Kalman Filter

A Kalman filter is an optimal estimation algorithm used to estimate states of a system from indirect and uncertain measurements. A Kalman filter is only defined for linear systems.

Let us design a self-driving car and we are trying to localize its position in an environment. The sensors of the car can detect cars, pedestrians, and cyclists. Knowing the location of these objects can help the car make judgements, preventing collisions. But on top of knowing the location of the objects, the car needs to predict their future locations so that it can plan what to do ahead of time. For example, if it were to detect a stop sign towards the road, it should stop and cautiously proceed. The Kalman filter can help with this problem, as it is used to assist in tracking and estimation of the state of a system.

The car has sensors that determines the position of objects, as well as a model that predicts their future positions. In the real world, predictive models and sensors aren't perfect. There is always some uncertainty. For example, the weather can affect the incoming sensory data, so the car can't completely trust the information. With Kalman filters, we can mitigate the uncertainty by combining the information we do have with a distribution that we feel more confident in.

Now let's discuss Kalman filter is extensively used in estimating the car position using Lidar and Radar sensors. Measurement of these sensors are not accurate as they are subject to drift or noisy. Kalman filter can be used to fuse the measurement of these sensors to find the optimal estimation of the exact position.

To track a moving car, we repeat a 2-step procedure:

- **Predict**: Apply a dynamic model to our belief to predict what is next.

  A simple implementation of this is:

```
def predict(mean1, var1, mean2, var2):
 new_mean = mean1 +mean2
 new_var = var1 + var2
 return [new_mean, new_var]
```

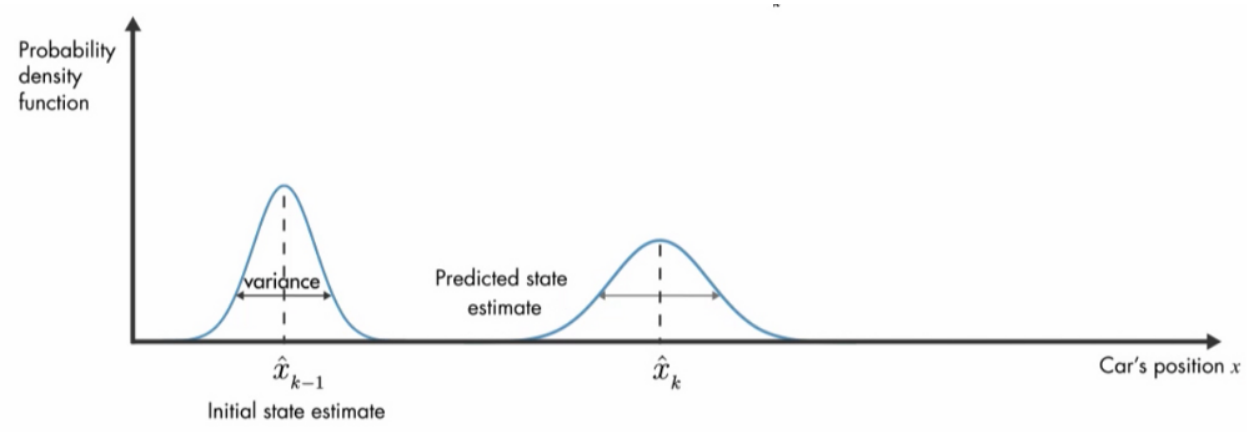
- **Update**: Take a measurement to update our prediction.

  A simple implementation of this is:

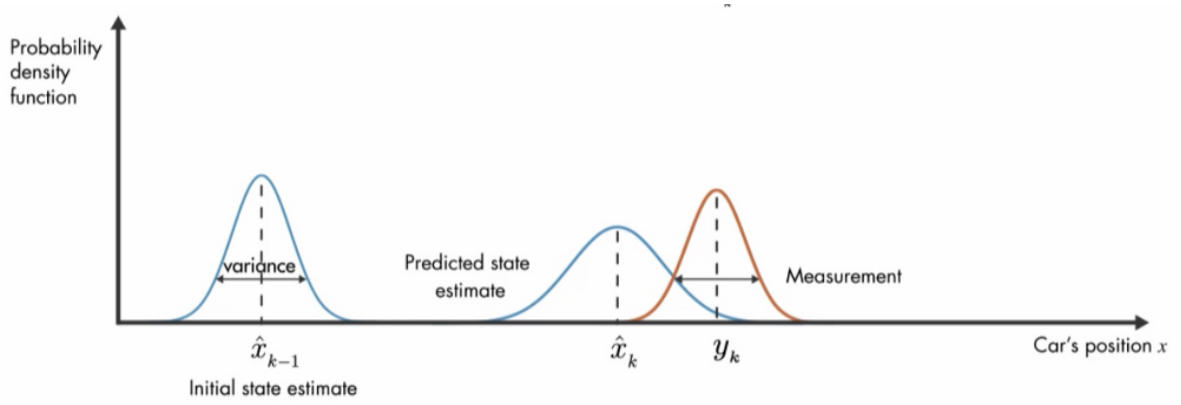
```
def update(mean1, var1, mean2, var2):

 sum = var1+var2
 pr_s = mean1*var2 + mean2*var1
 #print(pr_s)
 new_mean =1/(sum) * pr_s
 product = var1*var2
 new_var = product/sum
 return [new_mean, new_var]
```

Before getting into the details of the Kalman filter algorithm cycle, consider below example of estimating the car position based on the prediction result and measurement result using Kalman filter. We can explain the working principles of Kalman filters with the help of probability density function as shown below.

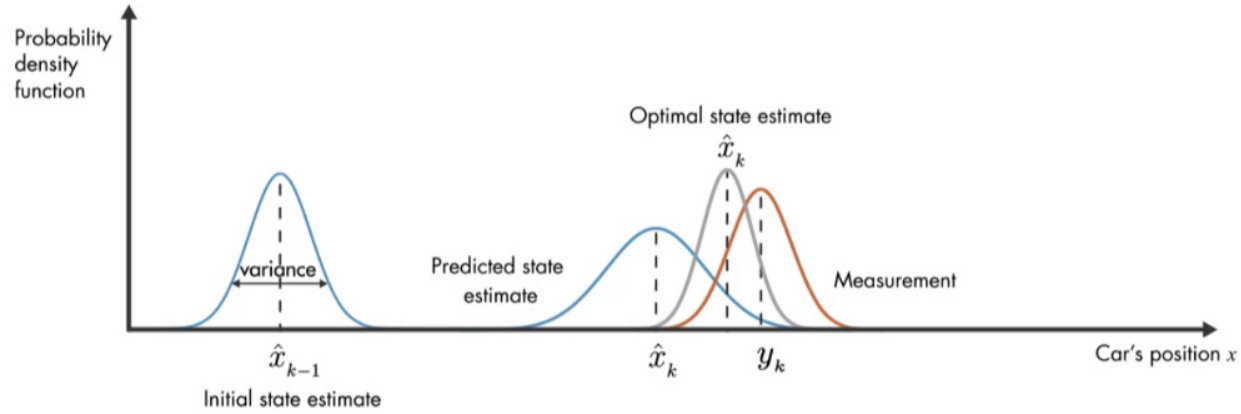
At initial time step **k-1**, car position can be in the mean position of gaussian distribution at ***xk-1*** position. At the next time step K, uncertainty at ***xk*** position increases which is shown below with larger variance.



Next one is the measurement result **y** from the sensors and noise represented in the variance of the third gaussian distribution as shown below.



Kalman filter says optimal way to estimate the position is combining these two results. This is done by multiplying these two probability functions together and the result will also be another gaussian function.



Let's look deep into dynamic model in predicting motion. The equations will look scary but actually pretty simple.

To simplify our illustration, we assume our car moves along the x-axis only.

State

We first define the state of our car. For simplicity, we will use the position p and the speed v only.

$$x_k = \begin{bmatrix} p_k \\ v_k \end{bmatrix}$$

where

p_k and v_k are the position and velocity along x-axis at $time = k$.

State-transition model

Without acceleration, we can calculate the equation of motion with some simple Physics.

$$p_k = p_{k-1} + v_{k-1}\Delta t$$
$$v_k = v_{k-1}$$

Rewrite this into a matrix form which derives states from the last previous state.

$$x_k = \begin{bmatrix} p_k \\ v_k \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_{k-1} \\ v_{k-1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} x_{k-1}$$

$$x_k = \begin{bmatrix} p_k \\ v_k \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_{k-1} \\ v_{k-1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} x_{k-1}$$

$$x_k = \mathbf{A}\, x_{k-1}$$

A, a matrix, becomes our state-transition model. In our example, **A** is:

$$\mathbf{A} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$$

Input controls

We have many controls on the car. For example, we control the throttle for acceleration. Let's modify our equations:

$$p_k = p_{k-1} + v_{k-1}\Delta t + \frac{1}{2}a\Delta t^2$$
$$v_k = v_{k-1} + a\Delta t$$

We pack all input **controls** into a vector **u** and the matrix form of motion becomes:

$$x_k = \mathbf{A}\, x_{k-1} + \mathbf{B}\, u_k$$

where, in our example,

$$x_k = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} x_{k-1} + \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix} [a]$$

$$\mathbf{B} = \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix}$$
$$u_k = [a]$$

(Since we have only one control for now, u has only one element a.)

To make our model more accurate, we add another term called **process noise**.

$$x_k = \mathbf{A}\, x_{k-1} + \mathbf{B}\, u_k + w_k$$

We use process noise w to represent random factors that are difficult to model precisely, for example, the wind effect and the road condition. We assume w is Gaussian distributed with a zero mean and covariance matrix $\boldsymbol{Q}$. (Variance is for 1-D Gaussian distribution while covariance matrix is for n-D Gaussian distribution)

$$w_k \sim \mathcal{N}(0, Q)$$

Observer model (measurement model)

We also model our measurements by applying a transformation matrix **C** on the state of the system.

$$y_k = \mathbf{C}\, x_k + v_k$$

Very often, we measure the state directly (for example, the car location). Hence, C is often an identity matrix. In some case, we do not measure it directly. We need a matrix C to describe the relationship between the state and the measurement. In other cases, **C** performs a coordinate transformation. For example, the state of a RADAR is stored in polar coordinates. We use **C** to convert it to Cartesian.

$$v_k \sim \mathcal{N}(0, R)$$

Putting it together

Here we have a dynamic model to predict a state and a measurement from its last previous state.

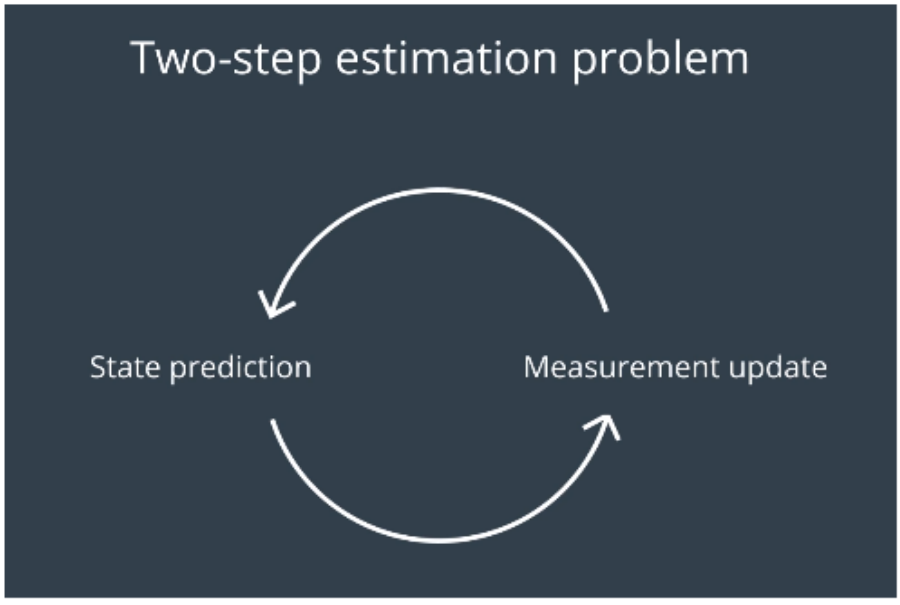
$$x_k = \mathbf{A}\, x_{k-1} + \mathbf{B}\, u_k + w_k$$
$$y_k = \mathbf{C}\, x_k + v_k$$

where

x_k and x_{k-1} are the states of the system at $time = k$ and $k - 1$ respectively.
A is the state-transition model from state x_{k-1} to x_k.
B is the input-control model that applies to the control vector u_k.
$w_k \sim \mathcal{N}(0, Q_k)$ is the sampled process noise, like wind.
Q_k is the covariance matrix of the process noise.

Observation/measurement:
y_k is the measurements made at $time = t$.
C is the observation model to convert the state x_k to measurements.
$v_k \sim \mathcal{N}(0, R_k)$ is the sampled measurement noise like sensor noise.
R_k is the covariance matrix of the measurement noise.

Now let's get into the two cycles of Kalman filters: Prediction and Measurement update.



The filter loop that goes on and on.

Prediction

So far, our observer world uses a deterministic model. Let's expand it to stochastic. We assume all estimated states are Gaussian distributed. So, we model our last estimated state with a mean x and a covariance matrix P.

$\hat{x}_{k-1}$ is the estimated state at *time* $= k - 1$.
P_{k-1} is the covariance matrix of the estimated state $\hat{x}_{k-1}$.

Then, we apply the car model in the observer world to make a new state estimate.

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$$

Back to linear algebra, if we apply a transformation matrix A to an input x with a covariance matrix $\sum$, the covariance matrix of the output (Ax) is simply:

$$A\Sigma A^T$$

Putting the process noise back, the Gaussian model for the estimated state before the update is:

Prediction:

$$\hat{x}_k^- = \mathbf{A}\hat{x}_{k-1} + \mathbf{B}u_k$$
$$P_k^- = \mathbf{A}P_{k-1}\mathbf{A}^T + Q$$

where,

$\hat{x}_k^-$ is the estimated state at *time* $= k$ before the update.
P_k^- is the covariance matrix of the estimated state $\hat{x}_k^-$.

Update

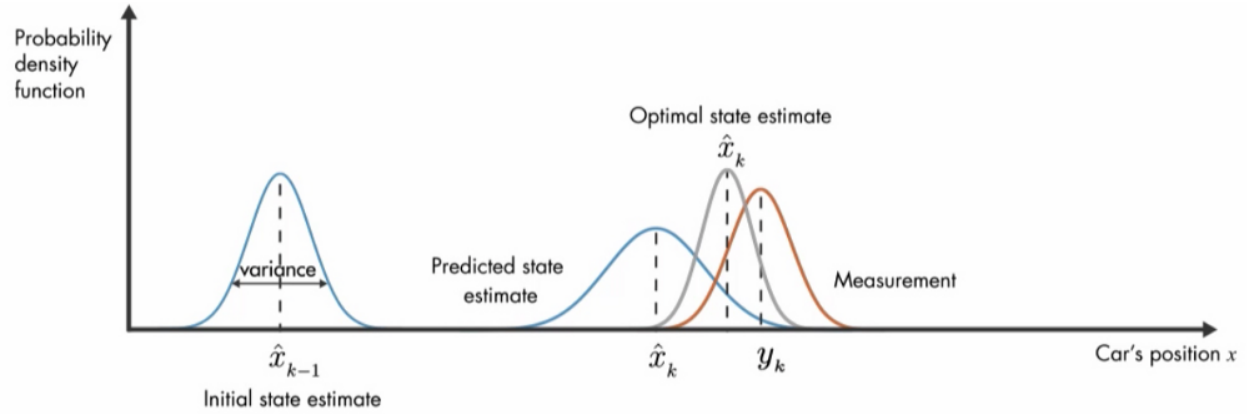
Finally, we will make the final prediction using Kalman filter. At time=k, we make a measurement y. For easier visualization, we always assume C is an identity matrix when we plot the measurement PDF in this article.

Since the estimated state (before the update) and the measurement are Gaussian distributed, the final estimated state is also Gaussian distributed. We can apply linear algebra again to compute the Gaussian model of our final prediction.

First, we calculate the Kalman gain which put the measurement noise back into the equation and map the error in the measurement estimate to the state estimate error.

$$K_k = \frac{P_k^- \mathbf{C}^\mathbf{T}}{\mathbf{C}P_k^- \mathbf{C}^\mathbf{T} + R}$$

Then the Gaussian model for the new estimated state is calculated based on the Gaussian model for the state estimate before the update, the Kalman gain *K*, the measurement and *C*. Here is our updated state estimation:



Update:

$$K_k = \frac{P_k^- \mathbf{C}^\mathbf{T}}{\mathbf{C}P_k^- \mathbf{C}^\mathbf{T} + R}$$
$$\hat{x}_k = \hat{x}_k^- + K_k(y_k - \mathbf{C}\hat{x}_k^-)$$
$$P_k = (I - K_k\mathbf{C})P_k^-$$

Where,

K_k is the computed Kalman gain to correct the observer estimation.
$\hat{x}_k$ is the estimated state at $time = k$.
P_k is the covariance matrix of the estimated state $\hat{x}_k$.

Let's do a recap:

- The red curve on the left: the estimated state at time=k-1.
- The red curve on the right: the estimated state before the update.
- The orange curve: the measurement.
- The black curve: the estimated state at time=k.

To track a moving car, we repeat a 2-step procedure below:

Prediction

$$\hat{x}_k^- = \mathbf{A}\hat{x}_{k-1} + \mathbf{B}u_k$$
$$P_k^- = \mathbf{A}P_{k-1}\mathbf{A}^T + Q$$

Update

$$K_k = \frac{P_k^-\mathbf{C}^\mathbf{T}}{\mathbf{C}P_k^-\mathbf{C}^\mathbf{T} + R}$$
$$\hat{x}_k = \hat{x}_k^- + K_k(y_k - \mathbf{C}\hat{x}_k^-)$$
$$P_k = (I - K_k\mathbf{C})P_k^-$$

where
$\hat{x}_{k-1}$ is the estimated state at $time = k - 1$.
P_{k-1} is the covariance matrix of the estimated state $\hat{x}_{k-1}$.
u_k is the input control.
$\mathbf{A}$ is the state-transition model.
$\mathbf{B}$ is the input-control model.
$\mathbf{C}$ is the observer model for the measurement.
Q is the covariance matrix of the process noise.
R is the covariance matrix of the measurement noise.
y_k is the measurement at $time = k$.
$\hat{x}_k^-$ is the estimated state at $time = k$ before the update.
P_k^- is the covariance matrix of the estimated state $\hat{x}_k^-$.
K_k is the computed Kalman gain to correct the observer estimation.
$\hat{x}_k$ is the estimated state at $time = k$.
P_k is the covariance matrix of the estimated state $\hat{x}_k$.

Sensor fusion:

LiDAR fires rapid pulses of laser light (200K per second) to create a 3-D map of our environment. Its short wavelength lets us detect small objects with high resolutions. However, the measurement can be noisy in particular during rain, snow, and smog. Radar has longer range and is more reliable, but it has lower resolution. Sensor fusion combines measurements from different sensors using Kalman filter to improve accuracy. The measurement errors of many sensors are not co-related, i.e. the measurement error of a sensor is not caused by another sensor. For that, we can apply Kalman filter one at a time for each measurement to refine the prediction.

Resources

https://github.com/udacity/CarND-Extended-Kalman-Filter-Project

https://github.com/paul-o-alto/CarND-Extended-Kalman-Filter-Project

https://github.com/paul-o-alto/CarND-Extended-Kalman-Filter-Project

https://en.wikipedia.org/wiki/Kalman_filter

https://en.wikipedia.org/wiki/Extended_Kalman_filter

https://en.wikipedia.org/wiki/Kalman_filter#Unscented_Kalman_filter

https://ai2-s2-pdfs.s3.amazonaws.com/df70/ce9aed1d8b6fe3a93cb4e41efcb8979428c0.pdf

https://www.udacity.com/course/artificial-intelligence-for-robotics–cs373

http://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/