Kalman filter

What is it?

A Kalman filter can be applied whenever there's uncertain information about a dynamic system and a reasonable prediction about its next state. Even when unexpected events disrupt the predicted motion, the Kalman filter often accurately deduces what actually occurred. Additionally, it can utilize correlations between unpredictable factors that might otherwise be overlooked.

Kalman filters are especially effective for continuously changing systems. They are memory-efficient, as they only require the previous state rather than an extensive history, and are very fast, making them ideal for real-time applications and embedded systems.

What can we do with a Kalman filter?

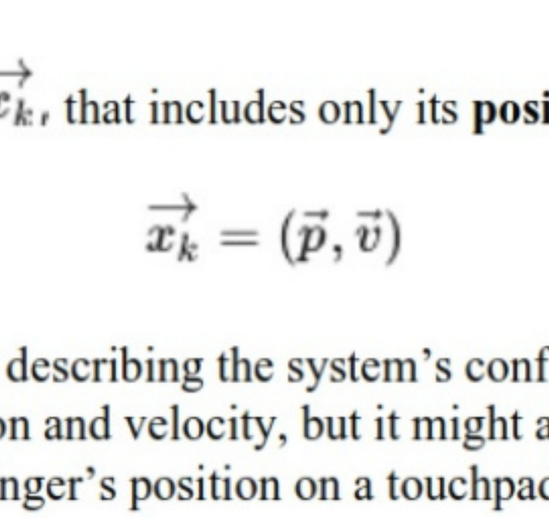
Here's a simple example: imagine you've built a small robot that roams around in the forest and needs to determine its exact location to navigate effectively.

Let's assume our robot has a **state** $\vec{x_k}$, that includes only its **position** and **velocity**.

$$\vec{x_k} = (\vec{p}, \vec{v})$$

The **state** is simply a set of numbers describing the system's configuration—it could represent anything. In our example, it's position and velocity, but it might also reflect the amount of fluid in a tank, a car engine's temperature, a finger's position on a touchpad, or any other data you need to monitor.

Our robot is equipped with a GPS sensor that's accurate to about 10 meters—helpful, but not precise enough. The woods are full of gullies and cliffs, so if the robot's location is off by more than a few feet, it risks falling. Thus, GPS alone isn't sufficient.



We may also have some insight into the robot's movement: it knows the commands sent to its wheel motors and understands that if it's moving in a given direction, it will likely continue along that path in the next moment—provided there are no disturbances. However, it lacks complete knowledge of its motion. Wind may push it, the wheels might slip, or uneven ground may affect its progress, meaning the wheel rotation might not fully represent the actual distance traveled, so the prediction won't be flawless.

The GPS sensor provides information about the state, though indirectly and with some inaccuracy. Similarly, our prediction offers insight into the robot's motion, but also indirectly and with potential errors.

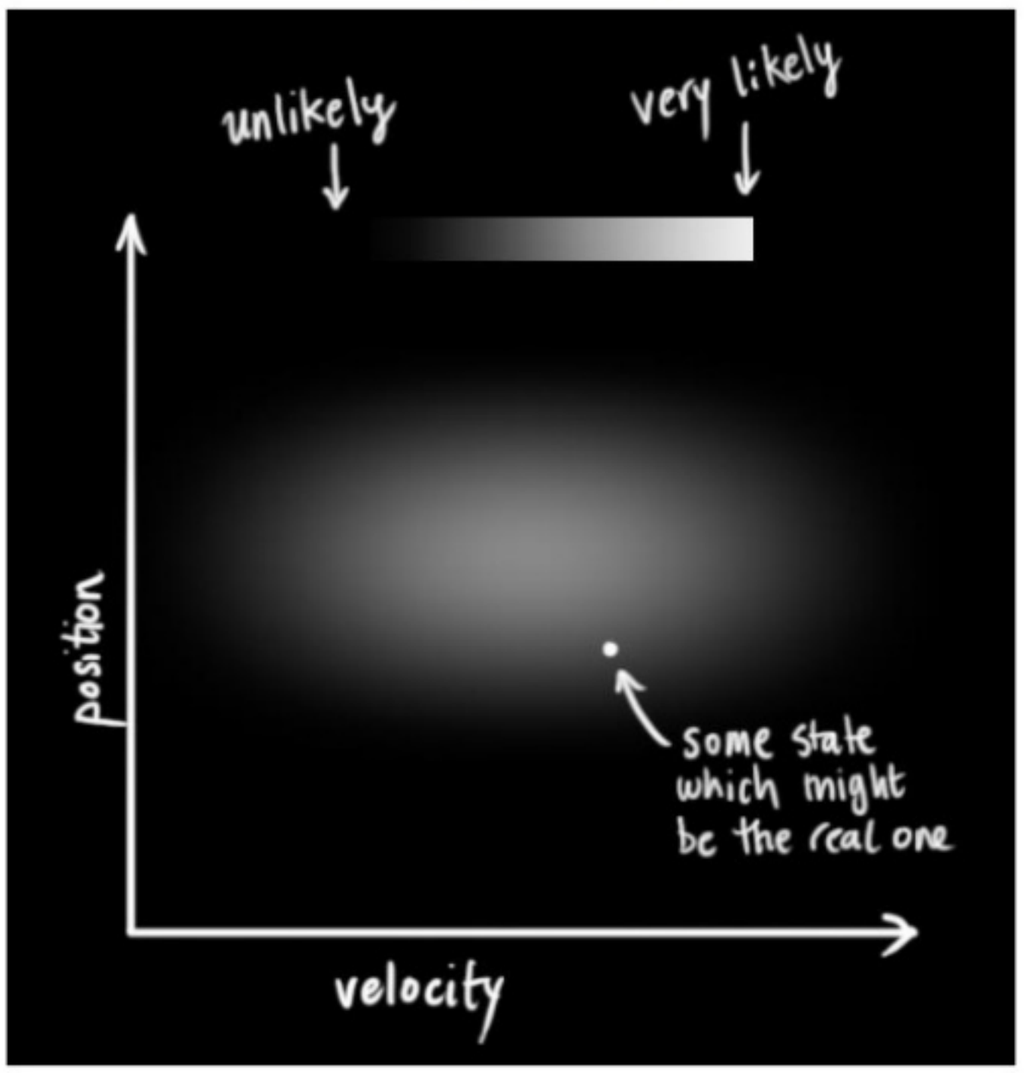
If we combine all the available information, can we obtain a more accurate estimate than either source alone? Yes, we can—and that's precisely the purpose of a Kalman filter.

How a Kalman filter sees your problem

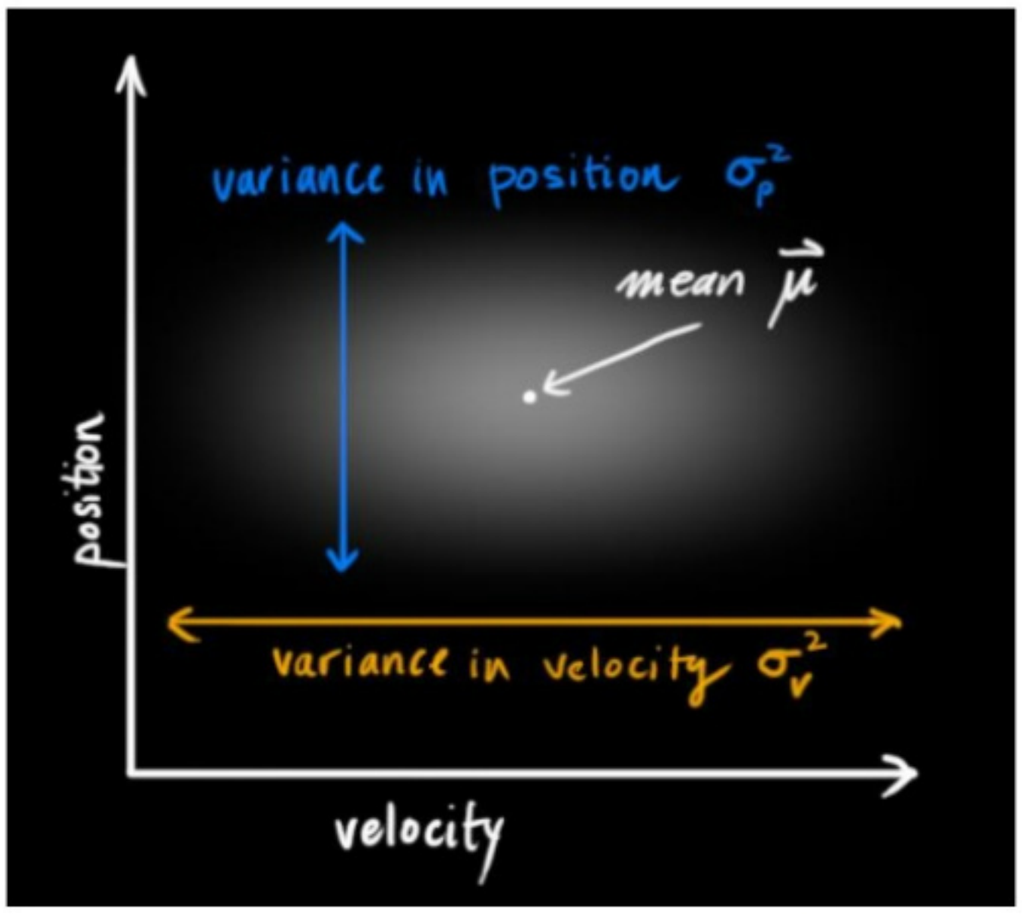
Let's examine the framework we're trying to analyze, sticking with a straightforward state that includes only position and velocity.

$$\vec{x} = \begin{bmatrix} p \\ v \end{bmatrix}$$

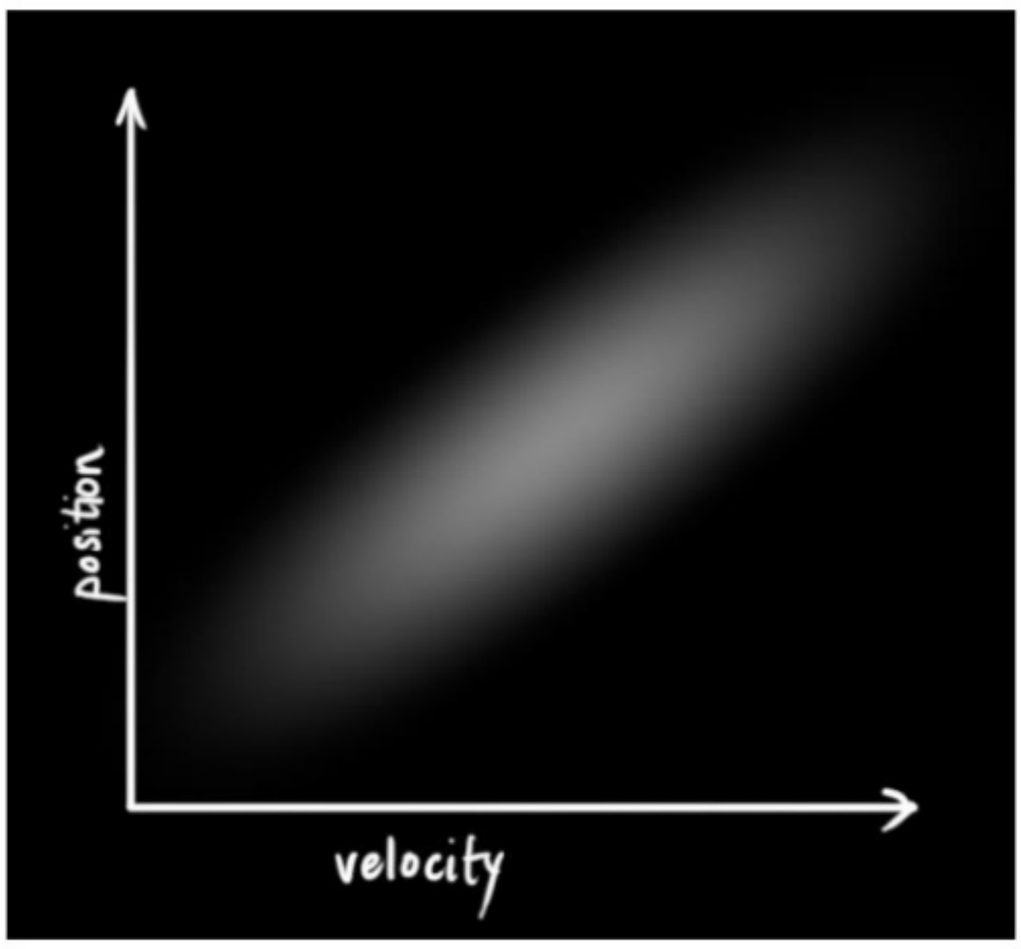
We don't know the exact position and velocity; instead, there's a range of possible position-velocity combinations that could be true, with some being more likely than others.



The Kalman filter operates under the assumption that both variables—position and velocity in our example—are random and follow a Gaussian distribution. Each variable has a mean value, μ, representing the central or most probable state of the distribution, and a variance, σ2, which quantifies the level of uncertainty:



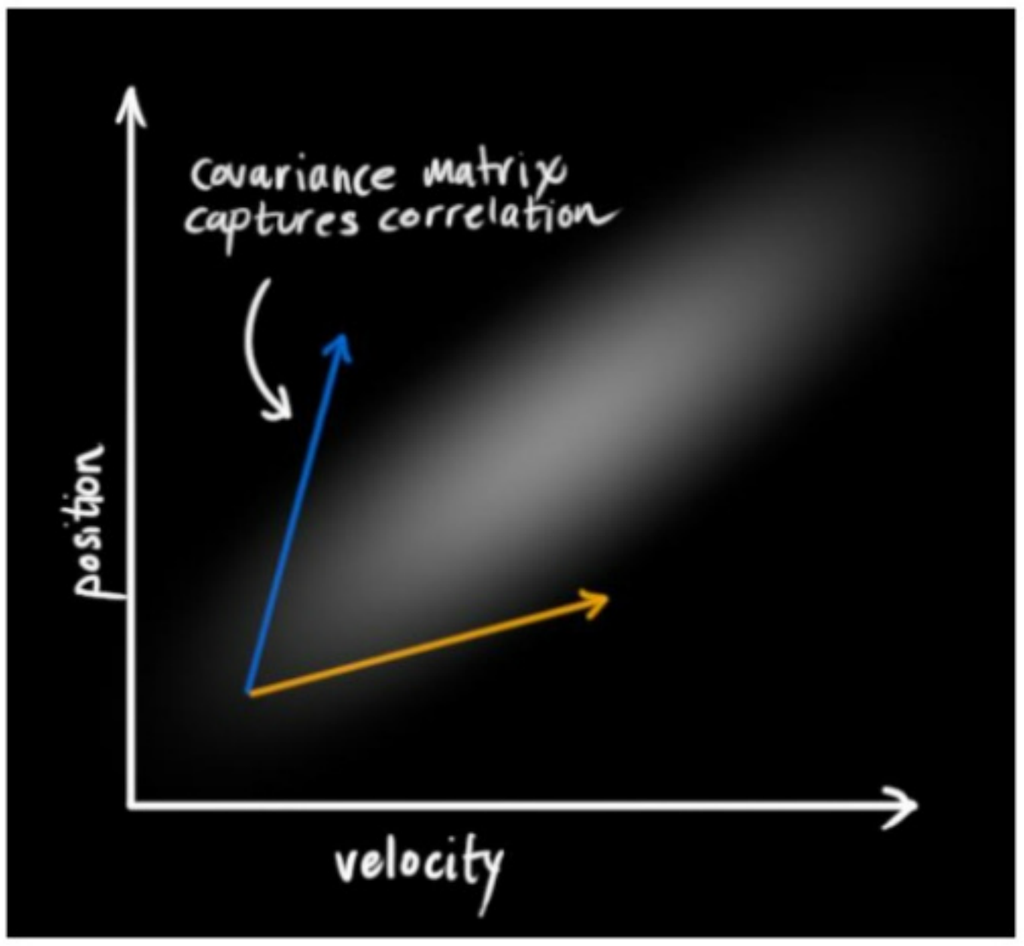
In the image above, position and velocity are **uncorrelated**, meaning that knowing the state of one variable provides no information about the other. However, in the example below, position and velocity are **correlated**, so the probability of observing a certain position depends on the velocity.



This scenario could occur if we are estimating a new position based on a previous one. For instance, if the velocity was high, the position would likely be farther along, whereas with a slower velocity, the position wouldn't have changed as much.

Tracking this type of relationship is crucial because it provides additional insights: one measurement offers clues about others. The purpose of the Kalman filter is to extract as much information as possible from these uncertain measurements.

This correlation is represented by a **covariance matrix**. Each element in the matrix, denoted Σij, indicates the level of correlation between the i-th and j-th state variables. The covariance matrix is **symmetric**, meaning the order of iii and jjj does not affect the result. These matrices are typically labeled as Σ, with each component referred to as Σij.



Describing the problem with matrices

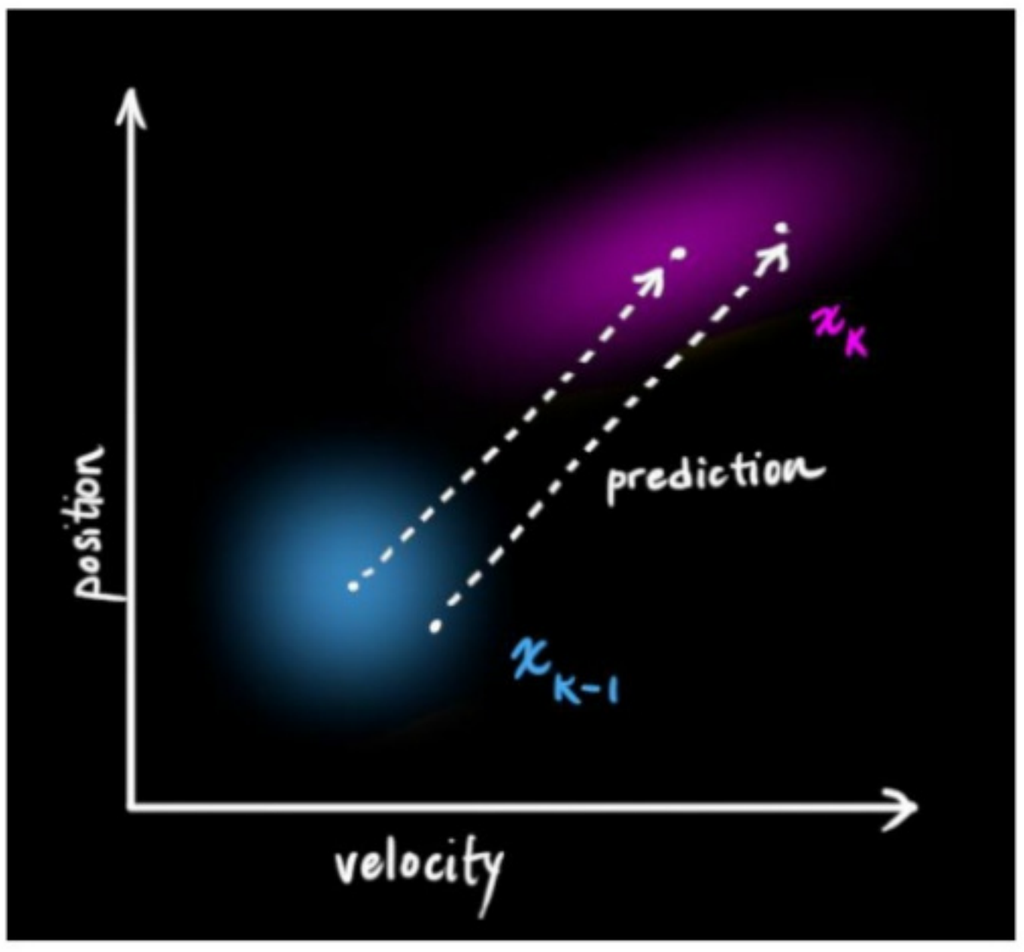
Our understanding of the state $\hat{\mathbf{x}}_k$ is modeled as a Gaussian distribution, or "Gaussian blob." At any given time k, we require two key pieces of information: our best estimate (the mean, also referred to as μ), and the covariance matrix, P_k.

$$\hat{\mathbf{x}}_k = \begin{bmatrix} \text{position} \\ \text{velocity} \end{bmatrix}$$

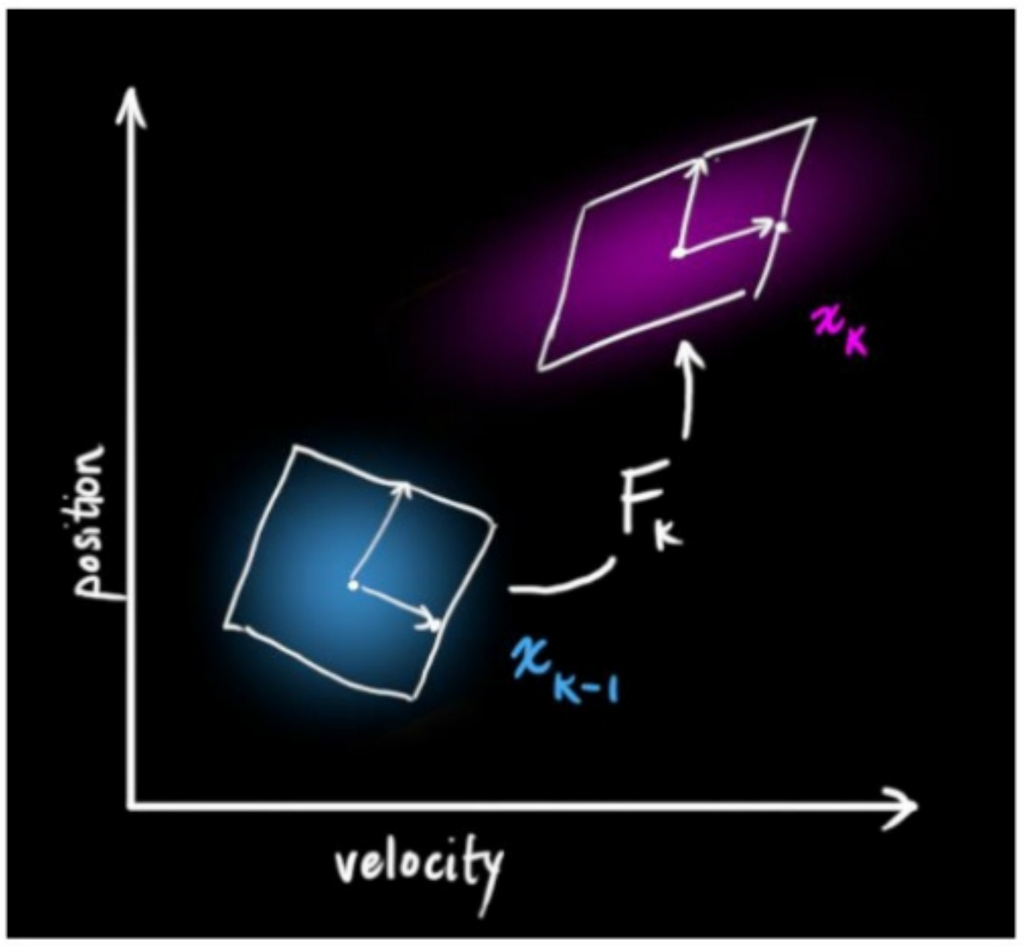
$$\mathbf{P}_k = \begin{bmatrix} \Sigma_{pp} & \Sigma_{pv} \\ \Sigma_{vp} & \Sigma_{vv} \end{bmatrix}$$

(Of course, we're focusing on position and velocity here, but it's important to note that the state can include any number of variables and represent any desired data).

Next, we need a method to examine the **current state** (at time k−1) and **predict** the **next state** at time **k**. Keep in mind, we don't know which state represents the "true" state, but our prediction function doesn't rely on this; it processes all possible states and produces a new distribution.



This prediction step can be represented using a matrix, F_k:



The matrix F_k shifts each point in our initial estimate to a new predicted position, representing where the system would go if the original estimate were accurate.

Let's put this into practice. How would we use a matrix to predict position and velocity for the next moment in time? We'll apply a simple kinematic formula:

$$\begin{aligned} p_k &= p_{k-1} + \Delta t v_{k-1} \\ v_k &= v_{k-1} \end{aligned}$$

To put it differently:

$$\hat{\mathbf{x}}_k = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \hat{\mathbf{x}}_{k-1} \tag{2}$$

$$= \mathbf{F}_k \hat{\mathbf{x}}_{k-1} \tag{3}$$

We now have a **prediction matrix** that provides the next state, but we still need a way to update the covariance matrix. This requires another formula. When every point in a distribution is multiplied by a matrix A, how does that affect its covariance matrix Σ?

The answer is straightforward, and here's the identity:

$$\begin{aligned} Cov(x) &= \Sigma \\ Cov(\mathbf{A}x) &= \mathbf{A}\Sigma\mathbf{A}^T \end{aligned} \tag{4}$$

Thus, by combining equation (4) with equation (3), we get:

$$\begin{aligned} \hat{\mathbf{x}}_k &= \mathbf{F}_k \hat{\mathbf{x}}_{k-1} \\ \mathbf{P}_k &= \mathbf{F}_\mathbf{k} \mathbf{P}_{k-1} \mathbf{F}_k^T \end{aligned} \tag{5}$$

External influence

We haven't accounted for everything yet. **External factors,** unrelated to the state itself, could be influencing the system.

For instance, if the state represents a train's movement, the operator could accelerate by adjusting the throttle. Similarly, in our robot example, the navigation system might issue commands to turn the wheels or stop. If we have additional information about such external influences, we can include it in a vector, $\vec{u}_k$, use it accordingly, and incorporate it into our prediction as a correction.

Suppose we know the expected acceleration ***a*** based on throttle settings or control commands. Using basic kinematic equations, we get:

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

In matrix form:

$$\begin{aligned} \hat{\mathbf{x}}_k &= \mathbf{F}_k \hat{\mathbf{x}}_{k-1} + \begin{bmatrix} \frac{\Delta t^2}{2} \\ \Delta t \end{bmatrix} a \\ &= \mathbf{F}_k \hat{\mathbf{x}}_{k-1} + \mathbf{B}_k \vec{\mathrm{u}_k} \end{aligned} \tag{6}$$

Here, B_k is referred to as the **control matrix** and $\hat{u}_k$ as the **control vector**. (For simpler systems with no external influence, these can be omitted).

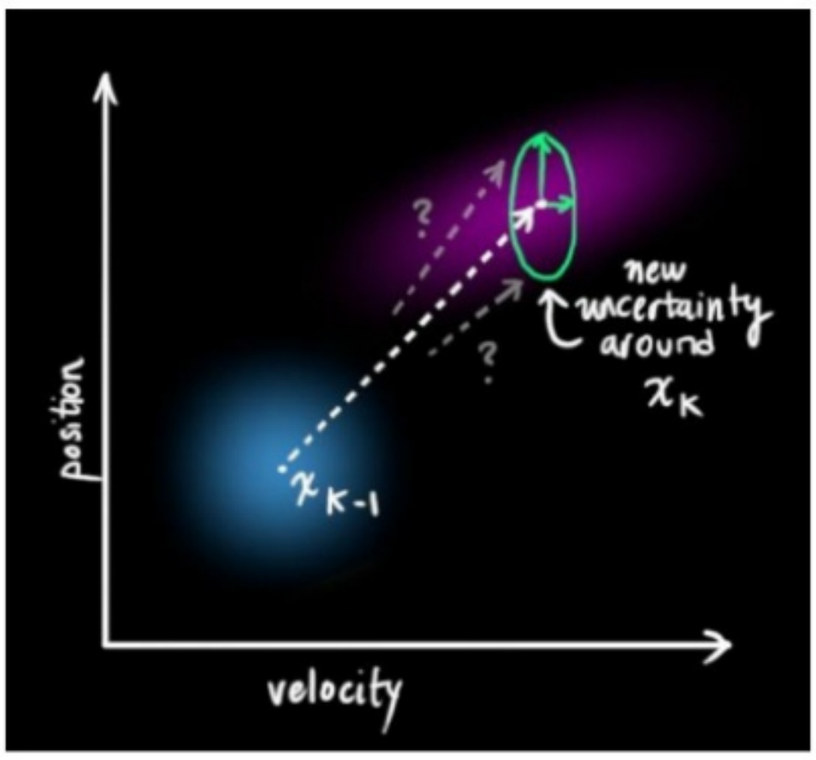
Now, let's consider one more detail: what if our prediction isn't a completely accurate reflection of the actual situation?

External uncertainty

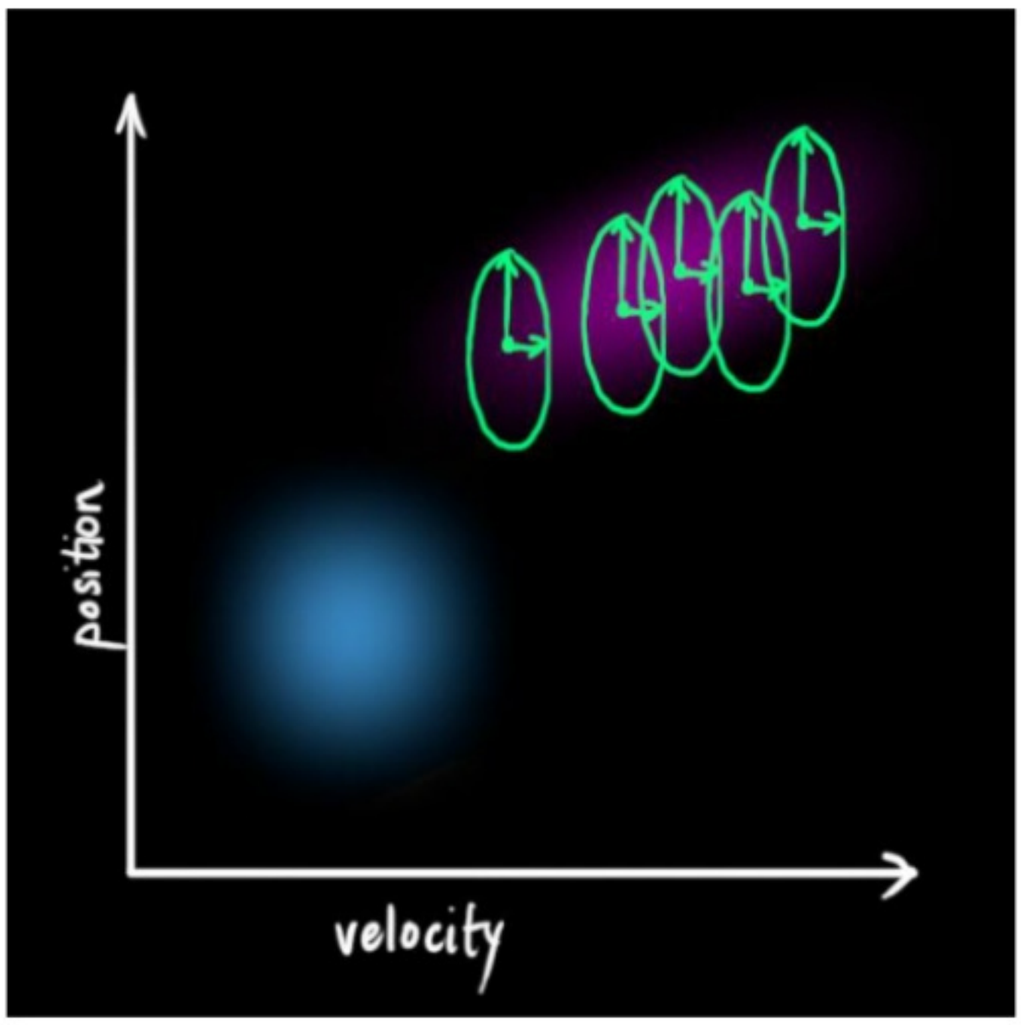
If the state evolves purely based on its internal properties, everything is straightforward. It's still manageable if the state changes due to known external forces.

But what about unknown forces? For instance, a quadcopter might be affected by gusts of wind, or a wheeled robot could encounter wheel slip or rough terrain. These untracked forces can cause deviations in our prediction, as they weren't factored in.

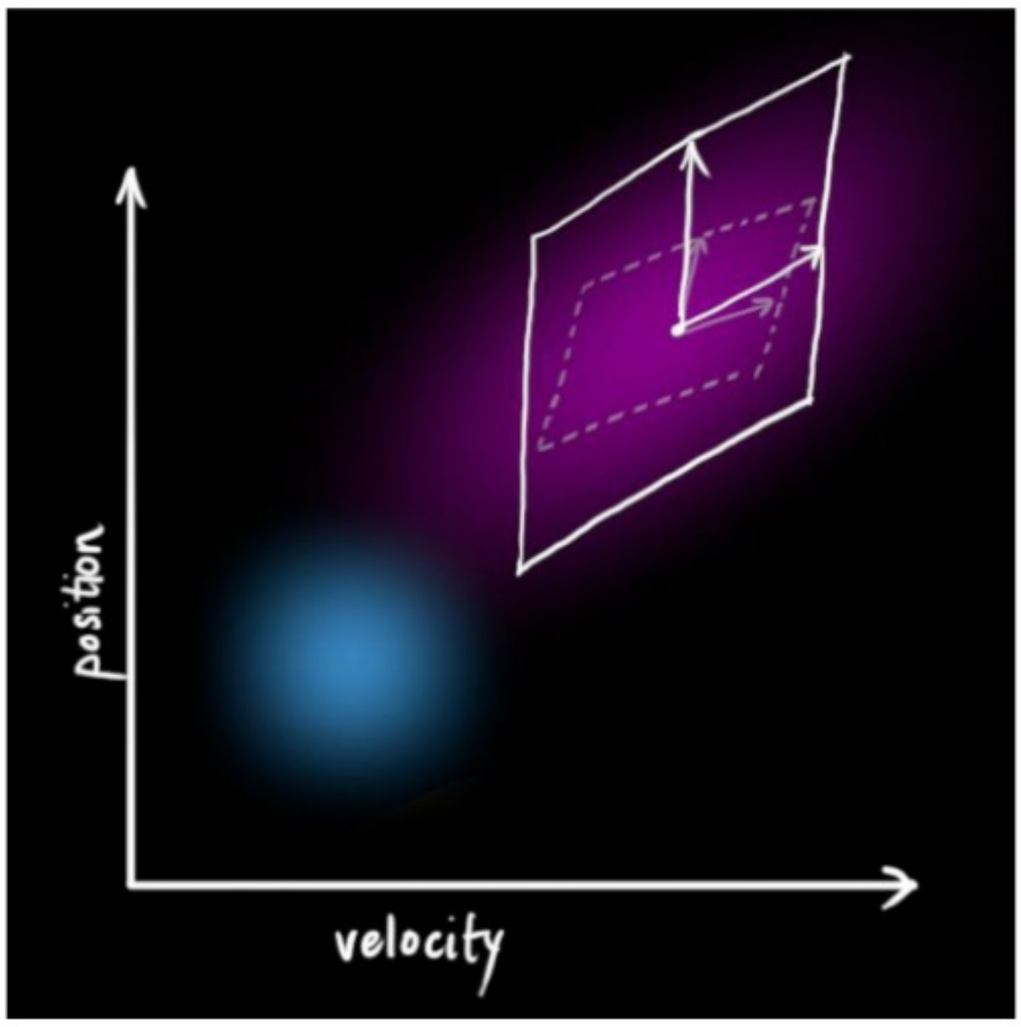
To account for this "world" uncertainty—factors we aren't actively monitoring—we can add a bit of extra uncertainty after each prediction step.



Each state in our original estimate could shift to a range of possible states. Since we often use Gaussian distributions, we'll assume each point in $\hat{x}_{k-1}$ moves within a Gaussian distribution with covariance Q_k. In other words, we treat these untracked influences as noise, characterized by the covariance Q_k .



This results in a new Gaussian distribution, now with an updated covariance but retaining the same mean.



We obtain the expanded covariance by adding Q_k, which gives us the full expression for the **prediction step**:

$$\begin{aligned}\hat{\mathbf{x}}_k &= \mathbf{F}_k\hat{\mathbf{x}}_{k-1} + \mathbf{B}_k\overrightarrow{\mathbf{u}_k} \\ \mathbf{P}_k &= \mathbf{F}_k\mathbf{P}_{k-1}\mathbf{F}_k^T + \mathbf{Q}_k\end{aligned} \tag{7}$$

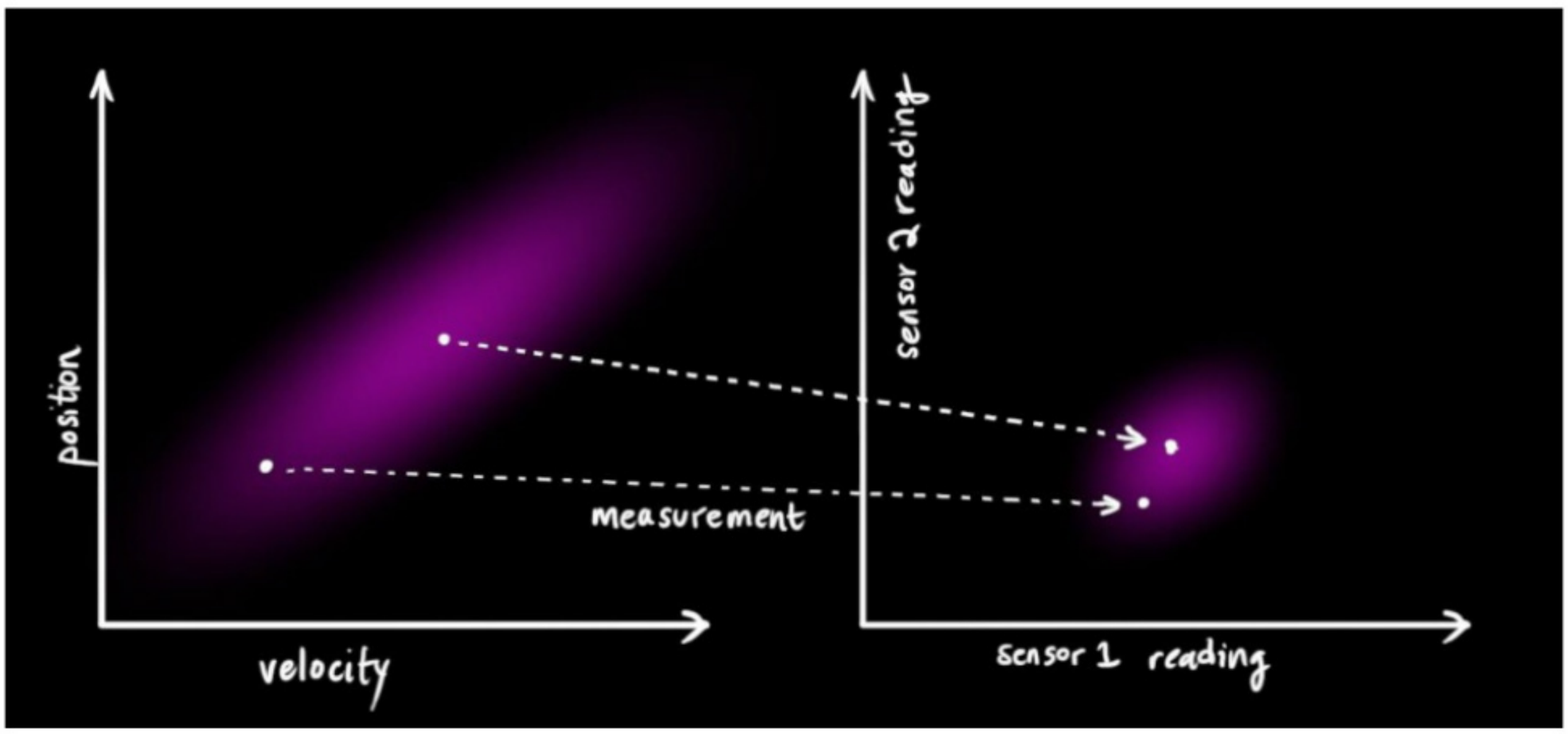
In other words, the new best estimate is based on the previous estimate with a **correction** for known external factors.

The updated uncertainty combines the prior uncertainty with extra environmental uncertainty.

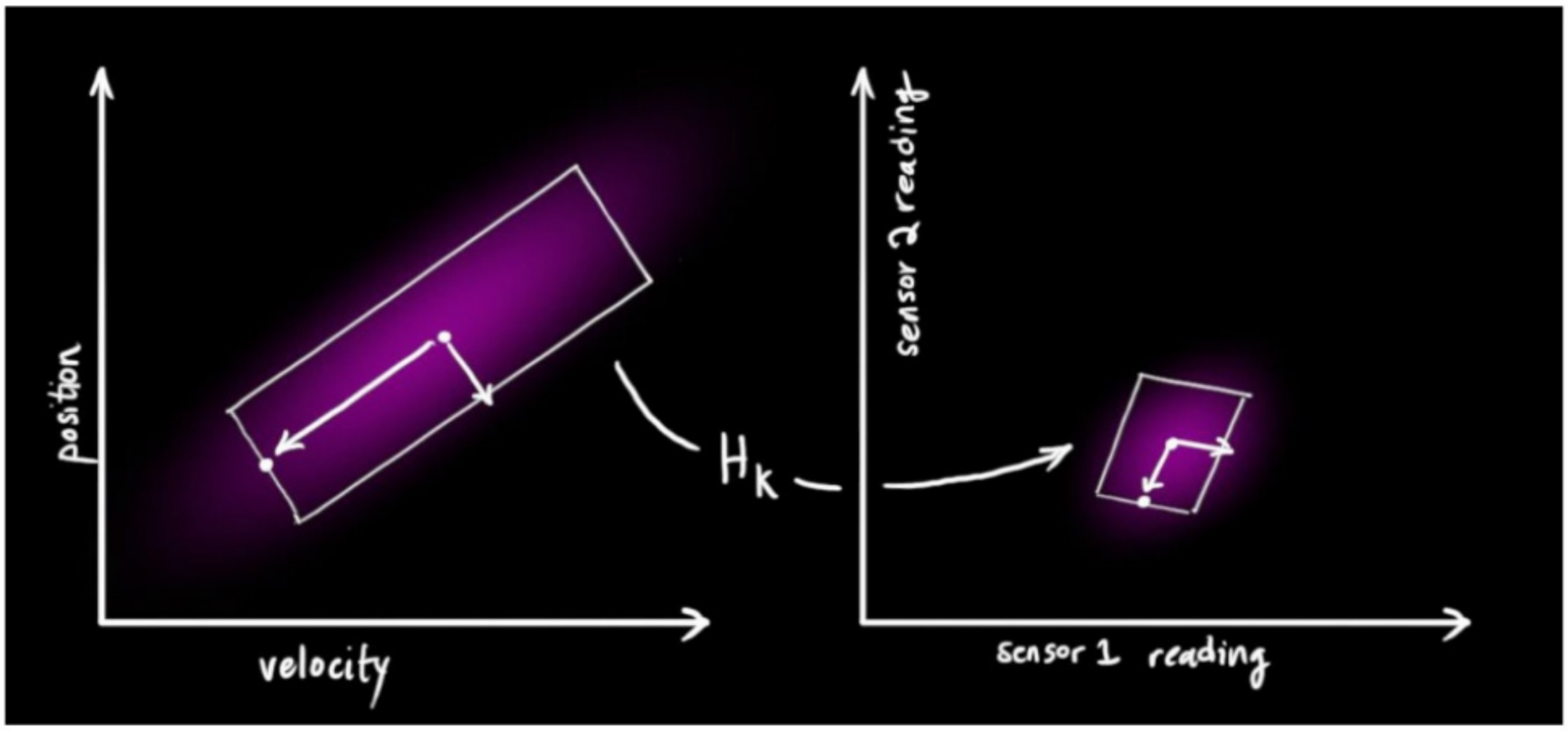
With this, we have a rough estimate of the system's state, represented by $\hat{x}_k$ and P_k Now, what happens when we receive sensor data?

Refining the estimate with measurements

We may have multiple sensors providing data about the system's state. For now, it doesn't matter exactly what they measure—one might detect position, while another measures velocity. Each sensor gives indirect information about the state, essentially processing it to produce a set of readings.



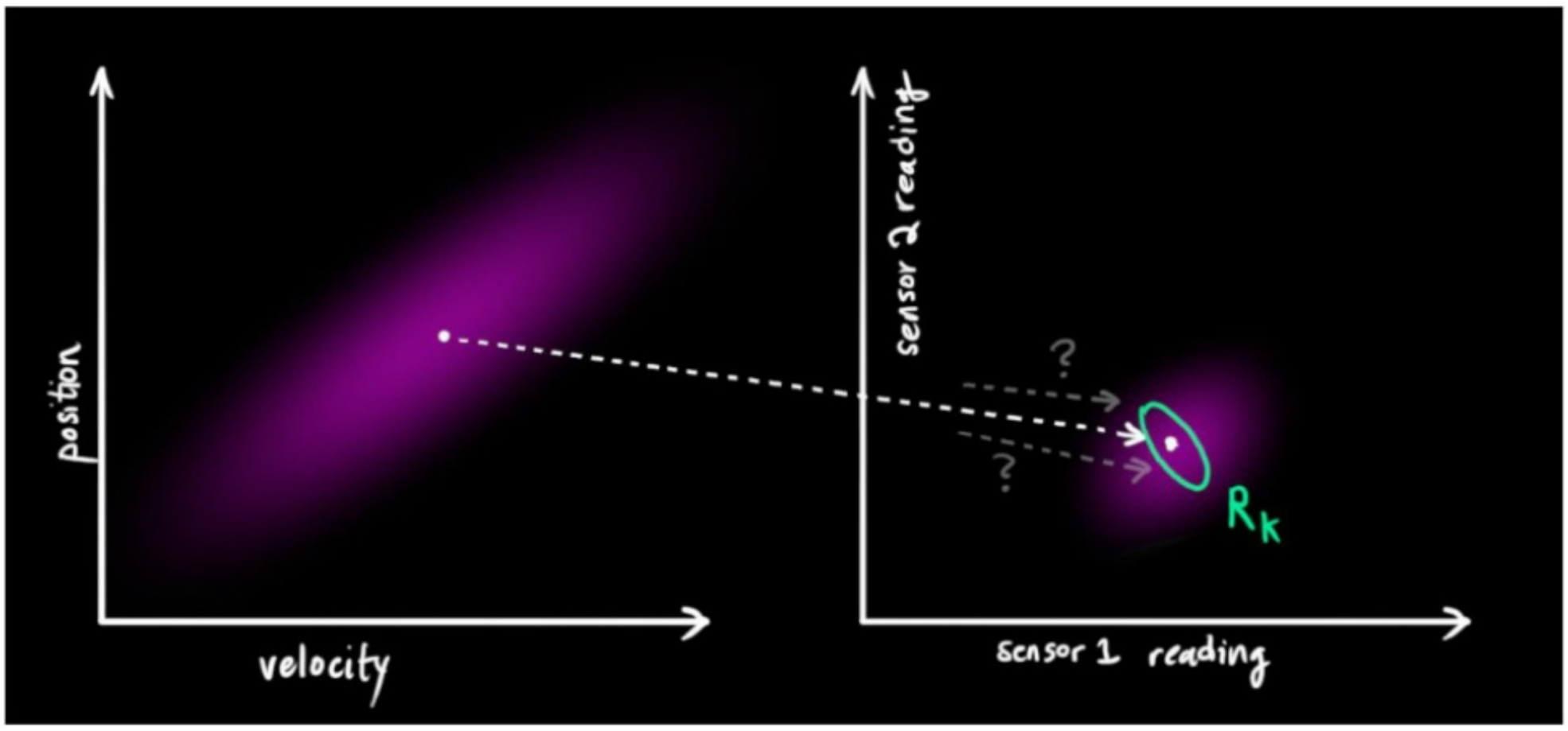
It's important to note that the units and scale of the sensor readings may differ from those of the state we're tracking. This leads us to model the sensors using a matrix, H_k.



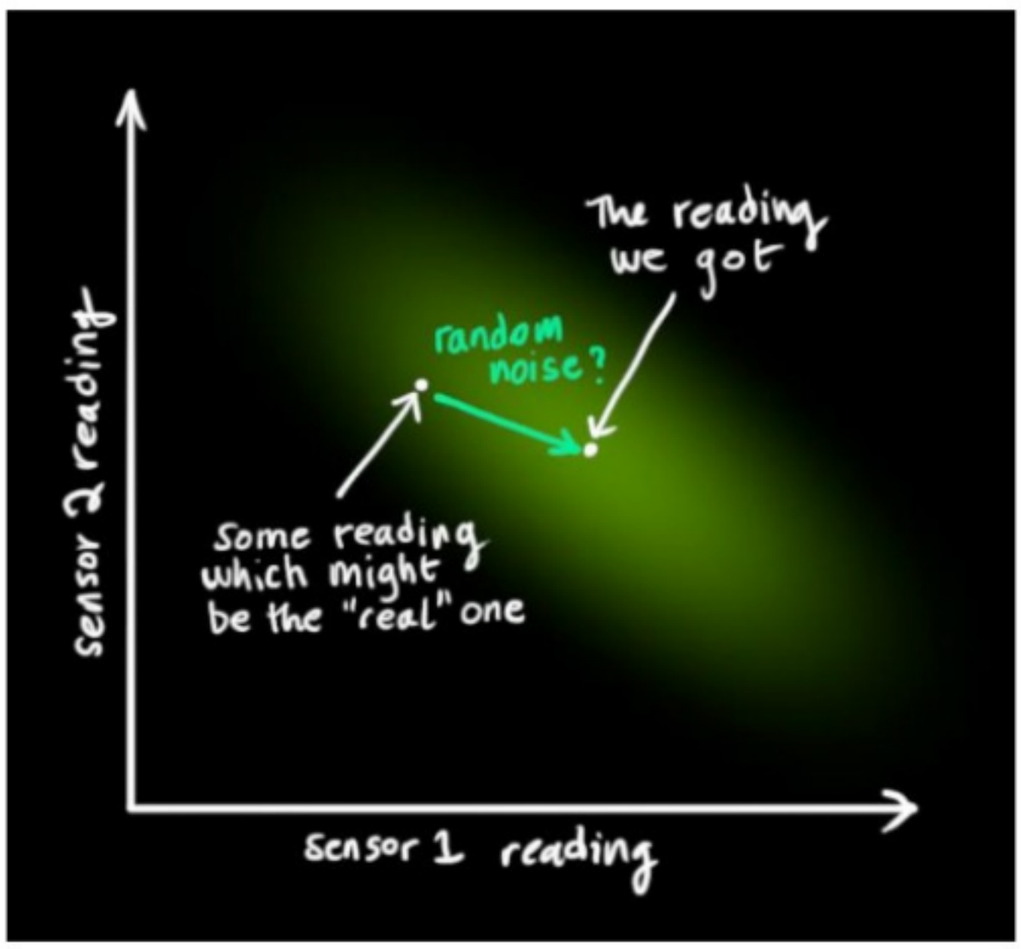
We can determine the expected distribution of sensor readings using the standard approach:

$$\vec{\mu}_{\text{expected}} = \mathbf{H}_k \hat{\mathbf{x}}_k$$
$$\mathbf{\Sigma}_{\text{expected}} = \mathbf{H}_k \mathbf{P}_k \mathbf{H}_k^T \quad (8)$$

A key strength of Kalman filters is their ability to handle sensor noise. In other words, since our sensors are somewhat unreliable, each state in our original estimate could correspond to a range of possible sensor readings.

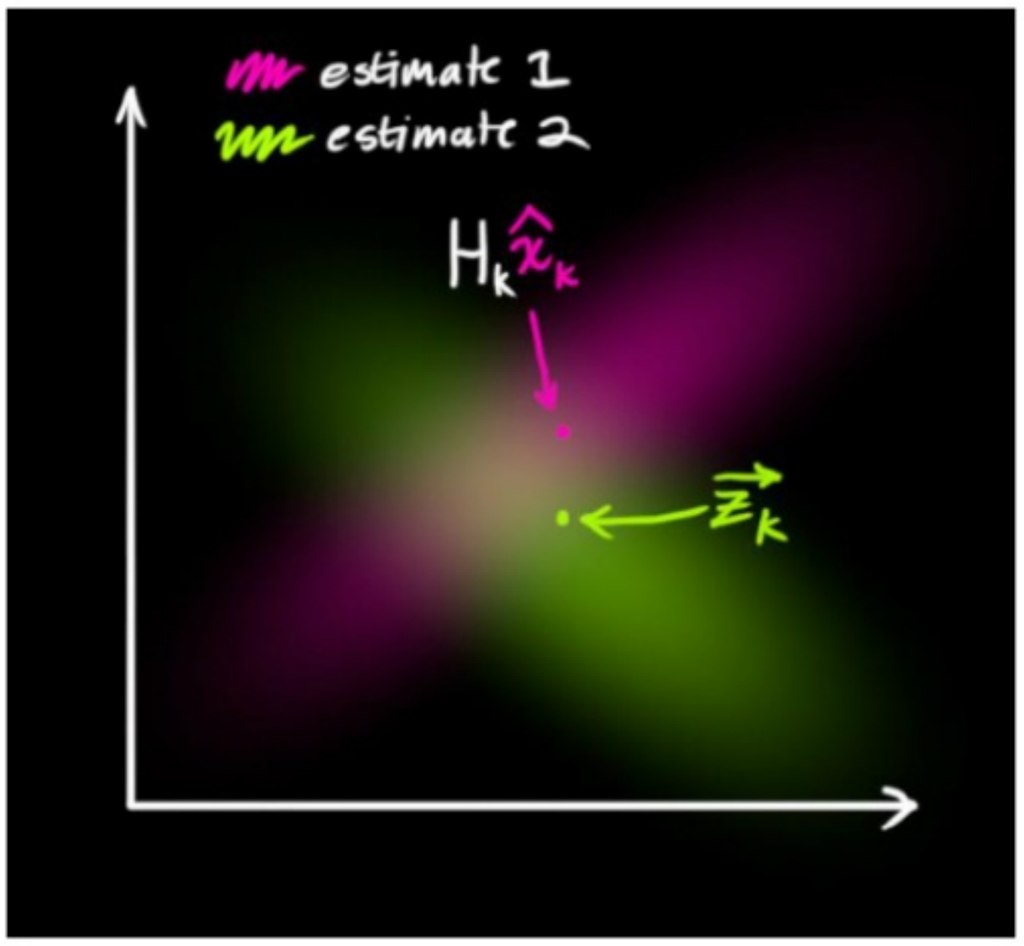


Based on each observed reading, we can infer a possible state of the system. However, due to uncertainty, certain states are more likely than others to have generated the observed reading.



We refer to the **covariance** of this uncertainty, or sensor noise, as R_k . This distribution has a mean that matches the observed reading, labeled $\hat{z}_k$.

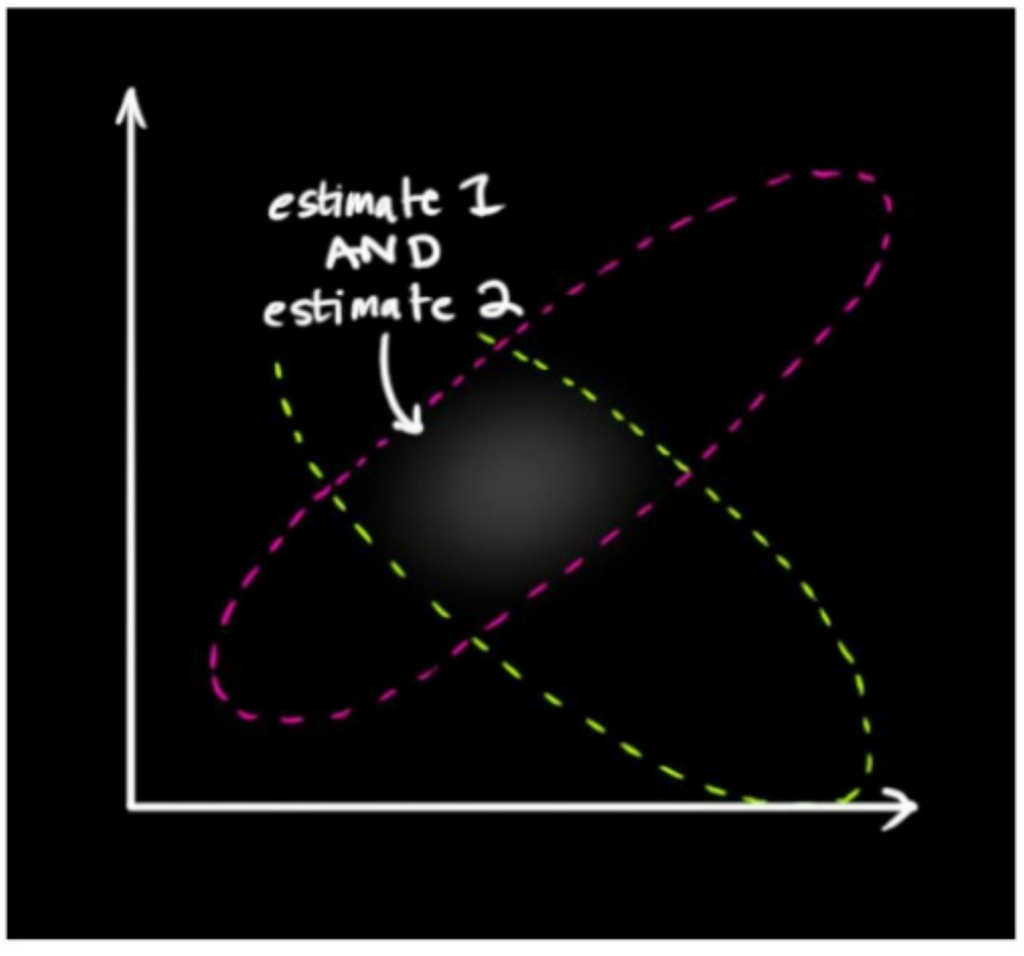
Now, we have two Gaussian distributions: one centered around the mean of our adjusted prediction and another around the actual sensor reading we received.



We need to reconcile our expected readings, based on the predicted state (shown in pink), with our actual sensor readings (shown in green).

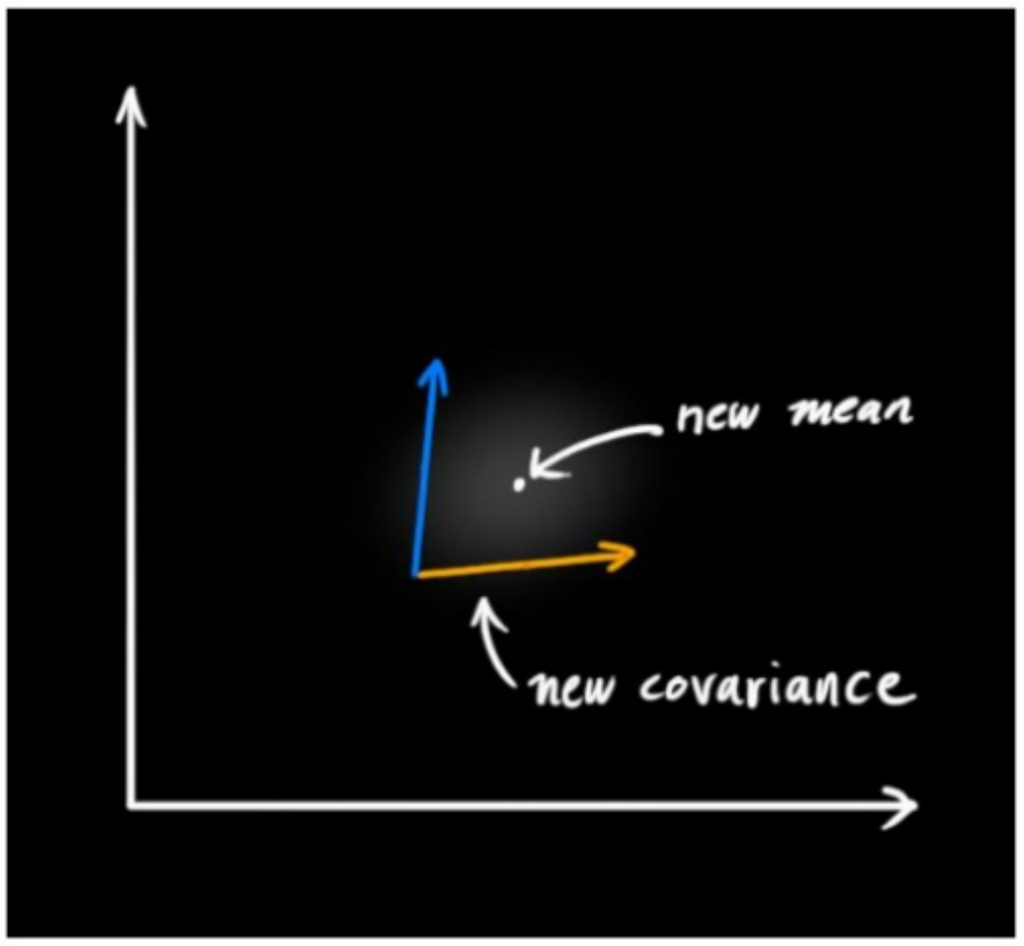
So, what is the new most likely state? For each potential reading (z_1,z_2), there are two associated probabilities: (1) the chance that our sensor reading zk is a (mis)measurement of (z_1,z_2), and (2) the probability that our previous estimate suggests (z_1,z_2) should be the expected reading.

To find the likelihood of both being true, we simply multiply these probabilities. Thus, we combine the two Gaussian distributions by multiplying them.



The result is the overlapping region where both distributions are most probable, providing a much more precise estimate than either one alone. The mean of this overlap represents the configuration with the highest likelihood based on both estimates, making it our best guess for the true state given all available information.

Interestingly, this result is also a Gaussian distribution.



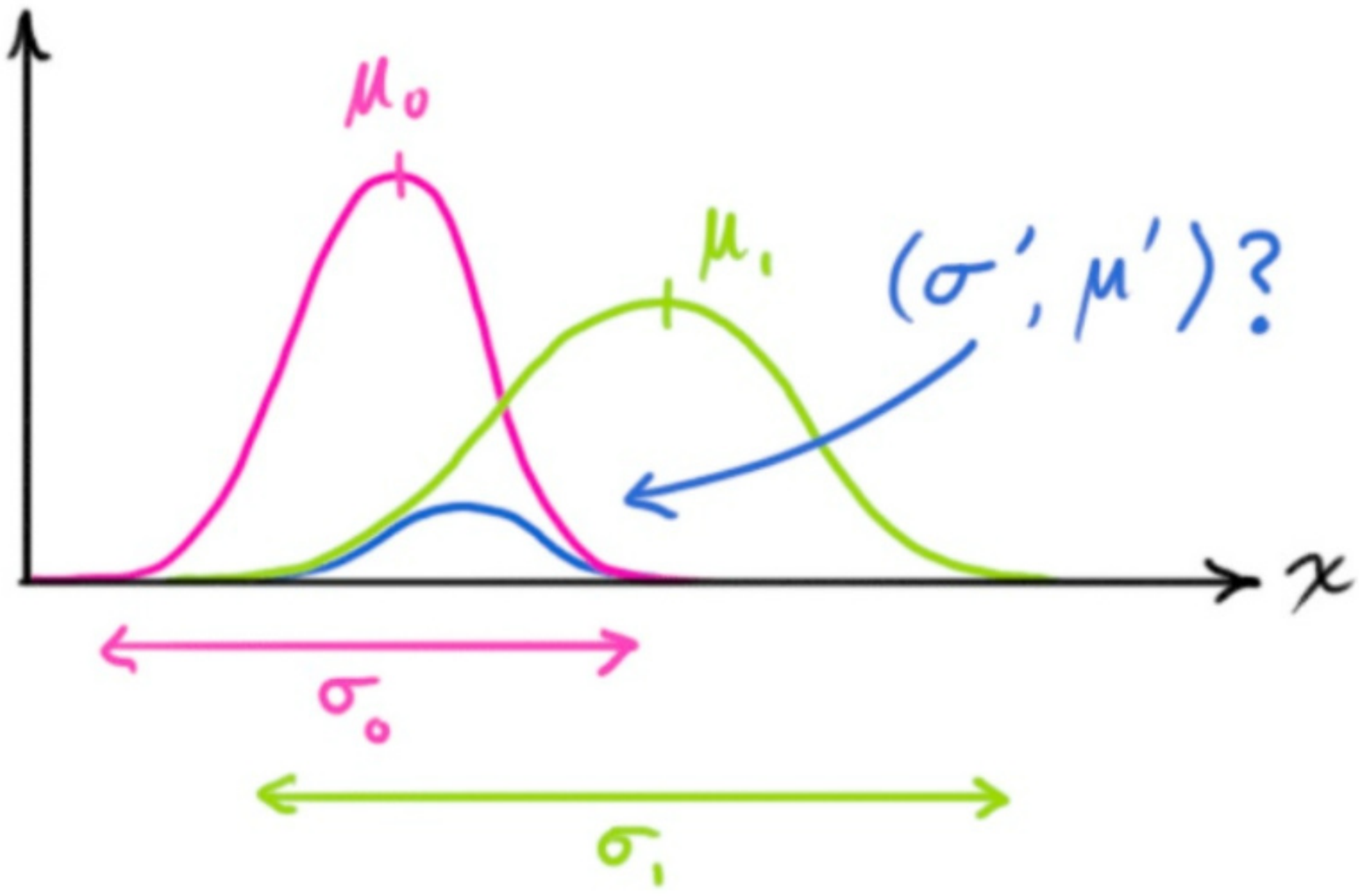
When you multiply two Gaussian distributions with different means and covariance matrices, the result is a new Gaussian distribution with its own mean and covariance matrix. You can probably guess where this leads: there must be a formula to derive these new parameters from the original ones!

Combining Gaussians

Let's work out this formula. It's simplest to start in **one dimension**. A 1D Gaussian bell curve with variance $\sigma 2$ and mean μ is defined as:

$$\mathcal{N}(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{9}$$

Our goal is to understand the result of multiplying two Gaussian curves. The blue curve below illustrates the (unnormalized) overlap of the two Gaussian distributions.



$$\mathcal{N}(x, \mu_0, \sigma_0) \cdot \mathcal{N}(x, \mu_1, \sigma_1) \stackrel{?}{=} \mathcal{N}(x, \mu', \sigma') \tag{10}$$

By substituting equation (9) into equation (10) and performing some algebra (taking care to renormalize so that the total probability equals 1), we arrive at:

$$\begin{aligned} \mu' &= \mu_0 + \frac{\sigma_0^2(\mu_1 - \mu_0)}{\sigma_0^2 + \sigma_1^2} \\ \sigma'^2 &= \sigma_0^2 - \frac{\sigma_0^4}{\sigma_0^2 + \sigma_1^2} \end{aligned} \tag{11}$$

We can simplify this by factoring out a small component and naming it k:

$$\mathbf{k} = \frac{\sigma_0^2}{\sigma_0^2 + \sigma_1^2} \tag{12}$$

$$\begin{aligned} \mu' &= \mu_0 + \mathbf{k}(\mu_1 - \mu_0) \\ \sigma'^2 &= \sigma_0^2 - \mathbf{k}\sigma_0^2 \end{aligned} \tag{13}$$

Notice how the formula allows us to update the previous estimate by adding a new term—keeping it remarkably simple!

Now, what about a matrix version? To do this, we can rewrite equations (12) and (13) in matrix form. If Σ represents the covariance matrix of a Gaussian distribution and $\vec{\mu}$ its mean along each axis, we get:

$$\mathbf{K} = \Sigma_0(\Sigma_0 + \Sigma_1)^{-1} \tag{14}$$

$$\begin{aligned} \vec{\mu}' &= \vec{\mu_0} + \mathbf{K}(\vec{\mu_1} - \vec{\mu_0}) \\ \Sigma' &= \Sigma_0 - \mathbf{K}\Sigma_0 \end{aligned} \tag{15}$$

K is a matrix known as the **Kalman gain**, and we'll apply it shortly.

Putting it all together

We have two distributions: the **predicted measurement** with

$$(\mu_0, \Sigma_0) = (\mathbf{H}_k\hat{\mathbf{x}}_k, \mathbf{H}_k\mathbf{P}_k\mathbf{H}_k^T)$$

and the **observed measurement** with $(\mu_1, \Sigma_1) = (\vec{\mathbf{z}_k}, \mathbf{R}_k)$. By plugging these values into equation (15), we can determine their overlap.

$$\begin{aligned} \mathbf{H}_k\hat{\mathbf{x}}'_k &= \mathbf{H}_k\hat{\mathbf{x}}_k &&+ \mathbf{K}(\vec{\mathbf{z}_k} - \mathbf{H}_k\hat{\mathbf{x}}_k) \\ \mathbf{H}_k\mathbf{P}'_k\mathbf{H}_k^T &= \mathbf{H}_k\mathbf{P}_k\mathbf{H}_k^T &&- \mathbf{K}\mathbf{H}_k\mathbf{P}_k\mathbf{H}_k^T \end{aligned} \tag{16}$$

And from (14), the Kalman gain is:

$$\mathbf{K} = \mathbf{H}_k\mathbf{P}_k\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_k\mathbf{H}_k^T + \mathbf{R}_k)^{-1} \tag{17}$$

We can remove an H_k from the beginning of each term in equations (16) and (17) (noting that one is embedded within K), and an H_k from the end of all terms in the equation for P_k'.

$$\begin{aligned}\hat{\mathbf{x}}'_k &= \hat{\mathbf{x}}_k + \mathbf{K}'(\vec{\mathbf{z}_k} - \mathbf{H}_k\hat{\mathbf{x}}_k) \\ \mathbf{P}'_k &= \mathbf{P}_k - \mathbf{K}'\mathbf{H}_k\mathbf{P}_k\end{aligned} \tag{18}$$

$$\mathbf{K}' = \mathbf{P}_k\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_k\mathbf{H}_k^T + \mathbf{R}_k)^{-1} \tag{19}$$

...resulting in the full equations for the **update step**.

And that's it! $\hat{x}'_k$ becomes our new best estimate, which we can cycle back—along with P_k'—into further rounds of prediction or update, repeating as needed.

To summarize, the **Kalman filter** is an algorithm for estimating the state of a dynamic system from a series of uncertain measurements. It combines predictions of a system's state with actual sensor readings, filtering out noise to produce accurate estimates. The filter works by using two phases: the **prediction** step, where the system's state is projected forward in time, and the **update** step, where new measurements adjust the prediction to refine the estimate. This iterative process allows it to track position, velocity, and other variables effectively in real-time applications, making it particularly valuable in areas like navigation, robotics, and signal processing.

Limitations of the Kalman Filter

Despite its versatility, the Kalman filter has some notable limitations:

1. **Linear Assumptions**: The Kalman filter assumes a linear relationship between the variables it models. However, many real-world systems exhibit non-linear dynamics, where a standard Kalman filter may not perform optimally.
2. **Gaussian Noise Requirement**: It requires that both process noise and measurement noise follow a Gaussian distribution. If the noise is non-Gaussian, the filter's performance can degrade.
3. **Sensitivity to Model Errors**: The Kalman filter's effectiveness depends on accurate system and noise models. If there are errors in these models, the filter may produce inaccurate estimates.
4. **Computational Cost for High-Dimensional Systems**: In high-dimensional state spaces, the Kalman filter can be computationally expensive due to matrix multiplications and inversions.

Solutions to Address Limitations

Several adaptations of the Kalman filter address these limitations:

1. **Extended Kalman Filter (EKF)**: For non-linear systems, the EKF linearizes the system around the current estimate, making it more effective for many non-linear applications.
2. **Unscented Kalman Filter (UKF)**: The UKF approximates the Gaussian distribution without linearization, making it more accurate in highly non-linear situations.
3. **Particle Filters**: For systems with non-Gaussian noise or highly non-linear dynamics, particle filters use a probabilistic approach, representing the state with a set of samples. They are computationally intensive but more flexible.

These variations and adaptations help expand the Kalman filter's use in complex, non-linear, and high-noise environments, making it a foundational tool in signal processing and control systems.