# Understanding the Kalman Filter

Robert Bragg
Computer Science & Electronics
University of Bristol
rb13234@my.bristol.ac.uk

## Abstract

The Kalman filter is considered an optimal algorithm for taking Gaussian noisy measurements and producing estimations of unknown variables for linear systems. This paper aims to present the Kalman filter in an intuitive, easy to follow manner and to demonstrate its application in the case of one dimensional movement.

## 1 Introduction

Developed by Rudolf E. Kálmán and published in 1960[1], the filter uses Bayesian inference and joint probability distribution to estimate new data in discreet time. The algorithm uses two principal steps; prediction of the new state from the old state along with any system inputs, followed by incorporation of all measurements taken. At all times the variance, or uncertainty, of the system is stored. Although this variance is not assumed to necessarily be Gaussian, it does yield an exact conditional probability estimate for that case.

In essence the process can be thought of as a form of weighted averaging where extrapolations or measurements with more certainty are given higher weighting.

## 2 Matrix Representation Preface

A good basis in linear algebra is advised when looking to implement the Kalman filter and while this paper will not go into the derivations of the filter, the reader should have at least a fundamental understanding of matrices and their purpose. Going through the calculations by hand with a very simple example such as the one used below is an excellent way to get a more intuitive feel for how this algorithm works; while the equations may seem fairly abstract, looking at how they behave with real numbers can help to see in what ways the matrices interact with one another.

## 3 Understanding the problem

The Kalman filter sees the world as a collection of variables with Gaussian distributed values and relies upon combination of these distributions to function. Therefore key to understanding and later implementing the filter is proper characterisation of the state and the measurements. Each variable has a mean $\mu$ representing the best estimate and a variance $\sigma^2$ modelling the uncertainty.

The state is the filter's representation of the variables to be estimated. For in-

stance, imagine the simple case where the one-dimensional position and velocity of an object needs to be estimated e.g. a train on a track. The state here would be a vector containing the estimation means as shown:

$$\chi = \begin{vmatrix} pos \\ vel \end{vmatrix}$$

with an associated covariance matrix:

$$P = \begin{vmatrix} \Sigma_{pp} & \Sigma_{pv} \\ \Sigma_{vp} & \Sigma_{vv} \end{vmatrix}$$

Let's also imagine that there are two sensors, one measuring the position of the train along the track in meters(m), and one measuring the velocity of the train in meters per second(m/s). Their values will be stored in the vector:

$$z = \begin{vmatrix} posMeas \\ velMeas \end{vmatrix}$$

also with an associated covariance matrix:

$$R = \begin{vmatrix} \Sigma_{pMpM} & \Sigma_{pMvM} \\ \Sigma_{vMpM} & \Sigma_{vMvM} \end{vmatrix}$$

It is worth noting that the accuracy of the filter relies heavily on the accuracy of the covariance matrices and hence on proper characterisation of the state and the measurement apparatus.

From here the filter proceeds sequentially with step *n* being derived from step *n-1* combined with the control inputs and measurement readings.

# 4 Step by Step Analysis of the Filter

The Kalman filter only ever uses the last time step to calculate the next, along with any information gained through control inputs or sensors. An initial state is required to begin the algorithm; while an accurate starting point is certainly very helpful, the filter will converge towards the true state fairly quickly.

## 4.1 Step One: Extrapolation with Control Input

### 4.1.1 Extrapolation

From the state values in the previous time step we can extrapolate to where we expect the state values to be now. Intuitively, the previous position and velocity will have an effect on the new position while only the previous velocity will effect the new velocity. Formally written from simple kinematic principals:

$$pos_n = pos_{n-1} + \Delta t vel_{n-1}$$

$$vel_n = vel_{n-1}$$

In matrix form:

$$\begin{vmatrix} pos_n \\ vel_n \end{vmatrix} = \begin{vmatrix} 1 & \Delta t \\ 0 & 1 \end{vmatrix} \begin{vmatrix} pos_{n-1} \\ vel_{n-1} \end{vmatrix}$$

This transformational matrix is known as the state transition matrix, denoted $F$, and details how each variable in the state matrix effects every other when we extrapolate to the next time step.

Having updated the prediction the covariance now also needs updating. Using the identity:

$$Cov(\chi) = \Sigma$$

$$Cov(A\chi) = A\Sigma A^T$$

It can be shown that since:

$$\hat{\chi}_{n|n-1} = F\chi_{n-1|n-1}$$

then:

$$P_{n|n-1} = FP_{n-1|n-1}F^T$$

### 4.1.2 Control input

At this stage any control levers can be introduced to the model. In this example, we will look at the idea of the train having control over its acceleration. The acceleration can be introduced into the system using a similar

style of matrix to that used in the extrapolation earlier where each element represents the effect of a control lever on every part of the state. This matrix is referred to as the input or control matrix and will be described as:

$$B = \begin{vmatrix} dt^2/2 \\ dt \end{vmatrix}$$

from the kinematic equations:

$$pos_n = pos_{n-1} + \Delta t vel_{n-1} + \frac{1}{2} u \Delta t^2$$

$$vel_n = vel_{n-1} + u\Delta t$$

where u = acceleration.

The control process has its own associated variance caused by imperfect application in a real world environment; in this example for instance variance could be introduced through wheel slippage on the tracks. Here the variance will simply be:

$$Q = \text{input variance} \times BB^T$$

#### 4.1.3 Overall

The combined overall equations for this step are:

$$\hat{\chi}_{n|n-1} = F\hat{\chi}_{n-1|n-1} + Bu$$

where $u$ is the input acceleration, and:

$$P_{n|n-1} = FP_{n-1|n-1}F^T + Q$$

describes the covariance update.

## 4.2 Step Two: Measurements Update

Now that there is a first estimate for where the train should be based on the information available from the last step, the measurement information acquired this step can be utilised. Once again there will be a mean estimate value for each sensor (i.e. the reading) and an associated covariance matrix for the sensory apparatus as a whole.

## 4.3 Mapping the Readings to the State

Firstly there needs to exist an association matrix which maps the measurements vector onto the state space. In this example where we have the measurement vector:

$$z = \begin{vmatrix} posMeas \\ velMeas \end{vmatrix}$$

the association matrix is simply:

$$H = \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix}$$

showing that the measured position has a one-to-one relationship with the state position, no relationship to the state velocity and vice versa for the measured velocity.

#### 4.3.1 The Innovation Vector

From the measurement vector containing the sensor readings the innovation vector is derived. This describes the measurement residual or, more intuitively, the difference between each reading and the state estimate from step one and is given by:

$$Inn = z - H\hat{\chi}_{n|n-1}$$

The covariance of this innovation vector is derived from the estimation covariance and the measurement covariance as shown:

$$\Sigma_{Inn} = HP_{n|n-1}H^T + R$$

#### 4.3.2 Combination Through the Kalman Gain

The innovation now needs to be combined with the step one estimate to give an overall prediction of the state. The Kalman gain is used to weight the two previous steps, and can be thought of as the proportional confidence of the measurements relative to the extrapolated state prediction from step one. It

should therefore be intuitive that the Kalman gain is derived from the covariances which describe the probable error. The Kalman gain is given in the equation below:

$$K = \frac{P_{n|n-1}H^T}{\Sigma_{Inn}}$$

The state estimate can now be updated simply with the equation:

$$\hat{\chi}_{n|n} = \hat{\chi}_{n|n-1} + KInn$$

The final step is to calculate the new covariance of the state estimate $P_{n|n}$ as to be used in the next algorithm iteration using:

$$P_{n|n} = P_{n|n-1} - KHP_{n|n-1}$$

### 4.4 Notes

A full mathematical walkthrough of one time step for the example used is available in appendix A. It will be shown that for the initial state at time $n = 1$ then the covariance of the state is equal to that of the process noise.

## 5 Kalman Filter Variants

While the Kalman filter can be utilised in a wide variety of applications, there are variants which allow use of the filter in other situations.

### 5.1 Extended Kalman Filter

The extended Kalman filter takes non-linear system models and uses mathematical techniques adapted from multivariate Taylor series expansion to linearise them. While this does mean that the filter can be used for higher than first order systems, performance benefits decrease with order.[2] There are downsides to the EKF; it is not considered to be an optimal estimator and errors in initial state or process modelling cause filter divergence.[3] Despite this, the EKF is currently the used standard for navigational systems and other non-linear system estimations.

### 5.2 Unscented Kalman Filter

For higher order non-linear systems where the efficacy of the EKF can deteriorate, the unscented filter can provide a better estimate of the true state. It also appears to be more robust than he EKF in all situations.[4] The UKF samples a minimal set of so called sigma points from the mean-covariance distribution and then propagates these through the non-linear functions, generating a new mean and covariance at the end of this process. The sample points are chosen using the unscented transform technique.[5][6]

## 6 Conclusion

## References


[1] R. E. Kalman *et al.*, "A new approach to linear filtering and prediction problems," 1960.

[2] G. A. Einicke, *Smoothing, filtering and prediction: estimating the past, present and future*. InTech, 2012.

[3] G. P. Huang, A. I. Mourikis, and S. I. Roumeliotis, "Analysis and improvement of the consistency of extended kalman filter based slam," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 473–479, IEEE, 2008.

[4] F. Gustafsson and G. Hendeby, "Some relations between extended and unscented kalman filters," *IEEE Transac-*

*tions on Signal Processing*, vol. 60, no. 2, pp. 545–555, 2012.

[5] S. J. Julier and J. K. Uhlmann, "New extension of the kalman filter to nonlinear systems," in *AeroSense'97*, pp. 182–193, International Society for Optics and Photonics, 1997.

[6] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004.

[7] R. Faragher, "Understanding the basis of the kalman filter," *IEEE Signal Processing Magazine*, pp. 128–132, September 2012.

[8] D. Simon, "Kalman filtering." `http://www.embedded.com/design/configurable-systems/4023342/Kalman-Filtering`, 2001. (Date last accessed 28-February-2017).

Appendix

# A Single Time Step Mathematical Walkthrough

This walkthrough will go through one full time step of the train system used previously. it will assume a starting position of 10m, velocity of 10m/s and constant acceleration of $1\text{m/s}^2$. The timestep will be 1s long. Measured position, measured velocity and control input variance will be 2m, 2m/s and $0.5\text{m/s}^2$ respectively. Assume measurement readings at time $n$ of position 21m and velocity 10.5m/s.

## A.1 Starting Equations at Time *n-1*

Kinematic equations:

$$pos_n = pos_{n-1} + \Delta t vel_{n-1} \tag{1}$$

$$vel_n = vel_{n-1} \tag{2}$$

give transition matrix:

$$F = \begin{vmatrix} 1 & \Delta t \\ 0 & 1 \end{vmatrix} \tag{3}$$

Further kinematic equations:

$$pos_n = pos_{n-1} + \Delta t vel_{n-1} + \frac{1}{2} u \Delta t^2 \tag{4}$$

$$vel_n = vel_{n-1} + u\Delta t \tag{5}$$

give control matrix:

$$B = \begin{vmatrix} dt^2/2 \\ dt \end{vmatrix} \tag{6}$$

Covariance that the control input introduces to the estimate through the equation:

$$Q = accelerationnoise \times BB^T = \begin{vmatrix} 0.0625 & 0.125 \\ 0.125 & 0.25 \end{vmatrix} \tag{7}$$

State vector:

$$\chi_{n-1|n-1} = \begin{vmatrix} pos \\ vel \end{vmatrix} = \begin{vmatrix} 10 \\ 10 \end{vmatrix} \tag{8}$$

with covariance matrix which at n=0 with no measurements is simply Q:

$$P = \begin{vmatrix} \Sigma_{pp} & \Sigma_{pv} \\ \Sigma_{vp} & \Sigma_{vv} \end{vmatrix} = \begin{vmatrix} 0.0625 & 0.125 \\ 0.125 & 0.25 \end{vmatrix} \tag{9}$$

Given measurement vector:

$$z = \begin{vmatrix} posMeas \\ velMeas \end{vmatrix} \tag{10}$$

with covariance matrix:

$$R = \begin{vmatrix} \Sigma_{pMpM} & \Sigma_{pMvM} \\ \Sigma_{vMpM} & \Sigma_{vMvM} \end{vmatrix} = \begin{vmatrix} 4 & 0 \\ 0 & 4 \end{vmatrix} \tag{11}$$

mapping (10) to (8) requires the state-measurement association matrix:

$$H = \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} \tag{12}$$

## A.2 Step One

$$\chi_{n|\hat{n-1}} = F\chi_{n-\hat{1}|n-1} + Bu = \begin{vmatrix} 20.5 \\ 11 \end{vmatrix} \tag{13}$$

where u = input acceleration

$$P_{n|n-1} = FPF^T + Q = \begin{vmatrix} 0.25 & 0.25 \\ 0.6875 & 0.625 \end{vmatrix} \tag{14}$$

## A.3 Step Two

$$Inn = z - H\hat{\chi}_{n|n-1} = \begin{vmatrix} 0.5 \\ -0.5 \end{vmatrix} \tag{15}$$

$$\Sigma_{Inn} = HP_{n|n-1}H^T + R = \begin{vmatrix} 4.0625 & 0.125 \\ 0.125 & 4.25 \end{vmatrix} \tag{16}$$

$$K = \frac{P_{n|n-1}H^T}{\Sigma_{Inn}} = \begin{vmatrix} 0.1504 & 0.1132 \\ 0.1196 & 0.1141 \end{vmatrix} \tag{17}$$

$$\hat{\chi}_{n|n} = \hat{\chi}_{n|n-1} + KInn = \begin{vmatrix} 20.5186 \\ 11.0027 \end{vmatrix} \tag{18}$$

$$P_{n|n} = P_{n|n-1} - KHP_{n|n-1} = \begin{vmatrix} 0.4744 & 0.3682 \\ 0.3682 & 0.3832 \end{vmatrix} \tag{19}$$

# B MATLAB Code

```
% function kalman(duration, dt)
%
% Kalman filter simulation for a train's 1D postion and velocity utilising two sensors.
%
% E = covariance
%
close all;

% INPUTS
duration = 20;    %length of simulation (seconds)
dt = 0.1;          %step size (seconds)

% Measurement noise due to imperfect sensors.
positionnoise = 10; % laser height measurement noise (m)
velocitynoise = 10;  % GPS velocity measurement noise (m/s)
% External noise due to imperfect replication of instruction wrt the environment.
accelnoise = 10; % acceleration noise (m/sec^2)

% Computational matrices.
F = [1 dt;
     0 1]; % transition matrix, used to extrapolate the new state from the previous position and velocity ignoring acceleration
B = [dt^2/2; dt]; % input matrix, used to apply the effect of acceleration on the new state when extrapolating from the old
H = [1 0;
     0 1]; % measurement matrix, maps the measurements onto the state space

% Initial state.
x = [0; 0]; % initial state vector, [position; velocity]
xhat = x; % initial state estimate

R = [positionnoise^2 0; 0 velocitynoise^2];  % measurement error E
Qu = accelnoise^2 * (B * B'); % process noise E (of xhat); describes the effect of process noise on estimated position and velocity
P = Qu; % initial estimation E (of xhat)

```

```
% Initialize arrays for later plotting.
pos = [];       % true position array
poshat = [];    % estimated position array
posmeas = [];   % measured position array
vel = [];       % true velocity array
velmeas = [];   % measures velocity array
velhat = [];    % estimated velocity array

for t = 0 : dt: duration,
    % Use a constant commanded acceleration in m/sec^2.
    u = 1;
    % Simulate the linear system.
    ProcessNoise = accelnoise * [(dt^2/2)*randn; dt*randn];
    x = F * x + B * u + ProcessNoise;
    % Simulate the noisy measurements
    MeasNoise = [positionnoise * randn; velocitynoise * randn];
        %randomly weighted measurement noise
    z = H * x + MeasNoise;   %actual position with added noise to
        create simulated (mean) measurement readings
    % Extrapolate the most recent state estimate to the present
       time;
    % this is the new prediction based on extrapolation from the
       position
    % and velocity of the previous state plus any compensation
       that needs
    % to be made due to acceleration.
    xhat = F * xhat + B * u;
    % Form the Innovation vector;
    % measured position - extrapolated position, describes the
       measurement
    % residual.
    Inn = z - H * xhat;
    % Compute the E of the Innovation;
    % Epp + measurement error E.
    InnE = H * P * H' + R;
    % Form the Kalman Gain matrix;
    % this represents the proportional confidence we have in our
    % measurements vs our extrapolated prediction.
    K = (((F * P * F') + Qu) * H') / InnE;
    % Update the state estimate;
    % the estimate plus the confidence weighted innovation.
    xhat = xhat + K * Inn;
    % Compute the E of the estimation error for next iteration.
    P = ((F * P * F') + Qu) - (K * H * (F * P * F' + Qu));
    % Save some parameters for plotting later.
```

```
    pos = [pos; x(1)];
    posmeas = [posmeas; z(1)];
    poshat = [poshat; xhat(1)];
    vel = [vel; x(2)];
    velmeas = [velmeas; z(2)];
    velhat = [velhat; xhat(2)];

end

% Plot the results
t = 0 : dt : duration;

subplot(2,2,1);
plot(t,pos, t,posmeas, t,poshat);
grid;
xlabel('Time (sec)');
ylabel('Position (m)');
title('Figure 1 - Vehicle Position (True, Measured, and Estimated)')
legend('True ','Measured','Estimated');

subplot(2,2,2);
plot(t,pos-posmeas, t,pos-poshat);
grid;
xlabel('Time (sec)');
ylabel('Position Error (m)');
title('Figure 2 - Position Measurement Error and Position Estimation Error');
legend('Measurement Error','Estimation Error');

subplot(2,2,3);
plot(t,vel, t,velmeas, t,velhat);
grid;
xlabel('Time (sec)');
ylabel('Velocity (m/sec)');
title('Figure 3 - Velocity (True, Measured, and Estimated)');
legend('True ','Measured','Estimated');

subplot(2,2,4);
plot(t,vel-velmeas, t,vel-velhat);
grid;
xlabel('Time (sec)');
ylabel('Velocity Error (m/sec)');
title('Figure 4 - Velocity Measurement Error and Velocity Estimation Error');
```

```
115 legend('Measurement Error','Estimation Error');
```

Code heavily modified from that written by Simon, D.[8]