Kalman Filter: Simple Object Tracking

Benjamin Brown

ECSE 456 - Notes

February 20, 2015

1 Background

A Kalman filter is used to combine continuous predictions from a theoretical system model with continuous measurements from a real implementation of the same system. This is done to help reduce the effects of noise on system measurements, and can provide a prediction of the system's next state if for some reason the measurements fail or contain lots of noise.

Implementation of a Kalman filter is highly intuitive for a sensor that has been characterized with testing. But for object tracking, it is much more difficult to understand as all of the Kalman filter jargon is focused around "sensors" and "measurements" that sound odd when referring to an object tracking system. The goal of these notes is to break down the Kalman filter theory and apply the theory to a single, 2D object tracking system.

The following two sources were used to develop these notes:

- http://en.wikipedia.org/wiki/Kalman_filter
- http://robotsforroboticists.com/kalman-filtering/

2 Kalman Equations

The Kalman filter equations can be divided into two key sets: the predication equations and the update equations.

The Prediction Equations:

$$\vec{x_{k+1}} = F \cdot \vec{x_k} + B \cdot \vec{u_k} \tag{1}$$

$$P_{k+1} = F \cdot P_k \cdot F^T + Q \tag{2}$$

Intermediate Calculations:

$$\vec{y_{k+1}} = \vec{z_k} - H \cdot \vec{x_{k+1}} \tag{3}$$

$$S_{k+1} = H \cdot P_{k+1} \cdot H^T + R \tag{4}$$

$$K_{k+1} = P_{k+1} \cdot H^T \cdot S_{k+1}^{-1} \tag{5}$$

The Update Equations:

$$\hat{\vec{x_{k+1}}} = \vec{x_{k+1}} + K_{k+1} \cdot \vec{y_{k+1}} \tag{6}$$

$$\hat{P_{k+1}} = (I - K_{k+1} \cdot H) \cdot P_{k+1} \tag{7}$$

Here I have chosen to only use subscripts on vectors and matrices that are going to change over time (i.e. dynamic). I have also chosen to break it up into three sets of equations, because the values produced in the intermediate calculations aren't very interesting to a beginner. The recursive calculation is essentially make a predication of the state of the system, take a measurement of the system, and adjust the predication based on the measurement.

3 Variables Explained

The equations present a lot of variables, which can be daunting, but most of them are very simple or arbitrary for a simple object tracking system.

The State & Input Vector:

- $\vec{x}$ (nx1): The state vector containing the state variables of the system. These are the values you want to control and use, like position, velocity, and acceleration.
- $\vec{u}$ (mx1): The controlled inputs to the system that effect the states, for instance force or voltage which effect the system's velocity or acceleration.

Measurement Vector:

- $\vec{z}$ (px1): This vector represents the values that you measure. These may differ from the states and are related to the state vector through the H matrix and the measurement error vector v discussed below.

Error Co-variance Matrix:

- P (nxn): This matrix is also very important, and it measures how accurate the estimate is. So each iteration it updates and should converge to specific values regardless of what it is set to initially.

Static Variables: These are the values that aren't going to change as time goes on, and you obtain new measurements.

- F (nxn): This is the state transition matrix, sometimes denoted A, and describes how the system states change over a time-step (i.e. from k to $k+1$). The entries of this matrix are derived using physical equations like kinematic equations, Lagrange equations, or Newton's laws. If you have a state space model of your system, or have a transfer function of the system then this matrix is found with ease. Essentially, this matrix describes how your system should act in theory.
- B (nxm): This is the control input model. If you derived the state transition matrix using equations mentioned above, if there are any external inputs being added to the states, this matrix would have the coefficients on those inputs.
- H (pxn): This is the measurement model. This relates the values you measure, $\vec{z}$, to the state vector $\vec{x}$ through the equation $\vec{z} = H \cdot \vec{x} + \vec{v}$.
- Q (nxn): This matrix is a measure of the process error, meaning the error in your theoretical system model. This could be something like friction or air resistance. The matrix is a diagonal (all zeros except along diagonal) with the variance of each state error in the entry. For instance, if your process error is characterized by some vector $\vec{w} = [w_{x_1}, w_{x_2}]$ ($\vec{w}$ would be added onto the first Kalman equation) where each entry is the error for each state variable, then this matrix has the variance of each entry of $\vec{w}$ along the diagonal.

- R (pxp): This is the exact same idea of Q except with your measurements. This matrix is characterized by the error vector $\vec{v}$ (px1). So think of Q as theoretical error and R as practical error.

Intermediate Variables:

- y (px1), S (pxp), K (nxp): These variables all have physical meanings (see Wikipedia), but I think for the purpose of a beginner they can just be thought of as steps to get to the update equations.

4 The Object Tracking Scenario

Now that we have decomposed the Kalman equations and understand the variables, it is time to think about what these variables mean in the context of object tracking. The problem can be framed as follows:

We are able to identify an object in a single image (frame) by means to be discussed below. But if we rely solely on these measurements we are subject to (1) the error in the measurement, (2) only knowing where the object is at the present time. For instance, if the object goes behind an obstacle obstructing its view from the camera, we cannot know where it is solely based on measurements. The Kalman equations incorporate a theoretical model of the system that could provide a means of tracking an object even when events (1) and (2) are present.

4.1 System Model

The system model is the constant velocity model. We assume that the object exists in the 2D image reference frame and travels with constant velocity. Meaning that the position can be found using the following kinematic equations:

$$x_{k+1} = x_k + v_{x,k} \cdot t_{step} \tag{8}$$

$$y_{k+1} = y_k + v_{y,k} \cdot t_{step} \tag{9}$$

$$v_{x,k+1} = v_{x,k} \tag{10}$$

$$v_{y,k+1} = v_{y,k} \tag{11}$$

Where t_{step} is the time between two updates. We will see from the measurement model below that an update occurs every frame (since we get a new measurement each frame). So this time step will equal the length of the video divided by the number of frames. From a real-time perspective this value is the frame rate. So the state vector $\vec{x}$ is:

$$\vec{x} = \begin{bmatrix} x, y, v_x, v_y \end{bmatrix}^T$$

The state transition matrix, control input, and the control input model is:

$$F = \begin{bmatrix} 1 & 0 & t_{step} & 0 \\ 0 & 1 & 0 & t_{step} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{12}$$

$$\vec{u} = \vec{0} \tag{13}$$

$$B = \begin{bmatrix} \vec{0} & \vec{0} & ... & \vec{0} \end{bmatrix} \tag{14}$$

4.2 Measurement Model

By means of a Delta Frame Generation algorithm presented in [8], we were able to perform object recognition and rudimentry tracking in the MATLAB script `ObjectTracking_FP_v2.m`. This works by taking the first frame of the video as the base-frame, and subtracting the current frame to identify the object (all frames are converted to gray-scale). Essentially this method makes all pixels that are not the object black (0), and any that are the object non-zero (with some noise). We then perform edge detection to find the perimeter of the object.

This algorithm is the basis of our measurement model. Since we can identify the object in the frame, we can find its position in the 2D image reference frame (x,y) where x is the column the pixel lives in and y is the row it lives in. We define the position of the object as the center most pixel inside of its edges. So we have our measurement vector:

$$\vec{z} = \begin{bmatrix} x, y \end{bmatrix}^T \tag{15}$$

We come up with the measurement model, H, by thinking about what matrix would bring the state vector, $\vec{x}$, to the measurement vector, $\vec{z}$.

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{16}$$

4.3 Error Models

Not very sure how to determine the error models, but if the process error is given by:

$$\vec{w} = \begin{bmatrix} w_x, w_y, w_{v_x}, w_{v_y} \end{bmatrix}^T \tag{17}$$

And the measurement error is given by:

$$\vec{v} = \begin{bmatrix} v_x, v_y \end{bmatrix}^T \tag{18}$$

Then Q and R are given as:

$$Q = \begin{bmatrix} VAR(w) & 0 & 0 & 0 \\ 0 & VAR(w) & 0 & 0 \\ 0 & 0 & VAR(w) & 0 \\ 0 & 0 & 0 & VAR(w) \end{bmatrix} \tag{19}$$

$$R = \begin{bmatrix} VAR(v) & 0 \\ 0 & VAR(v) \end{bmatrix} \tag{20}$$

I guess for basic testing you could set the error vectors to 0 just to observe how the predictions and the measurements interact. These matrices aren't a big part of the algorithm (and by that I don't mean they aren't important) in the sense that they are only added on and are static in the Kalman equations.

4.4 Dynamic Equation Initialization

Now that we have determined the state vector, measurement vector, and static user provided variables, we must look at what the remaining variables must be set at the start. Looking at the Kalman equations, it is clear that $\vec{y}$, S, and K all don't have a $k = 0$ iteration, meaning it is completely arbitrary what they are initialized to (from a software and algorithm stand point) as they will be filled in the first iteration. However, P and $\vec{x}$ require initial values. For $\vec{x}$ the initial conditions can

be set to $\vec{0}$, as it can be assumed the object starts from the origin and rest. For P, I have seen this initialized to a zero matrix, or a diagonal matrix with arbitrary large values, since (like $\vec{x}$) it should converge to its usual values. After testing you can initialize P to values near the ones it converges to in order to save processing.