Kalman Filter Algorithm

Note: This section is currently under revision.

This section covers the Kalman Filter Algorithm. First we'll cover the State Space format of modeling and measuring a discrete-time dynamic system of estimated states, noisy inputs, and noisy measurements. Second, we'll explore all the different pieces of information about our system necessary to inform the algorithm. Third, the specific Kalman Filter Algorithm constructed based off of those parameters. Finally, we'll use some example state spaces and measurements to see how well we track.

Note: all images below have been created with simple Matlab Scripts. If seeing the code helps clarify what's going on, the .m files can all be found under internal location cs:localization:kalman.

Section 1 - State Space Format

The State Space Format is a universally standardized format for dynamic systems in the signals and controls community. In the continuous-time domain, the derivative of the state $\dot{x}(t)$ is a linear function of $x(t)$. In the discrete-time domain, where we'll be operating, the next state $x[k+1]$ is a linear function of the current state $x[k]$.

$$ X_{k+1} = AX_k + B(u_k + \sim\mathcal{N}(0,Q^2)) \\ Y_k = CX_k + \sim\mathcal{N}(0,\sigma_m^2) $$

Vector X_k is the State Vector which contains all states of the system at time-step k. These include things like position, velocity, orientation, voltage, etc. Matrix A is the Transmission Matrix, which contains the dynamics of the system, and calculates the next state given the current state. B is the input matrix, which describes the dynamics of inputs u.

Vector Q is the Process Noise of the system, which is the combination of the variance of the inputs σ_u^2 and an estimated degree of possible external forces σ_{ext}^2. $Q^2 = \sigma_u^2 + \sigma_{ext}^2$ Random forces from bumping into walls, random currents in the water, diver interaction, and other unpredictable perturbations. Put another way, Q describes the uncertainty involved when predicting into to future, even given perfect information about the present State. Were there no external forces, perfect actuators, and a perfect initial state x_0, the system state could be predicted perfectly into the future. This is obviously not the case in the real world, hence the need to specify uncertainty in the model, and thus in predicting the future states.

Vector Y_k is the measurement vector. It contains the values taken from the sensors. Matrix C is the Emission Matrix, which describes the linear function that relates the system state to the measurement values. σ_m is the noise vector which describes how noisy each individual sensor measurement is.

Section 2 - Kalman Filter Algorithm

The Kalman Filter is a two-stage process of prediction and measurement. First, based on the previous state estimate $\hat{X}_{k-1}$ and inputs u_{k-1}, an initial current state estimate $\hat{X}_{k'}$ is predicted. The confidence of the Previous Estimate is contained in the Covariance Matrix P From this estimate, a further estimate $\hat{Y}_k$ of what the sensors *should* be reading given $\hat{X}_{k'}$ is calculated using the Emission Matrix C.

Second, the true, noisy measurements Y_k are received and compared with the expected sensor values $\hat{Y}_k$. A compromise between the noisy measurements and the expected measurement is arrived at based upon the noise of the sensors σ_m and the uncertainty of the measurement predictions calculated from P. This compromise is encapsulated in a value $\hat{K}$ terms the 'Kalman Gain'. The official $\hat{X}_{k}$ is then calculated from the Emission Matrix C and the compromise of sensor values. The estimate of our state for this time step is made, and the process repeats to estimate the state X_{k+1} at the next time step.

$$ \hat{X}_{k'} = A\hat{X}_{k-1} + Bu_{k-1}, \qquad P_{k'} = AP_{k-1}A' + Q^2 \\ \hat{K} = P_{k'}H'/(HPH'+diag(\sigma_m^2)) \\ \hat{X}_k = \hat{X}_{k'} + \hat{K}(Y_k - C\hat{X}_{k'}),\qquad P_k = (I_{n} - \hat{K}C)P_{k'} $$

<inline code example to be explicit>

Section 3 - Kalman Filter Initialization

The above update loop requires initialization. Each execution of the loop relies on the model itself, and values from the previous loop. Upon start-up, any program must first build the model. To do so, we must construct our state vector X, construct our Transition Matrix A and Input Matrix B that describe the physical dynamics of the system, and construct an emission matrix C that ties our measured sensor values Y to be functions of the state X. As you will likely be changing the number of sensors and motors, as the submarine is tinkered with and modules are added, removed, or temporarily disabled, it is advisable to make this a dynamic process that takes information from a calibration file and constructs the B and C matrices.

Process Noise Q and sensor noise σ_m must be estimated. Q should likely be loaded from a calibration file, and σ_m dynamically created alongside C with information from such a file.

Once the model is created, initial parameters must be selected. A good default is to set your state estimate $\hat{X}_0$ to be all zeros, or their equivalent. For instance, if the 9 elements of the 3×3 orientation matrix are part of your state, they would be set as an identity matrix I_3. This initial guess at our state is accompanied by a the covariance matrix P_0 being a diagonal matrix of *very large* numbers. This will ensure that all predictions of the estimated state from the model are ignored in the face of *any* sensor measurement that might have some semblance of an idea of what the state is. The model should still begin to produce reliable results and be taken seriously in a short number of time steps, as even after a single measurement, the state estimate $\hat{X}$ has a better (smaller) variance than any sensor measurement, and raw sensor measurements typically are pretty good on their own.

Finally, calibration values must be estimated or directly measured and stored upon start-up. Things like the specific orientation and magnitude of the local magnetic field, the magnitude of gravity (direction is always tautologically 'down' along the z axis), and the drift of individual gyroscopes. These parameters, among others, are vital to the utilization of your sensor data to estimate the world, and are liable to change with location or time. Often times calculating these parameters require specific manipulation of the submarine - such as keeping the submarine perfectly still for measuring gyroscopic drift - and should potentially be made part of an executable calibration-script to be run at the push of a button once the submarine is appropriately staged.

Other sections will elaborate on how to build these matrices and perform these calibrations. Below is a simple example of a kalman filter code - the mechanics themselves are quite simple to set up.

Section 4 - Simulation

In the simulation, the vehicle on the 2D plane has thrusters to let it control orientation and forward motion. The process noise in the system is relatively low, but the sensors are noisy and ping less frequently. From a constant starting point and random orientation, the submarine attempts to reach its target, meandering less as it becomes more confident in its knowledge of the world.