



Photo by Gabriel Gurrola

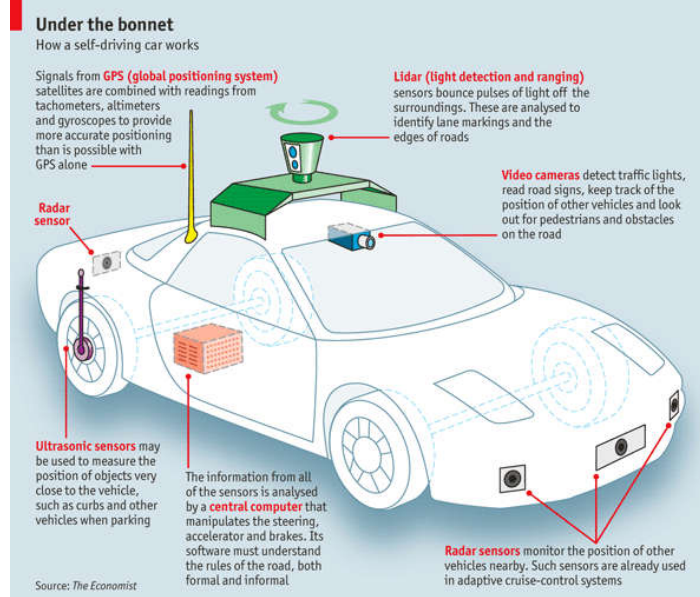
Self-driving car object tracking: Intuition and the math behind Kalman Filter



Jonathan Hui

Apr 9, 2018 · 12 min read

Three major components of autonomous driving are localization (where am I), perception (what is around me), and control (how to drive around). We are dealing with sensors to perceive what is around us. Those sensors offer many benefits but yet suffer some significant drawback. Cameras offer high resolution but cannot detect speed and distance easily. LiDAR creates a nice 3-D map. But LiDAR is huge and expensive. Also, both camera and LiDAR are vulnerable to bad weather and environmental factors. RADAR works better in bad conditions and detects speed well. But its resolution is not high enough for precise maneuver. When we are dealing with self-driving car sensors, we do not have a one-size-fits-all solution. Kalman filter is designed to fuse sensor readings to make more accurate predictions than each individual sensor alone. However, the math in Kalman filter can be unnecessary overwhelming. We will cover the intuition first followed by explanations on the math to make it easier.



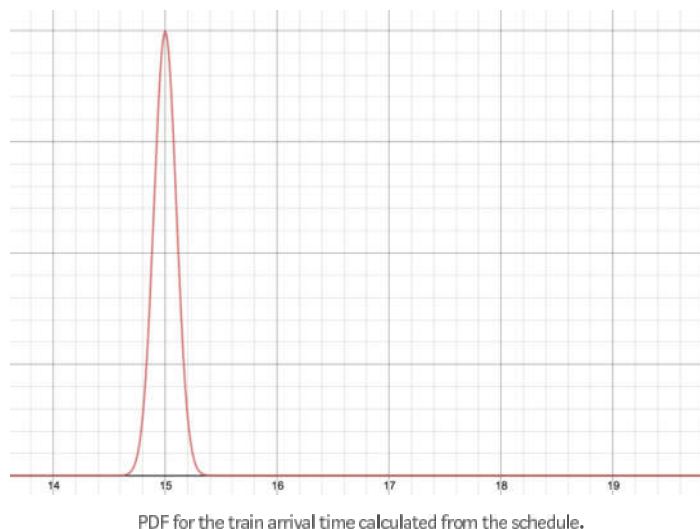
Source: The Economist

This is part of a 5-series self-driving. Other articles includes

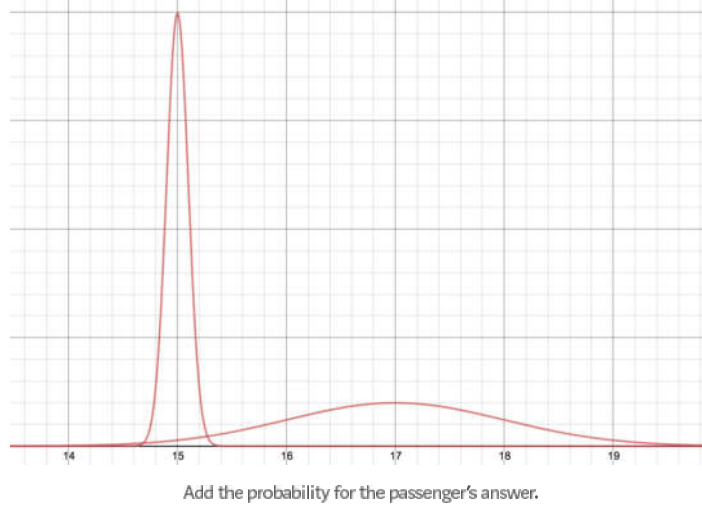
- Self-driving perception: [Sensor fusion with Kalman Filter](#).
- Self-driving perception: [Extended Kalman Filter and Unscented Kalman Filter](#).
- Self-driving localization: [Localization with Particle Filter](#).
- Self-driving control: [Control with Model Predictive Control & PID](#).
- Self-driving [Path finding](#).

Intuition

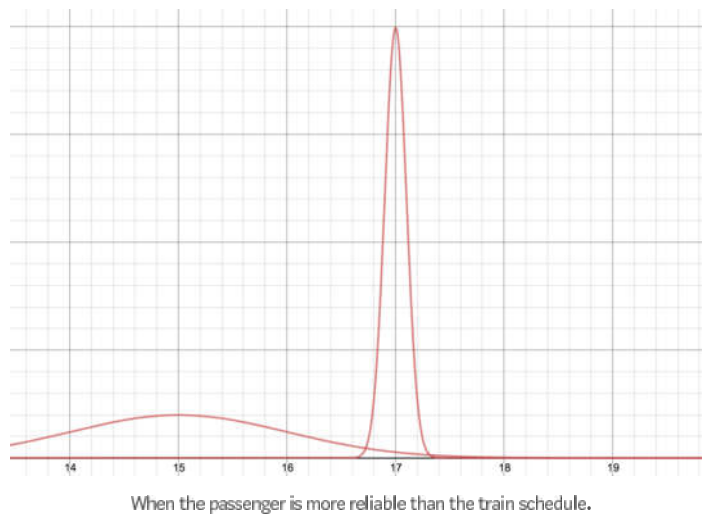
We are taking a train to work. The train comes every 15 minutes starting at 5 am. One day, you ask a fellow passenger when the next train arrives. He answers 7:17 am. Before settling down with his answer, let's dig down a little bit more. If this is the Japan Shinkansen bullet train, the train schedule is God and delays longer than 30 seconds are rare. The x-axis below represents the train arrival time (15 means 7:15 am) and the y-axis is the probability for each possible arrival time. The graph below indicates a strong certainty that the train will arrive at 7:15.



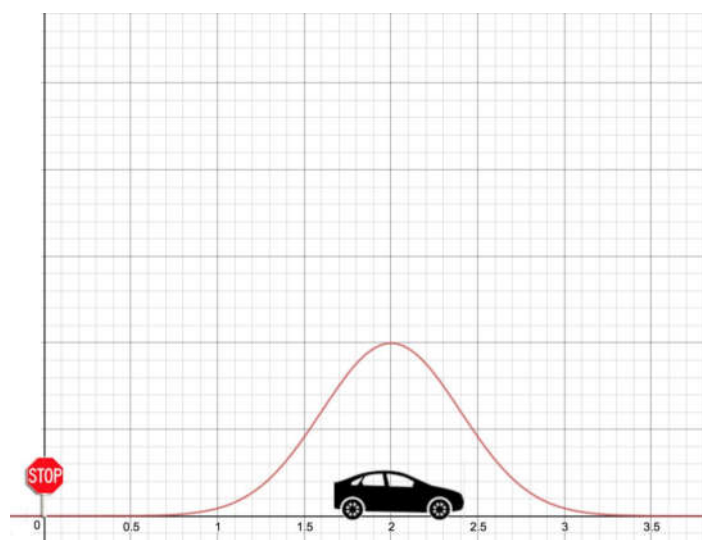
In the following graph, we add another curve modeling the passenger answer. In this case, the information is less certain and therefore, the curve is flatter.



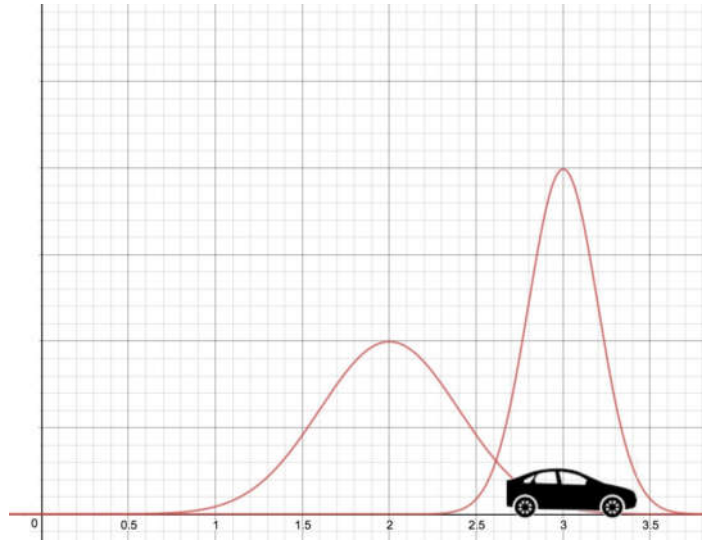
In many transport systems, the schedule is just a reference with frequent delays. Now, if the passenger just checks the real-time arrival time on the phone, the information is much more certain and the plot will just like the opposite of our previous plot.



Instead of blindly trusting the train schedule or the passenger, we want a better approach in making accurate predictions. Let's introduce a more realistic situation in predicting the location of a moving car. At a certain time, we **believe** our car is 2 meters away from the stop sign. To handle uncertainty, we use a stochastic model to describe our car location. The red curve below indicates the probabilities of finding the car at different locations. This idea may take a minute to sink in since we are so used to deterministic models.



To recalibrate our location, we take a GPS measurement. But remember, the measurement is noisy so we use a stochastic model to describe it also. Below are the two different probability curves. The left is our belief and the right one is the GPS measurement. Which one should we trust?

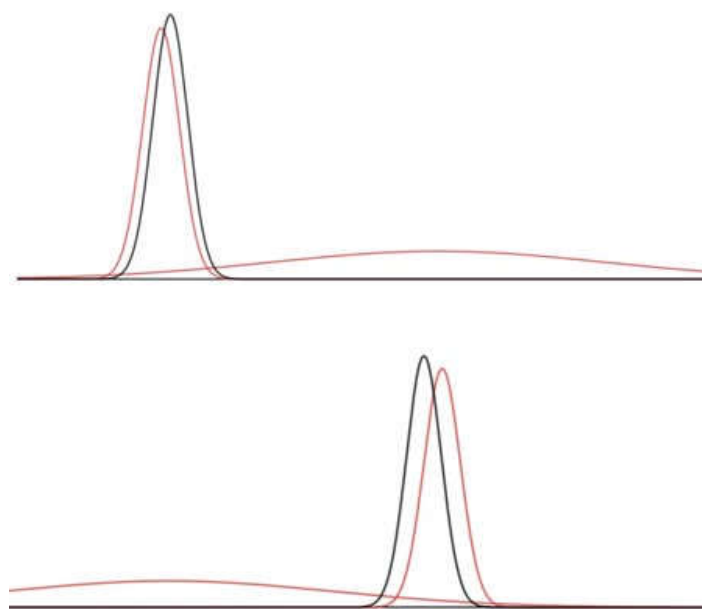


Let's multiply both curves together and renormalizing it to make the total probability equal to 1.

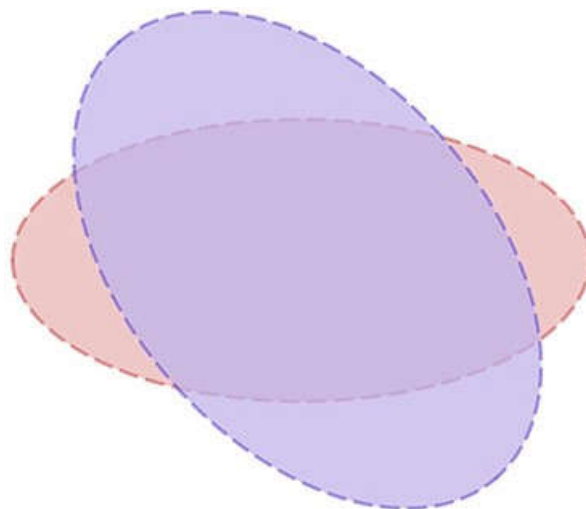
The orange curve below is our new location prediction combining our belief and the measurement. It has a peak at 2.6m where both red curves agree the most. The resulting curve is also sharper, i.e. we are more certain about the location now. The basic idea for Kalman Filter is that simple even it will take a while to explain the details. In particular, multiple probability curves are computationally intense. We need a more efficient approach to merge probability curves.



We can also view this approach as a weighted sum between the belief and the measurements. Let's revisit our train examples again. If our belief is strong (the Japan bullet train's schedule) while the measurements are weak, the final prediction (the black curve below) will resemble the probability curve of our belief. On the contrary, if the measurement is accurate but we are not certain about our belief, the final prediction will resemble the measurements.



How can we get a more accurate result with two less accurate information? In real-life, the measurements may have errors but their probability curves are accurate. For example, many measurement errors are Gaussian distributed with variances that can be determined by experiments. Therefore, we can derive our probability curves with good precision. By multiplying the probability curves, we locate where predictions agree and therefore reinforce the final predictions.



By overlapping belief and measurements, we reinforce what we agree.

So let's go through the process end-to-end. We start with an initial GPS reading. Since the measurement errors are Gaussian distributed, we can use it to build a probability curve (a **probability distribution function PDF**) for the car location. This is our **belief**. Then we use a **dynamic model** to predict where the car may be next. The most simple one will be $\text{location} = \text{previous location} + \text{velocity} \times \delta t$. Next, we take a **measurement** and develop a PDF for it. We merge both PDFs to make a final prediction. When Kalman filter is explained as a Bayes filter, the belief is also called **prior** and the final prediction is called **posterior**.

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

1. **Predict:** Apply a dynamic model to our belief to predict what is next.
2. **Update:** Take a measurement to update our prediction.

Model

Before we introduce the Kalman Filter, we need to detail the 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.

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

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

\mathbf{A} , a matrix, becomes our state-transition model. In our example, \mathbf{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 \mathbf{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 \mathbf{Q} . (Variance is for 1-D Gaussian distribution while covariance matrix is for n-D Gaussian distribution)

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

Observer model (measurement model)

We also model our measurements by applying a transformation matrix \mathbf{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, \mathbf{C} is often an identity matrix. In some case, we do not measure it directly. We need a matrix \mathbf{C} to describe the relationship between the state and the measurement. In other cases, \mathbf{C} performs a coordinate transformation. For example, the state of a RADAR is stored in polar coordinates. We use \mathbf{C} to convert it to Cartesian.

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

Putting it together

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

$$\begin{aligned} x_k &= \mathbf{A} x_{k-1} + \mathbf{B} u_k + w_k \\ y_k &= \mathbf{C} x_k + v_k \end{aligned}$$

where

x_k and x_{k-1} are the states of the system at *time* = k and $k - 1$ respectively.

\mathbf{A} is the state-transition model from state x_{k-1} to x_k .

\mathbf{B} is the input-control model that applies to the control vector u_k .

$w_k \sim \mathcal{N}(0, \mathbf{Q}_k)$ is the sampled process noise, like wind.

\mathbf{Q}_k is the covariance matrix of the process noise.

Observation/measurement:

y_k is the measurements made at *time* = k .

\mathbf{C} is the observation model to convert the state x_k to measurements.

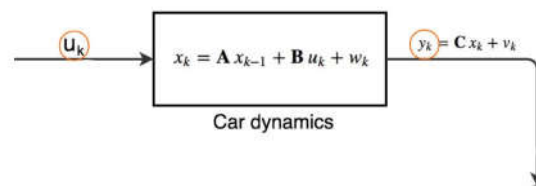
$v_k \sim \mathcal{N}(0, \mathbf{R}_k)$ is the sampled measurement noise like sensor noise.

\mathbf{R}_k is the covariance matrix of the measurement noise.

Kalman Filter

Real world

In the real world, we know the input control u and the measurement y . Through dynamic mechanic in Physics or experiments, it is not too hard to find \mathbf{A} , \mathbf{B} , and \mathbf{C} .



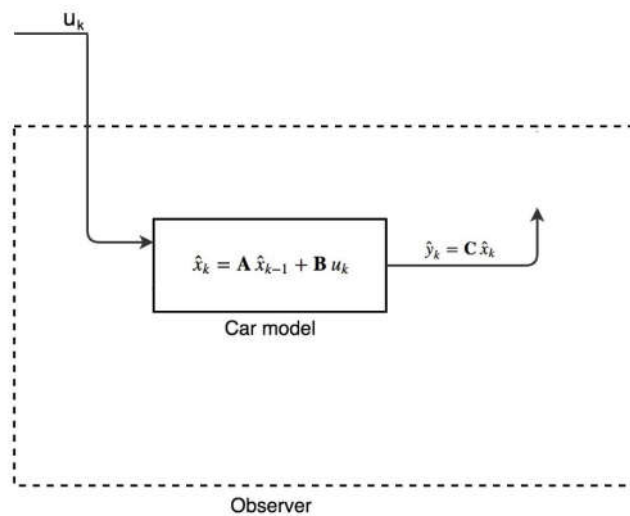
Observer world

Now, we use this information to build a model in an observer world to resemble the real world. In the observer world, we calculate the estimated measurement \hat{y} with the following equations:

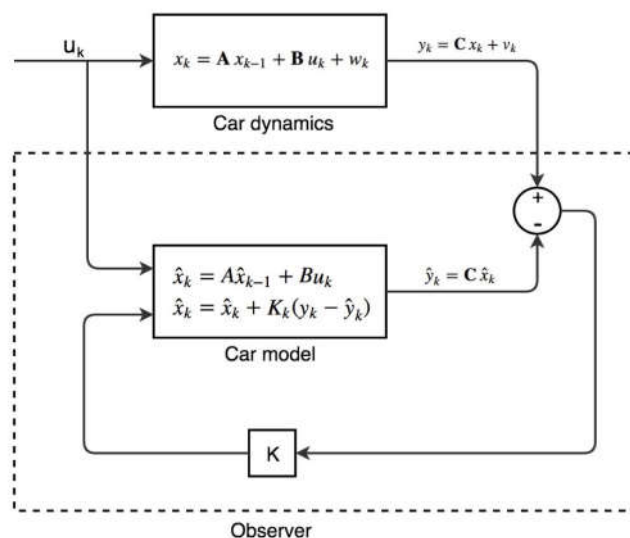
$$\begin{aligned}\hat{x}_k &= \mathbf{A} \hat{x}_{k-1} + \mathbf{B} u_k \\ \hat{y}_k &= \mathbf{C} \hat{x}_k\end{aligned}$$

Symbols with a hat, like \hat{y} , mean values estimated in the observer world.

Here is the visualization of the observer world.



\hat{y} is what the car model's estimation on the measurement. We know that \hat{y} will be off since our car model does not include process noise like the wind. By knowing the error ($y - \hat{y}$) between the measurement and the measurement estimate, we can refine the car model to make a better estimate for x . We just need to introduce a Kalman gain K to map the error in our measurement estimate to the error in our state estimate. Then our new x estimate will simply be the old estimate plus its error. In short, we use the error in our measurement estimate to make an adjustment to the state estimate.



Now our model involves 2 steps. The first step is the **prediction**:

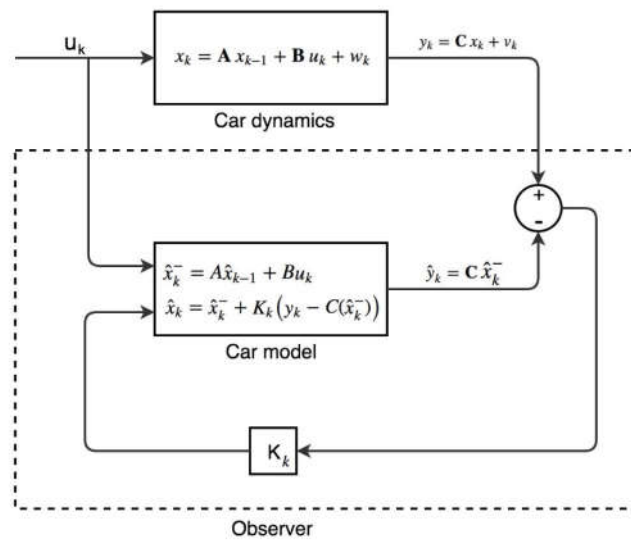
$$\hat{x}_k = A\hat{x}_{k-1} + Bu_k$$

The second step is the **update** of our estimated state with the error ($y - \hat{y}$):

$$\hat{x}_k = \hat{x}_k + K_k(y_k - \hat{y}_k)$$

Since we break the state estimation into 2 steps, there is an estimated state before the update and one after the update. To reduce confusion, we introduce the notation with a **minus** sign to indicate the **estimated state before the update**. Now our car model becomes:

$$\begin{aligned}\hat{x}_k^- &= A\hat{x}_{k-1} + Bu_k \\ \hat{x}_k &= \hat{x}_k^- + K_k(y_k - C\hat{x}_k^-)\end{aligned}$$



Let's take a quick peek at how K is calculated,

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

where R quantifies the measurement noise. Let's do an insanity check. In our location example, C is an identity matrix. If there is no noise, K becomes an identity matrix. Using our car model equation, it will output the measurement y as the estimated state. That is, when the measurement is 100% accurate, our prediction should equal the measurement.

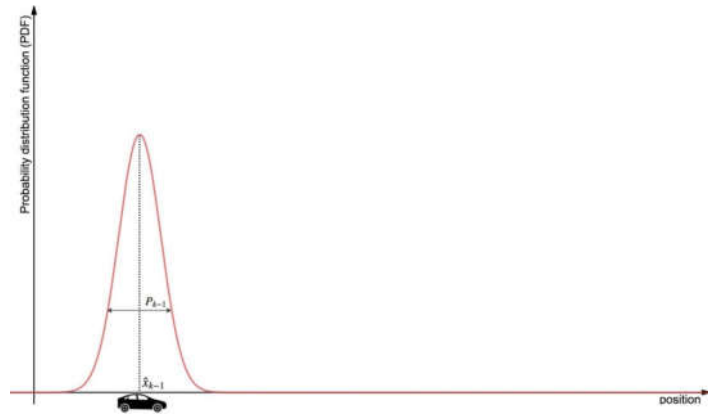
$$\begin{aligned}\hat{x}_k &= A\hat{x}_{k-1} + Bu_k + K_k(y_k - \hat{y}_k) \\ &= A\hat{x}_{k-1} + Bu_k + K_k(y_k - C(A\hat{x}_{k-1} + Bu_k)) \\ &= A\hat{x}_{k-1} + Bu_k + (y_k - A\hat{x}_{k-1} + Bu_k) \\ &= y_k\end{aligned}$$

Now, we create a car model in the observed world that take into the account of the measurement noise in the form of Kalman gain.

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} .



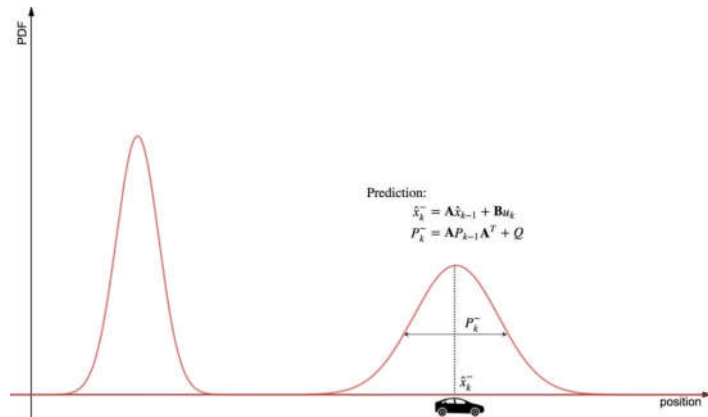
Then, we apply the car model in the observer world to make a new state estimate.

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

Back to linear algebra, if we apply a transformation matrix A to an input x with a covariance matrix Σ , 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:

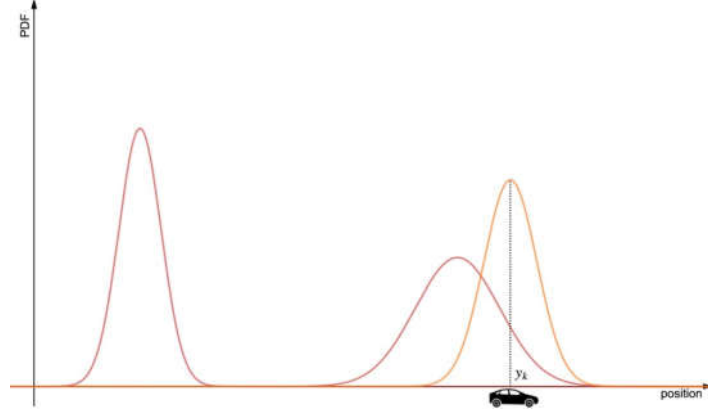


where,

\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.

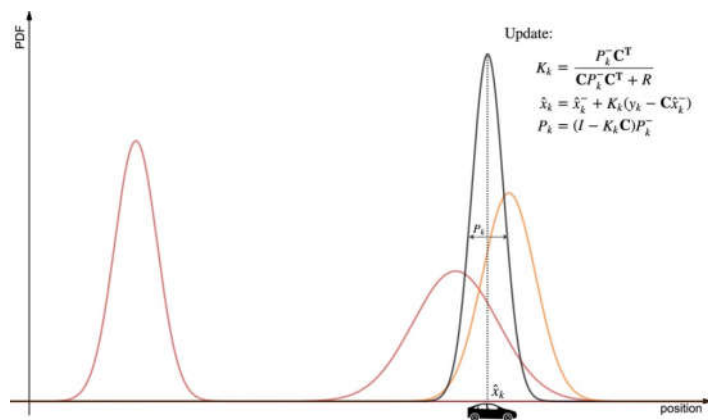


Since the estimated state (before the update) and the measurement are Gaussian distributed, the final estimated state is also Gaussian distributed. We can apply linear algebra again to compute the Gaussian model of our final prediction.

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.

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

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:



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 .

Congratulation! You survive the tedious notations and this is how we use Kalman filter to make better state estimation. Comparing with our previous explanation, we do not multiple curves together. Kalman Filter uses simple linear algebra and is much simpler.

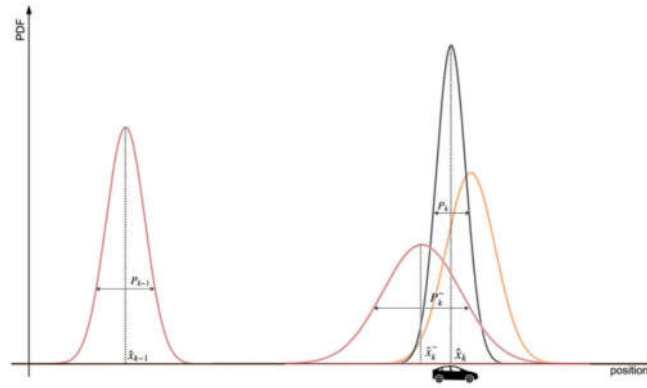
Recap

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 .

The diagram below shows the corresponding mean and the covariance matrix.



PDFs modeled with Gaussian distributions.

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

Prediction	Update
$\hat{x}_k^- = \mathbf{A}\hat{x}_{k-1} + \mathbf{B}u_k$ $P_k^- = \mathbf{A}P_{k-1}\mathbf{A}^T + \mathbf{Q}$	$K_k = \frac{P_k^- \mathbf{C}^T}{\mathbf{C}P_k^- \mathbf{C}^T + R}$ $\hat{x}_k = \hat{x}_k^- + K_k(y_k - \mathbf{C}\hat{x}_k^-)$ $P_k = (\mathbf{I} - K_k \mathbf{C})P_k^-$

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.

\mathbf{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.

More thoughts

We use linear algebra to model our car. i.e. A , B and C are simply matrix. It may not always true in the real world. For [next article](#), we

will talk about Extended Kalman Filter and Unscented Kalman Filter to overcome this problem.