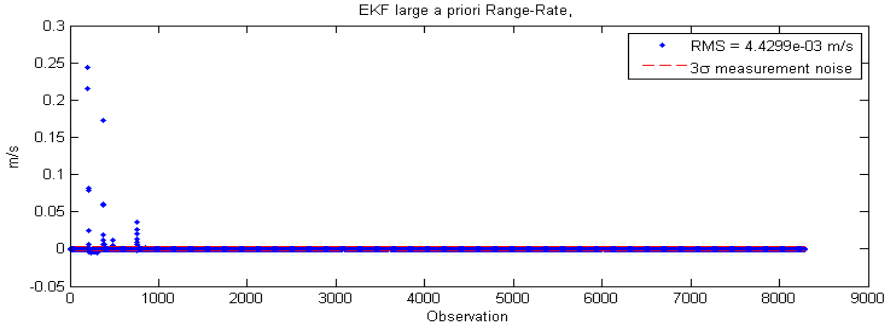






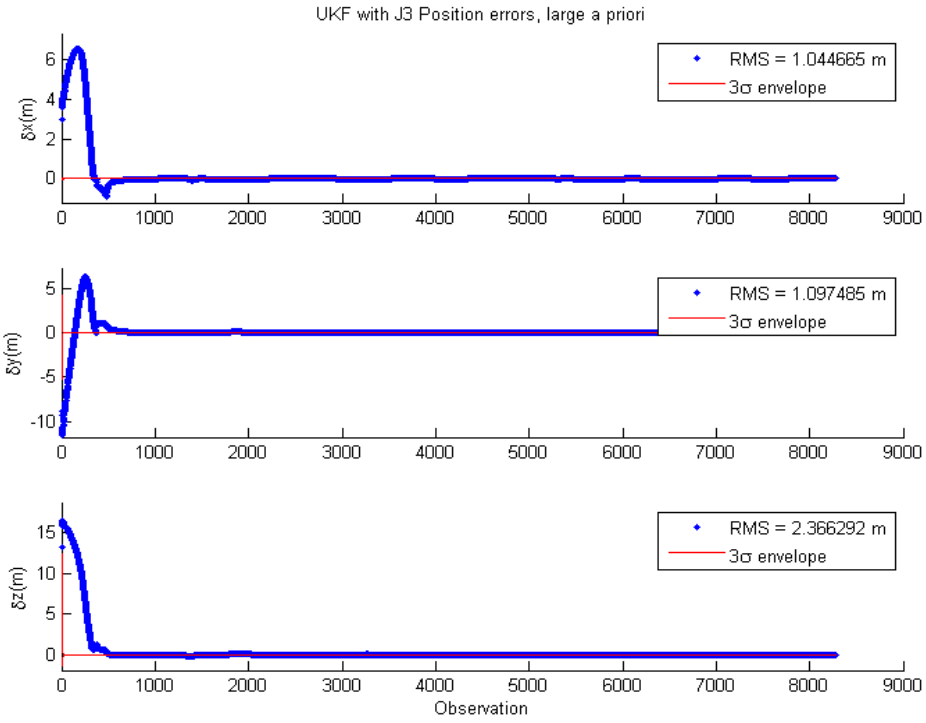

EKF Position errors, large a priori
RMS = 602.516569 m
3σ envelope
X: 5301
Y: -4.377
δx(m)
RMS = 1319.346347 m
3σ envelope
δy(m)
x 10^4
RMS = 975.979797 m
3σ envelope
δz(m)
Observation

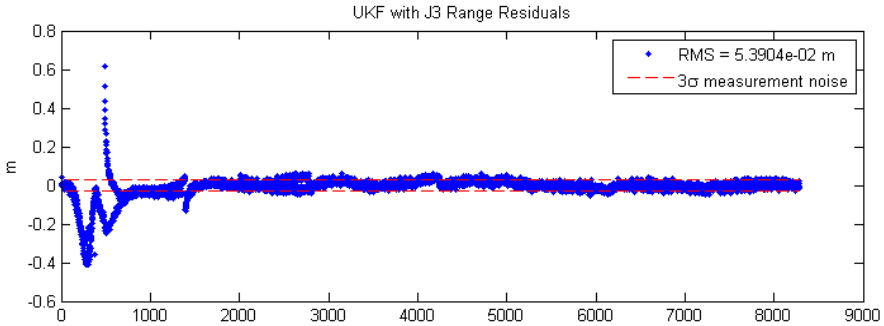

EKF large a priori Range Residuals
RMS = 8.8336e-02 m
3σ measurement noise
m

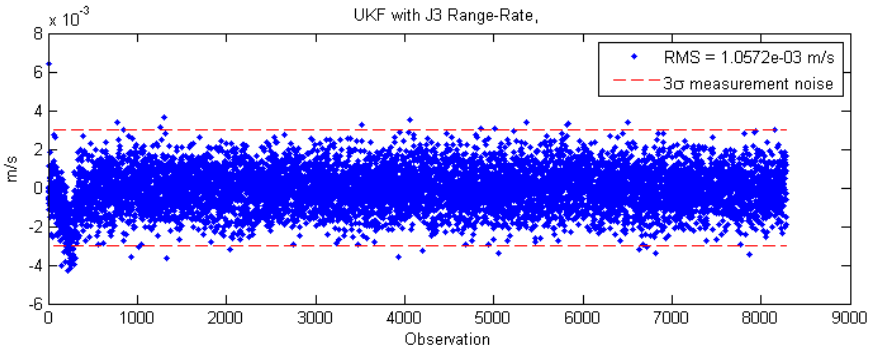

EKF large a priori Range-Rate,
RMS = 4.4299e-03 m/s
3σ measurement noise
m/s
Observation

While both filters had large residuals at the beginning, the EKF had more outside the noise envelope. The EKF also had much larger position error. The UKF, with its sigma-point spread and nonlinearity, was able to pick up on the initial state error and get pretty close. It didn't perform quite as well as the less-errored scenario. I would agree that the UKF is more robust to larger errors in initial conditions.

J3 was added to the filter, and SNC turned off. Results below:







Without the process noise, the covariance collapses immediately. However, the filter dynamics are accurate, so state error is removed pretty quickly for the best RMS. From the residuals one can see that the filter didn't really listen to the measurements much, especially at the beginning when the filter was removing the initial state error. After the error is removed, the residuals track well as a result of the

filter dynamics matching that of the modeled data.

It didn't take me long at all to implement this. I use the same propagator with a J3 option, so I just turned that on and made a wrapper function to call it 13 times for the propagated sigma points.