Applied State Estimation

Braeden Windham

August 2025

Contents

The G-H Filter

The key insight to Bayesian and Kalman filters is that we need to take some kind of blend of predictions and measurements, otherwise, we're not using one or the other. How do we choose to "blend" this data? That is the premise of the rest of the book.

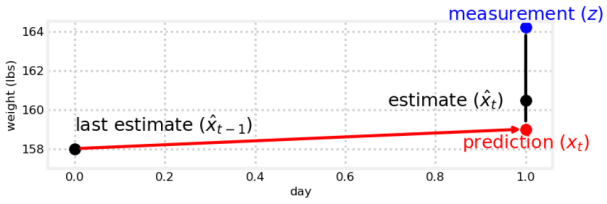


Figure 1: Estimate, measurement, and prediction.

In **Figure** 1, the difference between the measurement and the prediction is shown with a black line and is called the **residual**. The author arbitrarily chose a weighting factor here to blend the prediction and the measurement of $\frac{4}{10}$. This results in:

$$estimate = prediction + \frac{4}{10}(measurement - prediction) \tag{1}$$

where

$$prediction = current\ weight + \ 1 \tag{2}$$

as the assumption was made that a weight gain of 1 lb per day was known.

However, this rate of change (the 1 lb a day assumption) is fixed and if this doesn't match reality, then our estimates diverge from out measurements! What happens if we estimate this weight gain by using our last gain and the difference between the measured weight gain

and our predicted? This looks something like

$$new\ gain = old\ gain\ +\ \frac{1}{3}\frac{measurement - predicted\ weight}{1\ day}. \tag{3}$$

Now, our original equation becomes

$$estimate = prediction + \frac{4}{10}(measurement - prediction) \tag{4}$$

where

$$prediction = estimate\ +\ new\ gain \tag{5}$$

Discrete Bayes Filter

In Bayesian statistics, a probability before including a measurement is known as a **prior**. For a collection of all possible probabilities of an event, we call this the **prior probability distribution.** Let's say that we have a dog in a hallway with doors. Let's choose 10 discrete locations in that hallway.

When we first start listening to the sensor, there is no reason to believe that the dog is in any of the 10 possible positions vs any other one. So, the probability that the dog is in nay of the 10 positions is $\frac{1}{10}$. In this case, if we represented this with a histogram, it would be 10 bars of value 0.1. This is our **prior**. In three of those positions there is a door. Let's also assume that the tenth door wraps back around to the first door. We'll use 1 for doors and 0 for hallways.

Let's say that he first sensor reading comes in and says "door" and for the moment, we assume that the sensor readings are perfect. Using this information, we now know that the dog could only be in one of three places, in 0,1 or 9, where we placed the doors. Each bar of this histogram has a height of $\frac{1}{3}$.

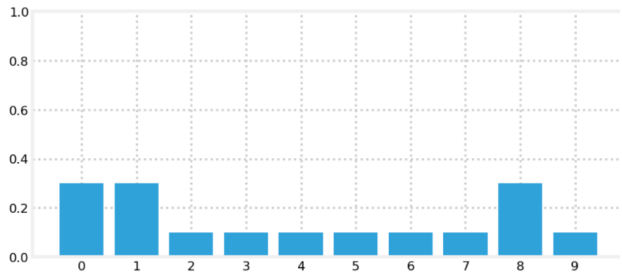
Now, let's say that we get a reading of door the dog moves to the right, and we get another reading of door. Now, we know the dog's location! The only pace that this sequence can occur is moving form position 0 to position 1. So, the takeaway here is that the first sensor reading gave us low probabilities ($\frac{1}{3}$) but after a position update and another sensor reading, we could be more confident (in this case exactly so) about the location of the dog. One could imagine that this could be extended to a longer hallway with a large number of doors. After several movements and sensor reading, we could narrow the dogs position down to a few possibilities.

Noisy Sensors

Let's consider the real world problem of noisy sensors. Now, when our dog is in front of a door, there is a chance that the sensor returns "hallway" instead. Thus, if we go back to our example hallway with 10 positions, we can't use the probability at each of the doors of $\frac{1}{3}$. Instead, our distribution may look something like this.

```
[.31, .31, .01, .01, .01, .01, .01, .01, .31, .01]
```

If we're uncertain about every measurement that we take, how can we ever draw any conclusions form it? The answer is with probabilities! we're already assigning a belief to the dogs location, we'll just do the same thing while incorporating the sensor error. Let's say that if we get a sensor reading of "door" and testing shows that our sensor is 3 times more likely to return "door" than "hallway when at a door. Let's look at the probability distribution again after our first reading of "door"



Note that this is not **truly** a probability distribution, because it doesn't sum to 1. However we can fix this by dividing each column by the sum of the columns! Saying "three times more likely" is also an odd way od talking about this. Let's instead use the scaling value given by

$$scale = \frac{prob_{correct}}{prob_{incorrect}} \tag{6}$$

At this point, we don't have any information that would lead us to believe that he dog is in front of any specific door, so the values should be the same for all potential door locations. This is called the **posterior** or the **posterior probability distribution** because this is the distribution *after* the incorporation of a measurement. Another term is **likelihood**.The likelihood was the "unscaled" version of the posterior, leading to the following relationship.

$$posterior = \frac{likelihood \times prior}{normalization} \tag{7}$$

Generally in a filter, we call the state after performing the prediction the *prior* or *prediction* and we call the state after the update the *posterior* or the *estimated state*. **The computation of the likelihood varies per problem!** Some sensors return distances, some return a 1 or 0, etc.

Incorporating movement

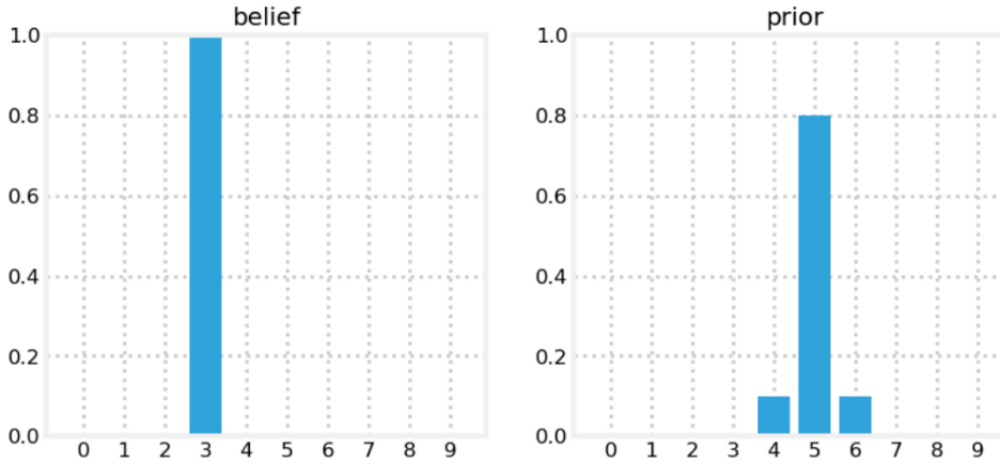
Recall when we had perfect information, how easily we could deduce that the dog was a the door in position 1. Let's again assume that we have perfect knowledge of the dogs motion. If we move one space to the right, then prior to incorporating a new sensor measurement, we would expect our belief to look like our belief prior to the movement, but shifted to the right one position.

As an example, let's assume that the position of a dog is 17m. our epoch is 2 seconds long and the dog is traveling at 15m/s. Where do we predict that he'll be in 2 seconds? Clearly

$$\bar{x} = 17 + (15 * 2) \tag{8}$$

Adding Uncertainty to the Prediction

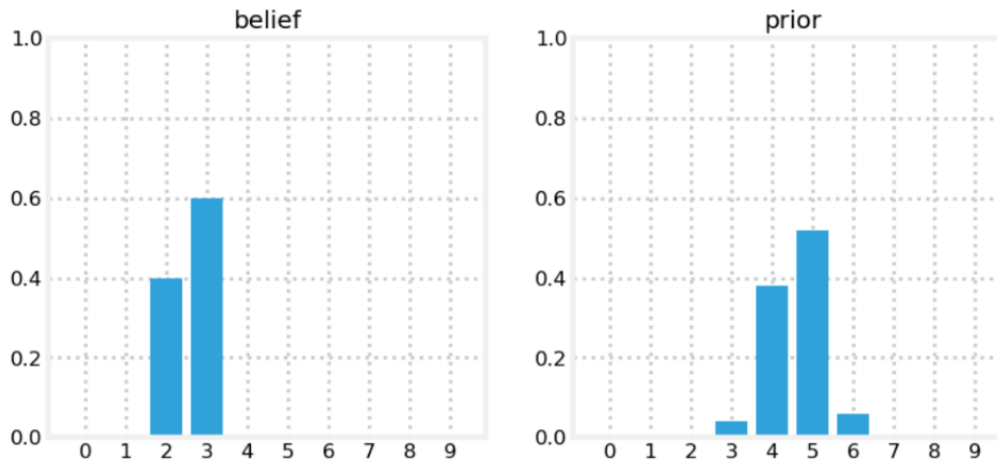
Instead of assuming perfect knowledge of the dog's motion, let's say that upon moving one space the the right, the sensor is is 80% likely to return 1 space, and there is a 10% chance to return either 0 spaces or 2 spaces. Then, assuming we know that we are at at position three and our imperfect motion sensor says that we've moved 2 spaces, our new belief will look like the following.



Now, what happens if we start with a belief that is not 100% certain? We can see from the

following that our motion prediction causes our belief to spread out more. If the probability of being at position 2 prior to prediction was 0.4 an being at position 3 was 0.6, then our prior can be interpreted as the following:

- The probability of being at position 3 is 0.04 as this would have to be an undershoot from starting at position 2
- The probability of being at position 4 is 0.38. we could have arrived at this position only two ways.
    1. we started at position and moved two spaces as predicted (0.4×0.8)
    2. We started at position 3 and undershot (0.6×0.1)
- The probability of being at position 5 is 0.52 with 0.48 coming from moving two spaces from position 3 and 0.04 coming from overshooting from position 2.
- Lastly, the probability of being at position 6 is 0.06 as the only way for this to happen is to overshoot form position 3.



Note that if we continued this trend step after step, eventually, this distribution would GREATLY flatten out leading us to be unable to have any certain belief about location. Note that what we have done is **convolved** our belief with the error function/distribution associated with our sensor! So, to recap, We're making a prediction of the dog's location from our model of the dog that says he'll move 2 positions and we're convolving our belief in his current location with the probability distribution associated with the sensors error.

If we weren't using probabilities to do this we would use

$$\bar{x}_{k+1} = x_k + f_x(\cdot) \tag{9}$$

where $f_x(\cdot)$ is the function that tells us how the state of the system changes from sample to sample. But, because we're using probabilities, we'll use the equation

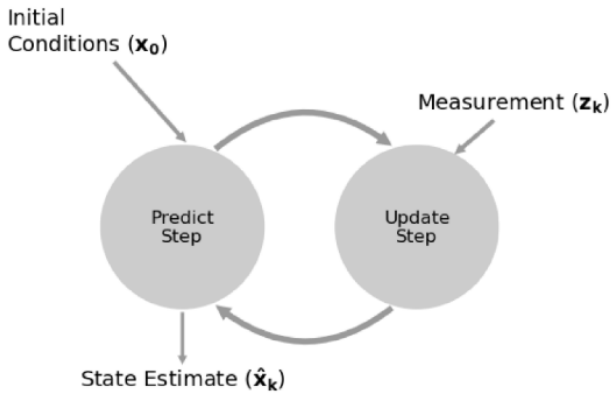
$$\bar{x}_{k+1} = x_k * f_x(\cdot) \tag{10}$$

because we don't have perfect knowledge of how our dog moves over time, but we have a distribution that shows how he likely moved.

Integrating Measurements and Movement Updates

The issue of losing information with every prediction due to uncertainty may seem like we can't reasonably deduce anything from imperfect sensor measurements. However, we also integrate measurements after each prediction which improves our knowledge! This is passed into the next prediction, and the cycle continues.

NOTE: between the **prior** and the **posterior**, you incorporate a **measurement** and between the **posterior** and the **prior**, you make a **prediction**. The author also makes liberal use of the word *belief*. Stick to prior and posterior.



The above diagram shows the following pseudocode

1. **Initialization**

   (a) Initialize our belief in the state

2. **Predict**

   (a) Based on the system behavior, predict state for the next time step

   (b) Adjust belief to account for uncertainty in prediction

3. **Update**

   (a) Get a measurement and associated belief about its accuracy

   (b) Compute how likely it is that the measurement matches the state

   (c) Update state belief with this likelihood

Written another way

$$\bar{x} = x * f_x(\cdot) \tag{11}$$

$$x = ||\mathcal{L} \cdot \bar{x}|| \tag{12}$$

where $\mathcal{L}$ is the likelihood function.

Drawbacks and Limitations

This is a perfectly viable filter if you need a multimodal, discrete filter. With that said, this filter isn't used often because it has many limitation. Getting around these is the motivation for studying follow-on filtering techniques.

One problem is scaling. We've looked at examples where we've tracked one variable. But most interesting filters will track multiple. This data is stored in a square grid meaning a problem with 3 variables becomes $\mathcal{O}(n^3)$. The second problem is that the filter is discrete. The histogram requires that you model your output as a set of discrete points. a 100 meter hallway requires 10,000 positions to model the hallway to 1cm accuracy.

Lastly, the filter is multimodal. This isn't always a problem, but imagine if your GPS said thatthere's a 30% chance that you're on Broadway Ave and a 40% chance that you're on Wapakoneta Ave.

Tracking and Control

Now, let's say that we're moving a robot on a track. We can place magnets every few feet and when the robot passes them we can detect this with a hall effect sensor. However, we also know that we've given a control input the the system and, assuming we have a good model of it, we know how much it *should* have moved, given this input. So now, during the

predict step, we will pass in the commanded movement that we gave the robot along with a kernel that describes the likelihood of that movement.

Bayes Theorem and teh Total Probability Theorem

Thus far, we've developed the necessary math via reasoning about it at each step. In doing so we've discovered *Bayes' Theorem* and the *Total Probability Theorem*. Bayes tells us how to compute the probability of an event given previous information.

Probabilities, Gaussians, and Bayes' Theorem

Each time a die is rolled, the **outcome** will be between 1 and 6. If we rolled a fair die an infinite number of time, we'd expect 1/6 of that infinite number of rolls to be a 1. The **probability** of the outcome 1 is 1/6. This combination of values and probabilities is called a **random variable**. The range of values is called the **sample space**, where a space is a term that means a set with structure. random variables such as dice rolls are **discrete random variables**, whereas something like the height of a population is a **continuous random variable**.

The **probability distribution** gives the probability that a random variable takes on a value in the sample space. this is denoted with a lowercase p such as

$$P(X = 4) = p(4) = \frac{1}{6} \tag{13}$$

for a fair, 6-sided die. Sample spaces are not unique. One sample space for a die is {1,2,3,4,5,6} whereas another might be {odd, even}. A sample space is valid as long as it covers all possibilities, and any event is described by only one element of the set. On order to be a probability distribution, the following must be true for discrete random variables

$$\sum_u P(X = u) = 1 \tag{14}$$

and for continuous random variables

$$\int_u f_X(u)\, du = 1 \tag{15}$$

Expected Value of a Random Variable

This is the average value that a random variable would take have if we took an infinite number of samples and averaged those samples. If we have $x = [1, 3, 5]$ and all of these values are equally probable, then what value would we expect x to have on average? It would be the average of 1, 3, and 5, or 3.now suppose that there is an 80% chance of 1, a 15% chance of 3, and a 5% chance of 5. Now, the expected value is given by

$$E[X] = 1 * 0.8 + 3 * 0.15 + 5 * 0.05 = 1.5. \tag{16}$$

For the discrete random variabel case, this can be written as

$$E[X] = \sum_{i=1}^{n} x_i \, p_i \tag{17}$$

and for the continuous case, this can be written as

$$E[X] = \int_a^b x \, f_X(x) \, dx \tag{18}$$

Variance of a Random Variable

The variance of a random variable is given by

$$\mathrm{Var}(X) = E\big[(X - E[X])^2\big] = E\big[(X - \mu)^2\big]. \tag{19}$$

In simple terms, it is the expected value of the **squared** difference of each element of a set with that sets expected value, or average. Note that this is squared because, if it weren't, positive and negative values could cancel one another out. An alternative might be the absolute value of the difference, but quadratic terms have "nice" properties that we like. Also, squaring the difference weighs outliers more heavily as compared to an absolute value. This may or may not be a good thing, and is a design decision.

The **standard deviation** is related to the variance by a square root

$$\sigma = \sqrt{\mathrm{Var}(X)} = \sqrt{E[(X - \mu)^2]}. \tag{20}$$

It's common when describing how much values in a population vary to use the standard deviation. The variance appears to be more common in computations. Lastly, the standard

deviation is commonly given by σ and the variance is given by σ^2.

Gaussians

A gaussian distribution is waht we calla probability density function or *pdf*. Gaussians have all of the qualities that we're looking for in a probability distribution. It is unimodal, continuous, and computationally efficient! Note that these gaussians represent the *probability density* of random variable.

Gaussians are given by

$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{21}$$

The reason that they're so prevalent is due to the **Central Limit Theorem** which states that if we make repeated measurements, these measurements will be normally distributed. We called these probability ***density*** functions because as we integrate from some value to another, it tells us the probability of an event occurring between them! The integral of this distribution as a function pevent ios called the *Cumulative distribution function* or the **CDF**. the notation for a **normal** distribution for a random variable X is

$$X \sim \mathcal{N}(\mu, \sigma^{\in}). \tag{22}$$

Since this is a probability density function, it is required that the are under the curve is equal to 1. Intuitively, this means that the probability of **something** happening is 1. Sounds pretty reasonable.

Computational Properties of Gaussians

The discrete bayes filter multiplies and adds arbitrary probability distributions. The Kalman filter does the same thing only with Gaussians instead. Importantly, the sum of two Gaussians is another Gaussian and the product of two Gaussians is not Gaussian, but is proportional to one i.e. it's not a "unit" Gaussian. This is **why** the Kalman filter uses Gaussians. They have **really** nice computational properties.

The product of two independent Gaussians is given by

$$\mu = \frac{\sigma_1^2\mu_2 + \sigma_2^2\mu_1}{\sigma_1^2 + \sigma_2^2}, \qquad \sigma^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2}.$$

The sum of two Gaussians is given by

$$\mu = \mu_1 + \mu_2, \qquad \sigma^2 = \sigma_1^2 + \sigma_2^2.$$

Bayes Theorem

in our update step, we have

$$posterior = \frac{likelihood \times prior}{normalization}. \tag{23}$$

It turns our that this is *Bayes Theorem*. The simple idea can be expressed as

$$updated\ knowledge = ||likelihood\ of\ new \times prior\ knowledge||. \tag{24}$$

Bayes theorem is given by

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{25}$$

where $P(A|B)$ is called a **conditional probability** and is read as "The probability of A occurring, given B has occurred." Let's rewrite this for our dog tracking problem.

$$p(x_i|z) = \frac{p(z|x_i)p(x_i)}{p(z)} \tag{26}$$

Note that here we've used p instead of P because this is not the probability that something has happened, but the probability distribution, given other probability distributions. This is maybe a bit abstract, but let's break down each of the terms. $p(z|x_i)$ is the likelihood, or the probability of the measurement for a position x_i. $p(x_i)$ the prior. This comes from predicting from inputs and our last posterior.

The last term is $p(z)$. This is the probability of getting the measurement z without taking the position x into account. This is sometimes called the **evidence**. I don't quite understand how dividing by the probability of the measurement z occurring here is the same as normalizing. Maybe look into how normalizing a Gaussian occurs?

Total Probability Theorem

To compute the posterior from a prior and a measurement, Bayes theorem was used. Now, in order to predict eh current prior from the last posterior, we will use the **total probability**

theorem. The probability of being at a position i at time t can be written as $P(X_i^t)$. This was computed as the sum of the prior at time $t-1$ multiplied by hte probability of moving from cell x_j to x_i or

$$P(X_t^i) = \sum_j P(X_{t-1}^j)\, P(x_i \mid x_j). \tag{27}$$

The Kalman Filter!

Let's again assume that we are trying to track the position of a dog in a hallway. We have an RFID sensor that gives us relatively accurate measurements from one end of the hallway. We can represent our belief that the dog is in a given position with a Gaussian. Recall that the discrete Bayes filter used a histogram of probabilities to track the dog. Each position had a bin and the value in that bin was the probability of the dog being in that place. Tracking was done with the following equations

$$\textit{Prediction step} \quad \bar{x} = x * f_x(\cdot) \tag{28}$$

$$\textit{Update step} \quad x = \mathcal{L} \cdot \bar{x} \tag{29}$$

but now, we will replace the histograms represented by x with Gaussians representing out beliefs! This allows us to replace hundreds or thousand of numbers with just two for each distribution, the mean μ and the variance σ^2. In the Bayes filter, in order to carry out the prediction step, we convolved the posterior from the prior step with a function describing how the state transitions with time. From the total probability theorem, we found this to be an addition. With our implementation with Gaussians, lets just add the two distributions as this will also result in another Gaussian. So now, our filter equations are

$$\textit{Prediction step} \quad \bar{x} = x + f_x(\cdot) \tag{30}$$

$$\textit{Update step} \quad x = \mathcal{L} \cdot \bar{x} \tag{31}$$

where x and $\bar{x}$ are given by normal distributions. Let's look at each of these steps individually.

Predictions with Gaussians

We use Newton's equations of motion to compute thew current position based on the current velocity and the previous position:

$$\bar{x}_k = x_{k-1} + v_k \Delta t = x_{k-1} + f_x. \tag{32}$$

We are uncertain about he dog's position and velocity, so we need to express these values as Gaussians. Let's say the dogs initial position was 10m with a standard deviation of 0.2m and the velocity is $15\frac{m}{s}$ with a standard deviation of 0.7m. than carrying out the following operation we arrive at

$$\mathcal{N}(10, 0.2^2) + \mathcal{N}(15, 0.7^2) = \mathcal{N}(25, 0.53). \tag{33}$$

This makes good sense! we expect the mean value to be at 25 if we thought that we we're at 10, then moved 25. Also, note that the standard deviation is **greater** than either of the two Gaussians that were added. Recall back to when we repeatedly predicted, we became less certain with each prediction if we didn't incorporate a measurement to reduce the uncertainty again.

Updates with Gaussians

For the normalized product of two Gaussians, the mean is given by

$$\mu = \frac{\sigma_1^2 \mu_2 + \sigma_2^2 \mu_1}{\sigma_1^2 + \sigma_2^2}, \tag{34}$$

and the variance is given by

$$\sigma^2 = \frac{\sigma_1^2 \sigma_2^2}{\sigma_1^2 + \sigma_2^2}. \tag{35}$$

Kalman Gain

The posterior is computed as the likelihood times the prior. We know that te solution to the product of two Gaussians is given by

$$\mu = \frac{\sigma_1^2 \mu_2 + \sigma_2^2 \mu_1}{\sigma_1^2 + \sigma_2^2}, \tag{36}$$

which can be written as

$$\mu = (\frac{\bar{\sigma}^2}{\bar{\sigma}^2 + \sigma_z^2})\mu_z + (\frac{\sigma_z^2}{\bar{\sigma}^2 + \sigma_z^2})\bar{\mu} \tag{37}$$

where a z denotes a mean and variance of a measurement and a bar represents the mean and variance of a prior. In this form, it's possible to see that the posterior is in fact a weighted sum of the measurement and the prior:

$$\mu = W_1\mu_z + W_2\bar{\mu}. \tag{38}$$

Let $K = W_1$. Then:

$$\mu = K\mu_z + (1 - K)\bar{\mu} \;=\; \bar{\mu} + K(\mu_z - \bar{\mu}) \tag{39}$$

K is the **Kalman Gain**. It's the term that chooses the value part way between the measurement and the prior. Note that the variance can be expressed in terms of the Kalman gain as well

$$\sigma^2 = \frac{\bar{\sigma}^2\sigma_z^2}{\bar{\sigma}^2 + \sigma_z^2} = K\,\sigma_z^2 = (1 - K)\,\bar{\sigma}^2$$

We can re-write our update and predict functions using the Kalman gain and a few new terms.

```
def update(prior, measurement):
    x, P = prior        # mean and variance of prior
    z, R = measurement  # mean and variance of measurement

    y = z - x        # residual
    K = P / (P + R)  # Kalman gain

    x = x + K*y      # posterior
    P = (1 - K) * P  # posterior variance
    return gaussian(x, P)

def predict(posterior, movement):
    x, P = posterior # mean and variance of posterior
    dx, Q = movement # mean and variance of movement
    x = x + dx
    P = P + Q
    return gaussian(x, P)
```

Here, R is used for the measurement noise, Q is the process noise, and P is the variance of the state.

Multivariate Gaussians

In the last chapter, our Kalman filter was only able to estimate a single variable of interest. In the case of motion, this would be the position of an object of interest. However, we know that position and velocity are related, and we shouldn't throw information away, so now we'll learn how to describe this relationship probabilistically and get better filter performance!

Before when specifying a normal distribution, the mean was a scalar, but what if we're interested in the x and y position of an object in the plane, and we want to represent this belief with a Gaussian? Well, instead of $\mu \in \mathbb{R}$ we would have $\mu \in \mathbb{R}^n$ or

$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix}. \tag{40}$$

Now, for the variances, we would **NOT** have a **vector** of variances $\sigma^2 \in \mathbb{R}^n$, but instead a **matrix** $\sigma^2 \in \mathbb{R}^{n \times n}$. This may seem a little unintuitive, but if we think of the two variables being tracked as height and weight of a population, there is surely some correlation between the height of a person and how much they weigh! Let's dig into this concept and see if we can make intuitive sense of it.

Covariance and Correlation

Covariance is a measure of how much two things vary with respect to one another. Height and weight are a great example of this. These are positively correlated. As height increases, generally, weight also increases. Temperature and heating bills are inversely correlated. Your shoe size and the price fo tea in China have **no** correlation to one another, so we might say that these are uncorrelated, or *independent* of one another. We'll assume linear correlation, but the concept of *nonlinear* correlation also exists. For two random variables X and Y, we define the covariance as

$$COV(X, Y) = \sigma_{xy} = E[(X - \mu_x)(Y - \mu_y)]. \tag{41}$$

The covariance matrix takes on the form

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1n} \\ \sigma_{12} & \sigma_2^2 & \cdots & \sigma_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{1n} & \sigma_{2n} & \cdots & \sigma_n^2 \end{bmatrix} \tag{42}$$

where the diagonal components are the *variances* and the off-diagonals are the covariances. Note that the covariance matrix is symmetric! For a dataset of height and weight, we would assume that every measurement or data point is equally likely, so to compute the expected value, we would use

$$E[X] = \frac{1}{N} \sum_{i=1}^{n} x_i. \tag{43}$$

This can be done in Numpy, **but** when doing so, it computes the expected value as such

$$E[X] = \frac{1}{N-1} \sum_{i=1}^{n} x_i. \tag{44}$$

This is because it assumes that we have a *biased estimator*. This happens you have, say, a population of 100 people, but you only sample 10 of them. In this case, it's better to use the default form of Numpy's covariance function. Otherwise, if you use data from the entire population, you would say that you have an *unbiased estimator*. In this case you would use **equation 43**.

Multivariate Normal Distribution Equation

Recall that for the scalar case, the *pdf* of a normal distribution is given by

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2} \frac{(x-\mu)^2}{\sigma^2} \right). \tag{45}$$

For the multivariate case, this is given by

$$p(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n \left|\Sigma\right|}} \exp\left(-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})\right) \tag{46}$$

Defining some terms, the *joint probability*, $P(x, y)$, is the probability of both x and y happening. The *marginal probability* is the probability of an even happening without regard to any other event.

Note that **independent** variables are also always **uncorrelated**, however, the opposite of this is not always true.

Multiplying Multidimensional Gaussians

Previously , during our update step, we combined an uncertain measurement with an uncertain prediction and the resulting probability distribution was a combination of the two with a smaller variance! The same happens in multiple dimensions. the equations for multiplying multivariate Gaussians ar as follows.

$$\mu = \Sigma_2(\Sigma_1 + \Sigma_2)^{-1}\mu_1 + \Sigma_1(\Sigma_1 + \Sigma_2)^{-1}\mu_2 \tag{47}$$

and

$$\Sigma = \Sigma_1(\Sigma_1 + \Sigma_2)^{-1}\Sigma_2. \tag{48}$$

Let's look at how we can use this with an example. Let's say that we're tracking an aircraft with two radar systems, ignoring altitude. Let's say that the radar's bearing measurement is accurate, but the range is inaccurate. Then if you have two radars, one "below" and to the left and one "below" and to the right of the aircraft, then your best estimate is the intersection of these two ellipses. the more orthogonal they are, the smaller the resulting ellipse whereas if they're aligned, the uncertainty doesn't change as much.

Hidden Variables

From our example above, we can see why multivariate Kalman filters can perform better than a univariate one, because the correlations can improve our estimate. If we're tracking an aircraft, let's say that we get x and y data from our sensor. If we got three readings roughly in a line at a similar distance apart, we could make the assumption that the next reading would be along that same line, at a similar distance. Really, what we're doing is inferring the current velocity of the object, and making an estimate of where it will be based on this. The position is an *observed variable* and the velocity is a *hidden variable.*

Multivariate Kalman Filters

When you design a Kalman filter, you start with differential equations that describe a system's dynamics. We'll start with Newton's classic equations of motion because we have a

closed for solution of these. As a bonus, these equations can be used to track moving objects, one of the main uses of Kalman filters!

Again, our algorithm is as follows:

Initialization

1. Initialize the state of the filter
2. Initialize our belief in the state

Predict

1. Use process model to predict state at the next time step
2. Adjust belief to account for the uncertainty in prediction

Update

1. Get a measurement and associated belief about its accuracy
2. Compute residual between estimated state and measurement
3. Compute scaling factor based on whether the measurement or prediction is more accurate
4. Set state between the prediction and measurement based on scaling factor
5. Update belief in the state based on how certain we are in the measurement

Mathematically, these turn out to be: **Predict**

Univariate	**Multivariate**
$\bar{\mu} = \mu + \mu_{f_x}$	$\bar{x} = Fx + Bu$
$\bar{\sigma}^2 = \sigma_x^2 + \sigma_{f_x}^2$	$\bar{P} = FPF^\top + Q$

Update

Univariate	**Multivariate**
$\mu = \frac{\sigma_z^2 \mu_x + \sigma_x^2 \mu_z}{\sigma_x^2 + \sigma_z^2}$	$y = z - Hx$
	$K = PH^\top (HPH^\top + R)^{-1}$
$\sigma^2 = \frac{\sigma_x^2 \sigma_z^2}{\sigma_x^2 + \sigma_z^2}$	$x = \bar{x} + Ky$
	$P = (I - KH)P$

Note: F is used instead of A and H is used instead of C. x and P are the state mean and covariance respectively. F is the state transition matrix, and Q is the process covariance (the term associated with the process noise). Obviously, u is the control input and B determines how the inputs affect the state of the system. Ignoring H in the update function, you can see that these functions are also extremely similar. While the details will be different, the concepts are the same.

Tracking an Object

Let's go back to the idea of tracking a dog. We'll assume that the dog is moving at approximately $1\frac{m}{s}$ with some process variation. Let's start with the **predict** step and design the filter in order. Here, x will be our state vector and P will be our covariance matrix that represents our certainty in the states and their correlation. In our case, we have a sensor that returns position, so position is our *observed variable* and the velocity is a *hidden variable*.

Predict Step

State

Now, to initialize our belief in the state of our dog we may initialize $\mathbf{x}$ to be $\mathbf{x} = [x\ \dot{x}]^T = [10\ 1]^T$. This says that when we initialize the filter, that we believe that our dog is 10 meters away form the sensor and that it has an initial velocity of 1m/s.

Covariance (P)

The other part of specifying the Gaussian is choosing an appropriate variance for both the position and the velocity. If we're very uncertain about the position of the dog, we may choose to initialize the variance of the position as $\sigma^2_{pos} = 500m^2$. This means that we believe that current position is accurate $\pm \approx 67$m. If we're alos uncertain of the velocity, we can sya that an upper bound on a dog's speed is around 21m/s. Then, $3\sigma_{vel} = 21 \rightarrow \sigma^2_{vel} = 49m/s^2$. This takes care of the diagonal entries, but what about he correlation between the velocity and position for a dog? We have no clue, so let's initialize these to zero and let the updating take care of this! So our covariance matrix, P is given by

$$P = \begin{bmatrix} 500 & 0 \\ 0 & 49 \end{bmatrix}. \tag{49}$$

The Process Model

Here is where we specify, given the state at the last time step and the inputs to the system, how do we determine the current state. For Newton's equations, this process is relatively simple. The state transition matrix is given by

$$\begin{bmatrix} \bar{x} \\ \bar{\dot{x}} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \tag{50}$$

This process model is used to perform the **innovation**, which up to now, we've called the *prediction*. Note that bars denote the **prior**.

Process Noise Design

Process noise can be thought of as disturbances and/or model error. We account for this by adding a process noise covariance matrix, Q, to our state's covariance P. Note that here we don't add anything to x because our noise is assumed to be *white*. For the univariate Kalman filter, we used $\sigma^2 = \sigma^s + \sigma^2_{process}$. We're essentially doing the same here, but with a few more terms. For now, know that $Q = \mathbb{E}[ww^T]$, or that the process noise is the expected value of the white noise w.

The Control Function

Th Kalman filter is not just used for data, we can use control inputs to our system as well. This is the B matrix if you're working with the linear state space representation of a system. For the example of the dog, let's say that the dog *doesn't* listen to you. Then, for a command, u, your B matrix may look like $B = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$.

Update Step

The Measurement Function

Often, we have sensors that report some quantity of interest such as a distance sensors. However these don't directly tell us a distance, they might return to us a voltage which corresponds to a distance. Thus, it's on us to ensure that during the update step, when we

compute the residual, that we're subtracting "like" quantities i.e. not

$$residual = distance - voltage,$$

as this is clearly nonsense. Let's say that we have a sensor that returns a voltage corresponding to distance and the state vector that we're working with is $\mathbf{x} = \begin{bmatrix} x \\ \dot{x}] \end{bmatrix}$, then we would use information from the sensor datasheet to turn the voltage into a distance and call that measurement z. Then we would choose our *measurement function* H, to be

$$H = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

Now, when computing the residual, y, we have

$$y = z - H\mathbf{x}, \quad y = z - \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$$

.

Measurement Design