Kalman Filter Practice Tests

Tres Reid

Update Notation

x: Current state vector; x’ prediction state vector; x* new state vector

P: Covariance matrix (Uncertainties) - start with identity matrix. This gets updated as the state evolves. Used in the Kalman gain to weight how heavily the measured values are taken into consideration

F: Prediction matrix - Used to predict how the current state should evolve into the new (prediction) state

B: Control matrix- used to convert the control vector into something that can be added to the state vector to help give the new (prediction) state

u: control vector- A vector of known quantities that can be used to help evolve the current state into the new predicted state outside of what is given by the current state

Update Notation (Cont)

Z: measured data

R: sensor noise - uncertainties from measured data

G(K): Kalman gain [PH^T (HPH^T + R)^-1] - used to recursively compute the prediction error. As the gain gets smaller, there will be less of an influence by the observed data.

C (H): Makes the sensor data compatible with the state vector.

Q: external noise: Covariance matrix of the external noise (noise of control vector quantities for example). Added to the covariance matrix during update

Kalman filter

Kalman filters work in two main steps

1) Predict step: Use the current state and our prediction about how the state evolves to predict the new state
    a) $x' = Fx + Bu$ (new prediction state from old state)
    b) $P' = FPF^{-1} + Q$ (new prediction uncertainties from old uncertainties)
2) Update step: Use the sensor information along with uncertainties to adjust the prediction
    a) $x^* = x' + K(z-Hx')$ (new state from prediction state, measurement and kalman gain)
    b) $P^* = P' - KHP'$ (new uncertainties from prediction uncertainties and Kalman gain)
    c) $K = P'H^T (HP'H^T + R)^{-1}$ (Kalman gain from prediction uncertainties)

Tests

I’ve run 3 “classes” of tests to help my understanding of kalman filters (https://github.com/tresreid/Kalman-Filter-practice)

1) Test 0: scalar test
    a) to help understand basic concepts
2) Test 1-3: state vector of position and velocity.
    a) Created particle trajectories (of increasing complexity) as my truth data
    b) Imposed noise on that to create my measurements
    c) From those measurements, I used my kalman filter to get my predicted particle trajectory and compared that to the truth data.
3) Test 4: predicted trajectory using real gps data as my measurement

Test 0

A simple test without using matrices to get a sense of how the KF works. States are just scalar values

Truth data starts with an initial value of 1000 and takes on ¾ of the previous value with each step.

Measured data takes that truth value and adds a random number between (-200 and 200).

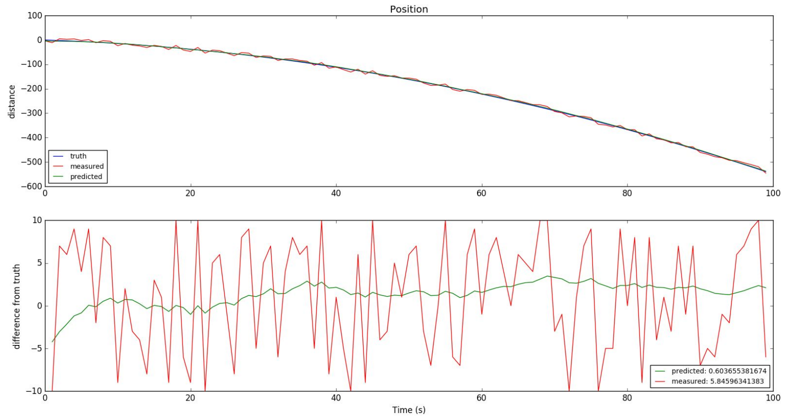
I then use the filter to make the predicted trajectory

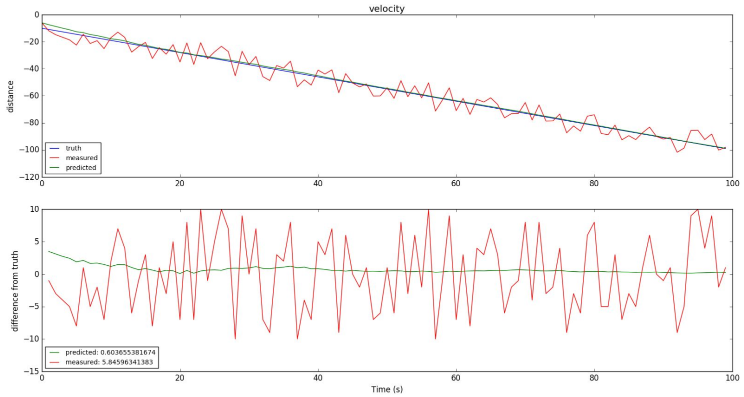
Test 1

State vector is now position and velocity.

I start with a random initial state (0<pos<10; -10<vel<10). I also make a constant random acceleration (between -10 and 10). My truth data takes this initial state with this acceleration and evolves it to give a particle trajectory. My measured data takes the truth data and imposes random noise at each step.

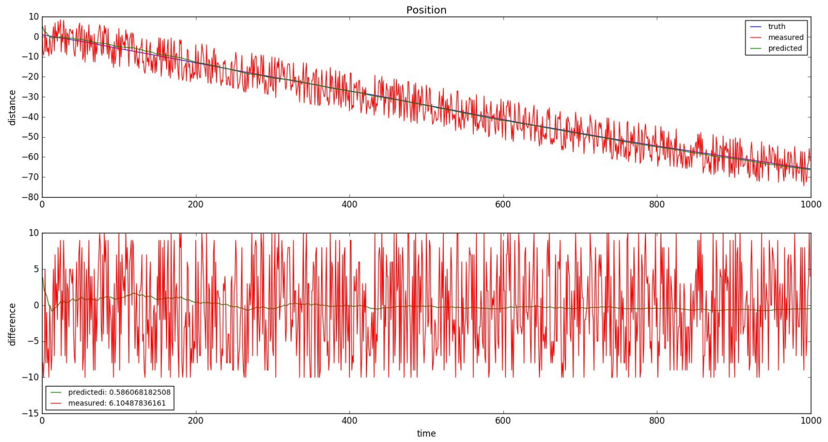
My filter starts out with a state that is equal to the first measured state. When the first measured state is far from the truth value, I found that the KF takes much longer to converge to the truth value (which makes sense). I assume the acceleration is the known value which is used as my control vector.

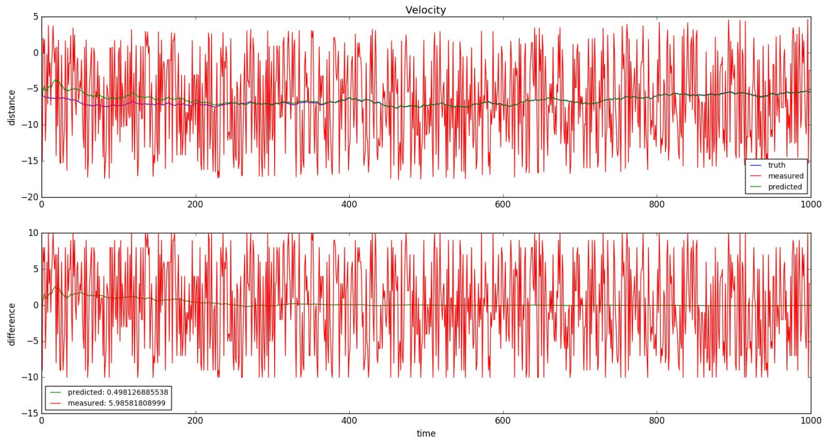




Test 2

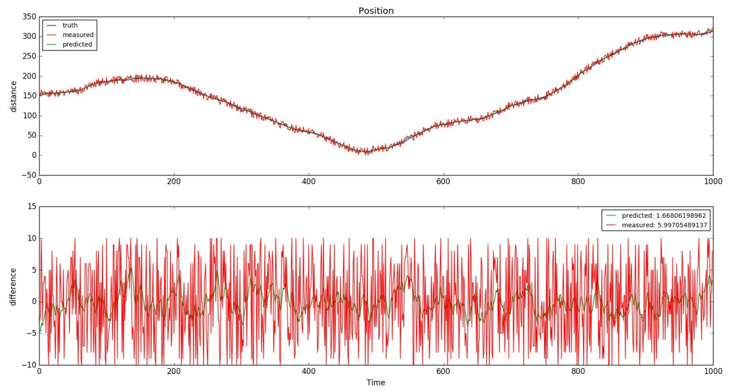
This is the same as test 1, however, instead of having constant acceleration, I randomly change my acceleration at each step. The acceleration is still used as my control vector. I found that the changing acceleration makes the KF take longer to converge to the truth data; thus, I had to extend the length of the run.

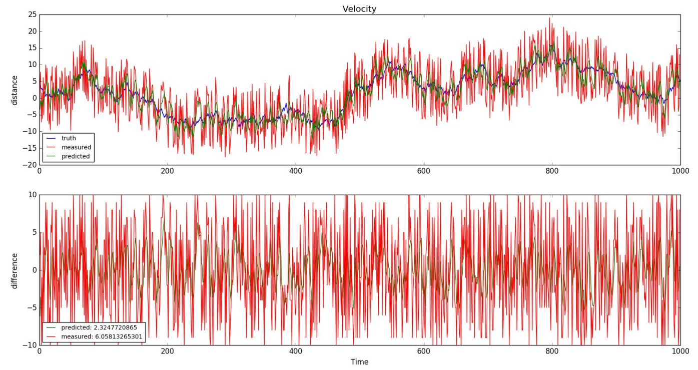




Test 3

Here I made the truth data with a greater time resolution. I evolve the truth state with .1*del_t time steps with additional random noise on the acceleration. I then completely change the acceleration at every del_t time step as before and take the measured value. Each new acceleration is considered as known, but the noise from the acceleration is not (contributes to Q). The results are much sloppier. You can see that the prediction no longer closely matches the truth. The standard deviation in the differences is also much higher than before.





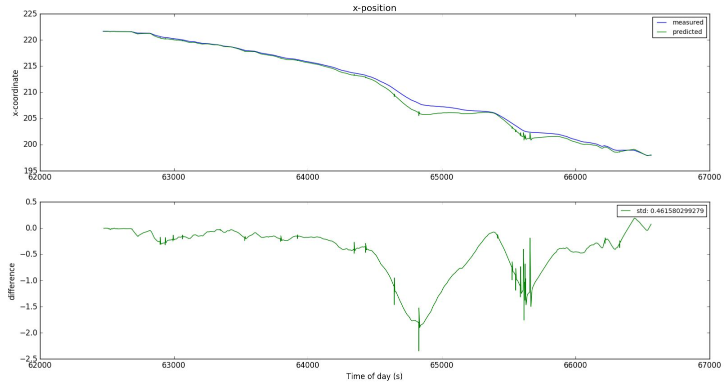
Test 4

Used my kalman filter to follow real gps data of a person driving back and forth.

The gps data was used as the measured data - only have x position, y-position and time to work with.

For my state, I used (current and previous) x position, x velocity, x- acceleration and time. I found that this gives me the best prediction, but it does make the matrices larger which would affect the run time (although that was not noticed here).

The prediction tends to stay fairly close to the measured data but there are these strange spikes that occur. I want to go back a “smooth” out the predicted track to account for this.



Next steps

1) Test 4 improvements
   a) I think I can modify test 4 so that I save less information in the state vector and so the matrices will be smaller decreasing the run time. This will also give me practice changing the C matrix.
   b) Include the y position information for test 4. Make 2 d trajectories
   c) Work out the track smoothing
2) Run with data closer to what would be used in CMS
   a) Helix trajectories
   b) Use same state as found in Fruhwirth Paper