Kalman Filter in C++

Introduction to Kalman Filters

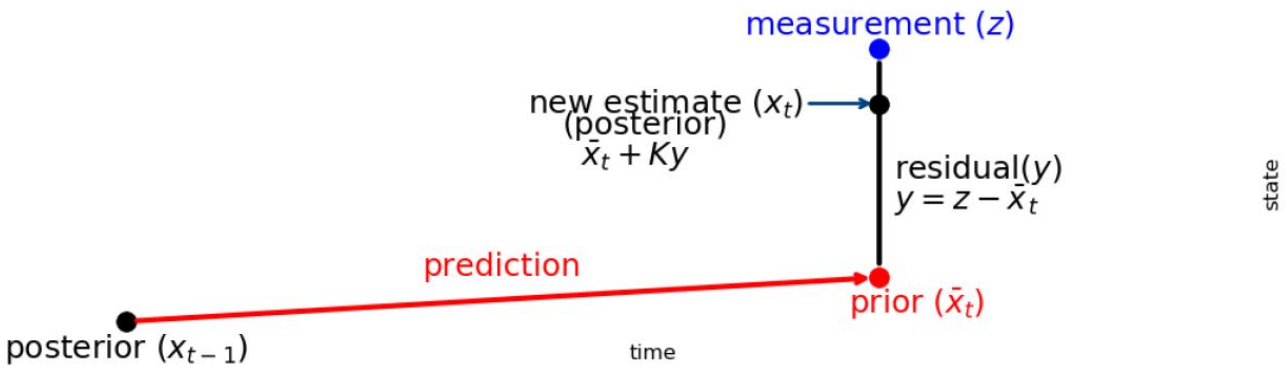
To estimate the state of a dynamic system, e.g., the map location of a moving vehicle, or the incidence of an infectious disease, scientists and engineers rely on measurements of the observable aspects of the system. However, sensors and other methods of data collection have sources of inaccuracy that introduce noise to our measurements.

A Kalman Filter is a recursive method to produce an optimal state estimation for a linear dynamic system with Gaussian noise. It combines information from a model of a dynamic system, and measurements of the system's observable state—both assumed to be noisy—to compute the best estimate under a set of assumptions. In practice, the basic Kalman Filter algorithm is simple to implement in a computer program, and the filters thereby created can execute quickly, with a small amount of memory, and perform well on problems that can be reasonably modelled as linear systems with Gaussian noise.

The basic Kalman Filter algorithm is a recursion of the following two steps.

- **Prediction:** Use a state transition model to predict the next state, based on the previous estimate.
- **Update:** Compute the residual of the prediction and a measurement, and produce a new estimate along the residual.

The prediction and update steps can be interpreted as the computation of a prior and posterior, respectively, in a recursive Bayesian estimation. This recursion is illustrated for a one-dimensional state variable in the book Kalman and Bayesian Filters in Python (https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python), by Roger R. Labbe, as follows.



Essentially, an iteration of this algorithm produces an estimate of a system's state as a weighted average of a prediction and a measurement. The prediction and the measurement are represented as the means of Gaussians, whose covariances determine the weights. The next section provides equations for the prediction and update steps. For derivations of the equations, and intuitive explanations of Kalman Filter theory, see the book linked to above.

Kalman Filter Equations

This section provides equations for a multivariate Kalman Filter's predict and update steps. Bold symbols indicate matrices.

Prediction Step

Here are the symbols used in the following equations, which can be modified with a bar to indicate predictions, or priors:

- $\mathbf{x}$: The state estimate as a $N \times 1$ vector of state variables.
- $\mathbf{P}$: The $N \times N$ state covariance matrix.
- $\mathbf{F}$: The $N \times N$ state transition matrix.
- $\mathbf{Q}$: The $N \times N$ process covariance matrix.

The state is modelled as a multivariate Gaussian with mean $\mathbf{x}$ and covariance $\mathbf{P}$. The state transition matrix $\mathbf{F}$ is a linear operator that models the evolution of the state variables $x_i, i \in [0, N-1]$ after a discrete time interval. The process covariance matrix $\mathbf{Q}$ models the uncertainty in the state transition model as additive white Gaussian noise.

$$\bar{\mathbf{x}} = \mathbf{F}\mathbf{x}$$
$$\bar{\mathbf{P}} = \mathbf{F}\mathbf{P}\mathbf{F}^{\mathbf{T}} + \mathbf{Q}$$

Notice that the prediction step can only add to the uncertainty in the state (increase the covariance $\mathbf{P}$.)

Update Step

The spaces from which values of state and measurement vectors may be drawn, i.e., the state-space and measurement-space, respectively, may be different for two primary reasons:

- the state vector $\mathbf{x}$ includes latent, or unobservable variables.
- measurements are of a different quality, or unit to the state variables.

Therefore, in a Kalman Filter, it is necessary to convert from the state-space to the measurement-space. Keep this slighltly complicating factor in mind when considering the equations below.

Here are the symbols used in the following equations:

- $\mathbf{z}$: A measurement as a $M \times 1$ vector of measurement variables.
- $\mathbf{R}$: The $M \times M$ measurement noise covariance matrix.
- $\mathbf{H}$: The measurement function as an $M \times N$ matrix.
- $\mathbf{y}$: The residual of the prediction and measurement as a $M \times 1$ vector.
- $\mathbf{S}$: The $M \times M$ system covariance matrix.
- $\mathbf{K}$: The Kalman gain as a $N \times M$ matrix.

A measurement is modelled as a multivariate Gaussian with mean $\mathbf{z}$ and covariance $\mathbf{R}$. The measurement function $\mathbf{H}$ is a linear operator that models the projection from state-space into measurement-space. The residual $\mathbf{y}$ is the difference between the measurement, and the prediction projected into measurement-space. The system covariance matrix $\mathbf{S}$ is a normalisation value, the sum total of uncertainties in the prediction and measurement. The Kalman gain $\mathbf{K}$ is the factor that determines the weight given to the measurement versus the prediction in computing the new estimate as a weighted average of the two. It is calculated as a ratio of uncertainties in the prediction and measurement.

$$\mathbf{y} = \mathbf{z} - \mathbf{H}\bar{\mathbf{x}}$$
$$\mathbf{S} = \mathbf{H}\bar{\mathbf{P}}\mathbf{H}^{\mathbf{T}} + \mathbf{R}$$
$$\mathbf{K} = \bar{\mathbf{P}}\mathbf{H}^{\mathbf{T}}\mathbf{S}^{-\mathbf{1}}$$
$$\mathbf{x} = \bar{\mathbf{x}} + \mathbf{K}\mathbf{y}$$
$$\mathbf{P} = (\mathbf{I} - \mathbf{K}\mathbf{H})\,\bar{\mathbf{P}}$$

Notice that the update step can decrease the uncertainty in the state by incorporating new information from the measurement into the prediction.

The next section proposes a C++ class to encapsulate the Kalman Filter predict and update steps described mathematically above.

Implementation in C++

This section offers a design of a C++ class to implement the basic Kalman Filter algorithm as described in previous sections.

Why Use C++?

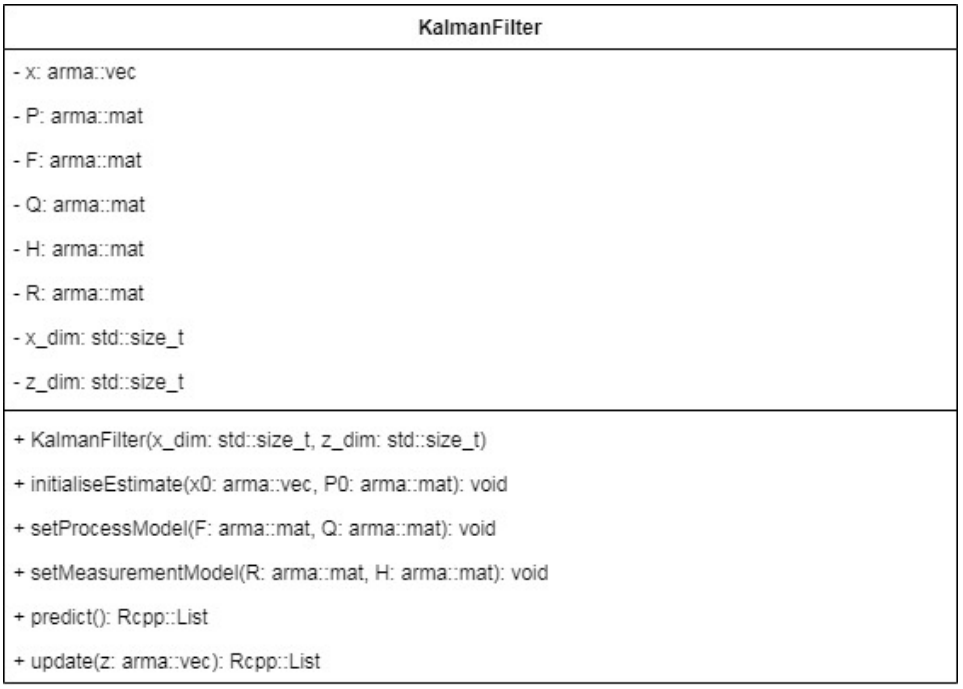
The basic Kalman Filter algorithm is simple to implement in any programming language with support for linear algebra, so why choose C++? There is an essential trade-off between a compiled language, like C++, and an interpreted language, like R: efficiency of execution versus efficiency of development. R programmers benefit from an interactive programming environment, a very high-level computing model, and loose constraints on the use and mixing of different types of data, to make program development a fast, iterative process. On the other hand, C++ programmers benefit from highly optimised compilers, a computing model that offers precise control over a computer's resources, and a strong type system, to enable completed programs to execute very efficiently, and on constrained systems. Beyond this trade-off, the following arguments can also be made for using C++ in various circumstances:

- C++ is well-suited to the development of robust, and safety-critical software.
- The vast majority of devices, from PCs to embedded systems, support programming in C++.
- Many other programming languages provide an API to allow C++ libraries to be used in their programs.

C++ KalmanFilter Class Design

In C++, the class mechanism is the primary means to define new types. A class encapsulates variables that encode the state of an entity the programmer wishes to model. The `KalmanFilter` class has member variables that correspond to matrices used in the equations of the previous section. By "encapsulate," I mean that the `KalmanFilter` class prevents direct access to these variables, and maintains some invariant conditions on the values of the variables to ensure the filter is always in a valid state, e.g., the matrices have the correct dimensions. The `KalmanFilter` class provides accessible member functions as an interface to the Kalman Filter algorithm. The implementations of these functions modify the member variables to track the state of the Kalman Filter, and they return values of interest.

Below is a class diagram that describes the member variables (middle section,) and member functions (bottom section,) of a C++ class called `KalmanFilter` (named in the top section.)



The `KalmanFilter` class uses types from the Armadillo (http://arma.sourceforge.net/) library for linear algebra, and the Rcpp (http://rcpp.org/) package for extending R with C++ programs.