The first thing we need to know when working with unreliable sensors is how unreliable do we expect them to be? This makes a great deal of intuitive sense; if a sensor is nearly always close to correct, then we can trust a single reading much more than if the sensor often gives readings far off from truth.

In our previous example, we were working with a binary sensor that would be wrong some percentage of the time. Many sensors aren’t binary, so the question isn’t is it wrong, but by how much is it wrong? For example, a sonar may tell you the wall is 1m away. But, of course, the wall isn’t exactly 1m away, it may be 1.003 m away, or .97 meters away. Hopefully, truth is very close to what the sensor tells you.

Frequently, we choose to model the error in a sensor with a Gaussian distribution, also known as a normal distribution. This is the “bell curve” you’ve likely seen before. A Gaussian distribution is determined by its mean \( \mu \), and its standard deviation \( \sigma \).

Figure 1: Gaussian distributions. Blue: \( \mu = 0, \sigma = 1 \), Red: \( \mu = 0, \sigma = 2 \), Green: \( \mu = 3, \sigma = 1 \).

The mean \( \mu \) determines where the peak of the curve is, and the standard deviation \( \sigma \) determines how wide it is. Gaussians are advantageous for at least two reasons:

1. It’s easy to convince yourself that it’s an accurate model (“the average of the error is close to zero, and most of the time I get small error. However, it’s possible, though unlikely, to have significant error”).

2. A Gaussian times a Gaussian is a Gaussian. This is unusual in distributions. We’ll see why this is really, really handy later.

We have two sources of error. First, we might have an incorrectly-calibrated sensor, and it may return an observation that is some multiple of the truth. For example, an observation may always be 1.5 times the truth: \( o = c \cdot x \), where \( c = 1.5 \).

Second, we have random error. In an accurate sensor, we would expect the random error to be around zero, meaning \( \mu = 0 \), and \( \sigma \) would be small. That would mean the range of likely amounts of random error is very, very small, and close to zero. An inaccurate sensor with a wide amount of variation on readings would have error with large standard deviation.

So, if \( o \) is the sensor reading, and \( x \) is the true distance, then we can model this as:

\[
o = c \cdot x + N(0, \sigma_o),
\]

where \( c \) is the coefficient for systemic error (hopefully close to 1), and \( N(0, \sigma_o) \) denotes the random error. In mathematics, we say that the random error is a sampling from the Gaussian distribution with a mean of 0 and a standard deviation of \( \sigma \).

For a Gaussian distribution with mean \( \mu \) and standard deviation \( \sigma \), the probability of sampling a value \( o \) from that Gaussian is given by the probability density function of that Gaussian:

\[
p(o) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(o-\mu)^2}{2\sigma^2}}.
\]

That’s ugly! Fortunately, in real life, it’s not necessary to have this memorized:
from math import *
def normpdf(d,mu,sigma):
    return (1/sqrt(2*pi*sigma*sigma))*exp(-.5*pow(d-mu,2)/pow(sigma,2))

So, if we know the real life measurement $x$, and we know $c$ and $\sigma$, it is easy to get the conditional probability $p(o|x)$ (the probability of getting observation $o$ given that the actual distance is $x$) by calling our function $\text{normpdf}(o,c*x,\sigma)$. As you know from Bayes' Filters, knowing this conditional probability is likely to be important.

A short note about continuous probabilities: Continuous probabilities, when you get down to the details, are a little bit weird. For example, if a sensor tells you something is 1.03 m away, what is the probability it is EXACTLY 1.3 meters away? 0! This is because we can always add more digits after the decimal point until we find a digit where our measurement is not exactly correct. So, if we're strict about it, when we talk about the probability of some $x$, it is always zero. However, the probability density function $p(x)$ is NOT always zero, as it is set up so that $\int p(x)dx = 1$.

So, even though we'll SAY “what is the probability of $x$,” what we MEAN is “what is the value of the probability density function of $x$.” This is one of the few cases where mathematical rigor loses out to common sense, and should be cherished.

What about moving the robot? Well, odometers are just sensors; they're just as wrong as anything else can be. So, if instructed to move $u$ meters, they actually move $b \cdot u + N(0,\sigma_u)$ meters. It is the same with rotation. These parameters $b$ and $\sigma_u$ are known as the robot's motion model.

Kalman Filter

The easiest way to extend the Bayes' Filter we learned about last time to a world with continuous probabilities is known as the Kalman Filter.

A robot with a sonar is facing a wall. It wants to know its location by measuring its distance from the wall. The sensor has an error model of $c = 1$ and $\sigma = 2$ (no systemic error, and random error with standard deviation of 2m). We start with the belief that the robot is somewhere around 20m away from the wall, but we're not too sure. So, we'll say that our belief on the location of the robot can be described by a Gaussian distribution $N(20,5)$. This distribution can be seen in Figure 2a; the higher the line is, the more we're willing to believe the robot is in that location.

We then take an action (because, remember, a filter is some belief, followed by an action that changes that belief, followed by an observation that changes that belief, repeated). In this case, we'll drive -1m, meaning 1 meter to the left. The mean of our Gaussian should move 1 meter left, correct? And, because we have uncertainty when we drive, our standard deviation should increase. Looking at figure 2b, we can see that's exactly what happened.

We then take a sensor reading, and get back a distance of 5.04. When we consider our sensor model, this means that $p(o = 5.04| x) = N(5.04,2)$. This can be viewed as the red, dotted line in Figure 2c.

What we're interested in is the posterior $p(x|o)$. What we HAVE is $p(o|x)$ and the prior $p(x)$. If we remember Bayes' rule, we merely have to multiply these together to get what we want. Now, a Gaussian times a Gaussian is a Gaussian, and it turns out what we get back is $p(x|o) = N(5.596,1.9611)$, which can be seen as the red, dotted line in Figure 2d. Notice this posterior has done the sensible thing, and has “averaged” the guesses of our prior and our sensor measurement. Our sensor was much more sure of itself than our prior was, though, so its measurement factored in much more than the prior did.

1D Kalman Filter Algorithm

The below Kalman Filter algorithm takes care of the math, taking care of the ugliness of multiplying and adding Gaussians. Note this is built around variances $\sigma^2$, not standard deviations $\sigma$. It is easy to switch
back and forth between the two.

First, our notation:

<table>
<thead>
<tr>
<th><strong>Belief Parameters</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu$</td>
<td>Mean of our belief</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>Standard deviation of our belief</td>
</tr>
</tbody>
</table>

**Motion Model**

| $u$ | Action taken |
| $b$ | Systemic error term in motion model |
| $\sigma_u$ | Standard Deviation of the motion model |

**Sensor Model**

| $o$ | Observation |
| $c$ | Systemic error term in sensor model |
| $\sigma_o$ | Standard Deviation of the sensor model |
\[ \mu = \mu + b \cdot u \] // calculate \( \mu \) after robot’s movement

\[ \sigma^2 = \sigma^2 + \sigma^2_u \] // calculate \( \sigma \) after robot’s movement

\[ k = \frac{\sigma^2 \cdot c}{c^2 \sigma^2 + \sigma_0^2} \] // \( k \) is known as the Kalman gain

\[ \mu = \mu + k(o - c \cdot \mu) \] // calculate \( \mu \) after considering sensor reading

\[ \sigma^2 = (1 - k \cdot c)\sigma^2 \] // calculate new \( \sigma \) after considering sensor reading