Comp 150 Probabilistic Robotics Homework 1: Kalman Filter

James Staley

Question 1.1: textitwhat is the minimal state vector for the Kalman filter so that the resulting system is Markovian?

The minimal state vector holds the position and velocity of the system: $x = \{p_n, v_n\}$ where n is the number of tracked dimensions. Position and velocity depend on the history of the system, and the condition probability distribution of future states depends on only these two variables. The acceleration does not need to be included because it's a random variable and doesn't depend on time or any other variable.

Question 1.2: *Design the state transition probability function $p(x_t|u_t, x_{t-1})$. The transition function should contain linear matrices A and B and a noise covariance R.*

The state transition probability function gives the liklihood of being in state x_t given the previous state x_{t-1} and the command signal u_t. The Kalman Filter models all uncertainty as gaussian noise. So the new state x_t is actually the mean of a gaussian distribution given by eqn. 1, where A is a matrix that converts the previous state forward by one timestep, and B is a matrix that converts the command signal to changes in state. The covariance of the gaussian distribution is given in eqn. 2, where the noise matrix R is constructed from the variance of the expected noise, and the vector G which describes how the random variable (acceleration in this case) will affect each state variable (eqn. 3). The state transition probability function is given by the mean x_t and covariance Σ_t. The state is initialized to zero while the covariance is initialized to an identity matrix. The values for A, B, and G can be found in the top of the attached python file.

$$x_t = A_t x_{t-1} + B_t u_t \tag{1}$$

$$\Sigma_t = A_t \Sigma_{t-1} A_t^T + R_t \tag{2}$$

$$R = \sigma^2 G G^T \tag{3}$$

Question 1.3 + 1.4: *Implement the state prediction step of the Kalman fil-*

ter, assuming that at time $t = 0$, we start at rest, i.e., $x_t = \dot{x}_t = \ddot{x}_t = 0.0$. Use your code to calculate the state distribution for times $t = 1, 2, \ldots, 5$. For each value of t in the previous question, plot the joint posterior over x and $\dot{x}$ in a diagram where x is the horizontal and $\dot{x}$ is the vertical axis. For each posterior, you are asked to plot the uncertainty ellipse which is the ellipse of points that are one standard deviation away from the mean. Some additional information about uncertainty ellipses and how to calculate them using MATLAB or C++ can be found here: `http://www.visiondummy.com/2014/04/draw-error-ellipse-representing-covariance-matrix/`.

I the predicted position of the sailboat at each timestep and drew an ellipse axis aligned with the eigenvectors of the covariance matrix to show the uncertainty in position and velocity. The size of the ellipse corresponds to roughly one-standard deviation of uncertainty. I could not find a chi-squared table value needed for 1 standard deviation (n where $1 - n = 0.68$), so I used 0.75 as an approximation when calculating the size of the ellipse.

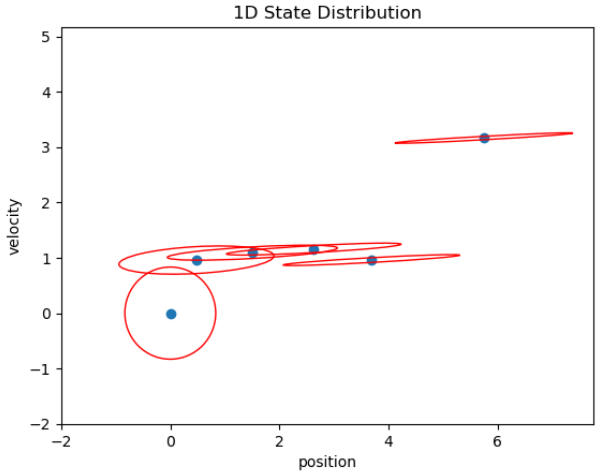

Figure 1: Five timesteps of prediction for the motion model with multivariate gaussian noise. Each sample has a confidence ellipse that shows how uncertain the estimate of position and velocity are for that prediction.

The above figure shows how uncertainty in the predictions evolves over time. Before any updates our uncertainty is a circle around our first point, but as we forward predict and incorporate multivariate noise the uncertainty stretches over position and rotates. We have much more uncertainty in our position than we do in our velocity because our position is influenced by the noise in velocity in addition to the noise from acceleration. Its two steps down the chain.

Question 2.1: *Define the measurement model. You will need to define matrices C and Q.*

The measurement model describes the likihood of observing a measurement z_t given the current state x_t, the sampled expected noise distribution δ_t, and matrix C_t which converts the state into the measurement [1]. In this case, C_t just passes through the spatial components of our state, since GPS data does not measure velocity. The exact values of C_t can be found in the attached python script.

$$z_t = C_t x_t + \delta_t \tag{4}$$

$delta_t$ is sampled from a multivariate gaussian distribution with mean 0 and variance σ^2_{gps}.

Question 2.2: *Implement the measurement update. Suppose at time $t = 5$, we query our sensor for the first time to obtain the measurement $z = 10$. State the parameters of the Gaussian estimate before and after incorporating the measurement. Afterwards, implement the sensor modal to randomly sample the true position, corrupted with noise σ^2_{gps}.*

The first five position updates drift around with estimates of the acceleration until the measurement of $z = 10$ relocates the estimate towards that position. The velocity estimate changes significantly as well because the agent must have been moving quickly to get to that position (i.e. the covariance matrix links the position and velocity). In the figure below, the dot in the top right is the prediction after the measurement update.

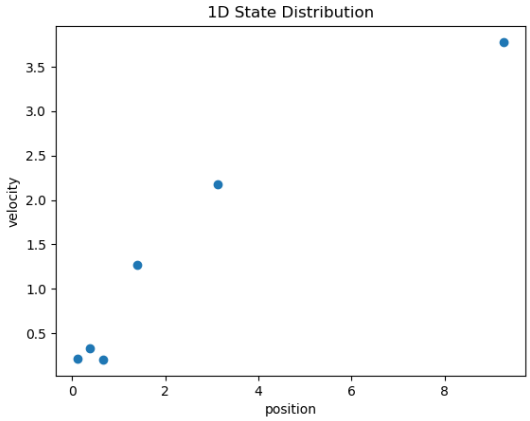

Figure 2: Five position updates without a measurement update followed by a single measurement of 10. The dot in the upper left reflects the measured estimate.

[1] C_t is a kxn matrix where k is the length of the measurement and n is the dimension of the state

Before the measurement $x = 3.1, 2.2$ and $\Sigma = [[67.2, 17.5][17.5, 6]]$. After the measurement $x = 9.3, 3.8$ and $\Sigma = [[7.1, 1.9], [1.9, 1.93]]$. The measurement moves the mean and reduces the variance.

Below is a sample position trajectory with measurement updates at every step. The Kalman Filter's estimate weaves back and forth but closely follows the ground truth thanks to the GPS.

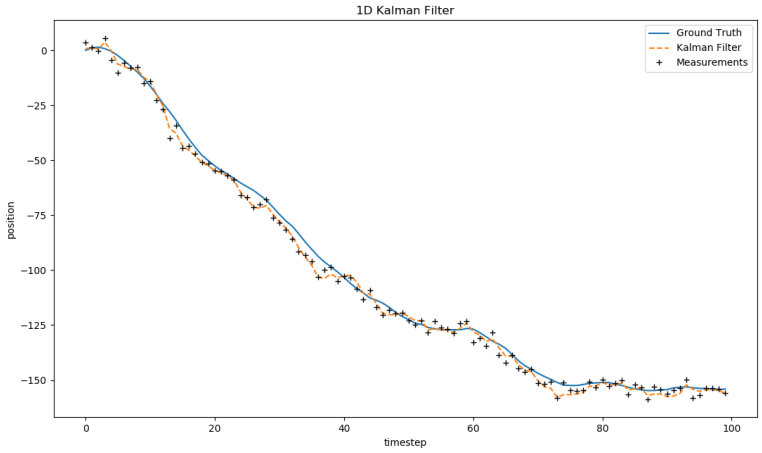

Figure 3: Position estimate with high-functioning GPS.

Question 2.3: *All of a sudden, the sky gets cloudy which may cause the sensor to fail and not produce a measurement with probability* $p_{gps-fail}$. *For three different values of this probability (e.g., 0.1, 0.5, and 0.9), compute and plot the expected error from the true position at time* $t = 20$. *You may do so by running up to* N *simulations and use the observed errors to obtain the expected error empirically.*

The following chart was produced by running 100 simulations at each error rate and averaging the error for each timestep.

Its notable that the error stays low and manageable for sensor failure rates up to 50%; the filter is robust to sensor error as long as it gets accurate readings occasionally. Once the failure rate reaches 90% its a different story. The Kalman filter error continuously rises as it does not have enough information to accurately estimate the state.

1 Extra Credit

For extra credit I modified the above system to work for 2 dimensions. Many of the matrices had to be modified along with other small changes, but its

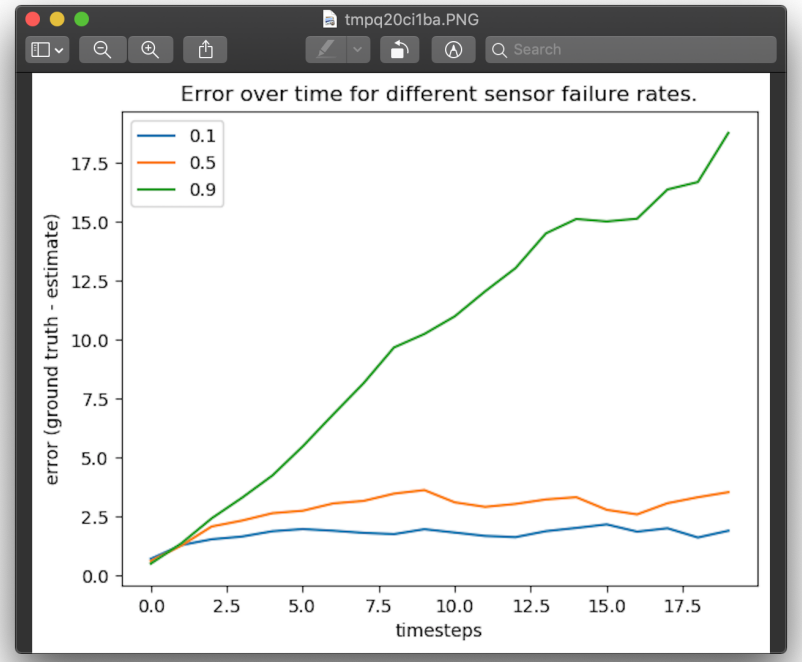

Figure 4: Error between ground truth and kalman filter estimation for various sensor failure rates.

conceptually identical to the 1D case.

The above figure just shows one trial, but the 2D filter seems much more error-prone than the 1D filter. It will typically track the ground truth well, but can get thrown off for long periods (as seen above). Its not clear to me whether this is a bug in my implementation or the increase in noise that comes with the larger state space.

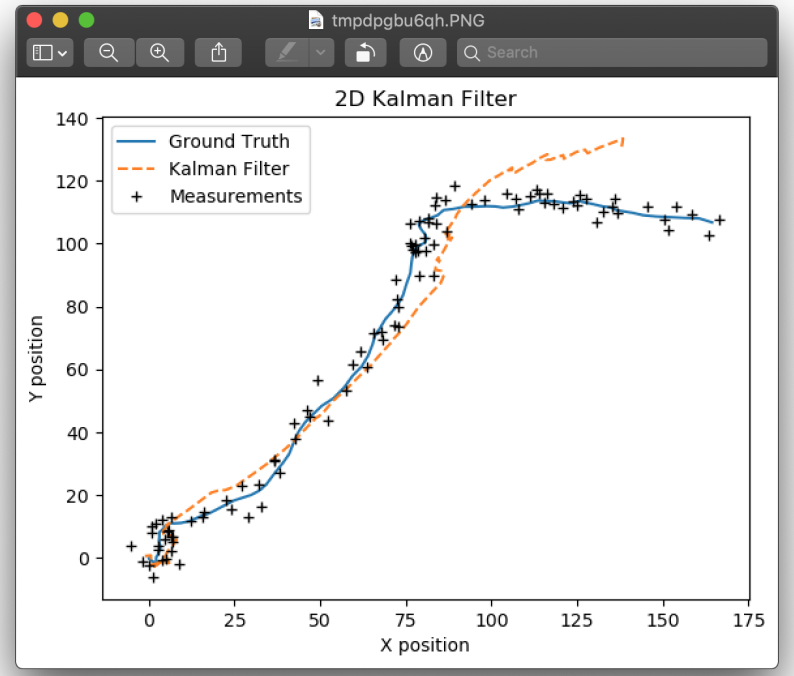

Figure 5: 2D Kalman Filter.