Ege University

Term Project

Kalman Filter

551: The Principles of Signals and Wireless Communication for Embedded Systems

Murat MULAYIM
91170000035

Lecturer: Assist. Prof. K. Sinan YILDIRIM

January 20, 2019

Abstract

In this report, we introduce the Kalman Filter and how object tracking is implemented on Javascript, a web page. Besides, Javascript is supported by HTML and CSS for purpose of user interface.

The Kalman Filter consists of two main steps: Prediction and Update. Each steps are introduced and implemented as function.

1 Introduction

Kalman Filter is a method of predicting the future state of a system based on the previous ones. The algorithm essentially implements a set of mathematical equations containing

o Prediction phase.

o Update phase.

Kalman filter is known as filter function. On the other hand, it is basically an estimator function, not only a filter function. It works recursively. That means the output of previous step can be used as the input of next step.

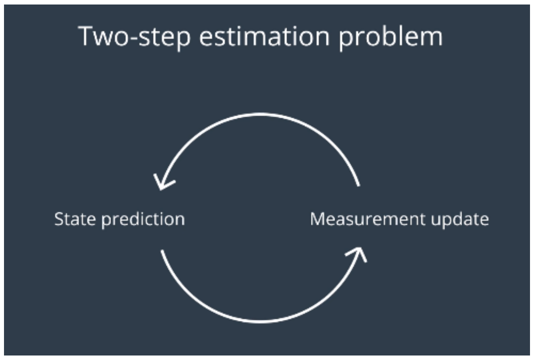


Figure 1: The Filter Flow

The model compares the estimation on the prediction phase and observation on the update phase. The difference between estimation and observation is scaled with a factor known as Kalman Gain. In order to improve the future estimation, Kalman Gain factor is used as an input into model for feedback.

Kalman filters are ideal for systems which are continuously changing.

2 Formulation of Kalman Filter

The Kalman filter addresses the general problem of trying to estimate the state $\mu \in$ R,

$$\mu_t = A\mu_{t-1} + Bu_{t-1} + w_{t-1}$$

with a measurement z $\in$ R,

$$z_t = C\mu_t + v_t$$

The random variables w_t and v_t represent the process and measurement noise. They are assumed to be independent of each other with normal probability distributions

$$p(w) \approx N(0, Q)$$

$$p(v) \approx N(0, R)$$

In practice, the process noise covariance Q and measurement noise R covariance matrices might change with each time step or measurement. The *nxn* matrix A relates the state at the previous time step to the state at the current step, in the absence of either a driving function or process noise. The *nx1* matrix B relates the optional control input u $\in$ R to the state x. The *mxn* matrix C in the measurement equation relates the state to the measurement z_t.

3 How does Kalman work ?

The Kalman filter consists of two phasess:

1. Prediction phase, where the next state of the system is predicted with given previous measurements z_{t-1}

$$\mu_t^- = A\mu_{t-1} + Bu_t$$
$$\sum\nolimits_t^- = A \sum\nolimits_t A_{t-1}^T + Q_{t-1}$$

2. Update phase, where the current state of the system is estimated with given the measurement z_t

$$K = \sum\nolimits_t^- C^T (R + C \sum\nolimits_t^- C^T)^{-1}$$
$$\mu_t = A\mu_{t-1}^- + K(z_t - CA\mu_{t-1}^-)$$
$$\sum\nolimits_t = \sum\nolimits_t^- - KC \sum\nolimits_t^-$$

where,

* μ_t^- and $\sum_t^-$ are the predicted mean and covariance state before measurement on step t

* A and B are state transition and control matrices

* u is control variables

* Q is process noise covariance matrix, represents error due to process

* R is measurement noise covariance matrix, represents error from measurement

* C is measurement matrix, known as mapping measurements onto state

* μ_t and $\sum_t$ are the estimated mean and covariance state after measurement on step t

* z_t is mean of the measurement on step t

* K is the Kalman Gain on step t. Kalman Gain shows how much the predictions should be corrected.

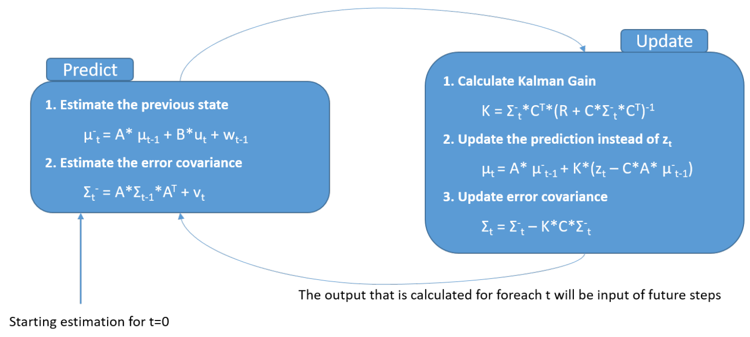

Predict
1. Estimate the previous state
$\mu^-_t = A^* \mu_{t-1} + B^* u_t + w_{t-1}$
2. Estimate the error covariance
$\Sigma_t^- = A^*\Sigma_{t-1}{}^*A^T + v_t$
Update
1. Calculate Kalman Gain
$K = \Sigma^-_t{}^*C^T{}^*(R + C^*\Sigma^-_t{}^*C^T)^{-1}$
2. Update the prediction instead of z_t
$\mu_t = A^* \mu^-_{t-1} + K^*(z_t - C^*A^* \mu^-_{t-1})$
3. Update error covariance
$\Sigma_t = \Sigma^-_t - K^*C^*\Sigma^-_t$
The output that is calculated for foreach t will be input of future steps
Starting estimation for t=0

Figure 2: Kalman Equations

On the update phase, we compute the real value of μ_t at step t and it is the purpose of we want to deal with.

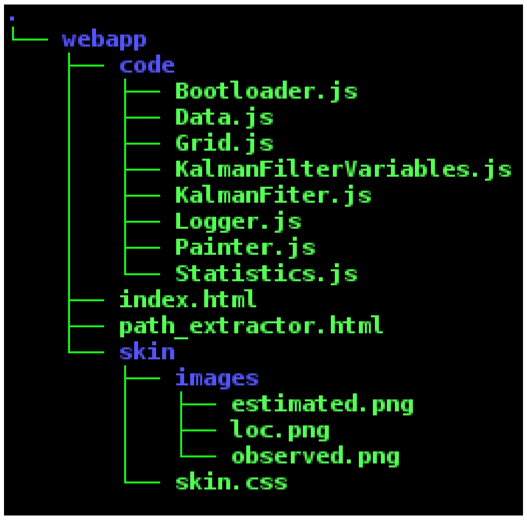
The applications of the Kalman Filtering in real world can be listed as following:

- $\surd$ A radar application where one is interested in tracking a target. The system also can provide some information about location, speed and acceleration of the target. There could be noise but Kalman Filter and eligible equations will deal with the noise.
- $\surd$ Navigation system can use Kalman Filtering and prediction techniques to locate the object with tolerance.
- $\surd$ Many computer vision applications such as feature and cluster tracking.
- $\surd$ A vision-based real-time vehicle tracking system used vision camera to achieve target tracking. The number of tracked vehicle can be single or multiple.

4 Kalman Filter on Object Tracking Problem

The claim of the project is to implement Kalman Filtering functionality on an object tracking system. The language has been chosen to implement is Javascript and supported by HTML5 and CSS3. Tree view of the project is constructed as Figure 3.

4.1 Project Structure



```
.
└── webapp
    ├── code
    │   ├── Bootloader.js
    │   ├── Data.js
    │   ├── Grid.js
    │   ├── KalmanFilterVariables.js
    │   ├── KalmanFiter.js
    │   ├── Logger.js
    │   ├── Painter.js
    │   └── Statistics.js
    ├── index.html
    ├── path_extractor.html
    └── skin
        ├── images
        │   ├── estimated.png
        │   ├── loc.png
        │   └── observed.png
        └── skin.css
```

Figure 3: Project Tree

In order to initialise the project, *index.html* file must be visited via an Internet browser just clicking double click, if default action of Operaing System is to open via browser. The explanation of JS, HTML codes and project tree is:

+ webapp
    + code
        - Bootloader.js : Handles the page load
        - Data.js : Stores the data about the path
        - KalmanFilterVariables.js : Stores the initial values of variables that are used in Kalman Filter equations
        - KalmanFilter.js : Implements both of Prediction and Update phases of Kalman Filter equations
        - Logger.js : Implements logging type and level
        - Painter.js : Handles path-extractor.html page load
        - Statistics.js : Implements statistic charts
    - index.html : Initial page of the project
    - path-extractor.html : A web page to create a pah by using mouse pointer. Mouse left button must be pressed down while drawing path. Once path is created, values will be printed at the end of yellow area. Those values must be copy-pasted into *locationMatrixActual* variable of *Data.js* module to activate the path created.
    + skin

+ images
  - estimates.png : The circle with painted with a shade of yellow to show estimated locations
  - loc.png : A red circle to represent the object
  - observed.png : The circle to show the real observed position of object. Painted with a shade of dark blue
- skin.css : Stores CSS properties of related HTML tags

4.2 How to Use

Whenenever index.html web page is navigated, Figure 4 will be shown on the screen. Object which is drawn with red color goes around the square are according to path information. The map can be considered as a coordinate system starting from 0 point and to 100 point. Object starts from [0, 0] location and moves around based on defined path. Once the path is completed, there will be shown a chart related to estimations and observations on the screen. If it is wanted to reload the scenario, the page must be reloaded.

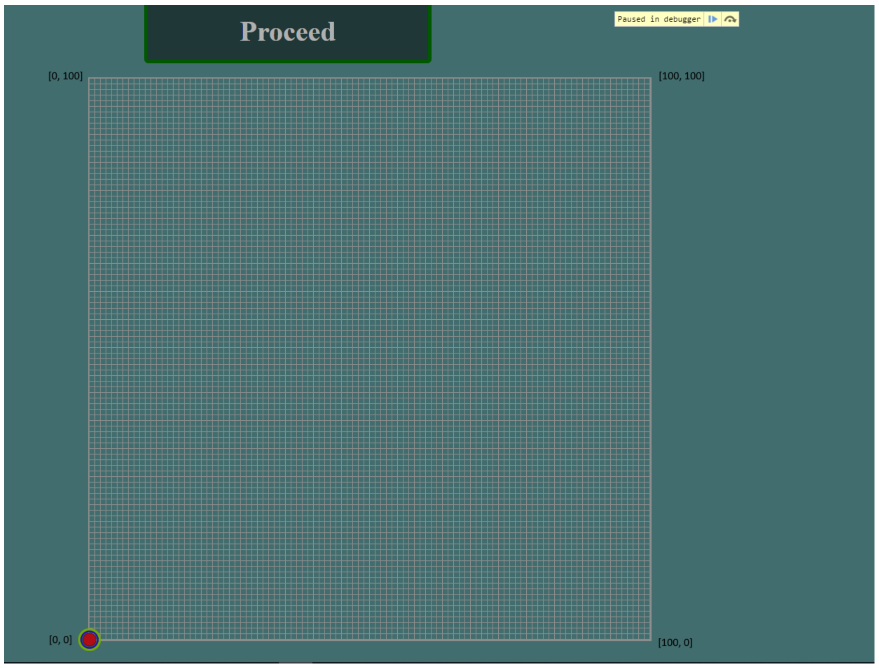


Figure 4: Project Main Page

The path can be reviewed in *Data.js* file.

If a new path is wanted to create, it is easily can be done by visiting *path-extractor.html* (shown as Figure 5) page. The map is similar to previous page and represents a coordinate system with a drawing area. Path can be drawn by keeping mouse left button down without exceeding drawing area. If path is ready, mouse left button must be released. Then path array will be printed on bottom of drawable area. It can now be copied as text and pasted into related *Data.js* file. After reloading the *index.html* page, the new path will be activated.

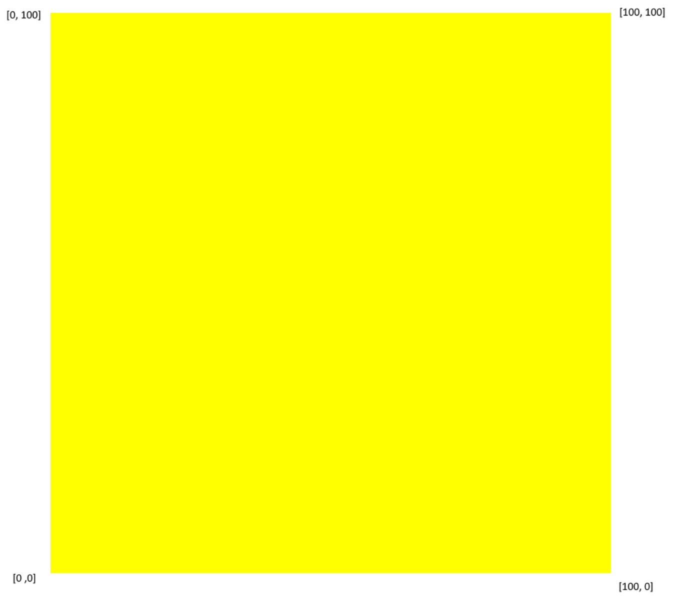


Figure 5: Path Creater Page

4.3 Implementation

Kalman Filter implementation is done by Javascript and supperted by HTML and CSS for UI purpose. Logic runs on *KalmanFilter.js* module. It has public and private functions.

o Public functions

- s load() : Initial method of the module.
- s predict() : Function to run prediction phase of Kalman Filter
- s update() : Function to run Update phase of Kalman Filter
- s doKalmanFiter() : Function to run entire Kalman Filter
- s getStatisticData() : Function to get stored data that will be used for statistic purpose

According to Kalman Filter equations, equations are:

- o Prediction Phase

$$predictedState = A * computedState_{t-1} + B * controlSignal$$

$$predicted\_P = A * computed\_P_{t-1} * A^T + processNoise$$

- o Update Phase

$$K = predicted\_P_t * C^T * (C * predicted\_P_t^T * C^T + measurementNoise)^{-1}$$

$$computedState = A * predictedState_t + K_t * (observedPosition_t - C * A * predictedState_t)$$

$$computed\_P = (1 - K_t * C) * predicted\_P_t$$

All the Kalman Filter logic runs inside *KalmanFilter.js* module and *Bootloader.js* feeds the data about observed location in terms of x and y as an 2x1 matrix.

Variable definitions stored inside *KalmanFilterVariables.js* module and on the load() function, initial values are copied to *KalmanFilter.js* module.

Variables that are used in equations and initializations are listed in the following.

- o dt : Represents sampling rate as number. Used for initializing *processNoise*, *A* and *B*.
- o A : State update transition matrix. Used in both phases. Dimension: 4x4
- o B : Optional control input to the predictedState. Dimension: 4x1
- o C : Measurement function that will apply to the state estimation to get expected next/new measurement. Dimension: 2x4
- o K : Known as Kalman Gain. It tells how much the predictions should be corrected. Dimension 4x2
- o initialState : Represents initial state of system containing x and y positions and velocities. Dimension: 4x1
- o predictedState : Stores predicted state of the system on prediction phase. Initial value is equal to initialState. Dimension: 4x1
- o computedState : Stores computed state of the system on update phase. Initial value is equal to initialState. Dimension: 4x1
- o measurementNoise : Measurement noise in directions on update phase. Some resources represent it as $\sum_v$.Used for calculation of K Kalman Gain. Dimension: 2x2
- o processNoise : Process noise covariance matrix. Dimension: 4x4
- o controlSignal : The control input.

- o predicted _P : The predicted error covariance matrix of previous step, computed in prediction phase before measurement. Initially equals to processNoise. Dimension: 4x4
- o computed _P : The computed error covariance matrix of current step, computed in update phase after measurement. Initially equals to processNoise. Dimension: 4x4

Kalman filter works in the project as an estimator. First of all, filter tries to predict the current state of the system. Operation runs according to previous observed state. Secondly and lastly, we now know the observed location. However, we are not sure about its observed position. Thus, we need to estimate its position by removing part of or entire noise in the equations.

The reason of using Kalman filter is that even we know observed position of object, we are not sure if it is correct or not because there could be several noises such as process noise and measurement noise while measuring signal on the environment where the object is.

5 Particle filter

Kalman filter can model only Gaussian distributed systems. Gaussian Distribution has regular possibility distribution like shown in Figure 6 .

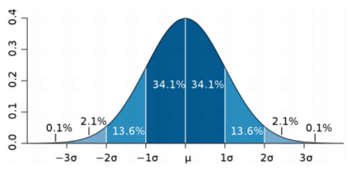


Figure 6: Gaussian Distribution

All the systems on the environment would not have Gaussian distribution. There could be several systems that have several non-linear and/or non-gaussian distributed possibilities such as shown in Figure 7.

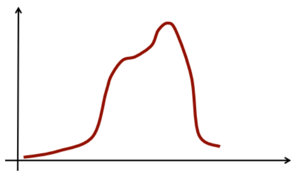


Figure 7: Non Gaussian Distribution

The key difference is that, instead of deriving analytic equations as the Kalman filter does, the Particle filter uses simulation methods to generate estimates of the state and the innovations.

The key idea of achieving the problem that is occured by distribution on noises is particle filter. Representation of arbitrary distributions is done by using multiple samples. Using multiple samples gives us a set of particle with weighted.

A large class of dynamic models can be represented by a state-space form: state estimate - X_t, associated weight - W_t

$$X_{t+1} = < X_t, W_{t+1} >_{t=1,2..n} \tag{1}$$

$$Y_t = < X_t, V_t >_{t=1,2..n} \tag{2}$$

This representation handles a stochastic process by finding three objects: a vector that describes the position of the system (a state, $X_t \in \mathrm{X} \subset R^l$) and two functions, one mapping the state today into the state tomorrow (*the transition equation,* (1)) and one mapping the state into observables, Y_t (*the measurement equation,* (2)).

The Kalman and Particle filters are algorithms that recursively update an estimate of the state and find the innovations driving a stochastic process given a sequence of observations. The Kalman filter accomplishes this goal by linear projections, while the Particle filter does so by a sequential Monte Carlo method. With the state estimates, we can forecast and smooth the stochastic process. With the innovations, we can estimate the parameters of the model.

Particle filter consists of three main phases.

1. Sample the particles using the proposal distribution

$$x_t^{[i]} \sim \pi(x_t|..)$$

2. Compute the importance weights

$$w_t^{[i]} = \frac{target(x_t^{[i]})}{proposal(x_t^{[i]})}$$

3. Resampling:Replace unlikely samples by more likely ones

An algorithm for particle filter is:

```
**Particle_filter**($X_{t-1}$, $u_t$, $z_t$)
    $\widehat{X}_t = X_t = \emptyset$
    for m = 1 to M do
        sample $x_t^{[m]} \sim \pi(x_t)$
        $w_t^{[m]} = \frac{p(x_t^{[m]})}{\pi(x_t^{[m]})}$
        $\widehat{X}_t = \widehat{X}_t + < x_t^{[m]}, w_t^{[m]} >$
    endfor
    for m = 1 to M do
        draw i with probability $\alpha$ $w_t^{[i]}$
        add $x_t^{[i]}$ to $X_t$
    endfor

    return $X_t$
```

In order to summarize particle filters

o Particle filters are non-parametric, recursive Bayes filters

0 Posterior is represented by a set of weighted samples

0 Particle filters are not limited to Gaussians

o Proposal to draw new samples

0 Weight to account for the differences between the proposal and the target

o Work well in low-dimensional spaces

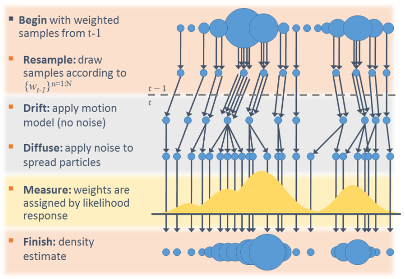
5.1 Object Tracking Problem using Particle Filter

Particle : A single particle is also known as a sample. The reason for this is due to its role (sample) for the target(posterior) distribution. Each particle contains:

1. State Estimate (X_t)
2. Weight (W_t)

Steps: We start with the previous estimation.

Figure 8: A Flow for Object Tracking Problem using Particle Filter


Begin with weighted samples from t-1
Resample: draw samples according to $\{w_{t-1}\}^{n=1:N}$
$t-1$
t
Drift: apply motion model (no noise)
Diffuse: apply noise to spread particles
Measure: weights are assigned by likelihood response
Finish: density estimate

The first step is the particle resampling and weight normalization (red). Then we apply state transition (e.g. motion model) to each particle (green). Those two steps are included into the prediction steps.

The update step is formed of measurement and weight update. The measurement is taken using the observation model (gold).

Finally, the particles' weights are updated using the observation model in order to give a posterior distribution (red).

5.1.1 Prediction

Prediction is the first step which includes particle selection according to their weight **resample** , the next state transition (e.g. applying motion model) **drift** (applying motion model without noise) and applying noise **diffuse** (applying noise to spread particles).

The first step is the particle resampling. The particles are resampled according to their weights, where the particles which have higher weight have a greater chance to be selected.

We begin with the weighted particles from a previous iteration. Resampling draws new samples from the previous particle set. Resampling step is needed in order to stop degeneracy. Degeneracy happens when a single sample has an extremely dominant weight. After the particles are resampled the weight distribution is set to uniform.

Apply the motion model p$(x_t|x_{t-1})$ to every particle:

$\Rightarrow$ $x_t = F_t x_{t-1} + w_t$, where F_t is linear motion model and w_t is noise

$$\Rightarrow \begin{bmatrix} x_t \\ y_t \\ x_t^- \\ y_t^- \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ x_{t-1}^- \\ y_{t-1}^- \end{bmatrix} + w_t$$

5.1.2 Update

After the noisy measurement has been obtained, the update (correction) step begins. Each particle is evaluated and each particles' weight is updated according to obtained likelihood. Observation model models the likelihood that the measurement z_t would appear assuming the we know the state x_t of the object.

- **Obtain observation** z_t for each state estimate x_t
- **Evaluate likelihood** that an x_t gave rise to z_t using observation model

  $$\mathrm{p}(z_t|x_t^n) = e^{-\lambda dist(z_t,c)}$$

- **Assign weights** are proportional to the likelihood response

  $$w_t^n = p(z_t|x_t^n)$$

5.2 Particle Filter versus Kalman Filter

In a linear system with Gaussian noise, the Kalman filter is optimal. In a system that is nonlinear, the Kalman filter can be used for state estimation, but the particle filter may give better results at the price of additional computational effort. In a system that has non-Gaussian noise, the Kalman filter is the optimal linear filter, but again the particle filter may perform better.

Kalman filters have much lower computational requirements than particle filters, but are less flexible. Basically, the math works out so that estimators for this sort of system have a very nice solution