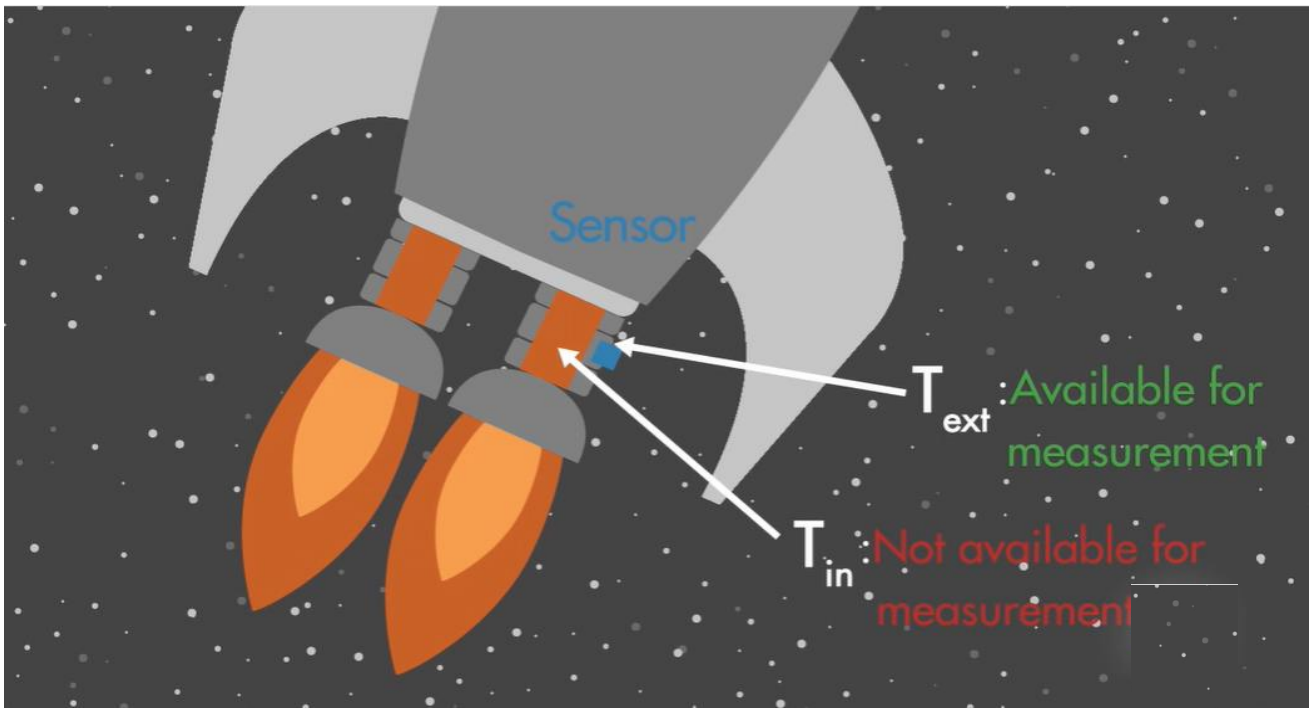
Kalman filter

Why Use Kalman Filters?

A Kalman filter is an optimal estimation algorithm.



To illustrate this, let's go to Mars before anyone else does. If your spacecraft's engine can burn fuel at a high enough temperature, it can create thrust that will let you fly to Mars. By the way, according to NASA, liquid hydrogen is a light and powerful rocket propellant that burns with extreme intensity at 5,500 degrees Fahrenheit.

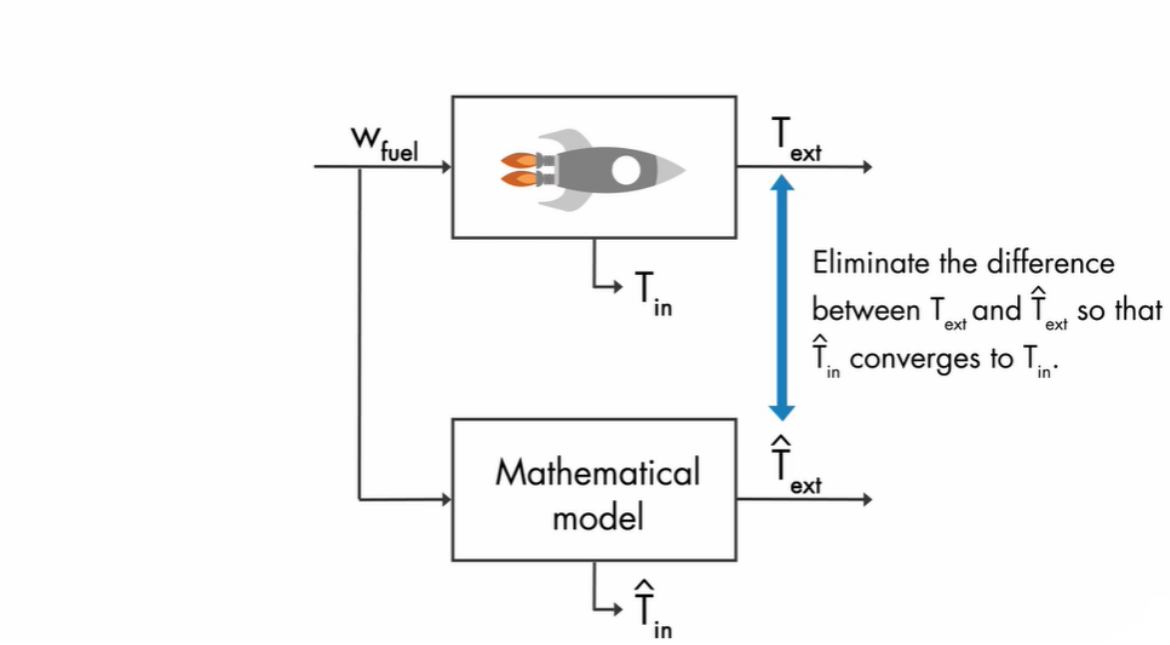
But be careful, because at too-high temperature it can put the mechanical components of the engine at risk, and this can lead to the failure of some of the mechanical parts. If that happens, you might be stuck in your small spacecraft where you've got to eat from tubes. To prevent such a situation, you should closely monitor internal temperature of the

combustion chamber. This is not an easy task, since a sensor placed inside the chamber would melt.

Instead, it needs to be placed on a cooler surface close to the chamber. The problem you are facing here is that you want to measure internal temperature of the chamber, but you can't. Instead you have to measure external temperature. In this situation, you can use a Kalman filter to find the best estimate of the internal temperature from an indirect measurement. This way you're extracting information about what you can't measure from what you can.

If it gets too hot, it could damage your spacecraft. However, there isn't any feasible way of measuring the internal temperature, since a sensor placed inside the engine would melt.

What you can do is to place the sensor on a colder surface and measure the temperature there. Let's call this temperature t external, and the one you can't measure but you need to estimate, t internal. This is your rocket. You're wondering how high the internal temperature of the engine is, since this will tell you how you should regulate the fuel flow to your rocket.



However, you don't have access to the state t internal. Instead, you can measure t external. The signals that are available to you are the fuel flow and your measurements. How do you estimate the internal temperature? Actually, you have access to more information. Since your method's pretty good, you can derive the equations that will give you the mathematical model of your real system.

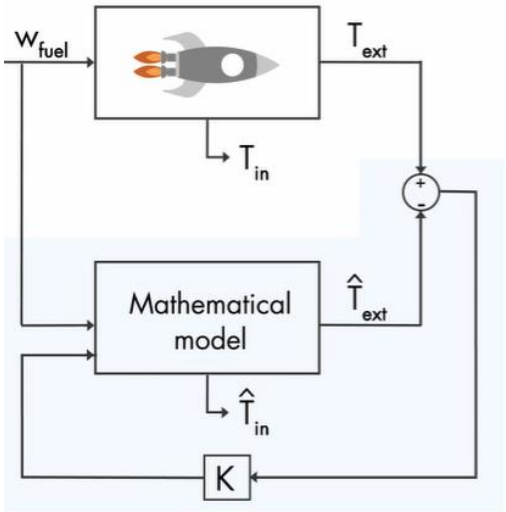
You already know how much fuel you're adding to your rocket, so if you now input this fuel flow to your mathematical model, this will give you an estimate of your output. And also, notice that since you have all the governing equations, you can even calculate the internal state of the system.

Okay, does this solve your problem now? Unfortunately, it doesn't. No doubt that you're good at math, but , the mathematical model you found is only an approximation of your real system. It is subject to uncertainties. If you had a perfect model without any uncertainties, and if your real system and your model had the same initial conditions, then your measurement and estimated output values would match each other.

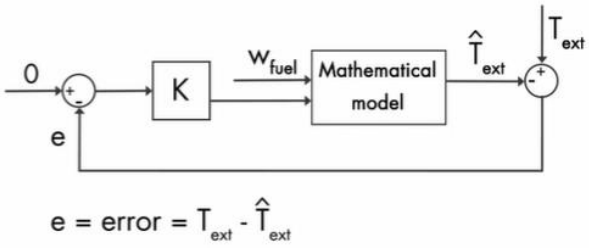
And therefore, the estimate of the internal temperature would match the true internal temperature as well. But in real life, this is an unlikely scenario, and therefore, the estimated external temperature won't match the measured temperature. That's why you need to use a state estimator to estimate your internal states.

Let's see how a state estimator works. Here, our goal is to match the estimated external temperature with the measured external temperature. We know that if these two are equal, then the model will converge to the real system. So the estimated internal temperature will converge to its true value.

What we're trying to do is to minimize the difference between the estimated and measured external temperature.



STATE OBSERVER

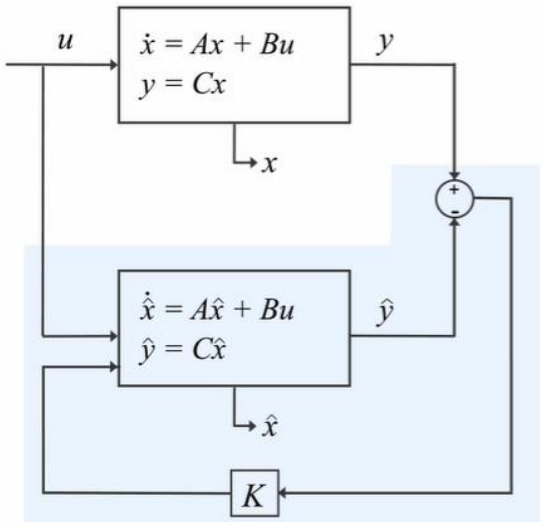


Actually, we're talking about a feedback control system, where we try to control the error between the measured and estimated external temperature at 0 using a controller k.

If we now update our diagram on the left-hand side based on what we've discussed here, it will look like this. This part represents the state observer. By closing the loop with a controller k around the observer, we try to eliminate the error between the estimated and measured external temperature such that the estimated internal temperature is driven to its true value.

In summary, you can't directly measure the internal engine temperature, but you know how much fuel

you're supplying to your rocket. So you can run this through your mathematical model and estimate the output, which you can then use along with your real measurements to estimate the internal state of the system.



STATE OBSERVER

$$e_{obs} = x - \hat{x}$$

$$\dot{x} = Ax + Bu \qquad y = Cx$$

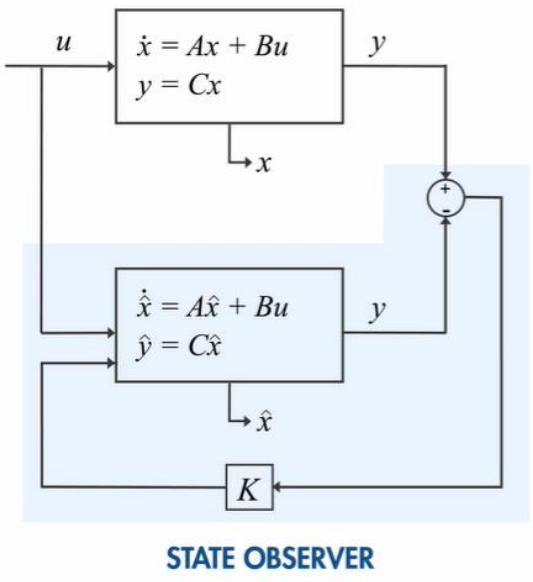
$$\dot{\hat{x}} = A\hat{x} + Bu + K(y - \hat{y}) \qquad \hat{y} = C\hat{x}$$

$$\underbrace{\dot{x} - \dot{\hat{x}}}_{\dot{e}_{obs}} = \underbrace{Ax - A\hat{x}}_{Ae_{obs}} + \underbrace{Bu - Bu}_{0} - \underbrace{K(y - \hat{y})}_{K(C\underbrace{(x - \hat{x})}_{e_{obs}})} \qquad y - \hat{y} = C\underbrace{(x - \hat{x})}_{e_{obs}}$$

$$\dot{e}_{obs} = (A - KC)e_{obs} \qquad y - \hat{y} = Ce_{obs}$$

let's look at how we can explain the state observer mathematically. We will generalize the problem and show the input as u, the output as y, and any states we want to estimate as x.

Our goal is to derive x hat to x, so we can define the difference between these values as an error. Next, let's write down the equations for the system and the observer. If we subtract these equations from each other, this will give us the error dynamics. By rearranging terms, we see that the aerodynamics can be shown by this equation.



STATE OBSERVER

$e_{obs} = x - \hat{x}$

$\dot{x} = Ax + Bu$ $\quad y = Cx$

$\dot{\hat{x}} = A\hat{x} + Bu + K(y - \hat{y})$ $\quad \hat{y} = C\hat{x}$

$\dot{e}_{obs} = (A - KC)e_{obs}$ $\quad y - \hat{y} = Ce_{obs}$

$\rightarrow e_{obs}(t) = e^{(A-KC)t}e_{obs}(0)$

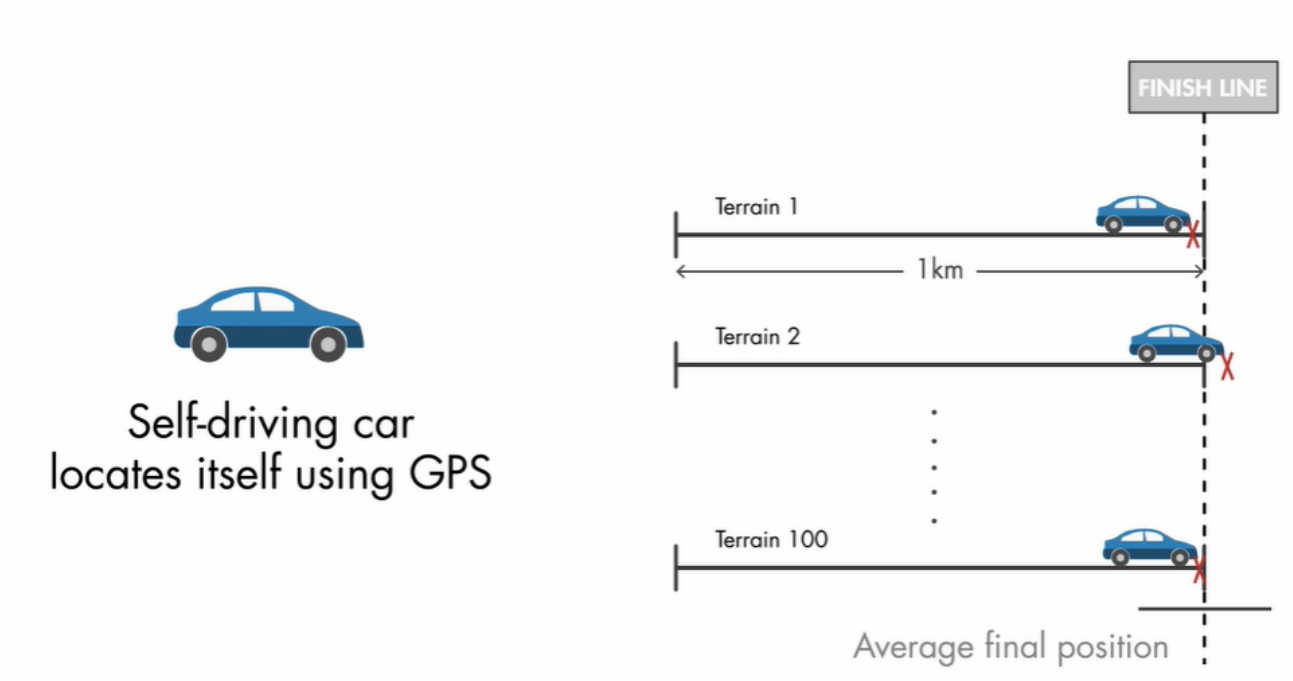
If $(A-KC) < 0$, then $e_{obs} \rightarrow 0$ as $t \rightarrow \infty$. So, $\hat{x} \rightarrow x$.

The solution to this equation is an exponential function. What this means is if this term is less than zero, we're good because we know that our error will vanish over time, and x hat will converge to x. At this point, you may be asking if we really need the kc term in this equation, because even without the feedback loop that adds the kc term to the equation, we will have a decaying exponential function for the error.

The significance of having a feedback loop around the observer is that we can control the decaying rate of the error function by selecting the controller gain k accordingly. However, here the decay rates solely depends on the matrix a. And if there are some

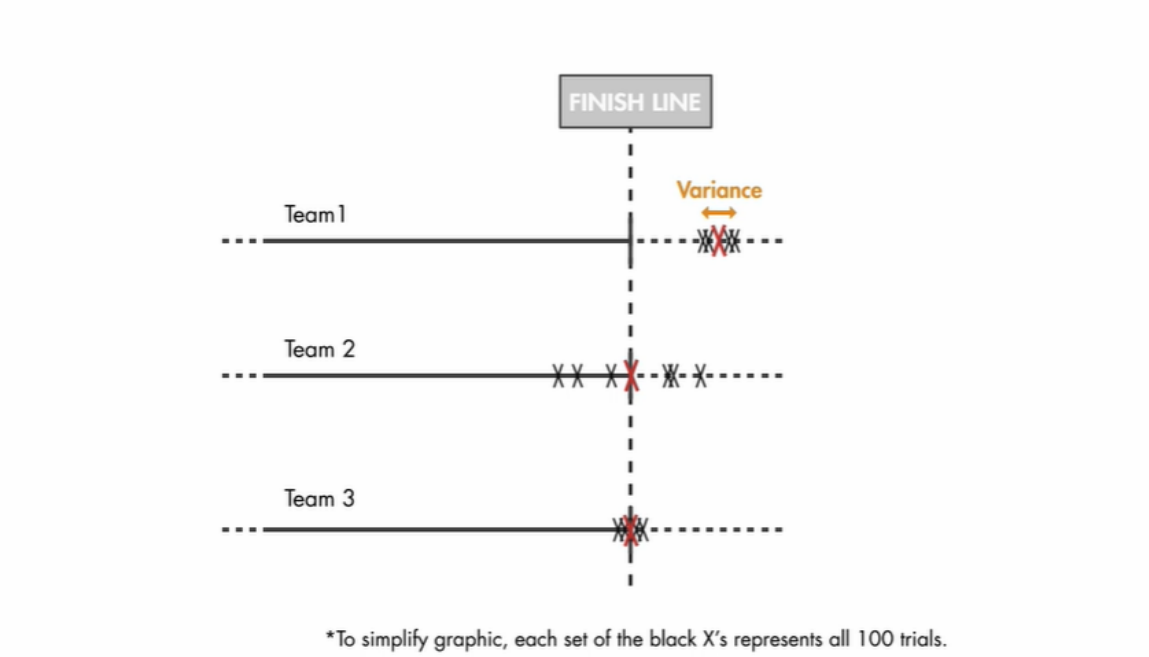
uncertainties in the mathematical model, this means you don't know a exactly.

Therefore, you can't control how quickly the error will vanish. Having the feedback controller gives you more control over this equation and guarantees a faster elimination of the error. And the faster the error vanishes, the faster the estimated states x hat converge to the true states x. An optimal way of choosing the gain k is performed through the use of Kalman filters.



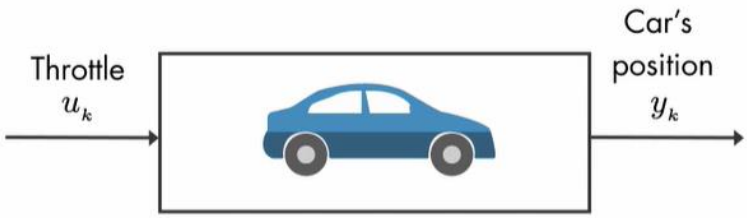
An optimal state estimator

Your car is supposed to drive 1 kilometer on 100 different terrains. Each time, it must stop as close as possible to the finish line. At the end of the competition, the average final position is computed for each team, and the owner of the car with the smallest error variance and average final position closest to 1 kilometer gets the big prize.

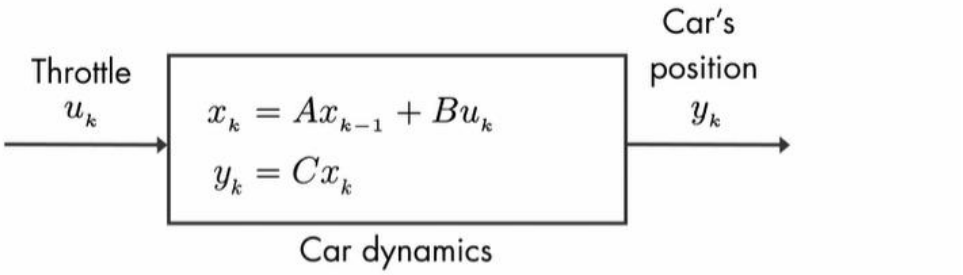


Here's an example. Let these points represent the final position, and the red ones the average final position for different teams. Based on these results, Team 1 would lose due to the biased average final position, although it has small variance. Team 2 would lose as well. Its average final position is on the finish line, but it has high variance. The winner will be Team 3, since it has the smallest variance, and its average final position is on the finish line.

In order to meet the required criteria to win the competition, you can estimate the car's position using a Kalman filter. Let's look at the system to understand how the Kalman filter works.

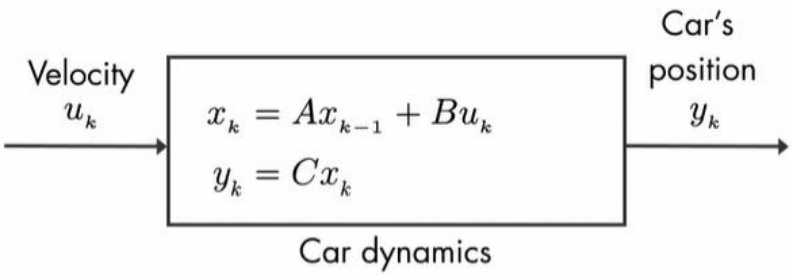


The input to the car is a throttle. The output that we're interested in is the car's position.



$$x_k = \begin{bmatrix} velocity \\ position \end{bmatrix}$$

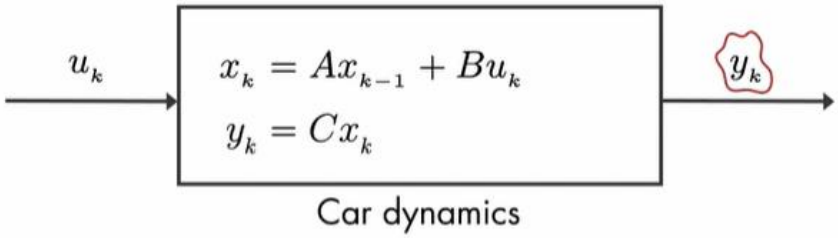
But here, to give you intuition, we'll assume an overly simplistic system, where the input to the car is the velocity.



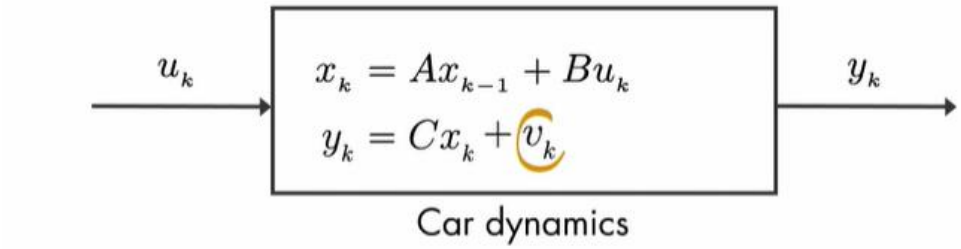
$$x_k = [position]$$

$$C = 1$$

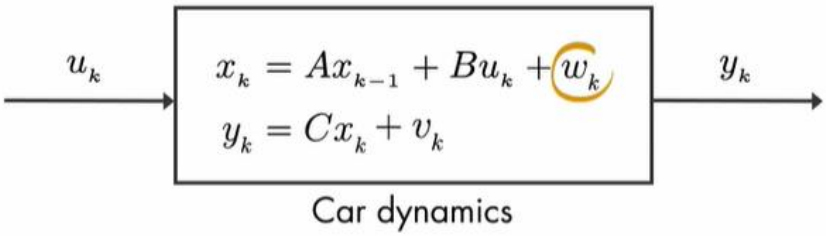
This system will have a single state, the car's position. And we're measuring the states so matrix C is equal to 1.



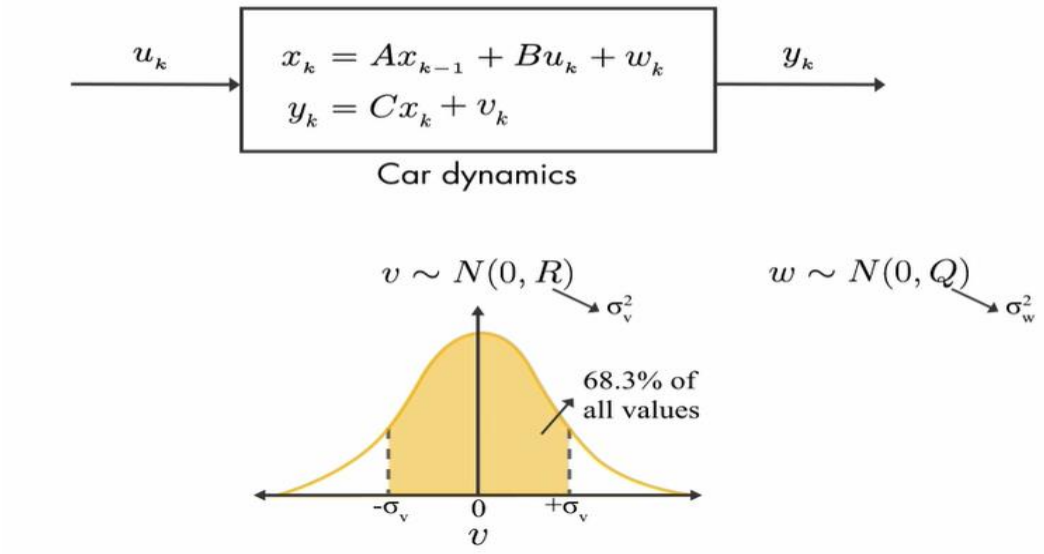
It's important to know y as accurately as possible, since we want the car to finish as close as possible to the finish line.



But the GPS readings will be noisy. We'll show this measurement noise with v, which is a random variable.



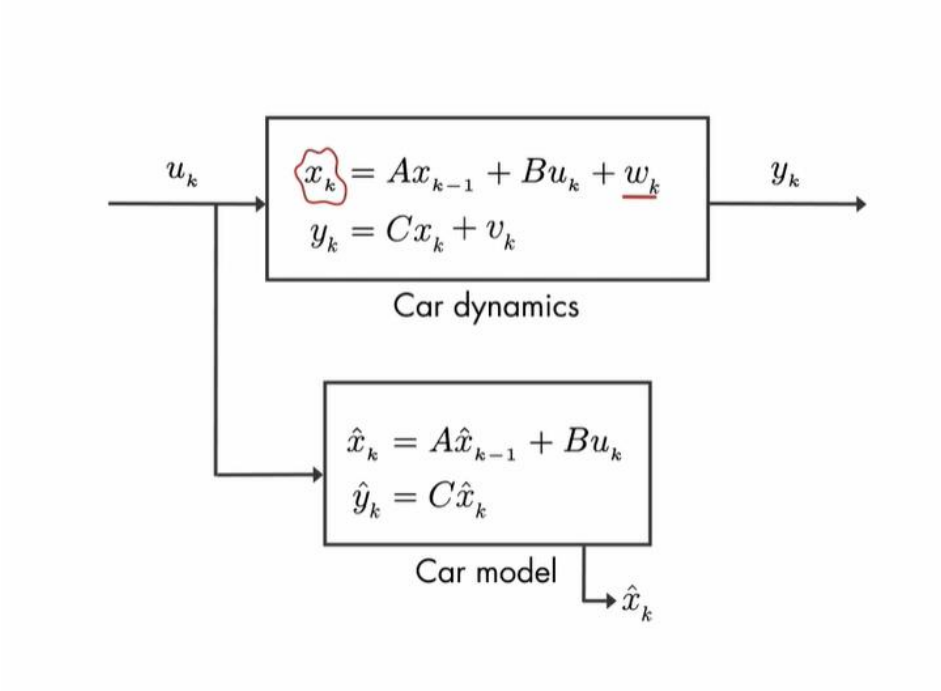
Similarly, there is process noise, which is also random and can represent the effects of the wind or changes in the car's velocity.



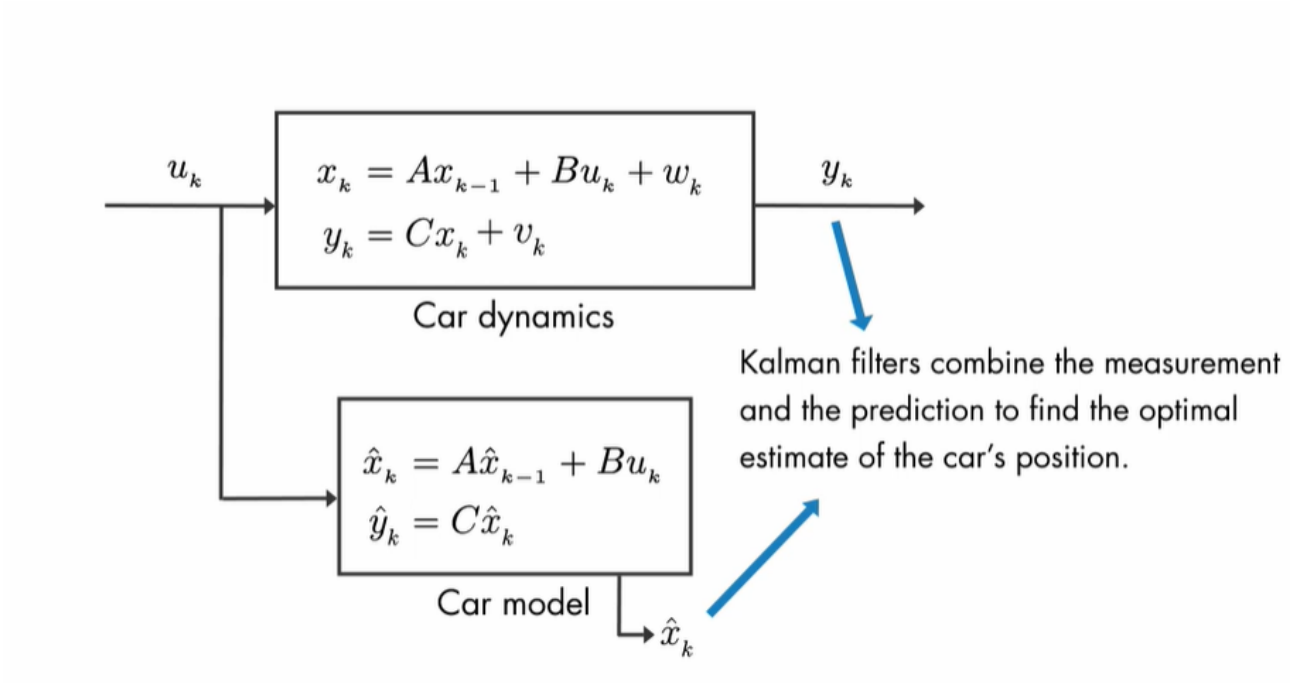
Although these random variables don't follow a pattern, using probability theory, we can tell something about their average properties. v, for example, is assumed to be drawn from a Gaussian distribution with zero mean and covariance R. This means if we measured the position of the car, let's say 100 times at the same location, the noise in these readings will take on values, with most of them located near to zero mean and fewer located farther away from it. And this results in the Gaussian distribution, which is described by the covariance R.

Since we have a single output system, the covariance R is scalar and is equal to the variance of the measurement noise. Similarly, the process noise is

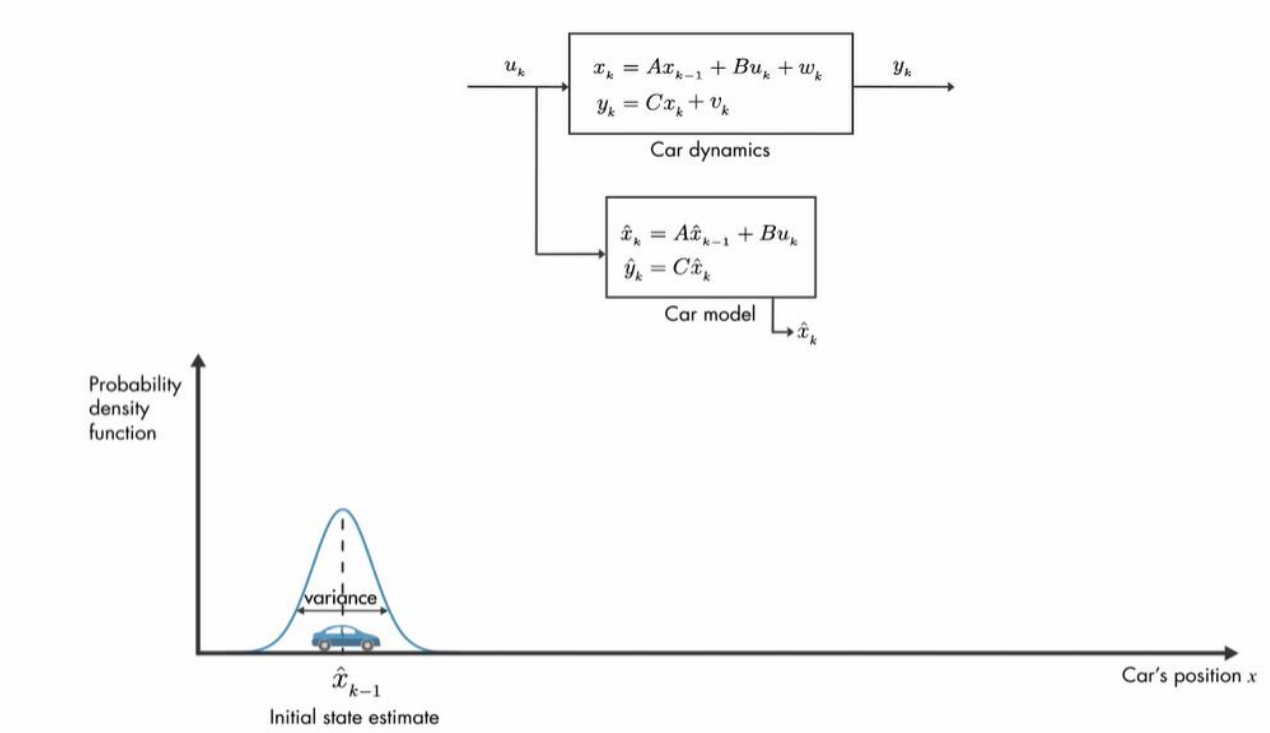
also random and assumes a Gaussian distribution with covariance Q. Now, we know that the measurement is noisy, and therefore what we measure doesn't quite reflect the true position of the car.



Now, we know that the measurement is noisy, and therefore what we measure doesn't quite reflect the true position of the car. If we know the car model, we can run the input through it to estimate the position. But this estimate also won't be perfect, because now we're estimating x, which is uncertain due to the process noise.

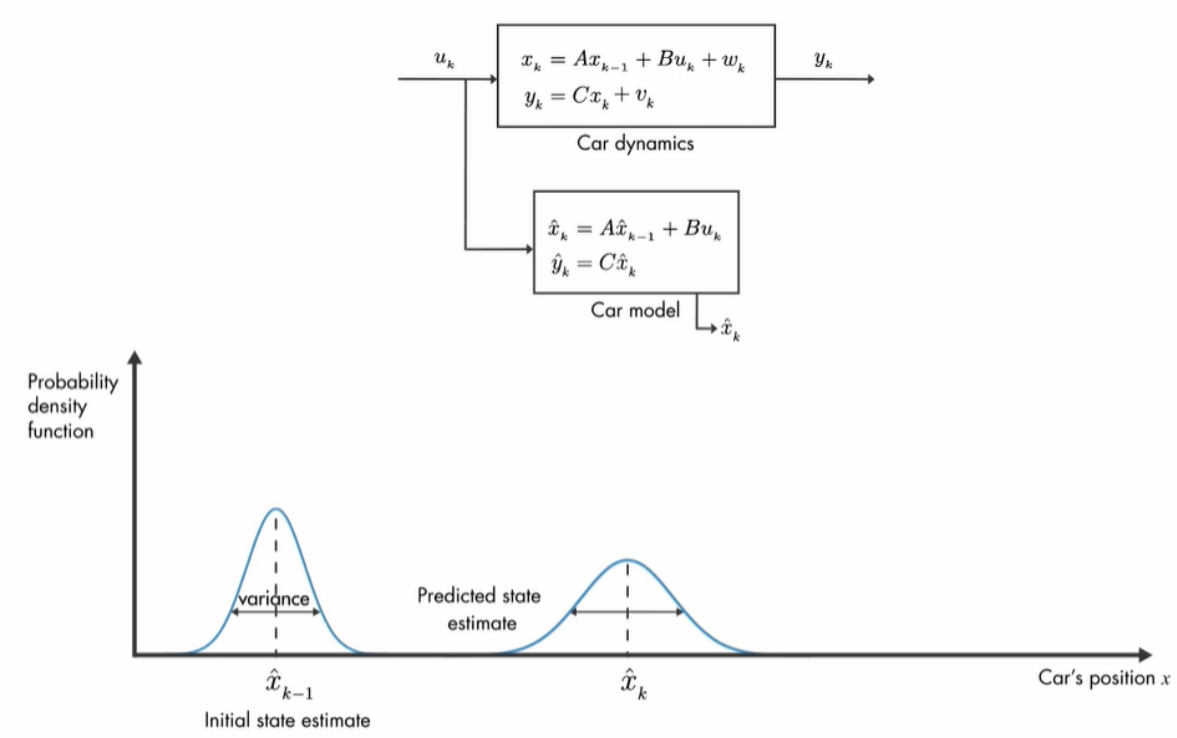


This is where the Kalman filter comes into play. It combines these two pieces of information to come up with the best estimate of the car's position in the presence of process and measurement noise.

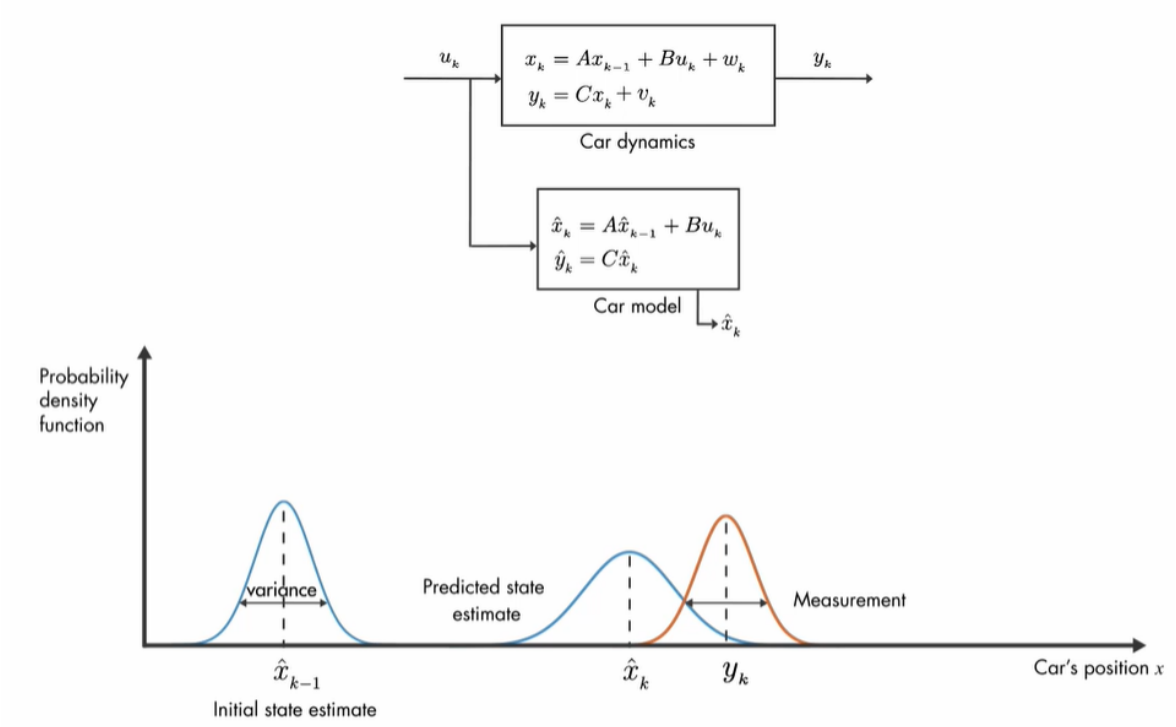


We'll discuss the working principle of the Kalman filter visually with the help of probability density functions. At the initial time step, k minus 1, the actual car position can be anywhere around the estimate x hat k minus 1. And this uncertainty is described by this probability density function.

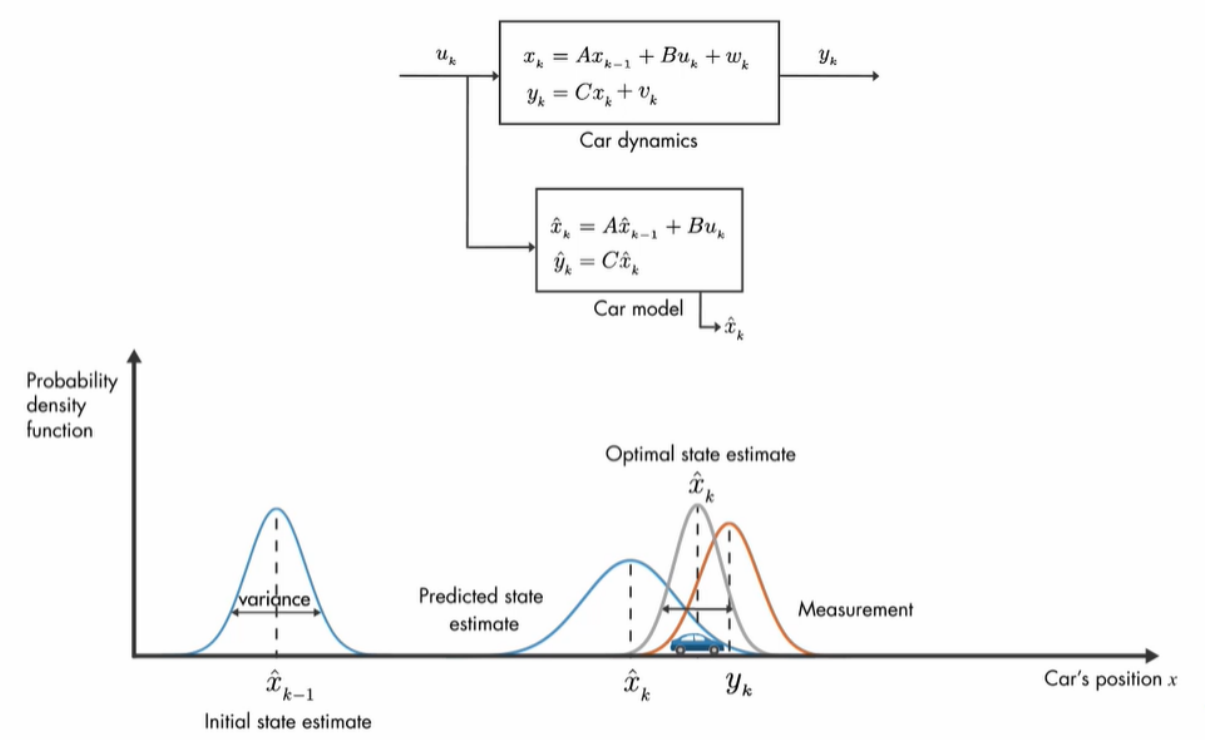
What this plot also tells us is that the car is going to be most likely around the mean of this distribution.



At the next time step, the uncertainty in the estimate has increased, which is shown with the larger variance. This is because between time step k minus 1 and k, the car might have run over a pothole, or maybe the wheels may have slipped a little bit. Therefore, it may have traveled a different distance than what we have predicted by the model.



As we discussed before, another source of information on car's position comes from the measurement. Here the variance represents the uncertainty in the noise measurement. Again, the true position can be anywhere around the mean.



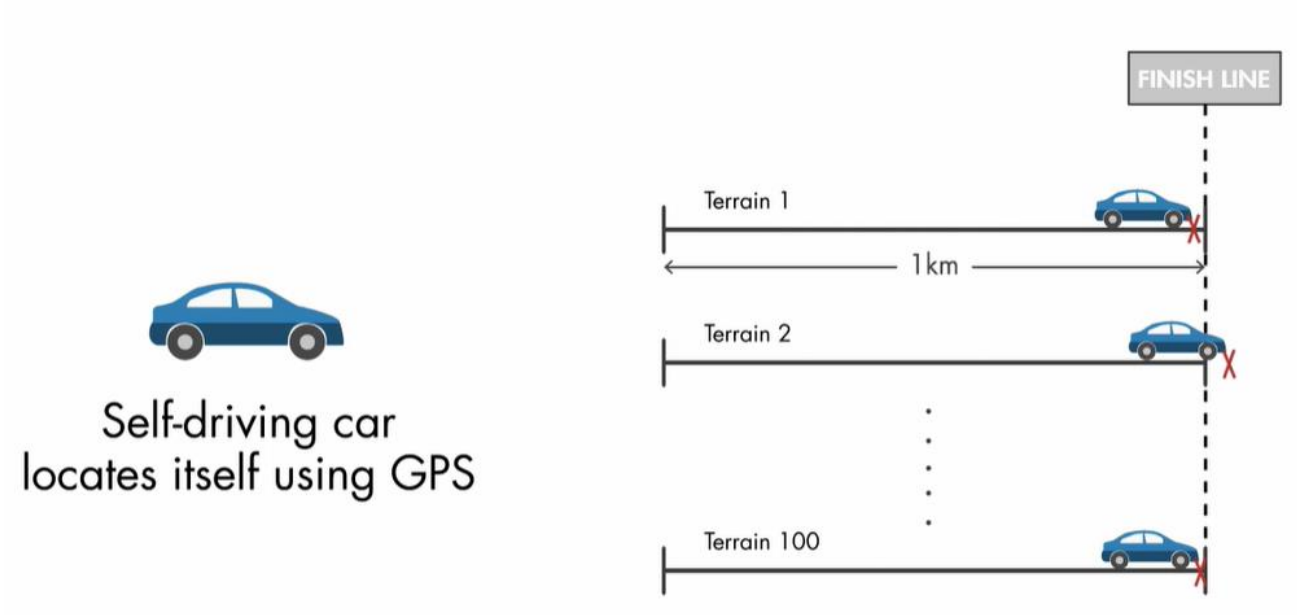
Now that we have the prediction and measurement, the question is, what is the best estimate of the car's position? It turns out that the optimal way to estimate the car's position is by combining these two pieces of information. And this is done by multiplying these two probability functions together. The resulting product is also a Gaussian function.

This estimate has a smaller variance than either of the previous estimates, and the mean of this probability density function gives us the optimal estimate of the car's position. This is the basic idea behind Kalman filters.

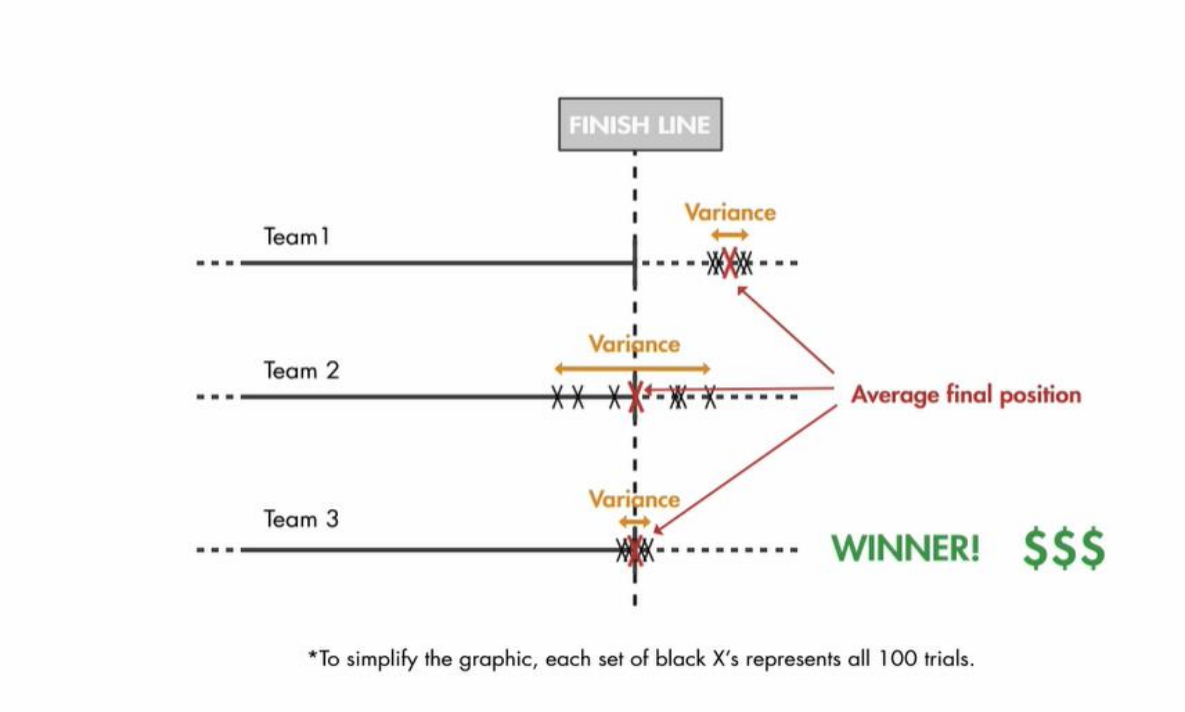
Optimal State Estimator Algorithm

we'll discuss the set of equations that you need to implement the Kalman filter algorithm. Let's revisit the example

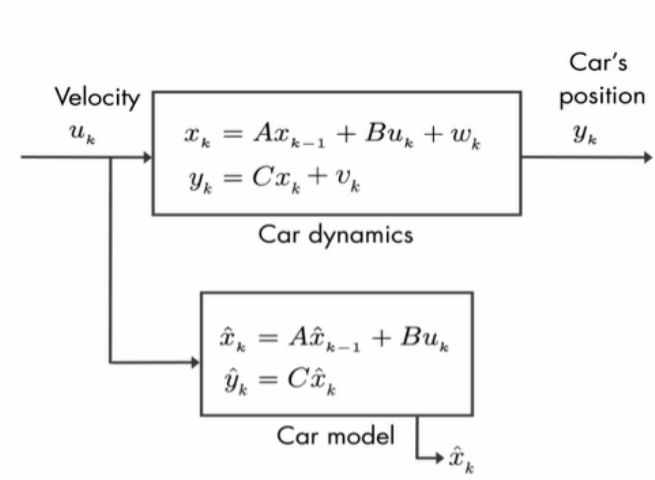
You join a competition to win the big prize. You're asked to design a self-driving car that needs to drive one kilometer on 100 different terrains.



n each trial, the car must stop as close as possible to the finish line.

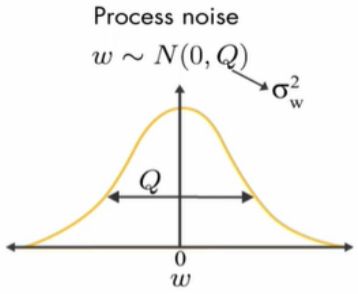


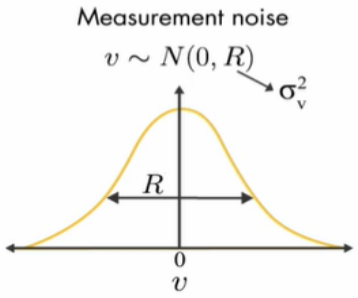
At the end of the competition, the average trial position is computed for each team, and the owner of the car with the smallest error variance in an average trial position closest to 1 kilometer gets the big prize.



$x_k = [position]$

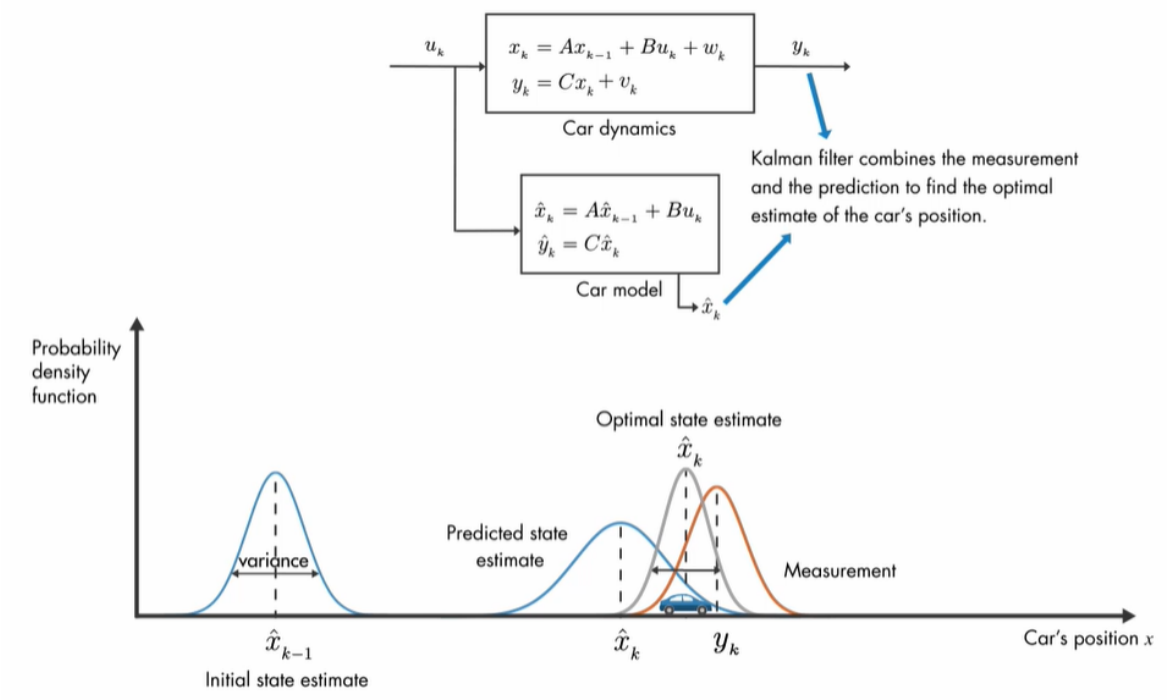
$C = 1$





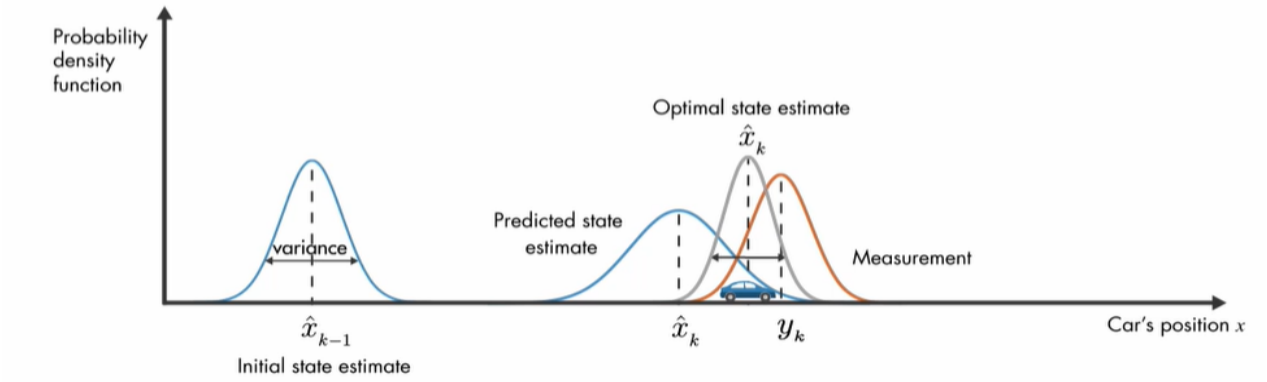
In that example, we also showed the car dynamics and the car model for our single-state system, and we discussed process and measurement noises along with the covariances.

Xk-1 is the just previous state, xk is the state we need to determine, yk is the output



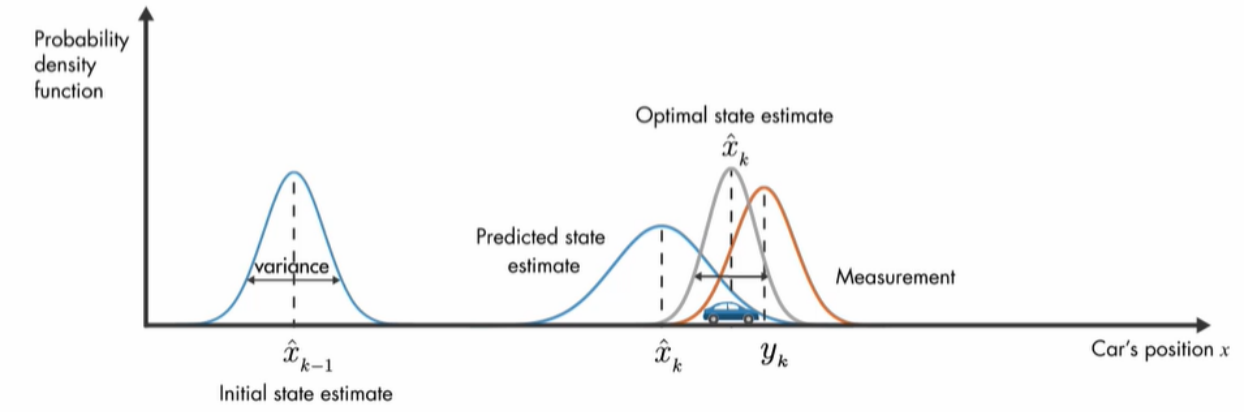
Finally, we said that you could win the competition by using a Kalman filter, which computes an optimal, unbiased estimate of the car's position with minimal variance.

Kalman filter $\hat{x}_k = A\hat{x}_{k-1} + Bu_k + K_k(y_k - C(A\hat{x}_{k-1} + Bu_k))$

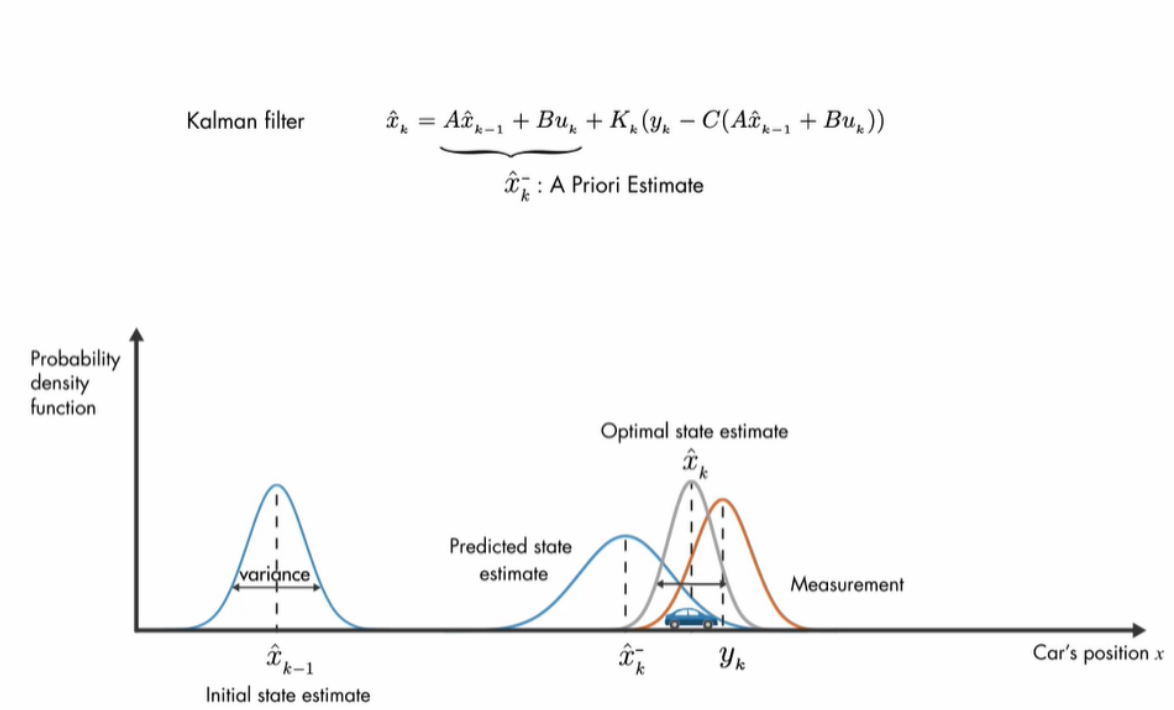


This optimal estimate is found by multiplying the prediction and measurement probability functions together, scaling the results, and computing the mean of the resulting probability density function. Computationally, the multiplication of these two probability density functions relates to the discrete Kalman filter equation shown here.

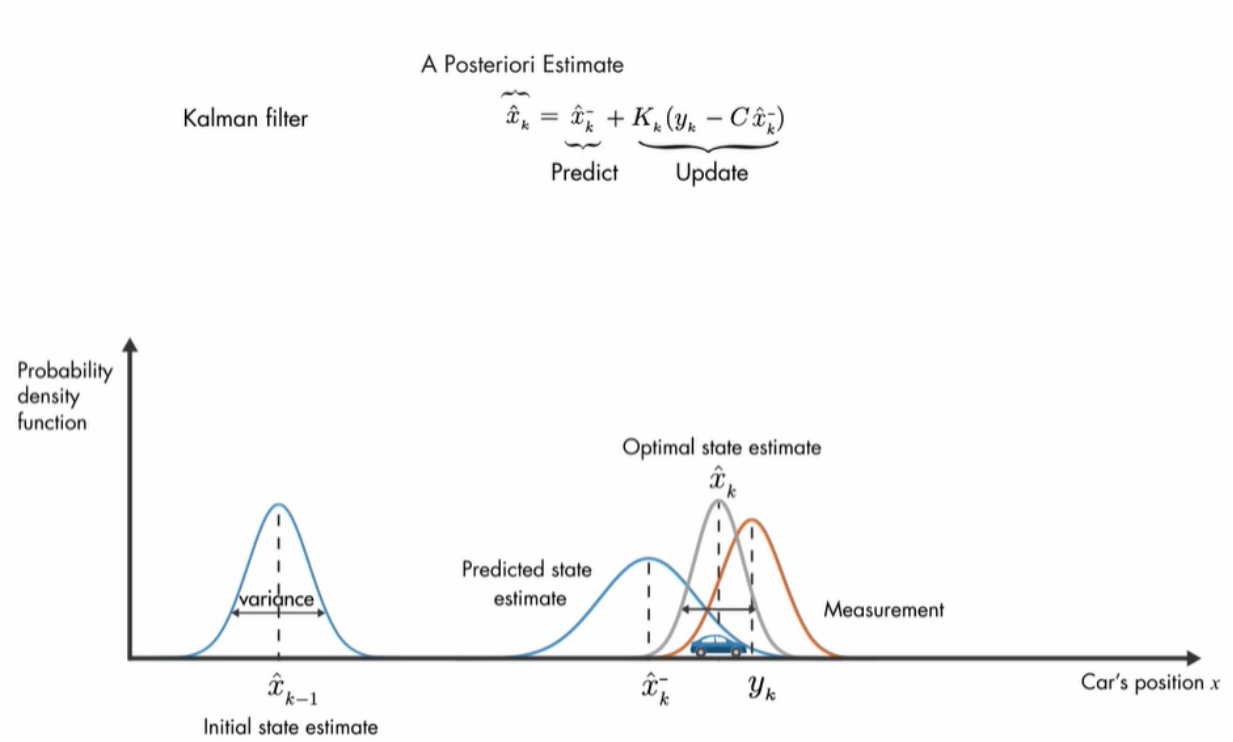
State observer	$\hat{x}_{k+1} = A\hat{x}_k + Bu_k + K(y_k - C\hat{x}_k)$	Deterministic system
Kalman filter	$\hat{x}_k = A\hat{x}_{k-1} + Bu_k + K_k(y_k - C(A\hat{x}_{k-1} + Bu_k))$	Stochastic system



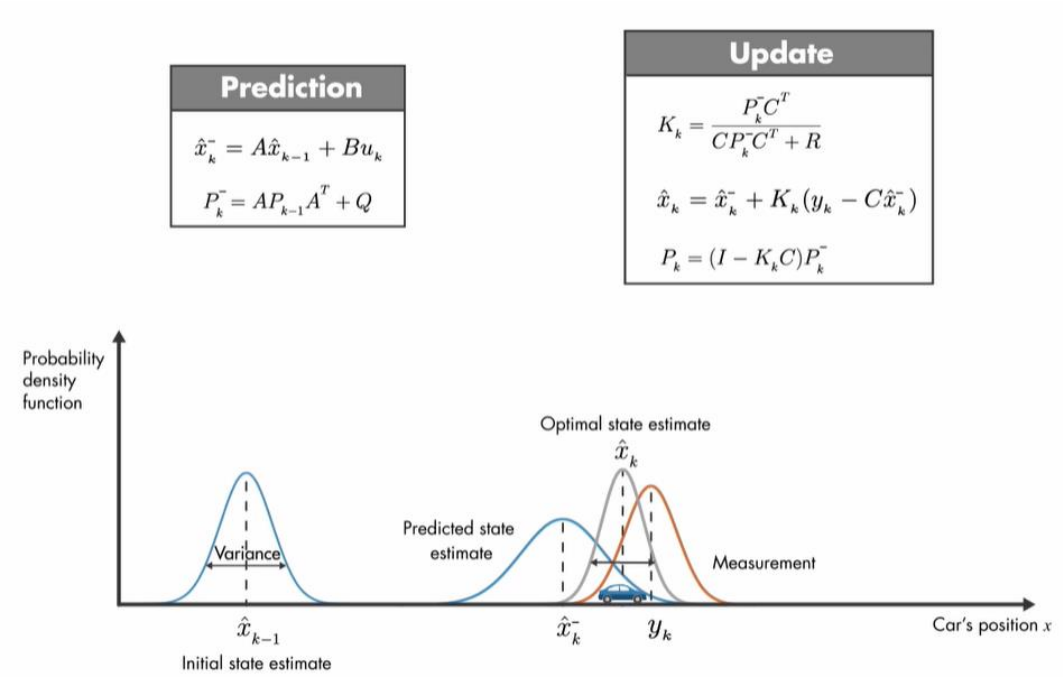
it look similar to the state observer equation that we discussed in previous videos? Actually, a Kalman filter is a type of state observer, but it is designed for stochastic systems. Here is how the Kalman filter equation relates to what we've discussed with the probability density functions.



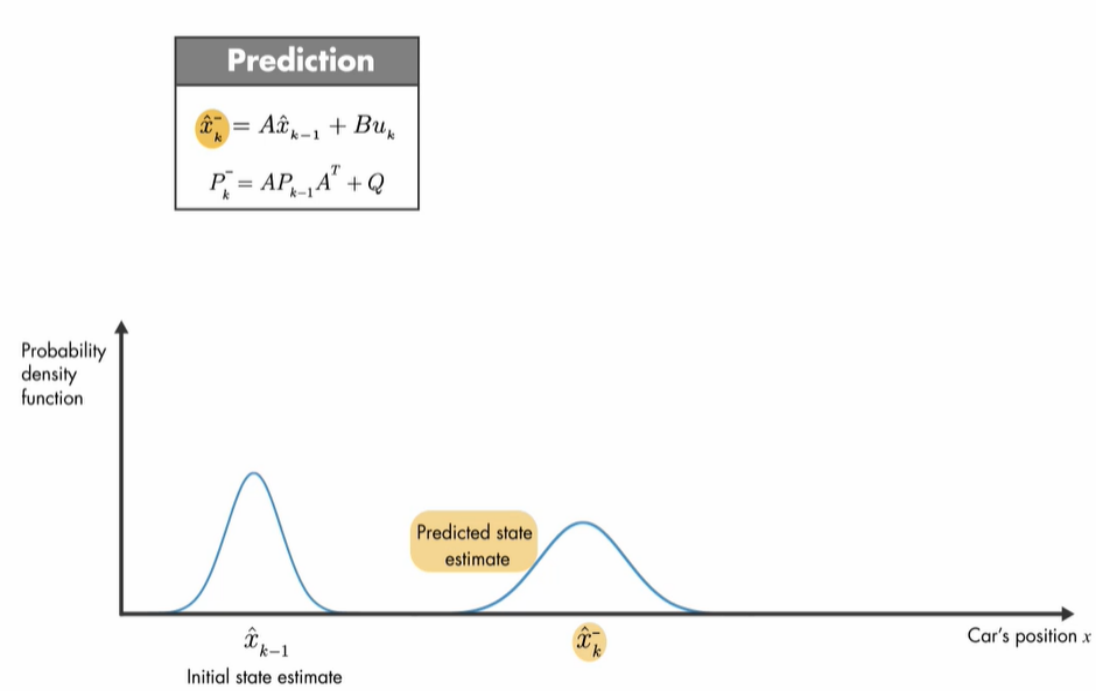
The first part predicts the current state by using state estimates from the previous timestep and the current input. Note that these two state estimates are different from each other. We'll show the predicted state estimate with this notation. This is also called the a priori estimate, since it is calculated before the current measurement is taken.



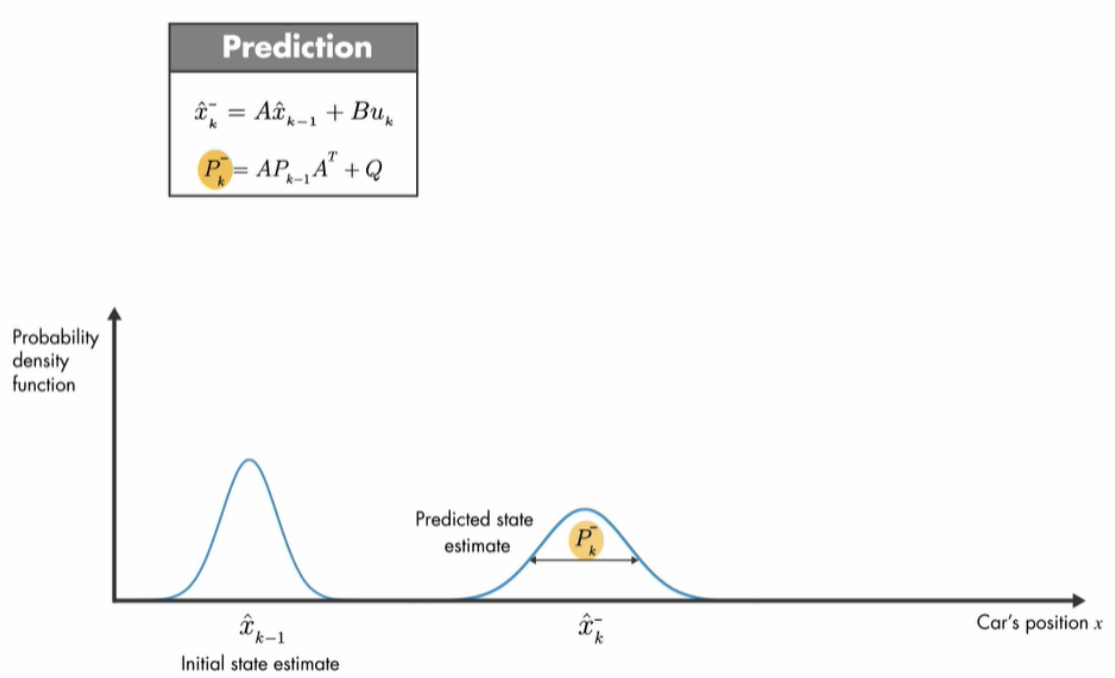
We can now rewrite the equation like this. The second part of the equation uses the measurement and incorporate it into the prediction to update the a priori estimate. And we'll call the result the a posteriori estimate.



we'll go over the algorithm equations step by step

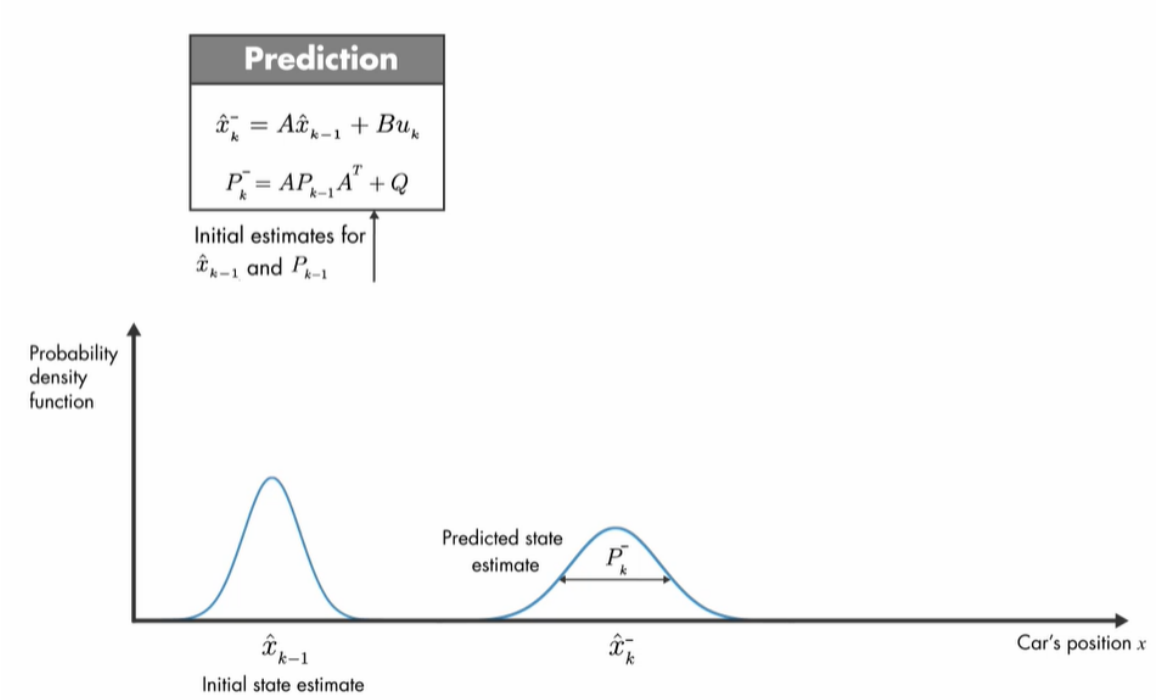


The Kalman filter is a two-step process. Let's first start with the prediction part. Here, the system model is used to calculate the a priori state estimate

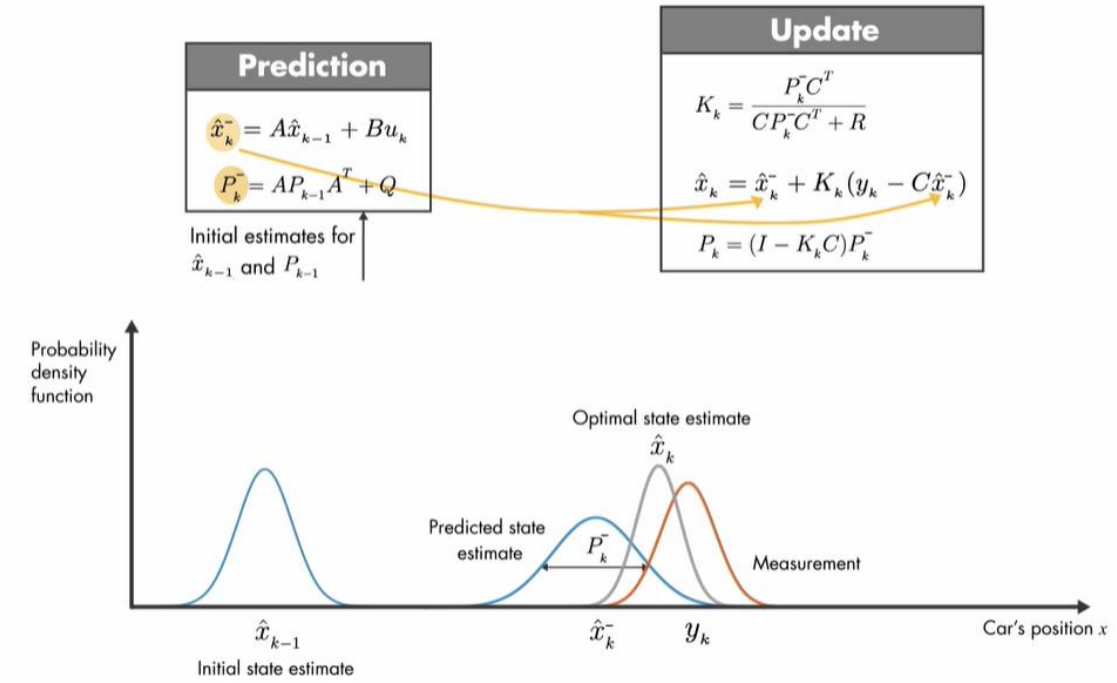


and the error covariance p.

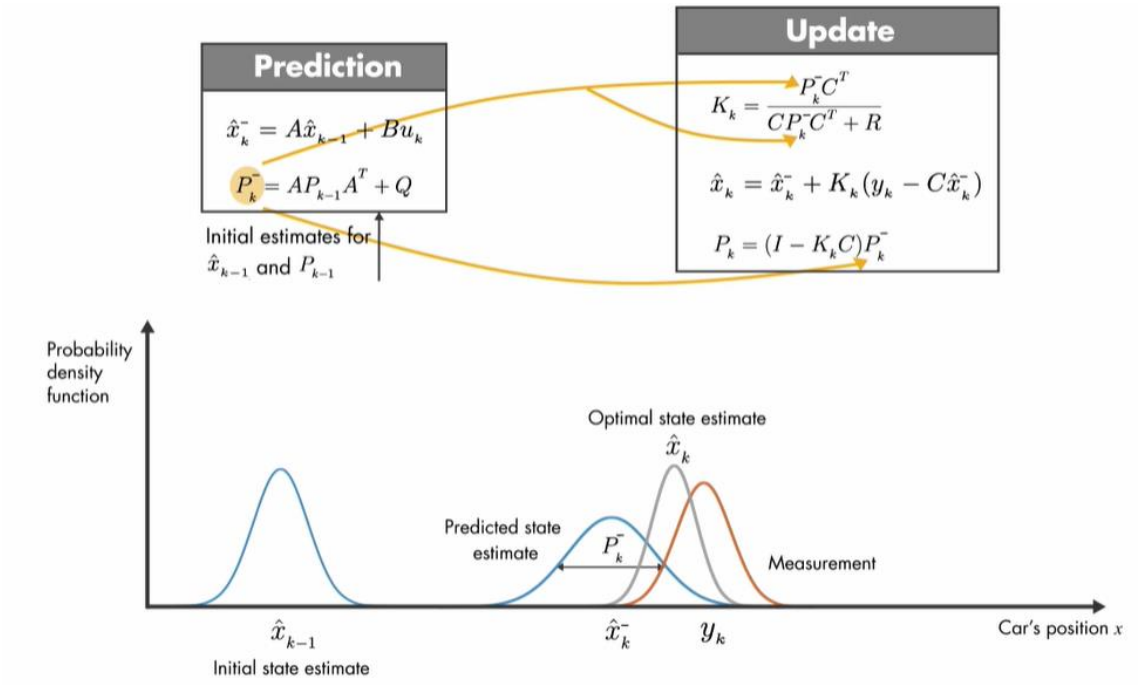
For our single-state system, p is the variance of the a priori estimate, and it can be thought of as a measure of uncertainty in the estimated state. This variance comes from the process noise and propagation of the uncertain x hat k minus 1.



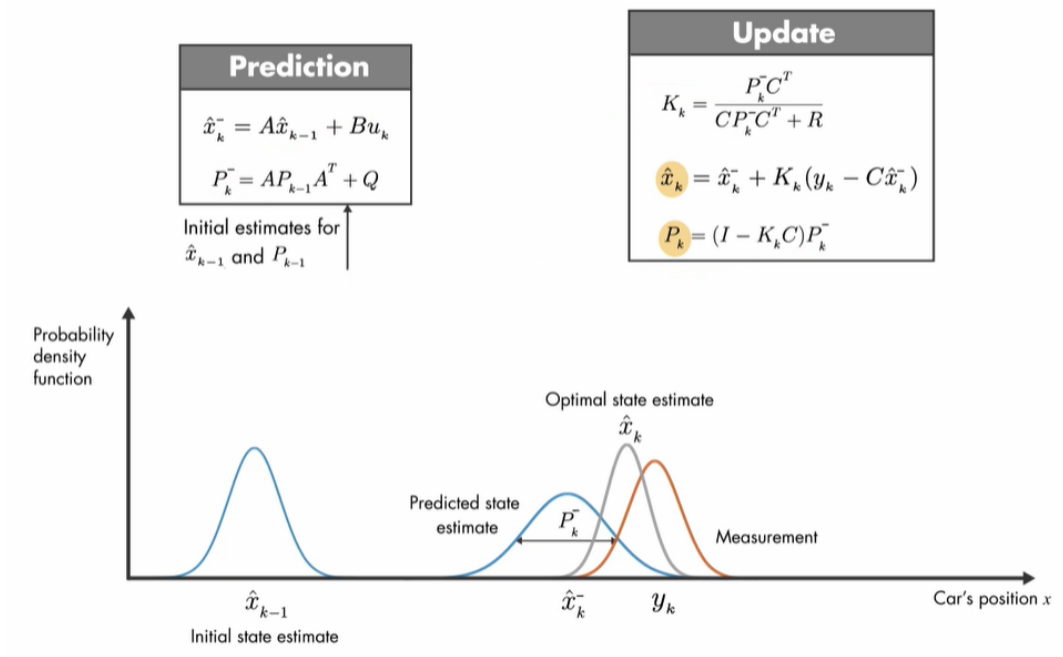
At the very start of the algorithm, the k minus 1 values for x hat and p come from their initial estimates.



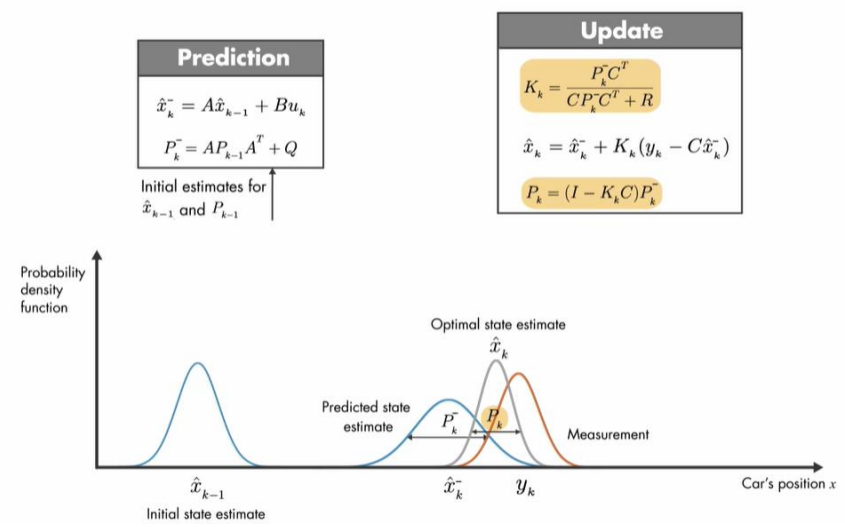
The second step of the algorithm uses the a priori estimates calculated in the prediction step and updates them to find the a posteriori estimates of the state and error covariance.



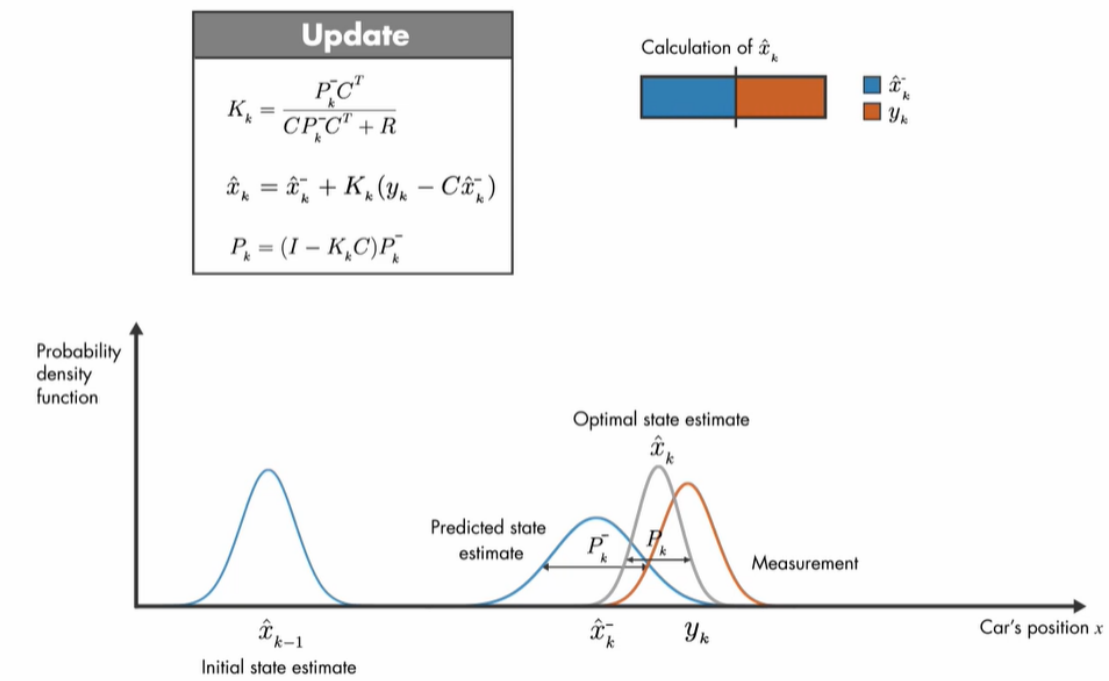
and updates them to find the a posteriori estimates of the state and error covariance



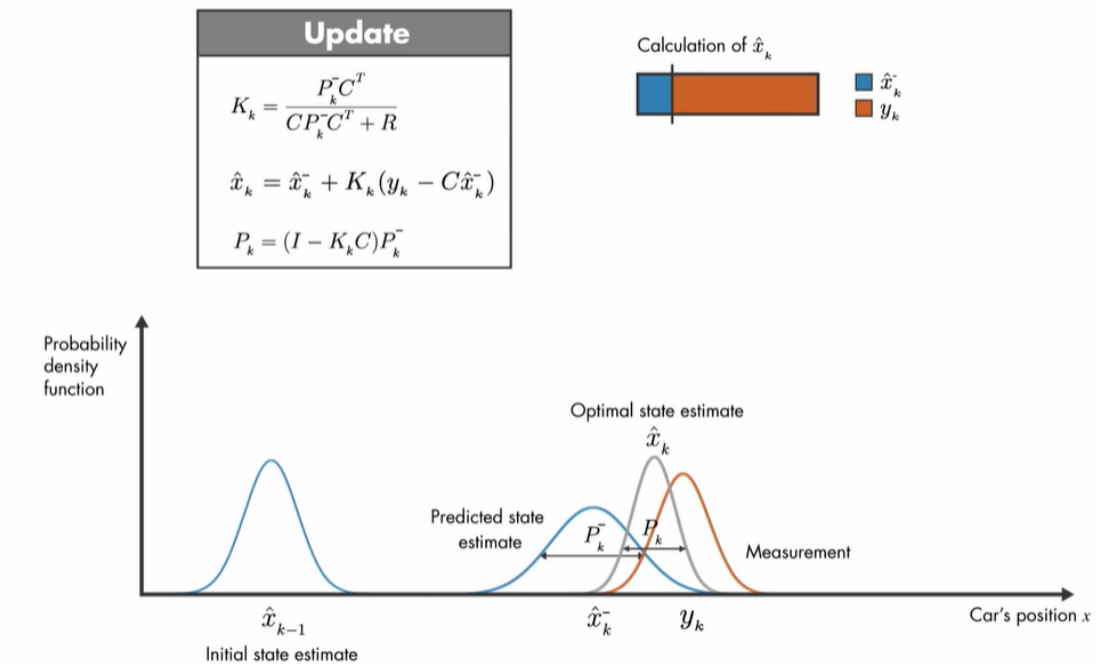
of the state and error covariance



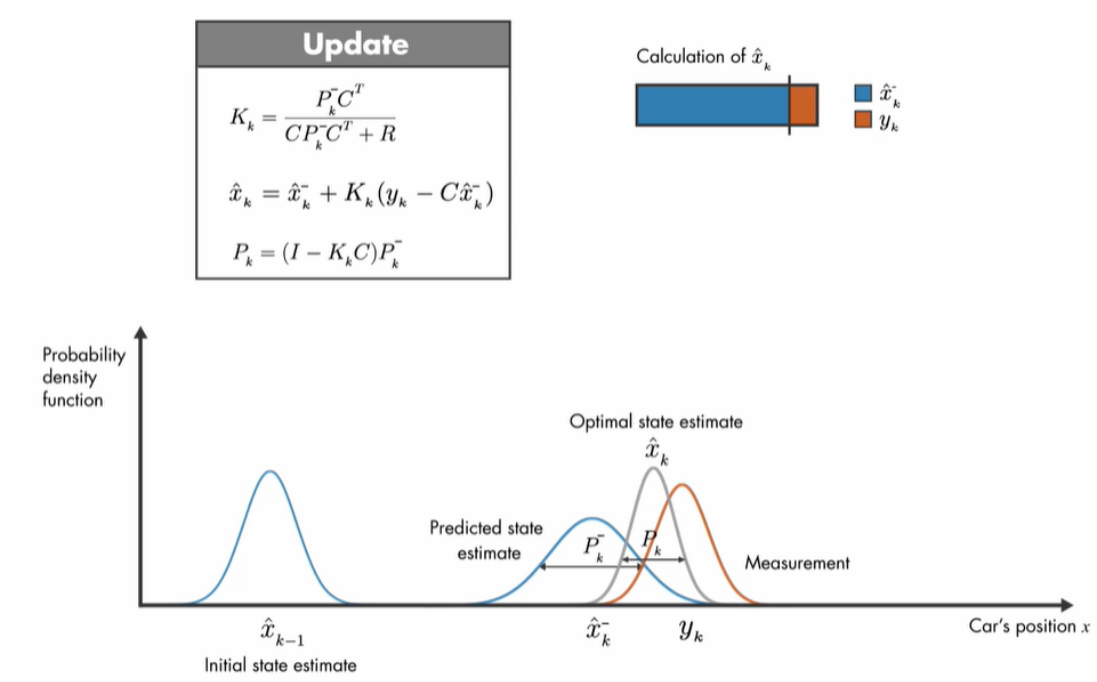
The Kalman gain is calculated such that it minimizes the a posteriori error covariance.



Let this bar represent the calculation of x hat k. By weighing the correction term, the Kalman gain determines how heavily the measurements and the a priori estimates contributes to the calculation of x hat k.

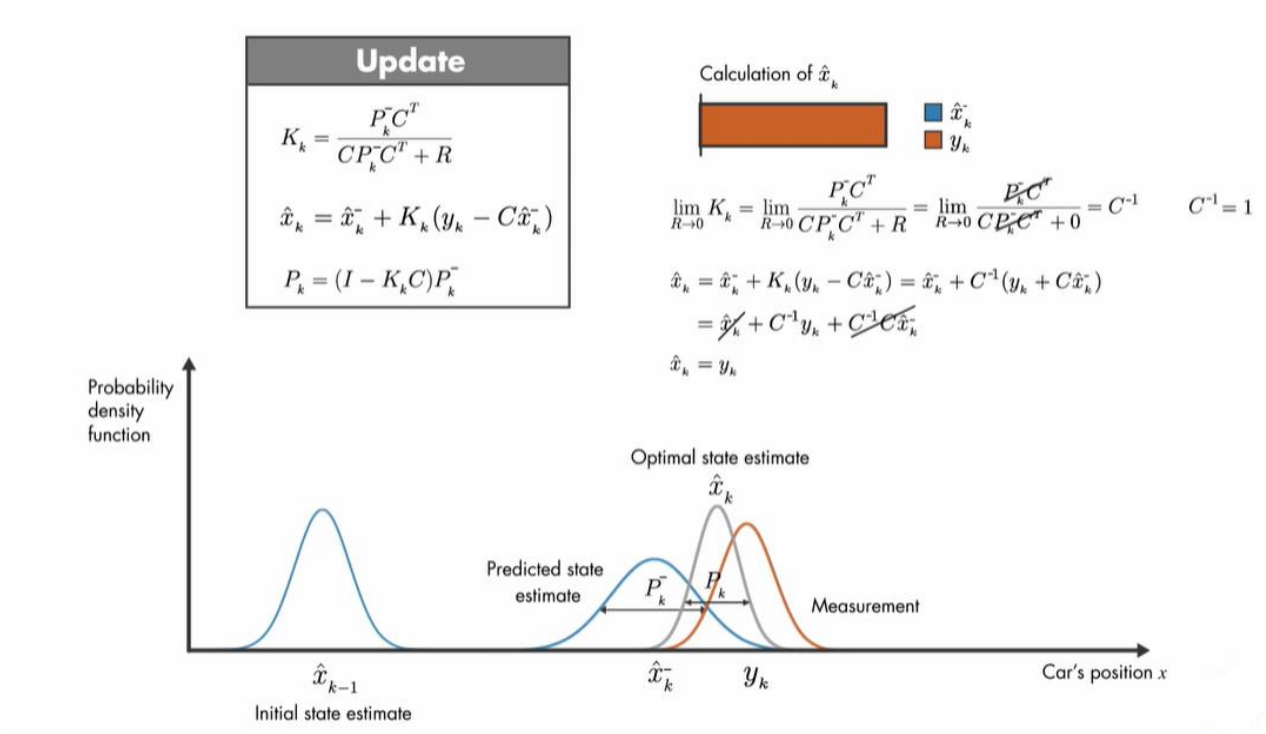


If the measurement noise is small, the measurements is trusted more and contributes to the calculation of x hat k more than the a priori state estimate does.

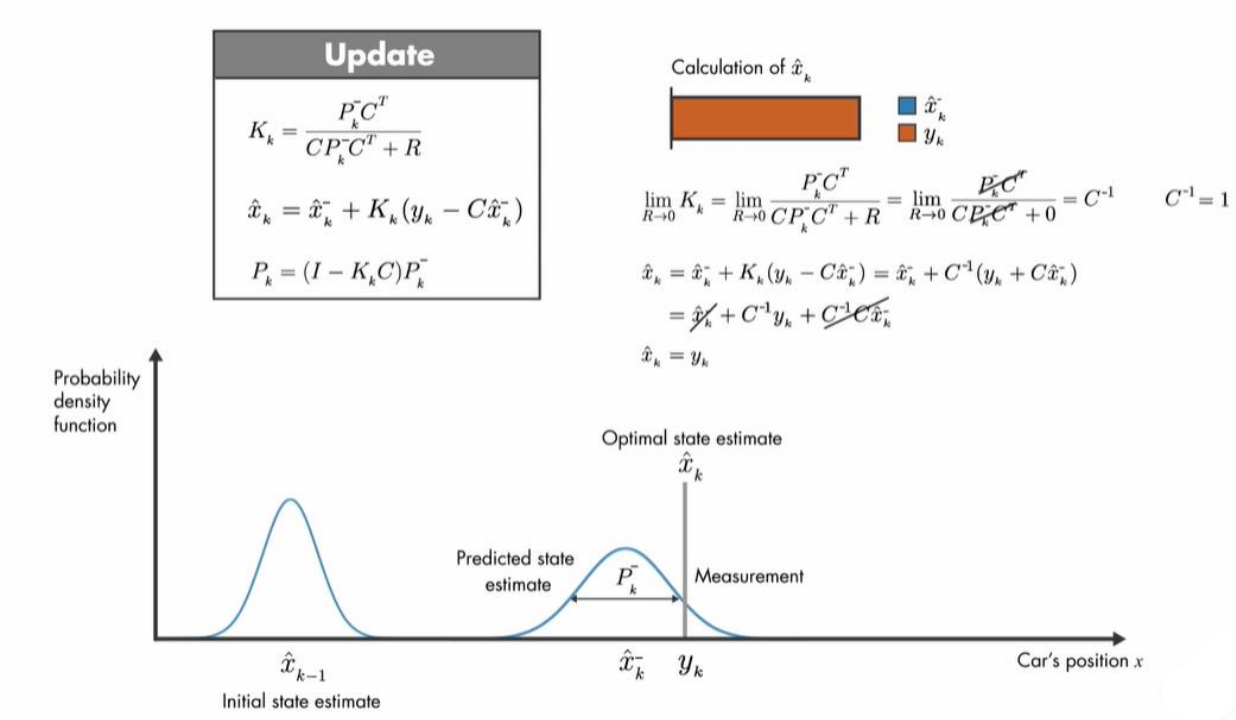


In the opposite case, where the error in the a priori estimate is small, the a priori estimate is trusted more. And the computation of x hat k mostly comes from this estimate.

.

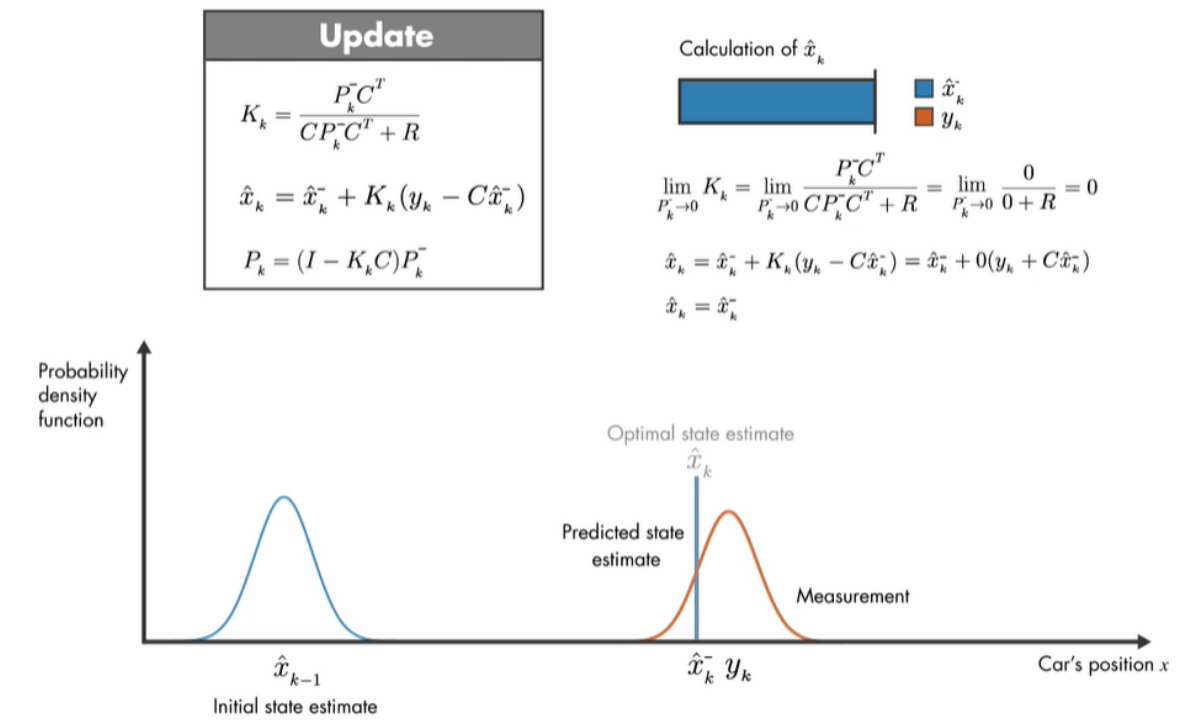


We can also show this mathematically by looking at two extreme cases. Assume that in the first case, the measurement covariance is 0. To calculate the Kalman gain, we take its limit as r goes to 0. We plug in 0 for r and see that these two terms cancel each other out.



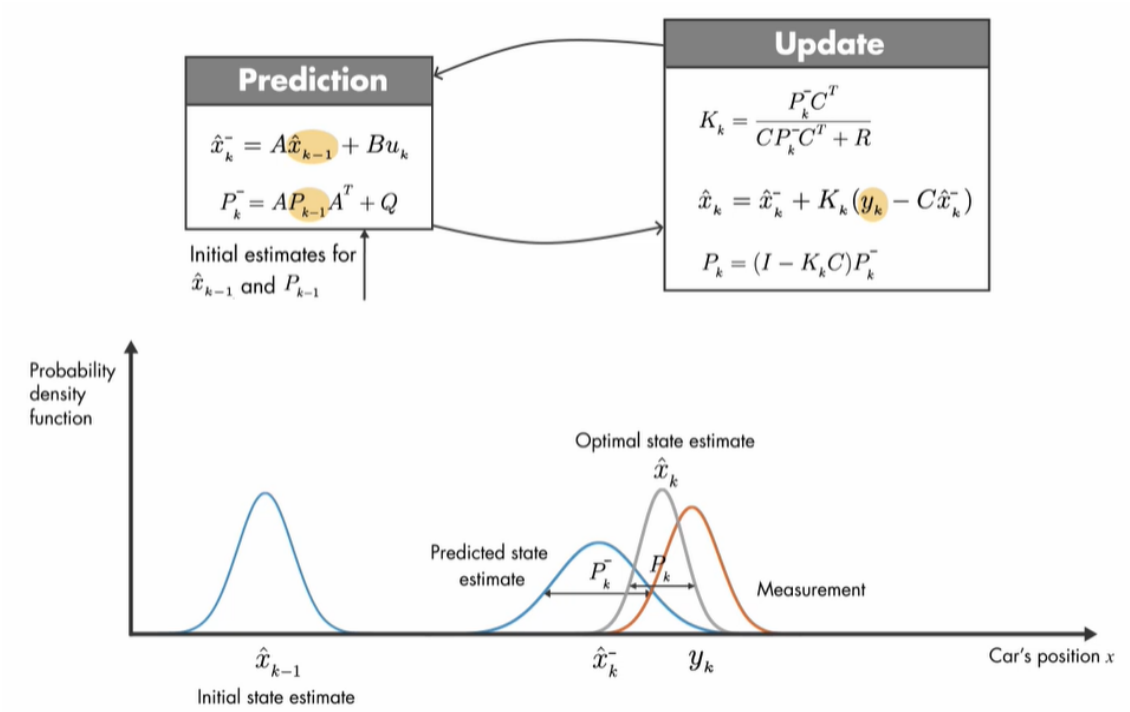
As r goes to 0, the Kalman gain approaches to the inverse of c, which is equal to 1 in our system. Plugging k, an inverse of c, into the a posteriori set estimate shows that x hat k is equal to yk. So the calculation comes from the measurement only as expected.

Now, if we update our plot, we can show the measurement with an impulse function, which is shown with this orange vertical line. Note that the variance in the measurement is 0, since r goes to 0. We found that the a posteriori estimate is equal to the measurement, so we can show it by the same impulse function.



On the other hand, if the a priori error covariance is close to 0, then the Kalman gain is form as 0. Therefore, the contribution of this term to x hat k is ignored, and the computation of x hat k comes from the a priori state estimates. On the plots, we'll show the a priori state estimate with an impulse function, which has zero variance.

And since the a posteriori estimate is equal to the a priori estimate, we'll show it with the same impulse function



Once we've calculated the updated equations, in the next timestep, the a posteriori estimates are used to predict the a priori estimates, and the algorithm repeats itself.

Notice that to estimate the current state, the algorithm doesn't need all the past information. It only needs the estimated state and error covariance matrix from the previous timestep and the current measurement. This is what makes the Kalman filter recursive.

http://www.ilectureonline.com/lectures/subject/SPECIAL%20TOPICS/26/190/1966