Kalman Filters

https://github.com/tobybreckon/python-examples-cv/blob/master/kalman_tracking_live.py

What are Kalman Filters?

- Sensor fusion and data fusion algorithms
- Also known as linear quadratic estimation (LQE)
- Is an algorithm that uses a series of measurements observed over time to produce estimates of unknown variables by estimating a joint probability distribution over variables of each time frame.
    - Joint probability distribution
        - Given two random variables defined in the same probability space, the joint probability distribution is the corresponding probability distribution on all possible pairs of outputs.
    - These measurements can include statistical noise and other inaccuracies
    - These estimates tend to be more accurate than if the algorithm was on only based on one measurement
- Two Phase Process
    - Prediction phase
        - Produces estimates of the current state variables with their uncertainties
        - Uses the state estimate from previous timestep to produce estimate of state at current timestep (a priori)
    - Update Phase
        - The difference between the current estimate of state and current observation information is multiplied by the optimal Kalman gain and then combined with the previous state estimate to refine the current state estimate.
        - Improved state estimate (a posteriori)
    - Using example below

Predict [edit]

Predicted (*a priori*) state estimate	$\hat{\mathbf{x}}_{k\mid k-1} = \mathbf{F}_k\mathbf{x}_{k-1\mid k-1} + \mathbf{B}_k\mathbf{u}_k$
Predicted (*a priori*) estimate covariance	$\hat{\mathbf{P}}_{k\mid k-1} = \mathbf{F}_k\mathbf{P}_{k-1\mid k-1}\mathbf{F}_k^\mathsf{T} + \mathbf{Q}_k$

Update [edit]

Innovation or measurement pre-fit residual	$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k\mid k-1}$
Innovation (or pre-fit residual) covariance	$\mathbf{S}_k = \mathbf{H}_k\hat{\mathbf{P}}_{k\mid k-1}\mathbf{H}_k^\mathsf{T} + \mathbf{R}_k$
Optimal Kalman gain	$\mathbf{K}_k = \hat{\mathbf{P}}_{k\mid k-1}\mathbf{H}_k^\mathsf{T}\mathbf{S}_k^{-1}$
Updated (*a posteriori*) state estimate	$\mathbf{x}_{k\mid k} = \hat{\mathbf{x}}_{k\mid k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k$
Updated (*a posteriori*) estimate covariance	$\mathbf{P}_{k\mid k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\,\hat{\mathbf{P}}_{k\mid k-1}$
Measurement post-fit residual	$\tilde{\mathbf{y}}_{k\mid k} = \mathbf{z}_k - \mathbf{H}_k\mathbf{x}_{k\mid k}$

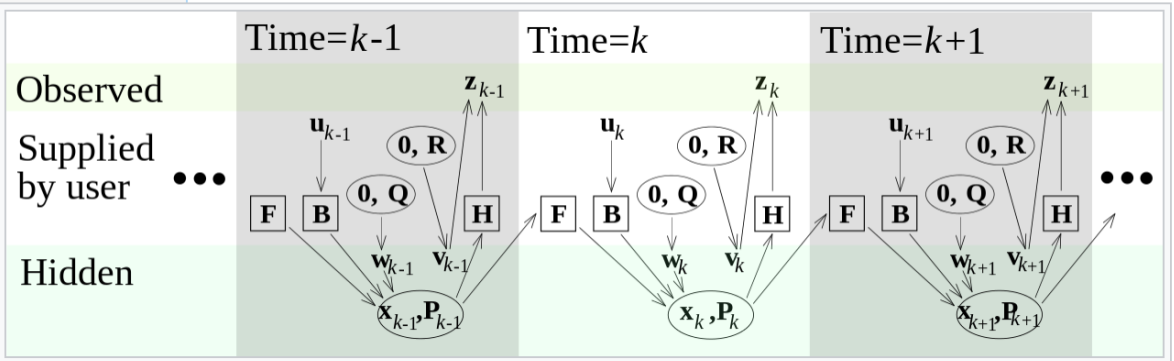
-

How do they work?

- Uses a system's dynamic model, known control input to that system, and multiple sequential measurements to form an estimate of a system's varying states.
- The Kalman filter produces an estimate of the state of the system as an average of the system's predicted state and of the new measurement using a weighted average.
    - Weights are so that the more certain values are "trusted" more
    - Weights calculated by covariance
        - Measure of the estimated uncertainty of the prediction of the system's state
- Works recursively
    - Requires only best guess rather than entire history of the system's state to calculate a new state
- Kalman Filter's gain
    - Weight given to measurements and current state estimate
    - Can be adjusted to achieve a certain performance
    - High gain = more weight on recent measurements
        - Conforms more responsively to recent measurements
        - Extremes: high gain = more jumpy estimated trajectory
    - Low gain
        - Conforms to model predictions
        - Extremes: low gain = smooth out noise but reduces responsiveness
    - State estimate and covariances are put into matrices because of calculations involving multiple dimensions
- Based on a linear dynamic systems discretized in the time domain.

In order to use the Kalman filter to estimate the internal state of a process given only a sequence of noisy observations, one must model the process in accordance with the following framework. This means specifying the matrices, for each time-step k, following:

- $\mathbf{F}_k$, the state-transition model;
- $\mathbf{H}_k$, the observation model;
- $\mathbf{Q}_k$, the covariance of the process noise;
- $\mathbf{R}_k$, the covariance of the observation noise;
- and sometimes $\mathbf{B}_k$, the control-input model as described below; if $\mathbf{B}_k$ is included, then there is also
- $\mathbf{u}_k$, the control vector, representing the controlling input into control-input model.



Model underlying the Kalman filter. Squares represent matrices. Ellipses represent multivariate normal distributions (with the mean and covariance matrix enclosed). Unenclosed values are vectors. For the simple case, the various matrices are constant with time, and thus the subscripts are not used, but Kalman filtering allows any of them to change each time step.

The Kalman filter model assumes the true state at time k is evolved from the state at (k – 1) according to

$$\mathbf{x}_k = \mathbf{F}_k\mathbf{x}_{k-1} + \mathbf{B}_k\mathbf{u}_k + \mathbf{w}_k$$

where

- $\mathbf{F}_k$ is the state transition model which is applied to the previous state $\mathbf{x}_{k-1}$;
- $\mathbf{B}_k$ is the control-input model which is applied to the control vector $\mathbf{u}_k$;
- $\mathbf{w}_k$ is the process noise, which is assumed to be drawn from a zero mean multivariate normal distribution, $\mathcal{N}$, with covariance, $\mathbf{Q}_k$: $\mathbf{w}_k \sim \mathcal{N}(0, \mathbf{Q}_k)$.

At time k an observation (or measurement) $\mathbf{z}_k$ of the true state $\mathbf{x}_k$ is made according to

$$\mathbf{z}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k$$

where

- $\mathbf{H}_k$ is the observation model, which maps the true state space into the observed space and
- $\mathbf{v}_k$ is the observation noise, which is assumed to be zero mean Gaussian white noise with covariance $\mathbf{R}_k$: $\mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R}_k)$.

The initial state, and the noise vectors at each step {$\mathbf{x}_0$, $\mathbf{w}_1$, ..., $\mathbf{w}_k$, $\mathbf{v}_1$, ... ,$\mathbf{v}_k$} are all assumed to be mutually independent.

Many real-time dynamic systems do not exactly conform to this model. In fact, unmodeled dynamics can seriously degrade the filter performance, even when it was supposed to work with unknown stochastic signals as inputs. The reason for this is that the effect of unmodeled dynamics depends on the input, and, therefore, can bring the estimation algorithm to instability (it diverges). On the other hand, independent white noise signals will not make the algorithm diverge. The problem of distinguishing between measurement noise and unmodeled dynamics is a difficult one and is treated as a problem of control theory using robust control.[23][24]

-

Extended Kalman Filter

- State transition and observation models don't need to be linear function of the state
    - Can be nonlinear functions of differentiable type

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k$$
$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k$$

The function *f* can be used to compute the predicted state from the previous estimate and similarly the function *h* can be used to compute the predicted measurement from the predicted state. However, *f* and *h* cannot be applied to the covariance directly. Instead a matrix of partial derivatives (the Jacobian) is computed.

At each timestep the Jacobian is evaluated with current predicted states. These matrices can be used in the Kalman filter equations. This process essentially linearizes the nonlinear function around the current estimate.

-

Implementing Kalman Filters

- We want to first implement a Guassian function which returns the gaussian value

```
# import math functions
from math import *
import matplotlib.pyplot as plt
import numpy as np

# gaussian function
def f(mu, sigma2, x):
    ''' f takes in a mean and squared variance, and an input x
       and returns the gaussian value.'''
    coefficient = 1.0 / sqrt(2.0 * pi *sigma2)
    exponential = exp(-0.5 * (x-mu) ** 2 / sigma2)
    return coefficient * exponential
```

-

- If multiply two Gaussians, as in Bayes rule, a prior and a measurement probability
    - Prior has a mean of Mu and a variance of Sigma square, and the measurement has a mean of Nu, and covariance of r-square
- Then, the new mean, Mu prime, is the weighted sum of the old means
- The Mu is weighted by r-square, Mu is weighted by Sigma square, normalized by the sum of the weighting factors. The new variance term would be Sigma square prime
- Clearly, the prior Gaussian has a much higher uncertainty, therefore, Sigma square is larger and that means the nu is weighted much much larger than the Mu
    - So the mean will be closer to the nu than the mu
    - Variance term is unaffected by the actual means, it just uses the previous variances

- Update function implementation

```
# the update function
def update(mean1, var1, mean2, var2):
    ''' This function takes in two means and two squared variance terms,
        and returns updated gaussian parameters.'''
    # Calculate the new parameters
    new_mean = (var2*mean1 + var1*mean2)/(var2+var1)
    new_var = 1/(1/var2 + 1/var1)

    return [new_mean, new_var]
```

- Motion is the change that occurred between the old mean and the new mean
    - A new mean is your old mean plus the motion often called u

```
# the motion update/predict function
def predict(mean1, var1, mean2, var2):
    ''' This function takes in two means and two squared variance terms,
        and returns updated gaussian parameters, after motion.'''
    # Calculate the new parameters
    new_mean = mean1 + mean2
    new_var = var1 + var2

    return [new_mean, new_var]
```

- Need a main function that takes in the update and predict functions, and feeds them into a sequence of measurements and motions

```
for n in range(len(measurements)):
    # measurement update, with uncertainty
    mu, sig = update(mu, sig, measurements[n], measurement_sig)
    print('Update: [{}, {}]'.format(mu, sig))
    # motion update, with uncertainty
    mu, sig = predict(mu, sig, motions[n], motion_sig)
    print('Predict: [{}, {}]'.format(mu, sig))
```

- As measurements are updated over time, the uncertainty trends downwards and the measurement trends closer to the actual value of 10

```
Update: [4.998000799680128, 3.99840063974410 23]
Predict: [5.998000799680128, 5.998400639744102]
Update: [5.999200191953932, 2.399744061425258]
Predict: [6.999200191953932, 4.399744061425258]
Update: [6.999619127420922, 2.0951800575117594]
Predict: [8.999619127420921, 4.09518005751176]
Update: [8.999811802788143, 2.0235152416216957]
Predict: [9.999811802788143, 4.023515241621696]
Update: [9.999906177177365, 2.0058615808441944]
Predict: [10.999906177177365, 4.005861580844194]
```

- Gaussian plot of resulting data