Artificial Intelligence for Robotics

Lesson 2: Kalman Filters

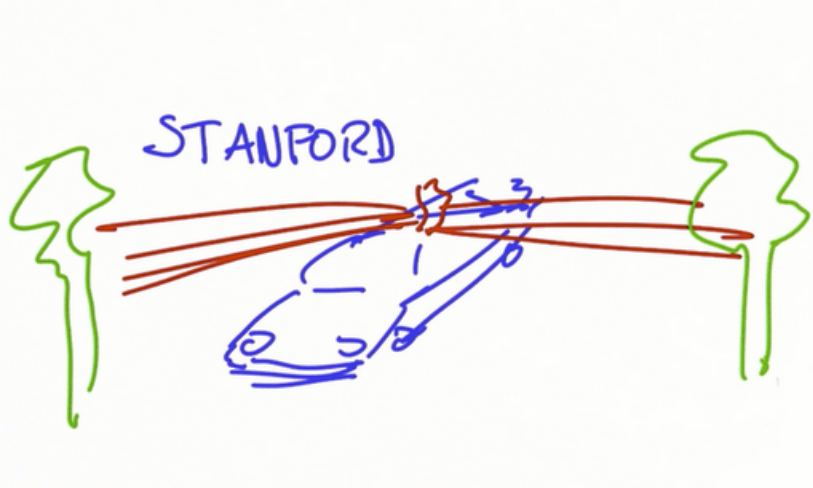
UDACITY

Introduction

Stanford's most recent self-driving car, Junior, is equipped with laser range finders that take distance scans ten times per second — about a million data points. The major function of collecting all of this data is to locate other cars. Data from the range finders provides key input for the Kalman filters, which is the topic of unit two!

Self-Driving Cars

Knowing how to detect other cars will keep our self-driving car from running into other cars. We need to know not just where other cars are - as in the localization case - but also, how fast they are moving in order to avoid collisions.

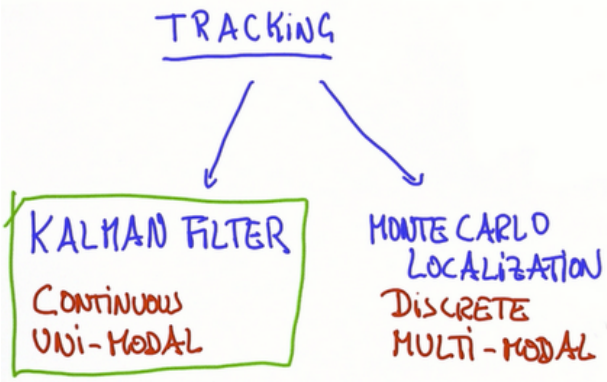


Tracking

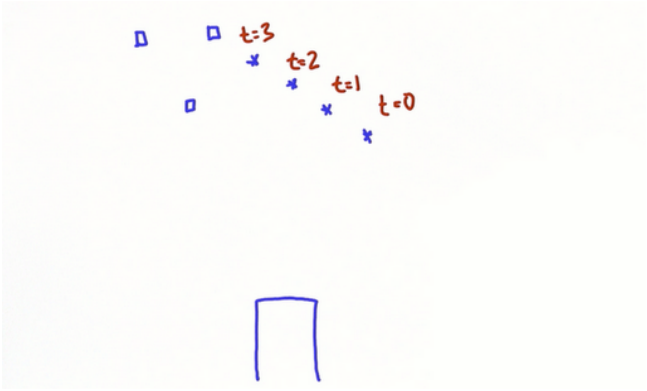
The Kalman filter is a popular technique for estimating the state of a system. Kalman filters estimate a continuous state and gives a uni-modal distribution.

The Monte Carlo Localization method is the method you learned in the first unit, though we did not call it by that name at the time.

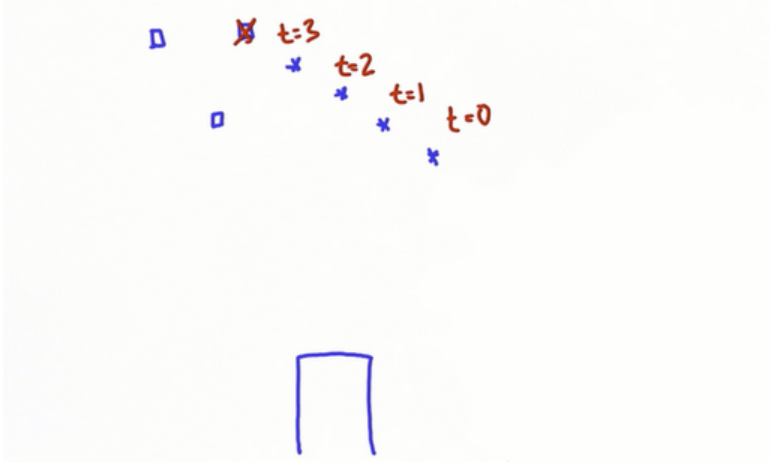
Example:



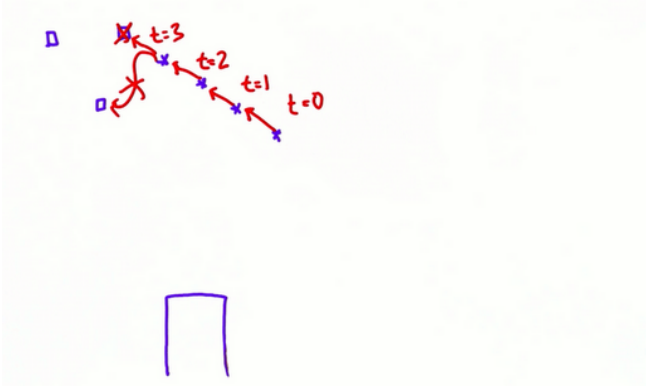
A car (the large rectangle) senses an object in space (*) at times *t=0, t=1, t=2, t=3*.Where would you assume the object would be at *t=4*? Check one of the three boxes.



Solution:



The velocity is changing according to a vector (red arrows). Assuming there is no drastic change in velocity, you can expect *t*=4 to be along the same vector line.

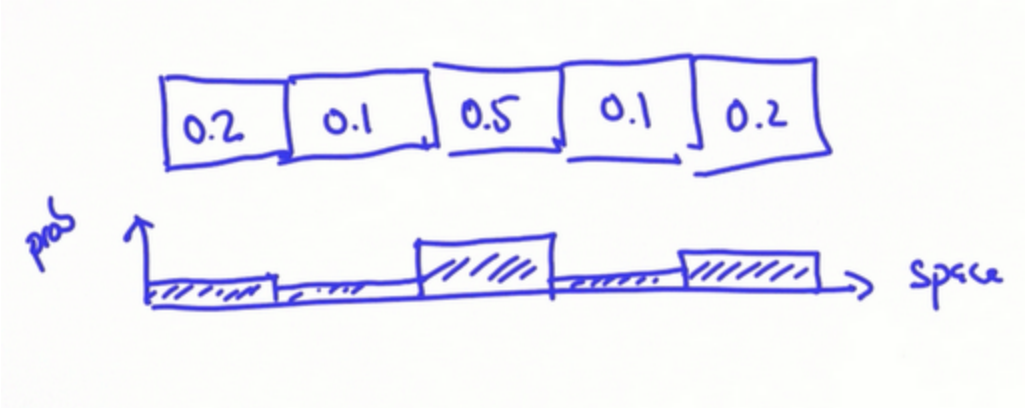


In this lesson we will learn to write a piece of software that enables you take points, even if they are uncertain, like in this example, and estimate where future locations might be. You will also be able to identify the velocity of the object in motion. The Google self-driving car uses methods like these to understand where

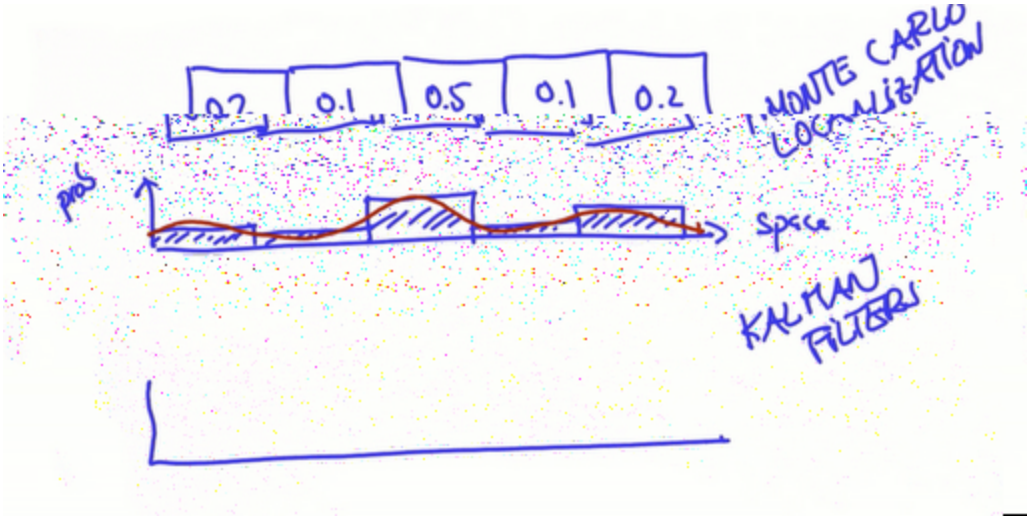
the traffic is based on laser and radar data.

Gaussians

A histogram divides continuous space into discrete regions. It is a way of approximating a continuous distribution.

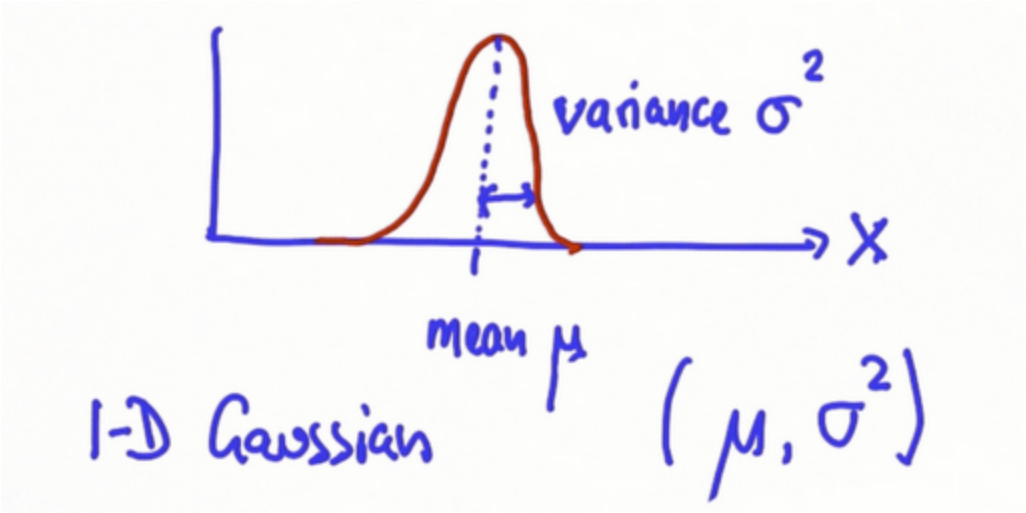


In Kalman filters the distribution is given in what is called a Gaussian, which is a continuous function over the space of locations. The area underneath the Gaussian adds up to one.



If we call the space (x-axis), x, then the Gaussian is characterized by two parameters - the mean (greek letter mu, μ), and the width of the Gaussian, called the variance (often written as a quadratic variable σ^2.

Any Gaussian in a one-dimensional parameter space, is characterized by (μ,σ^2). Our task when using Kalman filter is to maintain a μ and a σ^2, that serves as our best estimate of the location of the objects we are trying to find.

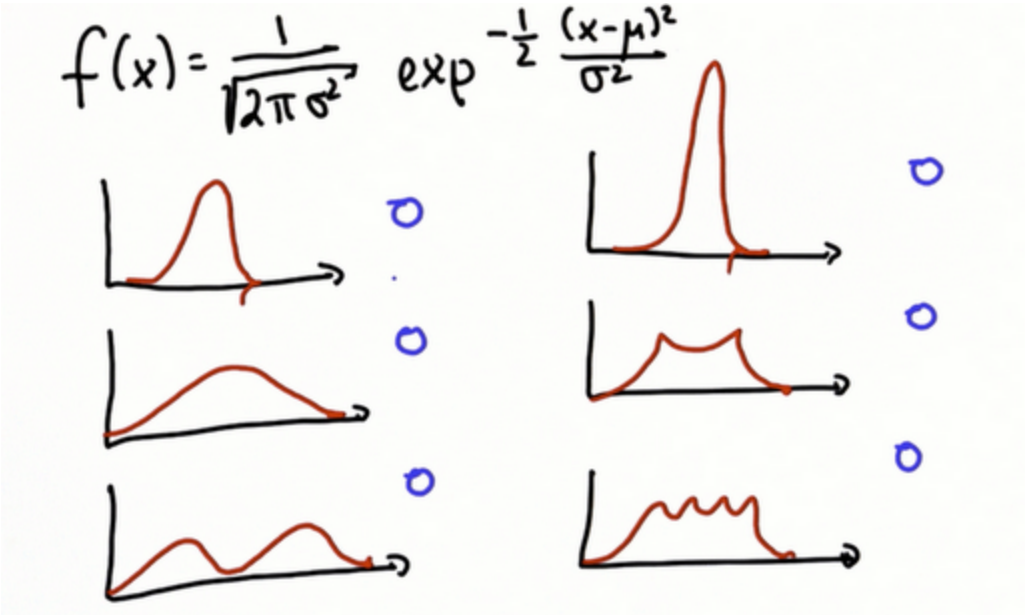


The exact formula is an exponential of a quadratic function:

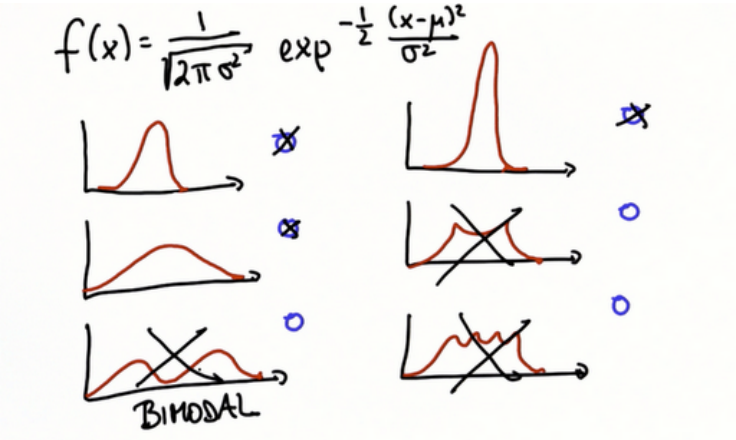
- $f(x) = \frac{1}{\sqrt{2\pi\sigma 2}} exp\ [\ \frac{-1}{2}\ \frac{(x-\mu)2}{\sigma 2}]$

Question 1 (Gaussian Intro):

Check which functions are Gaussians:



Answer: Gaussian Intro

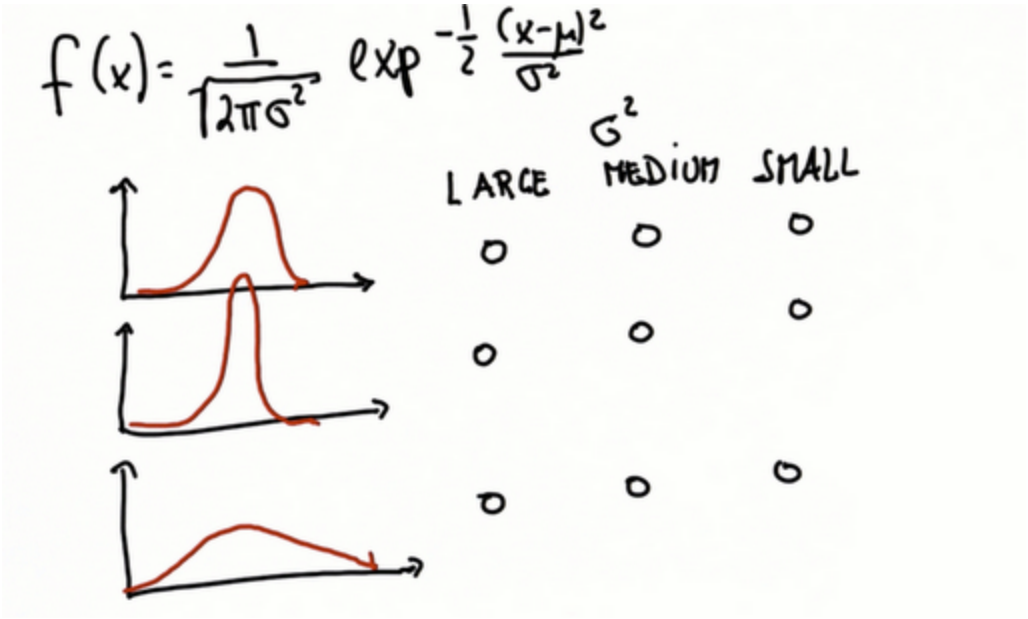


There are three functions that are Guassians. The signature traits of Gaussians illustrated by these three functions are symmetry on both sides with a single peak. The single peak indicates that the function is uni-modal.

What is this word, "unimodal?" Uni means one. Modal refers to the *mode* of the distribution. If you think back to beginning probability, you may remember learning about the mean, median, and mode of a set of data. The mode is the number that shows up most frequently. Stated in the language of probability, it is the most probable value, which shows up as a peak on a probability graph. This is why a *unimodal* distribution is one with only one peak, while a multimodal distribution has many peaks.

Question 2 (Variance Comparison):

Check the box to accurately describe the co-variance of the Gaussians.

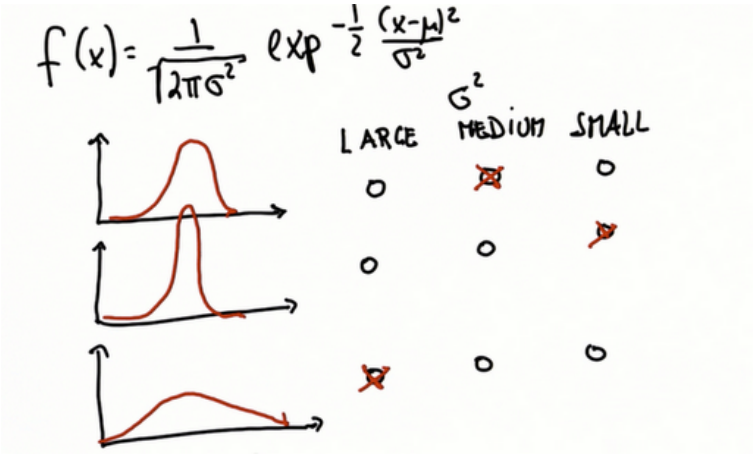


Answer: Variance Comparison

wide spread = large covariance; small spread = small covariance

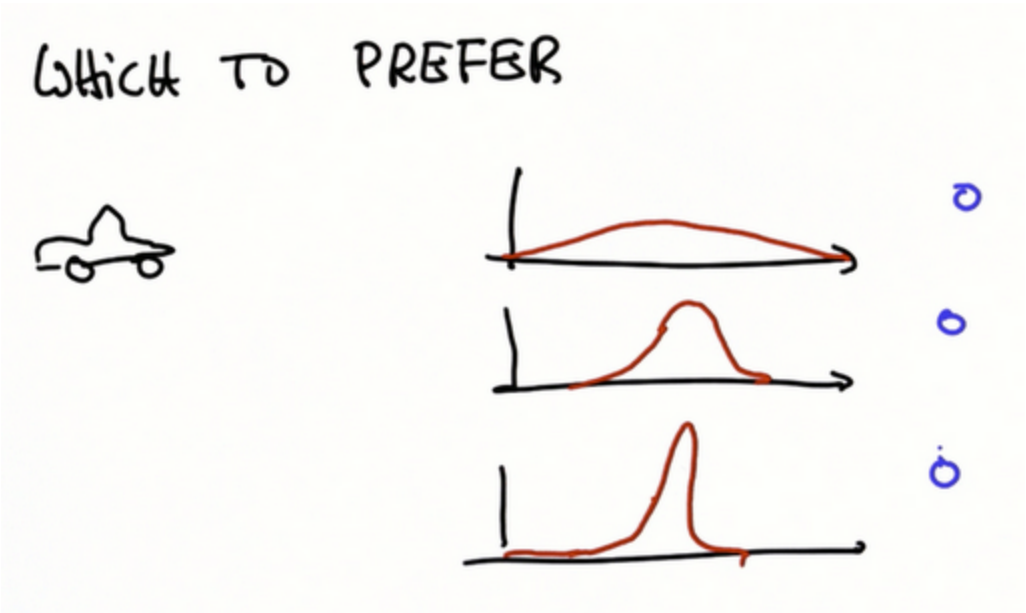
Note that the terms "variance" and "covariance" are being used more-or-less interchangeably here. For now, you can treat them as meaning the same thing.

Covariance is a measure of uncertainty. The larger σ^2, the more uncertain we are about the actual state.



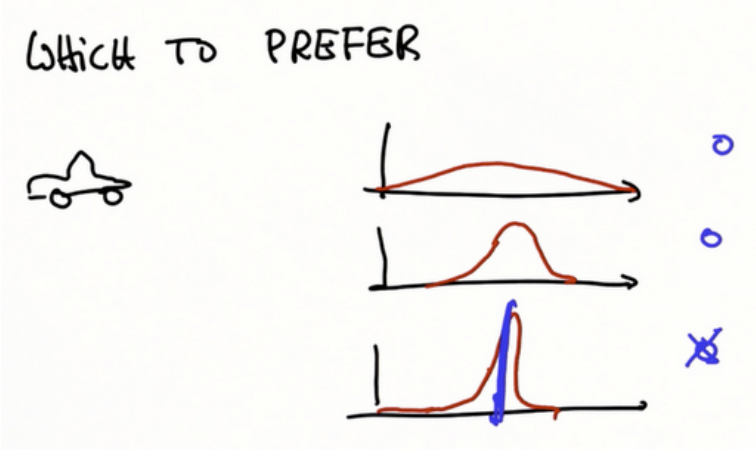
Question 3 (Preferred Gaussian):

Which Gaussian would we prefer to track a self-driving car?



Answer: Preferred Gaussian

The third is most certain, making the chance of hitting another car smaller



Finding the Peak of a Gaussian

Question 4 (Evaluate Gaussian):

Calculate the value of x, given the values (you will need a calculator)

- μ = 10
- $\sigma^2 = 4$
- x = 8

Answer: Evaluate Gaussian

For these values:

- *μ = 10*
- $\sigma^2 = 4$
- *x = 8*

$f(x) = ?$

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\ [\ \frac{-1}{2}\ \frac{(x-\mu)^2}{\sigma^2}]$$

$$f(x) = \frac{1}{\sqrt{2\pi 4}} exp\ [\ \frac{-1}{2}\ \frac{(8-10)^2}{4}]$$

$$f(x) = \frac{1}{\sqrt{8\pi}} exp\ [\ \frac{-1}{2}\ \frac{(-2)^2}{4}]$$

$$f(x) = \frac{1}{\sqrt{8\pi}} exp\ [-0.5]$$

$$f(x) = 0.19947 * 0.60653$$

$$f(x) = 0.120985$$

Programming a Gaussian

Using the values above and starting with the source code below, how would you complete the program so that you return an output of 0.12 — the peak of the Gaussian:

```
from math import *

def f(mu, sigma2, x):
    return #complete this line so that it returns the Gaussian function, given the arguments
print f(10.,4.,8.)
```

Solution:

```
from math import *

def f(mu, sigma2, x):
    #use the formula as we did manually above
    return 1/sqrt(2. * pi * sigma2) * exp(-.5 * (x - mu) ** 2 / sigma2)

print f(10., 4., 8.)
```

Question 5 (Maximize Gaussian):

How do you modify x, which is 8, to get the maximum return value for the function f?

Answer: Maximize Gaussian

The answer is essentially the same answer as mu, so that we get the peak of the Gaussian. So set x to the same value as mu, 10. From the graphs of the Gaussians we have seen so far, we can see that the peak always occurs at the mean. This is why we set x= when we want to maximize.

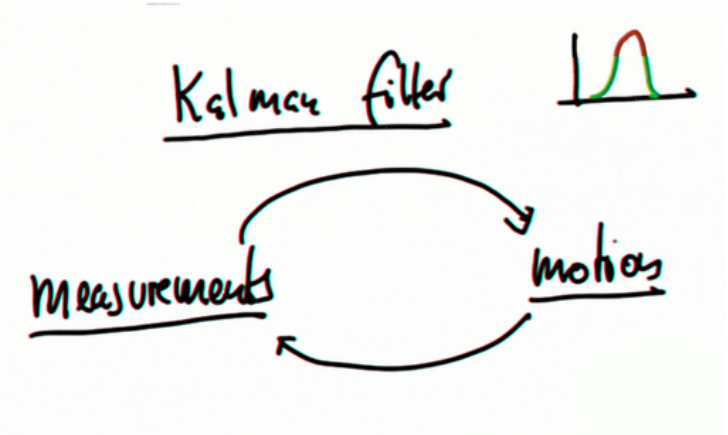
```
from math import *

def f(mu, sigma2, x):
    return 1/sqrt(2. *pi*sigma2) * exp(-.5 * (x-mu) ** 2 / sigma2)

print f(10., 4., 10.)
```

- *0.199471140201*

Measuring Motion

The Kalman Filter represents all distributions of the Gaussians and iterates two things:

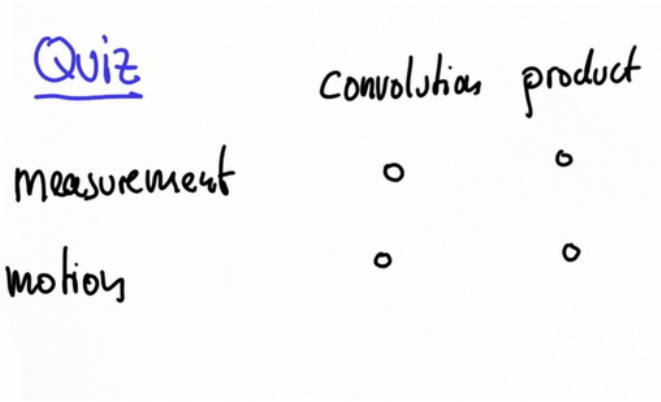
1. Measurement updates
2. Motion updates



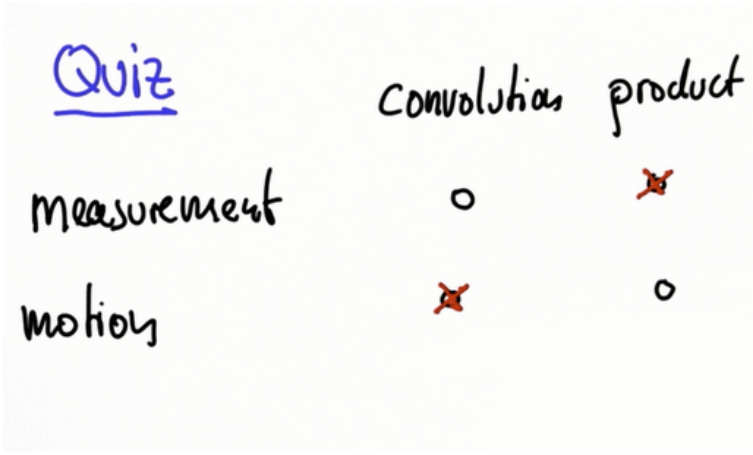
Question 6 (Measurement Motion 1):

Each of the two steps, measurement or motion, requires either a convolution or a product.

Do you remember which requires which? Check the corresponding box:



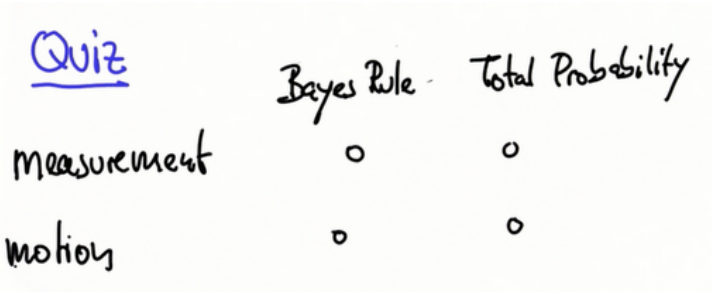
Answer: Measurement Motion-1



Remember from Unit one, that when we made a measurement we multiplied our beliefs by a certain factor (which we called pHit and pMiss earlier). When we made a motion we performed a convolution (which was essentially an addition of probabilities). If you need a refresher, please return to unit one.

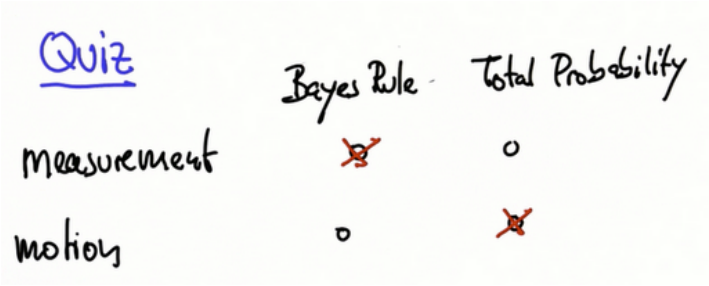
Question 7 (Measurement Motion 2):

Check whether Bayes Rule or Total Probability apply to measurements or motions:



Answer: Measurement Motion-2

In Unit 1 we used Bayes' Rule to calculate the posterior distribution given a measurement Z. We wrote this as P(X|Z). Total probability was used in motion when we calculated the probability of being in each cell after the move by tracking the probabilities from before the move. Please look at Unit One if you need a refresher.



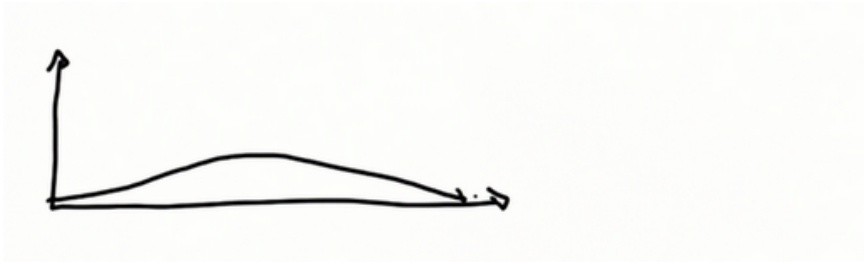
Kalman Filters

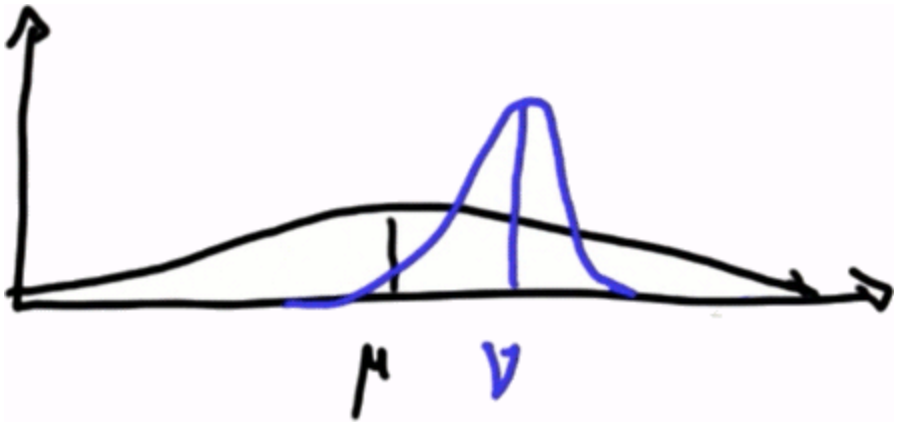
When using Kalman filters, the measurement cycle is usually referred to as a *measurement update*, while the motion cycle is usually called *prediction*. The *measurement update* will use Bayes Rule - a product, multiplication. The *prediction* update will use total probability - a convolution, an addition.

We can talk about the these two cycles using Gaussians.

Suppose you are localizing another vehicle and you have a prior distribution with a mean of mu that looks like:

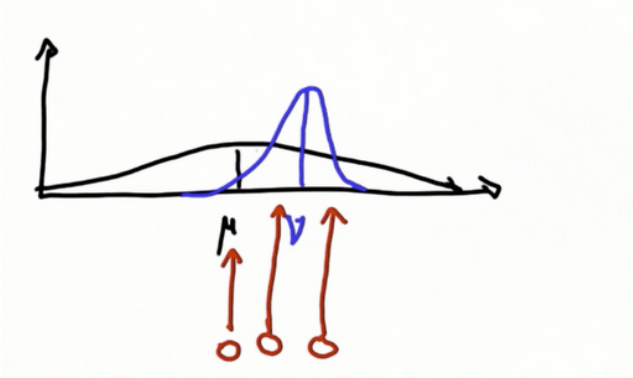
And then we get a measurement that tells us something about the localization of the vehicle, in blue, with a mean nu.





Question 8 (Shifting the Mean):

Where would the new mean of a subsequent Gaussian be? Check the correct location.

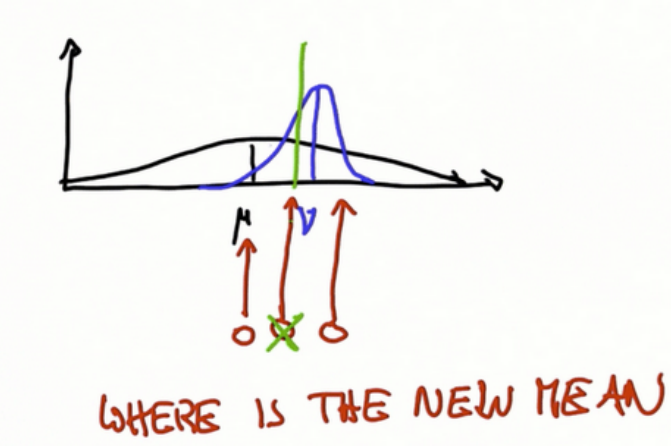


Answer: Shifting the Mean

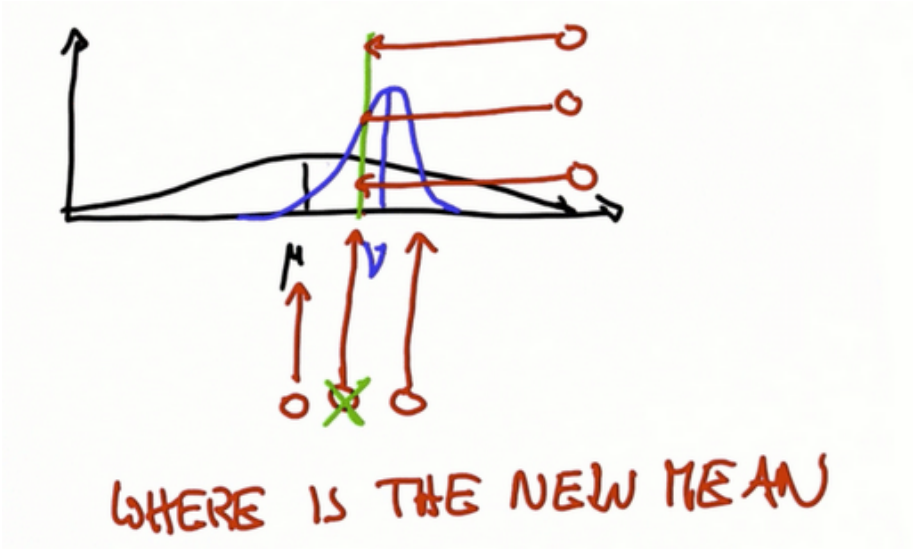
In between the two means, the mean of the prior and the mean of the measurement. It is slightly further on the measurement side because the measurement was more certain as to where the vehicle is, than the prior. The more certain we are, the more we pull the mean in the direction of the certain answer

Question 9 (Predicting the Peak):

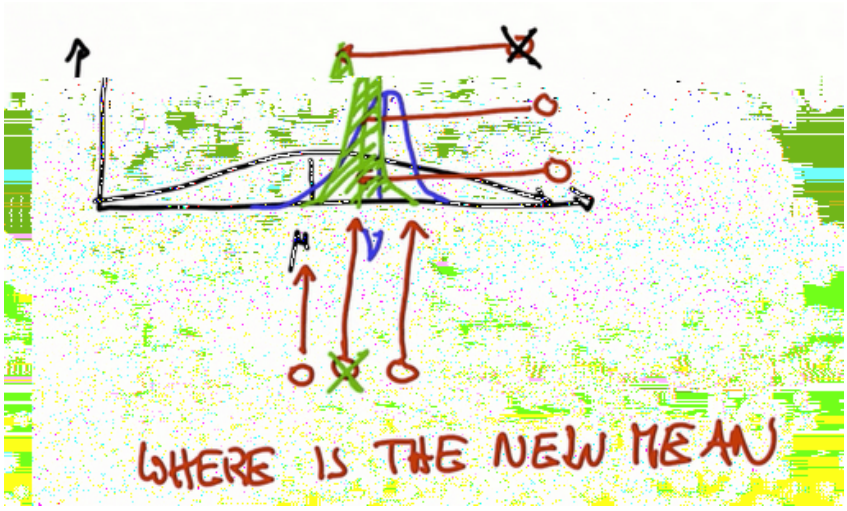
Where is the correct posterior after multiplying the two Gaussians? This question is hard, take your chances.



Answer: Predicting the Peak



The resulting Gaussian is more certain than the two component Gaussians. That is, the covariance is smaller than either of the two covariances in isolation. Intuitively speaking, this is the case because we can actually gain information from the two Gaussians.



This result may not seem very intuitive at first. It helps to think of these two measurements being made simultaneously (perhaps one is from a laser and the other from a radar sensor). Each of these sensors may make a mistake, but by using both sensors we MUST gain more certainty about our position than we would by using one alone (otherwise why would we even use a second sensor?). This means the resulting peak must be higher.

Proof

Warning: this proof is heavy on math! If you really want to understand it, you should work through it on your own, consulting this document when you get stuck.

Step 1: Multiply the Gaussians (remember that measurement always means multiplication)

$$exp(-\frac{1}{2}\,[\frac{(x-\mu)^2}{\sigma^2}+\frac{(x-v)^2}{r}])$$

For our sanity, let's ignore the normalization pieces in front of the exponential. They will not affect the piece of this calculation that we are interested in.

When we make this multiplication, we are left with something that is proportional to:

$$exp(-\frac{1}{2}[\frac{(x-\mu)^2}{\sigma^2} + \frac{(x-v)^2}{r^2}])$$

Note that the expression in brackets is quadratic. Let's solve this expression to find the new mean and covariance.

If we want to find the mean, we want to maximize the entire function (because the peak always occurs at the mean). We do this by minimizing the quadratic expression, which is done by setting the first derivative equal to zero. If you've never taken calculus, take this step for a given, and continue with the proof. When we take the first derivative, we obtain:

$$\frac{d}{dx} = 2\frac{(x-\mu)^2}{\sigma^2} + \frac{x-v}{r^2})$$

We set this equal to zero and solve for x. We find:

$$x = \frac{\mu r^2 + v\sigma^2}{r^2+\sigma^2}$$

This is a beautiful result! Let's talk about it.

First, don't pay too much attention to the denominator. The interesting stuff is going on upstairs. What we have in the numerator is a weighted sum, but notice that the mean of the *first* Gaussian is weighted by variance of the *second* Gaussian. Does this make sense?

It does! If the second distribution has a huge variance, then it isn't a very great measurement and we'd rather emphasize the first distribution. This is exactly what our equation says.

Okay, what about the variance? Let's take advantage of the fact that the second derivative of the quadratic expression in the exponent will be equal to two divided by the variance. Applying this, we find the following:

$$\frac{d^2}{dx^2} = \frac{2}{\sigma^2} + \frac{2}{r^2} = \frac{2}{\sigma^{2'}}$$

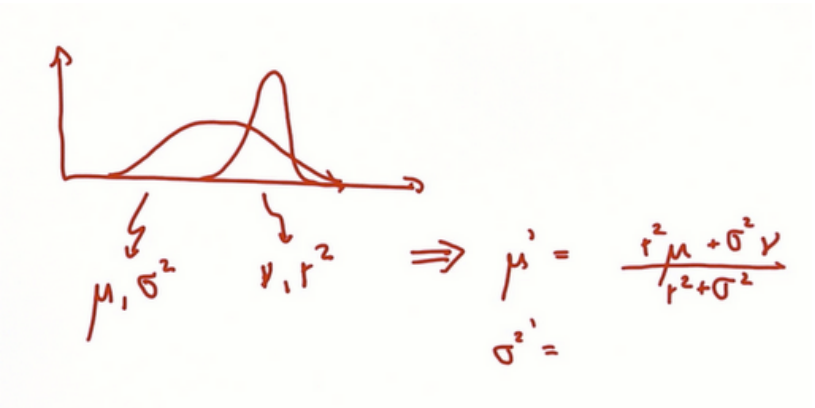
where $\sigma^{2'}$ is our new variance. When we solve this for $\sigma^{2'}$, we find:

$$\sigma^{2'} = \frac{1}{\frac{1}{\sigma^2}+\frac{1}{r^2}} = \frac{\sigma^2 r^2}{\sigma^2+r^2}$$

Note that this variance will always be smaller than either of our initial variances.

Equal Variances

Suppose we multiply two Gaussians, as in Bayes' Rule, a prior μ and a probability σ^2, and the measurement has a mean of v (nu) and a covariance of r^2, so that the new mean, μ' is the weighted sum of the old means, where μ is weighted by r^2, v is weighted by σ^2 and normalized by the sum of the weighting factors.



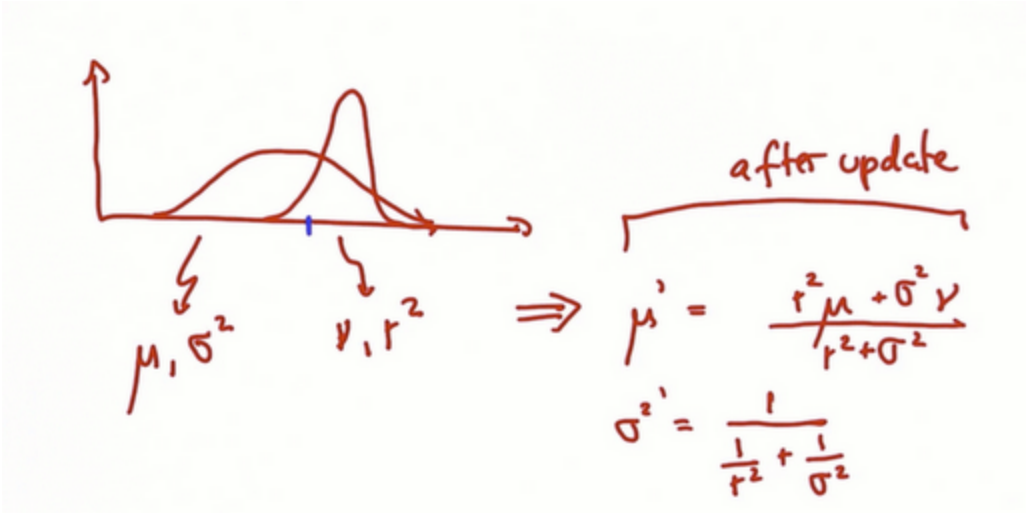
The new variance term after the update is given by the following equation:

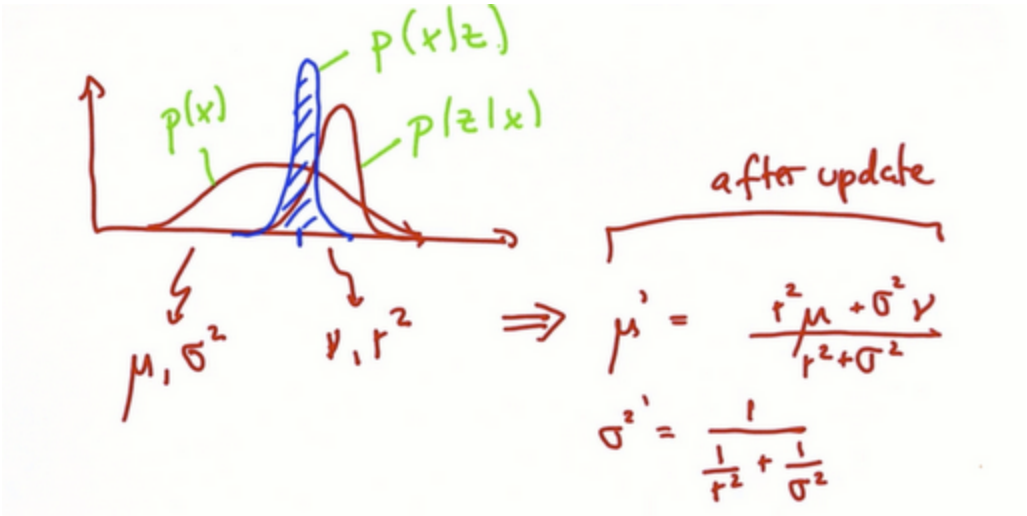
$$\frac{1}{\sigma^{2'}} = \frac{1}{\sigma^2} + \frac{1}{r^2}$$

Pending fix for MoinsMoins markup syntax to work nice with MathJax

We can determine the location of the new mean on the graph because we know that:

1. The prior Gaussian has a much higher uncertainty, a longer, lower slope, therefore σ^2 is larger
2. This means the v is weighted much larger than the μ
3. So the mean will be closer to the v than the μ (blue tick)





Question 10 (Parameter Update):

Using the following Gaussian's and the given equations, compute the new mean and the new σ^2 after the update.

$\mu' = \frac{1}{\sigma^{2+r^2}}[r^2\mu+\sigma^2 y]$

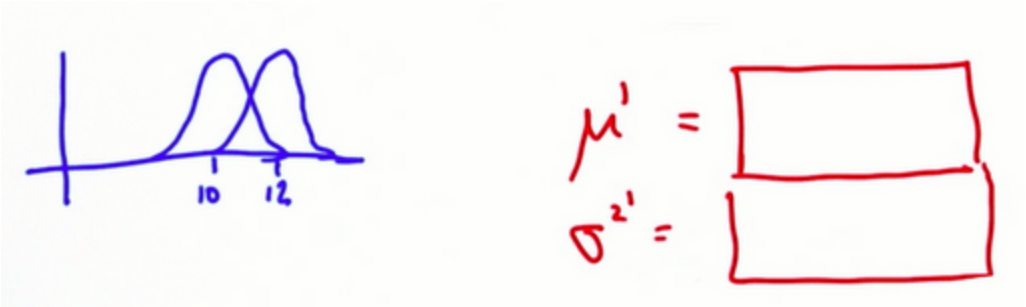
$\sigma^2 = \frac{1}{\frac{1}{\sigma^2}+\frac{1}{r^2}}$

$\mu^2 = 10$

$\nu = 12$

$\sigma^2 = 4$

$r^2 = 4$



Answer: Parameter Update

- $\mu = 11$
- $\sigma^2 = 2$

Unequal Variances

Question 11 (Parameter Update 2):

Once again, using the following Gaussian's and the given equations, compute the new mean and the new σ^2 after the update.

$\mu = \frac{1}{\sigma^{2+r^2}}[r^2\mu+\sigma^2 y]$

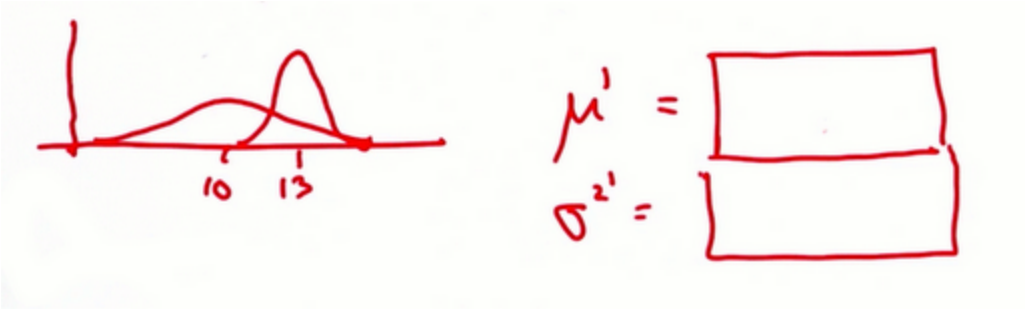
$\sigma^2 = \frac{1}{\frac{1}{\sigma^2}+\frac{1}{r^2}}$

$\mu^2 = 10$

$\nu = 13$

$\sigma^2 = 8$

$r^2 = 2$

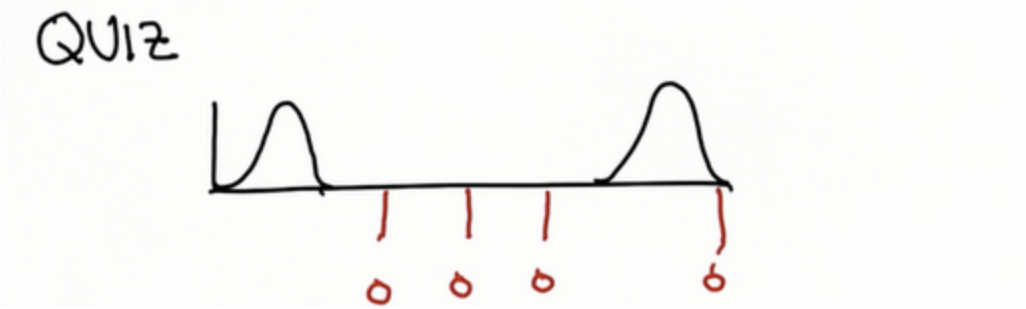


Answer: Parameter Update 2

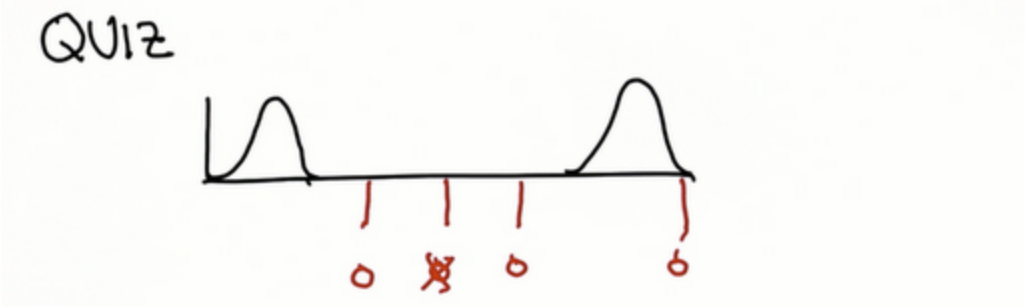
- $\mu = 12.4$
- $\sigma^2 = 1.6$

Question 12 (Separated Gaussians):

Suppose we have a prior and a measurement probability really far away from each other, and both have the same covariance. Where would the new mean be?



Answer: Separated Gaussians



The new mean is in the middle because the covariances are the same.

Question 13 (Separated Gaussians 2):

What will the Gaussian look like?

Answer: Separated Gaussians 2

The most peaky Gaussian, because the new σ^2 is obtained independent of the means. Again, this seems strange, but when we have two sources of information we expect our knowledge to be more precise. This explains the higher peak.

Answer: Gaussian Motion

- *μ=18*
- *σ^2 = 10*

New Mean Variance

Write a Python program that implements the following equations:

$$\mu = \frac{1}{\sigma^{2+r2}}\left[r^{2}\mu+\sigma^{2}y\right]$$

$$\sigma^{2} = \frac{1}{\frac{1}{\sigma^{2}}+\frac{1}{r^{2}}}$$

Here is a skeleton function, which has a function update and takes as its input a mean and a variance for the first distribution and a mean and a variance for the second distribution. The function outputs the new mean and the new variance of the product of the two distributions. Test with the values provided:

```
def update(mean2, var1, mean2, var2):
    new_mean = #enter your code here
    new_var = #enter your code here
    return [new_mean, new_var]

    print update(10., 8., 13., 2.)
```

You should obtain this result → *12.4, 1.6*

Solution:

```
def update(mean2, var1, mean2, var2):
    new_mean = (var2 * mean1 + var1 * mean2) / (var1 + var2)
    new_var = 1/ (1/ var1 + 1/ var2)
    return [new_mean, new_var}

    print update(10., 8., 13., 2.)
```

- *12.4, 1.6*

Run again with equal variances and use 10 and 12 as means:

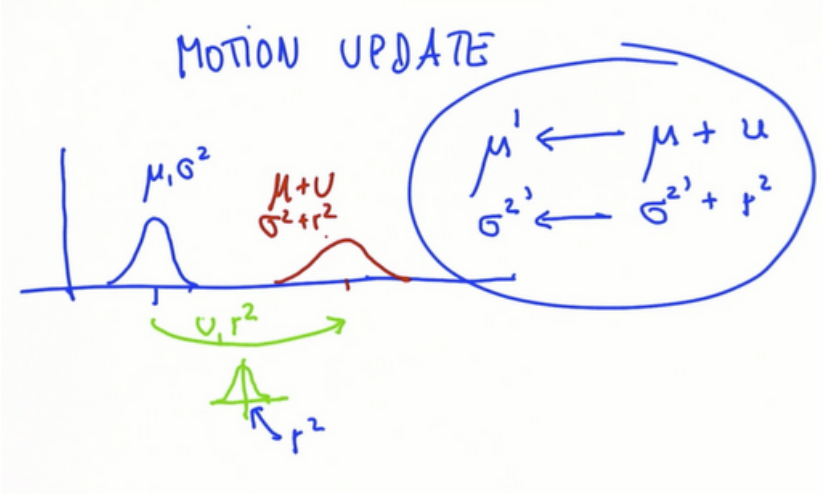
```
def update(mean2, var1, mean2, var2):
    new_mean = (var2 * mean1 + var1 * mean2) / (var1 + var2)
    new_var = 1/ (1/ var1 + 1/ var2)
    return [new_mean, new_var]

    print update(10., 4., 12., 4.)
```

- *11.0, 2.0*

Congratulations! You just programmed an essential update for the Kalman filter.

Motion Update

The graph below shows the best estimate of where you are (the blue tick on x-axis) and your uncertainty. Say you move to the right (green line), and that motion itself has its own set of uncertainty. You then arrive at a prediction that adds the motion command to the mean and has an increased uncertainty compared to the initial uncertainty (red line).



Intuitively, this makes sense, as you move to the right your uncertainty increases because you lose information about your location (as manifested by the uncertainty, green Gaussian).

The math: New mean is the old mean plus the motion, *u*. The new σ^2 is the old σ^2 plus the variance of the motion Gaussian, r^2. This is *very* similar to Unit One, where motion decreased our knowledge and updates increased our knowledge. This is a fundamental aspect of localization.

Question 14 (Gaussian Motion):

Given the following, find the new Gaussian values, μ^2 and σ^2

$\mu = 8$

$\sigma^2 = 4$

Move:

$u = 10$

$r^2 = 6$

$\mu^1 = ?$

$\sigma^2 = ?$

Question 15 (Predict Function):

Here is some skeleton code. Do the predict function to compute the new updated mean and variance, using the values given obtain the result [22.0, 8.0]

```
def update(mean2, var1, mean2, var2):
    new_mean = (var2 * mean1 + var1 * mean2) / (var1 + var2)
    new_var = 1/ (1/ var1 + 1/ var2)
    return [new_mean, new_var]
```

Answer: Predict Function

Just add the two means and the two variances:

```
def update(mean2, var1, mean2, var2):
    new_mean = (var2 * mean1 + var1 * mean2) / (var1 + var2)
    new_var = 1/ (1/ var1 + 1/ var2)
    return [new_mean, new_var]

def predict(mean1, var1, mean2, var2):
    new_mean = mean1 + mean2
    new_var = var1 + var2
    return [new_mean, new_var]

print predict(10., 4., 12., 4.)
```

- *22.0, 8.0*

The entire program above, implements a one-dimensional Kalman filter.

Kalman Filter Code

Write a main program that takes the two functions, update and predict, and feeds them into a sequence of measurements and motions.

Set the initial measurement, μ to 0, with a very large uncertainty of 10,000.

Assume the measurement uncertainty, measurement_sig, is constant at 4, and the motion uncertainty, *motion_sig,* is 2.

```
def predict (mean1, var1, mean2, var2):
    new_mean = mean1 + mean2
    new_var = car1 + var2
    return [new_mean, new_var]

    measurements = [5., 6., 7., 9., 10.]
    motion = [1., 1., 2., 1., 1.]
    measurement_sig = 4.
    motion_sig = 2.
    mu = 0.
    sig = 10000.
```

You should get these outputs:

update: [4.99, 3.99] predict: [5.99, 5.99] update: [5.99, 2.39] predict: [6.99, 4.399] update: [6.99, 2.09] predict: [8.99, 4.09] update: [8.99, 2.02] predict: [9.99, 4.02] update: [9.99, 2.00] predict: [10.99, 4.00]

Below is our Kalman Filter code:

```
for n in range(len(measurements)):
    [mu, sig] = update(mu, sig, measurements[n], measurement_sig)

    print 'update: ', [mu, sig]
    [mu, sig] = predict(mu, sig, motion[n], motion_sig)

    print 'predict: ', [mu, sig]
```

It is very important that we understand what we are doing here, so let's go through it together:

This code says "for each measurement that we have, repeat two steps." Those two steps are the update and the prediction step. In the update step, we incorporate the new measurement into our previous belief to generate a new belief (which is more certain than our previous belief). In the prediction step, we use our inferred velocity to once again create a new belief (which is less certain than our previous belief).

This is what makes Kalman filters really amazing: they never have to worry about more than two things! In the case of the update step, those two things are the previous belief and the measurement. With prediction they are the previous belief and the inferred velocity. This keeps the computational power required of a Kalman filter relatively low and allows it to run in real time.

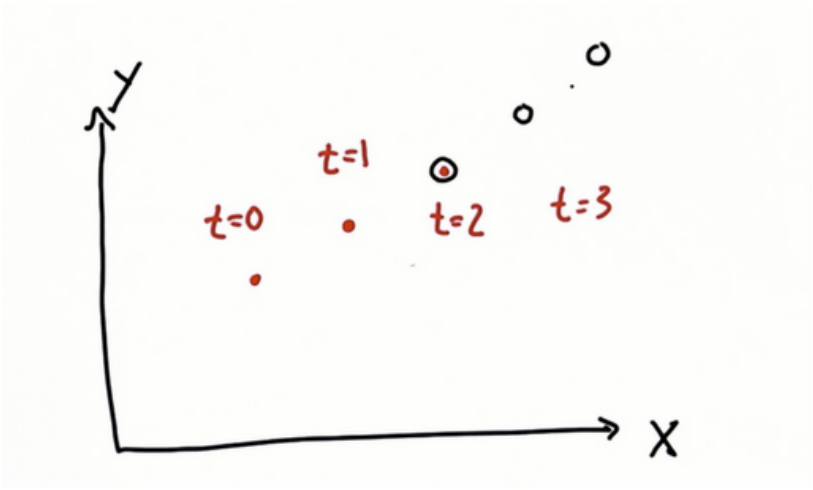
Kalman Filter with Many Dimensions

Example:

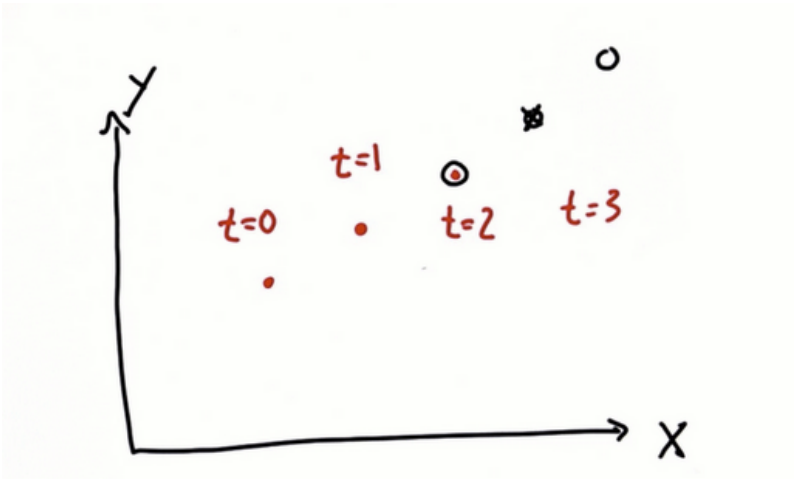
Suppose you have a two dimensional state space, like a camera image. Suppose at time *t*=0, you observe the object of interest to be at a specific coordinate. One time stamp later you see the object in another spot.

Question 16 (Kalman Prediction):

Where would you expect the object to be at *t*=3? Fill in the most likely location.



Answer: Kalman Prediction



In multi-dimensional spaces (like the real world, for example!) the Kalman filter not only allows you to estimate your positions, but also to infer your velocity from these position measurements. These inferred velocities then allow you to predict your future position.

Notice that the sensor itself only sees position, not the actual velocity, the velocity is inferred from seeing multiple positions.

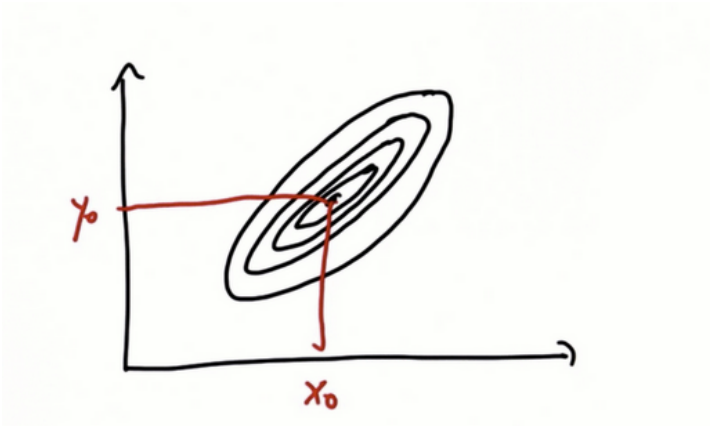
By being able to figure out the velocity of an object the Kalman filter can make predictions about future locations that incorporate velocity. This is why Kalman filters are so popular in Artificial Intelligence.

Multivariate Gaussians

So far we have discussed one dimensional motion, and though this captures all the essentials of a Kalman filter, we should at least briefly discuss what happens when we go from one dimension to higher dimensions.

1. Instead of being a number, the mean becomes a vector with one element for each of the dimensions.
2. The variance is replaced by the covariance, which is a matrix with d rows and d columns when the space has d dimensions.
3. This gives us a complicated formula, but it is not something you need to remember.

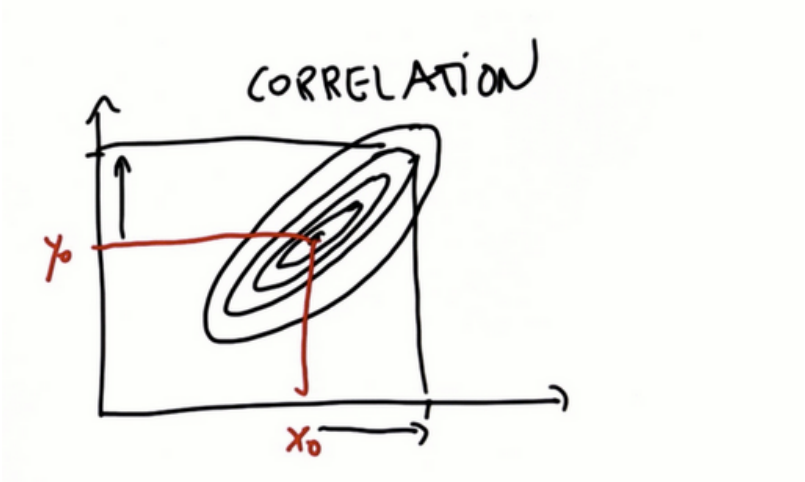
Here is an illustration of multivariate Gaussians. In this image the mean is indicated by x_0 and y_0 and the covariance is the contour lines:



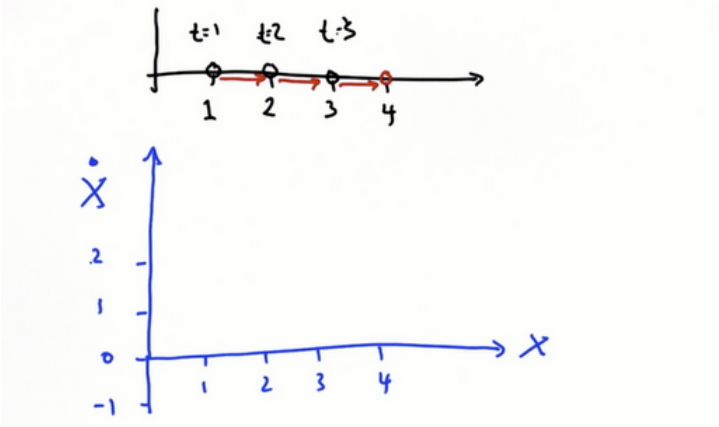
Gaussians with a smaller amount of uncertainty would have smaller contour lines:

It is also possible to have a small uncertainty in one dimension and a huge uncertainty in another:

When the Gaussian is tilted it means the x and y are correlated:



Build a two-dimensional estimate, where the x-axis is the location and the y-axis is the velocity, which we will denote as v, though keep in mind that Sebastian uses an x with a dot over it. (An x with a single dot over it represents the first derivative of x (position) with respect to t (time), which is another way to express velocity).



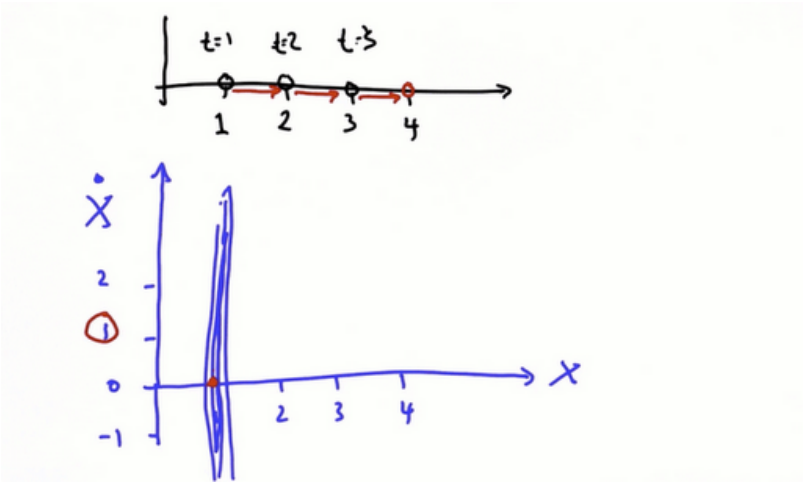
If, initially, you know location but not velocity, you can represent it with an elongated Gaussian around the correct location, but really broad in the space of velocities.

In the prediction step, since you do not know the velocity, you cannot predict

what location you are going to assume. However, there are some interesting correlations.

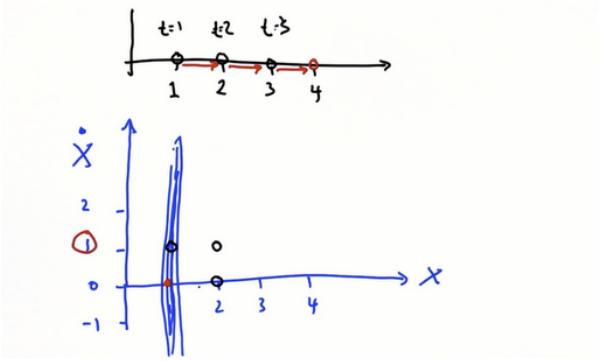
Assume the velocity is 0, where would the posterior be after the prediction?

If you know you start at location 1, then a velocity of zero would leave you in the exact same place, (1, 0).

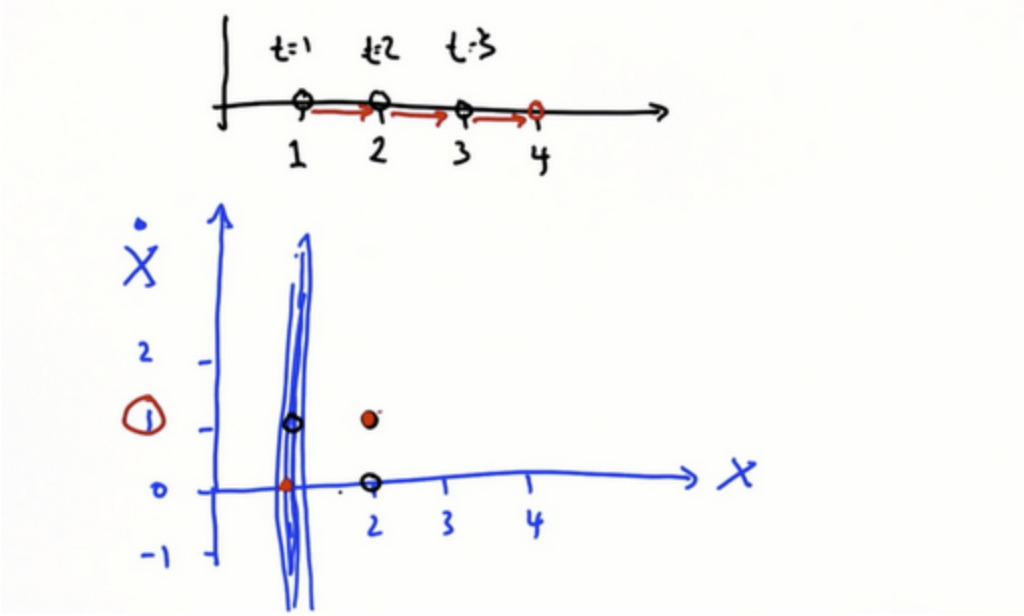


Question 17 (Kalman Filter Prediction):

Using the same graph, assume the velocity is one, where would the prediction be one timestamp later, starting at location one (1, 0), and velocity one? Select the one that makes the most sense.

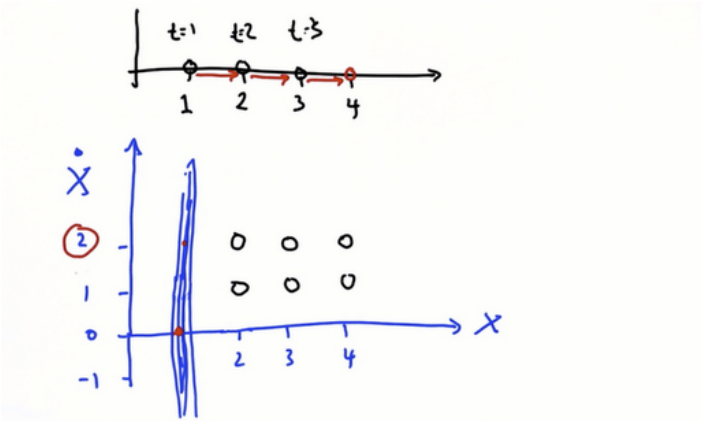


Answer: Kalman Filter Prediction

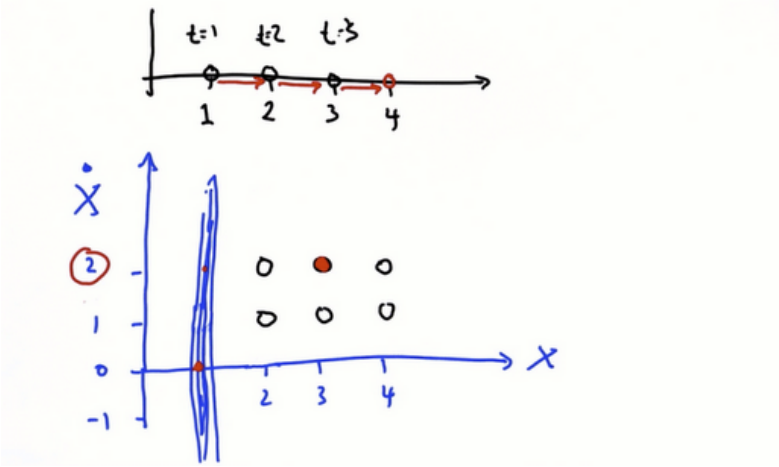


Question 18 (Another Prediction):

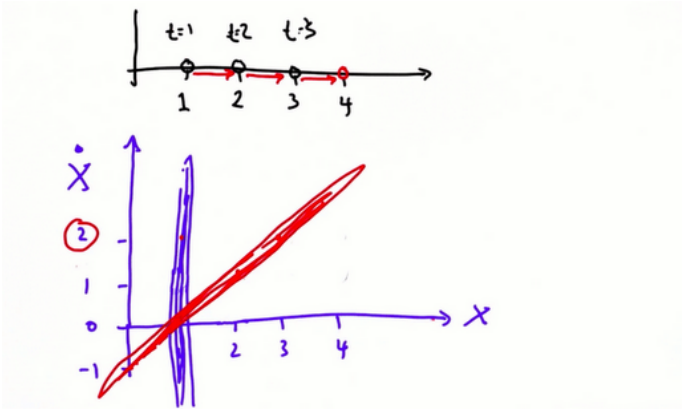
Consider a velocity of two with the same givens as before. Where would you expect the most plausible prediction to be?



Answer: Another Prediction



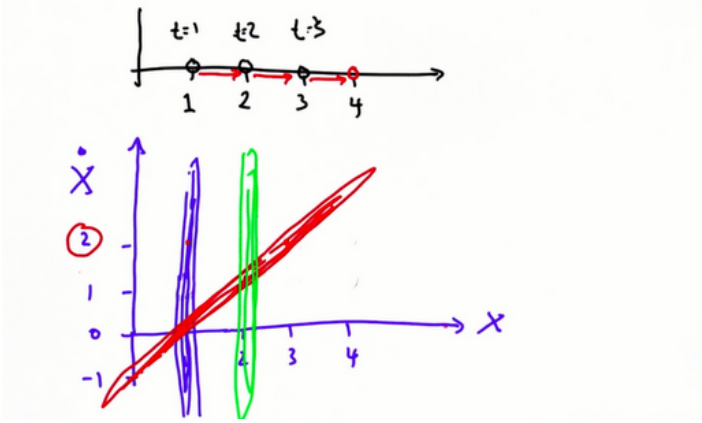
Keys Kalman Filters



When you find multiple data points, you find that all of the possibilities on the one-dimensional Gaussian (the blue one), link to a two-dimensional Gaussian (the red one).

Fold in the second observation t=2, which says nothing about velocity, but only about the location.

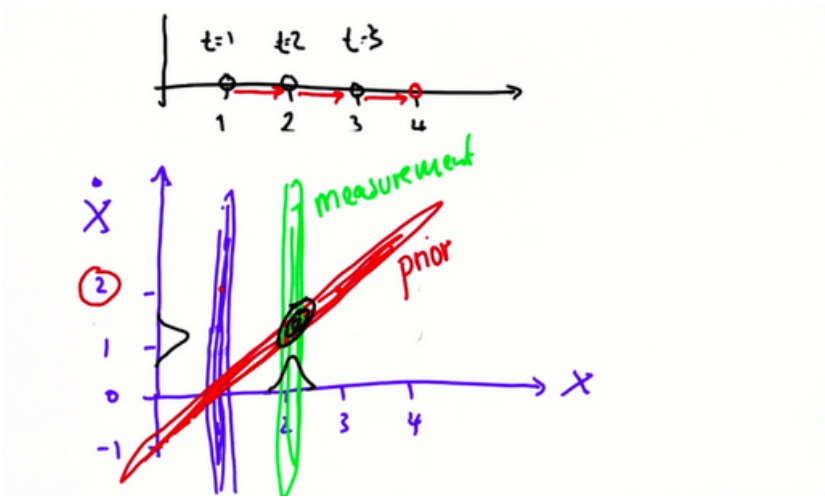
Multiply the prior (the red Gaussian) and the measurement (the green Gaussian) to obtain a really good estimate of an object's velocity and location. We aren't going to cover the math involved in multiplying together Gaussians of multiple dimensions, but you can try it on your own if you'd like or consult the supplementary material for more information on the subject. When we do the math, we find that we get a peakier distribution in both the x and y directions. How convenient! We now know our position and velocity with pretty high certainty!



So far we have only been able to observe one variable, and use that observation to infer another variable using a set of equations that

say that the location after a time-step, is the old location plus the velocity.

Our prediction for our new position, x', would then just be the sum of our previous location belief, x, and our motion, u.



$x'=x+u$

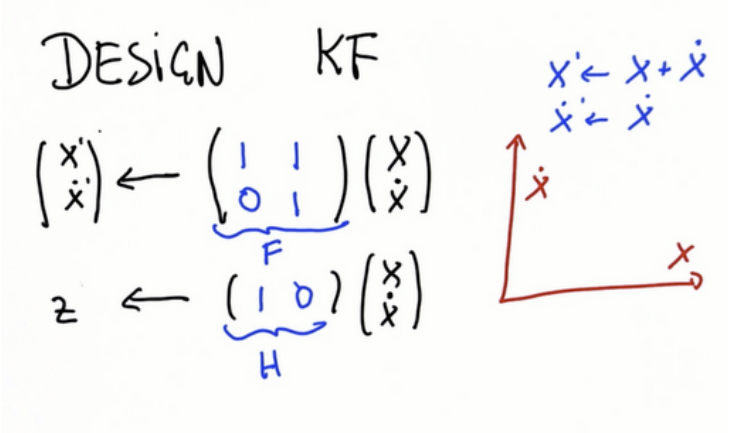
Variables of a Kalman filter are often called "States," because they reflect states of the physical world, like where an object is and how fast it is moving. Within Kalman filters there are two sub-sets:

1. Observables - momentary location
2. Hidden - in our example this was the velocity. We could not measure it directly, only infer it from our position measurements.

When these two sub-sets interact, subsequent observations of the observables give us information about the hidden variables, so that we can make an estimate of what the hidden variable are.

From multiple observations of the places of the object's location, we can discover how fast it is moving. Other filters can generate this information, but the Kalman filter is the most efficient.

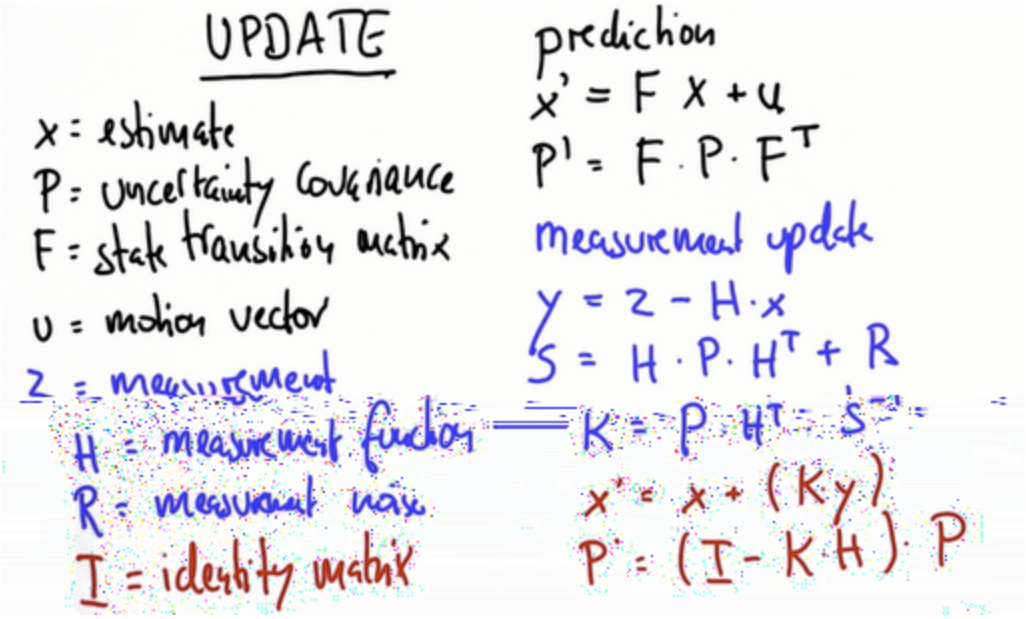
Kalman Filter Design



When designing a Kalman Filter, you effectively need two things, a state transition function and a measurement function, both of which are represented as matrices. We call the state transition matrix F and the measurement matrix H.

If you are a little rusty with linear algebra, feel free to check out the videos from Khan Academy or MIT Open Courseware to refresh your understanding. The MIT OCW link may be more helpful if you have a little bit of math background, while Khan Academy is better if you have never seen a matrix before.

We separate our Kalman filter into two steps: the update step and the prediction step. The equations for these steps and the names of the matrices and variables are given below. You might also want to check out the excellent write-up by one of students of the first iterations of CS373 class Kalman Filter Matrices.



This is just a generalization of the simple one-dimensional case that we have been talking about. If you would like to learn more about the details of a real Kalman filter, I recommend you check out the course notes from MIT's class on identification, estimation, and learning. Notes 5-8 are relevant to this discussion.

Kalman Matrices

Here we have a challenging programming problem. You are going to write a multidimensional Kalman filter! You can start by using the matrix class implemented by Professor Thrun . This class can perform all of the matrix manipulations that we need to utilize the formulas that we discussed in the previous question.

You can find a small version of this in libraries. Make a matrix with a command like this:

```
a = matrix([[10.],[10]])
```

The argument in parentheses is a two dimensional matrix and in this case it is a vertical vector. You can print out the result of the vertical vector with the show command:

```
a = matrix([[10.],[10]])
a.show()
```

You can compute the transpose to find the horizontal vector:

```
a = matrix([[10.],[10]])
b = a.transpose()
a.show()
```

If you want to multiply a matrix by a vector:

```
a = matrix([[10.],[10]])
F = matrix([[12., 8.], [6.,2.]])
b = a.transpose()
F.show()
```

You can also multiply F and a, and set that equal to b, to show the result of the vector:

```
a = matrix([[10.],[10]])
F = matrix([[12., 8.],[6.,2.]])
b = F * a
F.show()
```

[200.0] 80.0]

Here is what a matrix library looks like:

```
measurements = [1,2,3]

x = matrix([[0.], [0.]]) # initial state (location and velocity)
p = matrix([[1000., 0.], [0., 1000.]]) # initial uncertainty
u = matrix([[0.], [0.]]) # external motion
F = matrix([[1., 1], [0,1.]]) # next state function
H = matrix([[1., 0]]) #measurement function
R = matrix([[1.]]) # measurement uncertainty
I = matrix([[1., 0.], [0., 1.]]) # identity matrix

filter(x, P)
```

When you run the filter with the given measurements, you can estimate the velocity to make better predictions.

When you run the filter you want to return the measurement update first and then the motion update.

Fill in this empty procedure to print out the resulting estimates for *x* and *P*:

```
def filter(x, P):
    for n in range(len(measurements)):

    #measurement update

    #prediction

    print 'x= '
    x.show()
    print 'P= '
    P.show()
```

Once you fill it in you get an output. This output iteratively displays *x* and *P* as they update. Notice that at the first output of *x,* we know nothing about the velocity. This is because we need multiple location measurements to estimate velocity. As we continue making measurements and motions, our estimate converges to the correct position and velocity.

Question 19 (Kalman Matrices):

Write the algorithm filter(x, p) that outputs the exact values, by implementing the given equations and familiarizing yourself with the matrix class. Fill in the value in accordance to the things that were shown for the multivariate Kalman filter in the previous video.

Answer: Kalman Matrices

Here's the code Sebastian wrote:

```
Z = matrix(measurements[n])
y = Z - (H * x)
S = H * P * H.transpose() + R
K = P * H.transpose() * S.inverse()
x = x + (K * y)

P = (I - (K * H)) * P

# prediction
x = (F * x) + u
P = F * P * F.transpose()
```

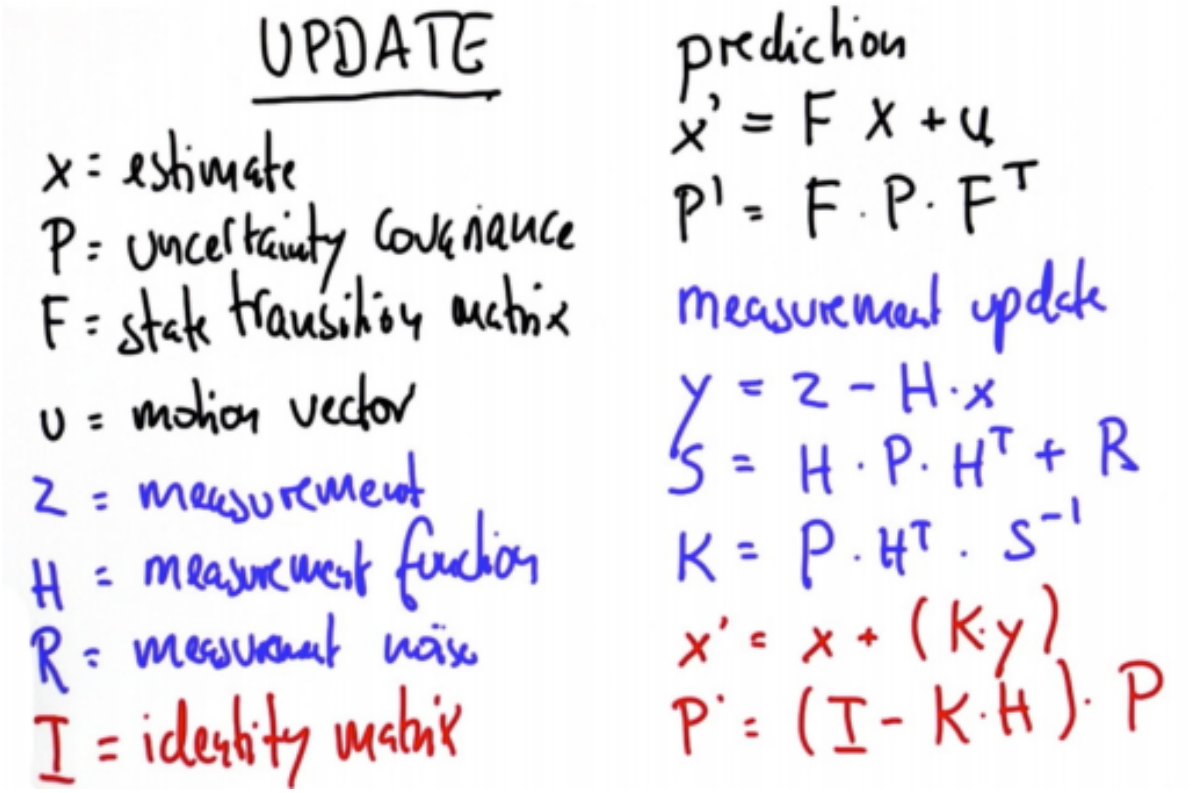
This code is not trivial! It is the core of the Google self-driving car's ability to track other vehicles. This is the line by line implementation of what you've seen before, the measurement update and the prediction.

First, a measurement update of measurement n. The error calculation *(y = Z - (H * x))*, the matrix *S*, the Kalman gain *K* with the inverse, and then back to the next prediction, *x* and the

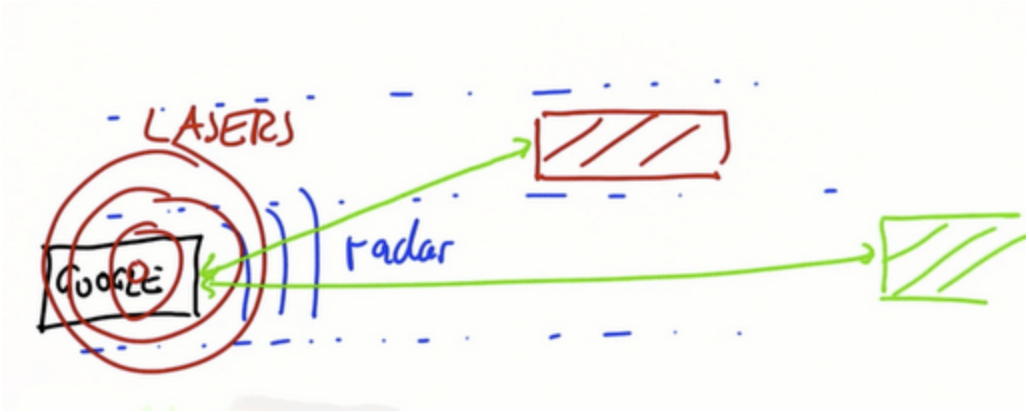
measurement update, *P*. Finally, you have the prediction step which implements the equations you have learned.

If you were able to write this program, which is really, really involved, you now understand Kalman filters and you have implemented a multi-dimensional Kalman filter all on your own using the fairly mechanical matrix class that was written for you.

Running this program returns a prediction and a velocity update. These are the equations you just implemented:



For the Google self-driving car this is the essential model for how it makes predictions about where other cars are and how fast they are going. The car uses radar on the front bumper that measures the distance to other vehicles and also gives a noisy estimate of the velocity. Additionally, the car uses its lasers, which are located on the roof, to measure the distance to other cars and gets a noisy estimate of the velocity.



So, when the Google car is on the road it uses radars and lasers to estimate the distance and the velocity of other vehicles on the road using a Kalman filter, where it feeds in range data from the laser and uses state spaces of the relative distance, *x* and *y*, and the relative velocity of *x* and *y*.

CONGRATULATIONS, You've finished Unit 2!!!