**Unscented Kalman Filter Notes**

**Intro**

Another technique called the UKF can achieve better results than the Extended Kalman Filter. It handles non-linear non-linear process/measurement models. An Unscented Kalman Filter uses something called Sigma points to approximate the probability distribution of the non-linear model instead of linearizing the model.

It has the advantages of sigma points approximating the non-linear model better than Linearization does, also it is not necessary to calculate a Jacobean anymore, making processing time faster.

**Motion Models**

In the Extended Kalman Filter, we used a **Constant Velocity model**, which is one of the most basic motion models used in object tracking.

There are however many different motion models:

* Constant Turn Rate and Velocity Magnitude Model (CTRV)
* Constant Turn Rate and Acceleration (CTRA)
* Constant Steering Angle and Velocity (CSAV)
* Constant Curvature and Acceleration (CCA)

Each model makes different assumption about the objects motion, so the application determines which one we should use.

Constant Velocity Motion Model has the limitation that it cannot predict a vehicles motion correctly when it comes to turns. A constant velocity model would draw a line tangential to the turn, meaning the next prediction of position would always be on the outside of the turn.

We can change the model to ensure that we don’t assume that the object is always going straight.

We can use **CTRV Model** to ensure we can assume the turn rate of a vehicle is constant, making our accuracy better when predicting the position of a turning vehicle.

With the change to the CTRV model, the states we need to track change. Our new state vector will be the position (px, py), as well as the speed (magnitude), and the yaw angle (psi). Because we also want the ability to estimate psi\_dot, we add it to the state vector too.

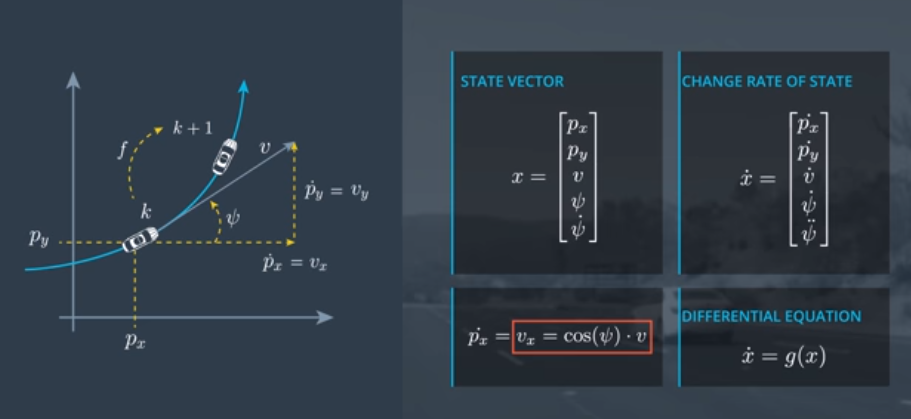


Note: When a car is going on a straight path, the psi\_dot is zero. When thinking about turning radius, it is the combination of both the turning angle, and the speed. If two vehicles have the same turning angle, the slower vehicle will have the smaller turning radius (make a complete circle in less distance).

**Change in Process Model**

The F function should transition us from time step k to k+1.

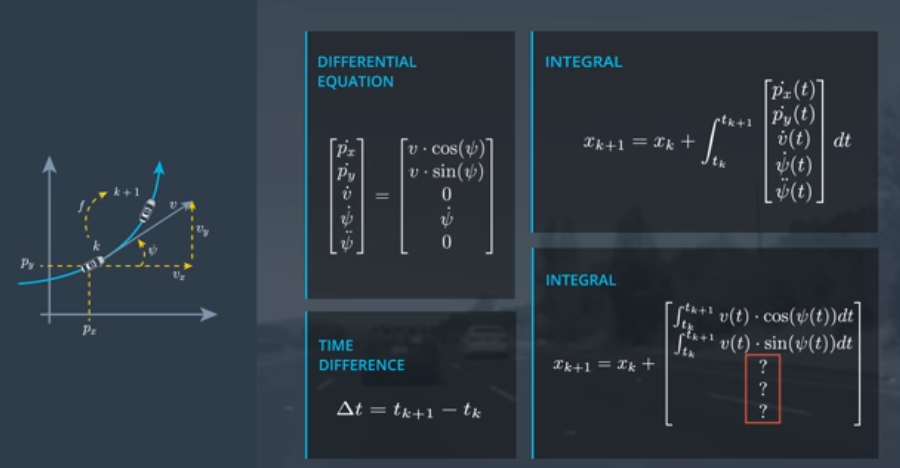
A technique we can use when trying to derive the equation that can take us from one state to the prediction of the state at the next time step is to look at the change rate of the state vector (derivative of the current state).



Py = v\*sin(phi) v\_dot = 0 (CTRV model) phi\_dot = phi\_dot (was already in prev state) phi\_dot\_dot = 0

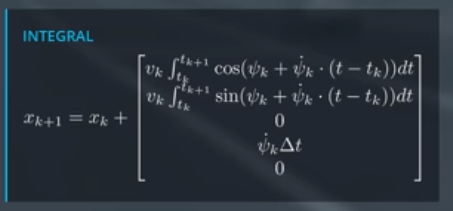
Now that we have the complete differential equation, we need to get from time step k to k+1. Looking at our time values as not only discrete steps but as continuous values, we can get from k to k+1 by using integrating our differential equation over the time difference.

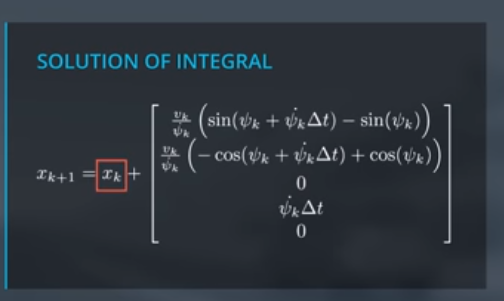
With the case of the CRTV, the integral can be solved pretty simply.



Answer: With the CTRV model, the last 3 are 0, phi\_dot \* delta\_t, and 0. This will make our assumption of the next time step follow the CTRV model.

Assuming the CTRV model we can solve the integral to be:





When looking at the solution, we see that there is an issue with the yaw rate being divided. It can lead to an issue when the yaw rate is 0. For this special case, we need to derive the process model again with the assumption that the yaw rate is 0. Or we can understand that because the yaw rate is zero, we are moving in a straight line and make our model with that knowledge.

With the yaw rate of zero, px = vk\*cos(phi\_k)\*delta\_t, py = vk\*sin(phi\_k)\*delta\_t.

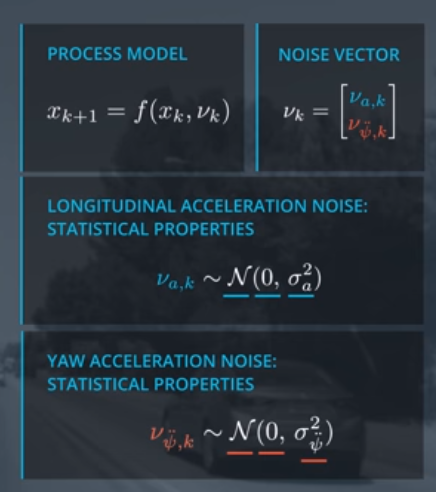
**CTRV Process Noise Vector**

The uncertainty can be described as a two-dimensional vector of two independent scalar noise processes.

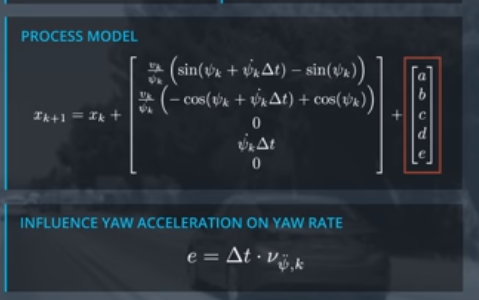
First Noise process is the Longitudinal Acceleration Noise, and the Second is Yaw Acceleration Noise.

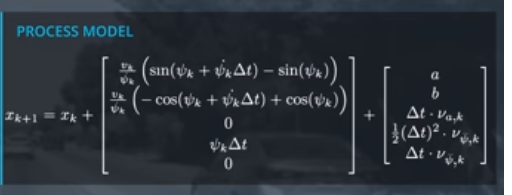
**Longitudinal Acceleration Noise:** Influences the Longitudinal Speed of the vehicle and randomly changes its value at every time step k. It is a normal distribution with 0 mean and a variance sigma\_A squared.

**Yaw Acceleartion Noise:** Also normal distribution with mean of 0 and variance of sigma\_phi\_dot\_dot squared.



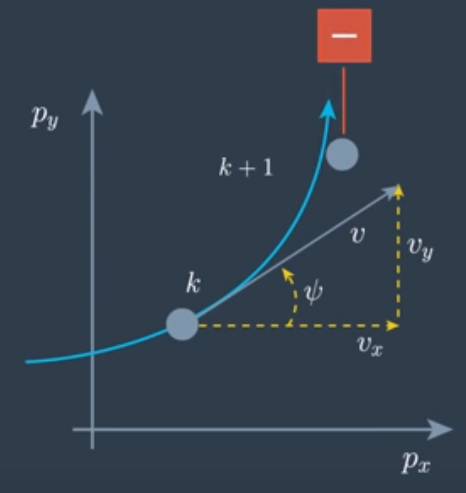
Under the CTRV model, and the assumption that the process noise from one time step to the next is the same, we can determine that the errors are:





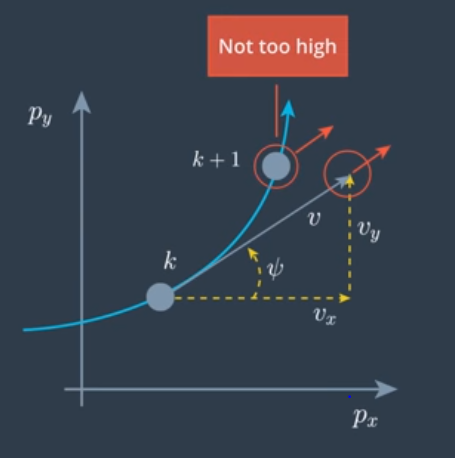
Process noise for position:

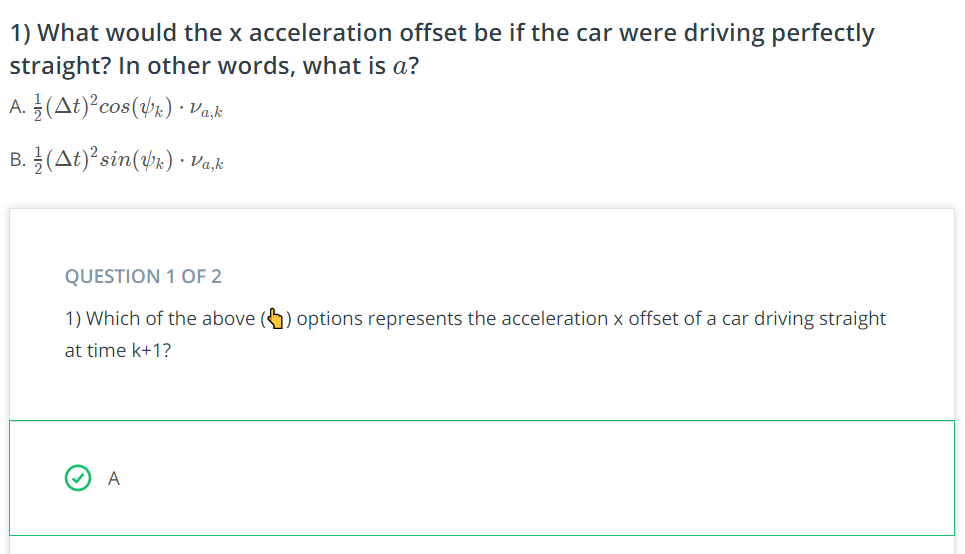
When thinking about a vehicles position as it is turning from one time step to the next, we can see that the yaw rate (phi\_dot) does effect the next position. The yaw rate effects the position, but for our application we can assume that because it has little effect overall on the position of a vehicle from one time step to the next, we can just ignore it.

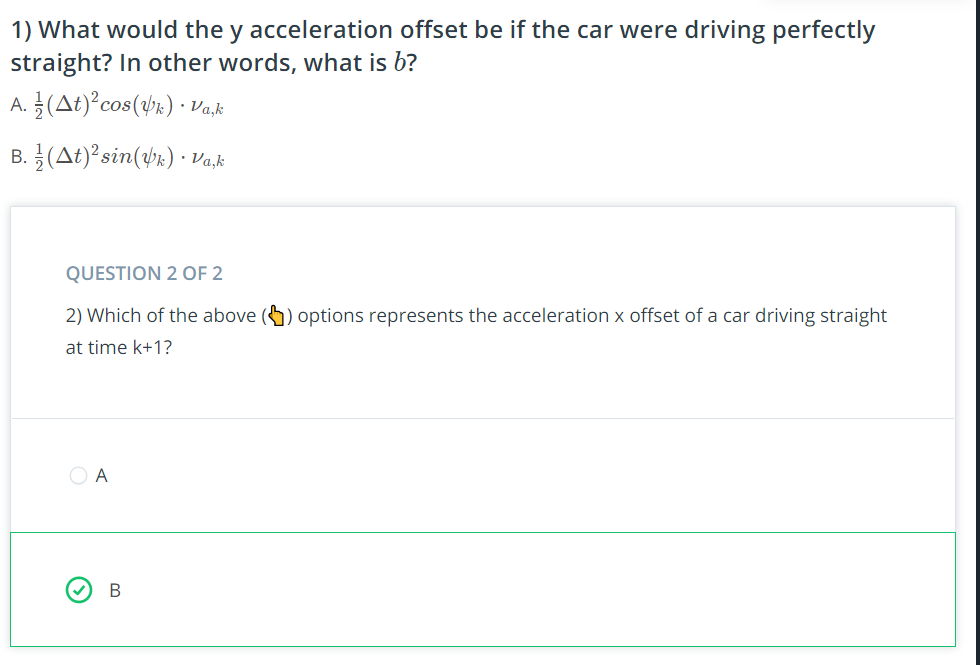


Calculating the effect of the Longitudinal Acceleration Noise on the position requires we go back to the differential equation that we started at, but this time make the assumption that instead of a constant velocity, we assume a constant acceleration and then solve the integral for that case.

We can solve the integral and get the correct solution, or just use this approximation to get the solution quicker. We can assume that the acceleration offset of the car is one where the car was driving exactly straight. This assumption works if the yaw rate is not too high.







**Unscented Kalman Filter Process Chain**

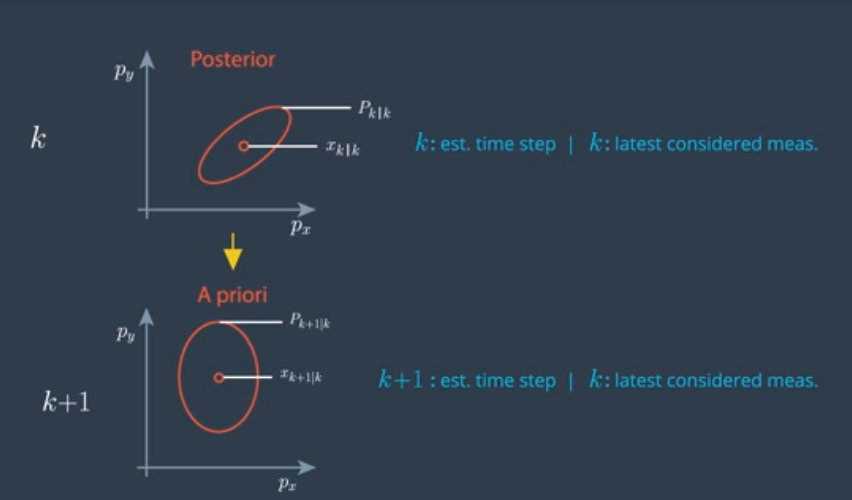
Prediction step and Measurement step in the same way as the Extended Kalman Filter, with new equations.

The update step varies depending on what time of measurement we receive.

The top level process chain is the exact same, just how the filter handles non-linear models changes.

Non-linear models are handled with Unscented Transformation.

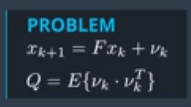
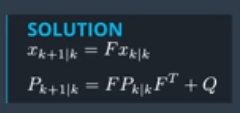
**What are the equation if we keep everything non-Linear?**



Each circle has the same probability distribution (This is a normal distribution). The ellipsis is also called the Error Ellipse.

It is the visualization of the covariance matrix P.

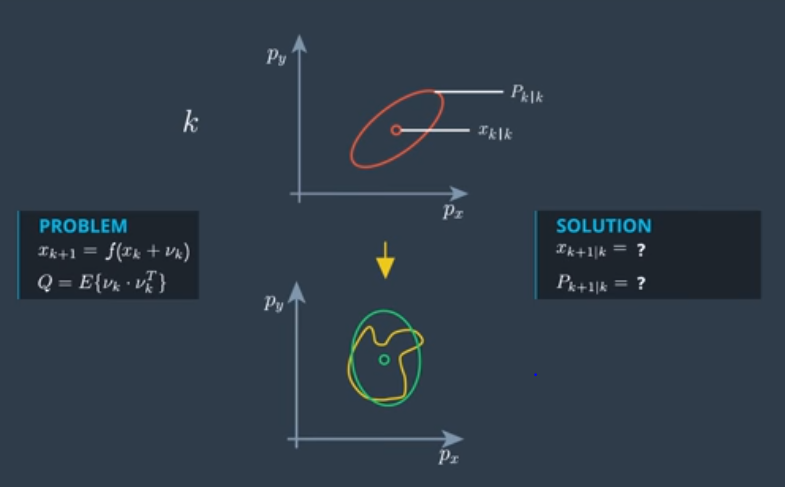
**Linear Process Model**

A Non-Linear Process model would result in a distribution that is not a Normal Distrbution. The prediction would be defined by the non-linear function F that we just described. The predicted probability distribution is difficult to calculate, and it generally can only be calculated numerically. In order to get a process model for the non-linear case an algorithm is required to get us the new distribution. The algorithm that does this numerical calculation is called a **Particle Filter**.

What an Unscented Kalman Filter does, is keep going as if the probability distribution was still **Normally Distributed**. (Of course this is an approximation)

So what we need to find is the Normal Distribution that represents the real predicted distribution as closely as possible. This means we want our estimate to the same mean and covariance matrix as the real predicted distribution.



**Unscented Kalman Filter Basics of Unscented Transformation**

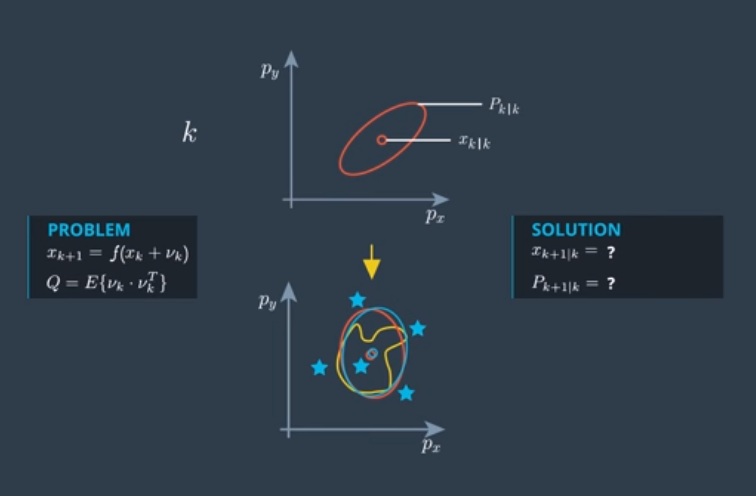
We can get an accurate normal distribution using **Sigma Points**.

It can be hard to transform the entire state distribution through a nonlinear function, but it is easy to transform individual points.

**Sigma Points**: Are points chosen around the mean state and in a certain relation to the standard deviation, sigma, of every state dimension.

These points serve as a representation of the entire distribution.

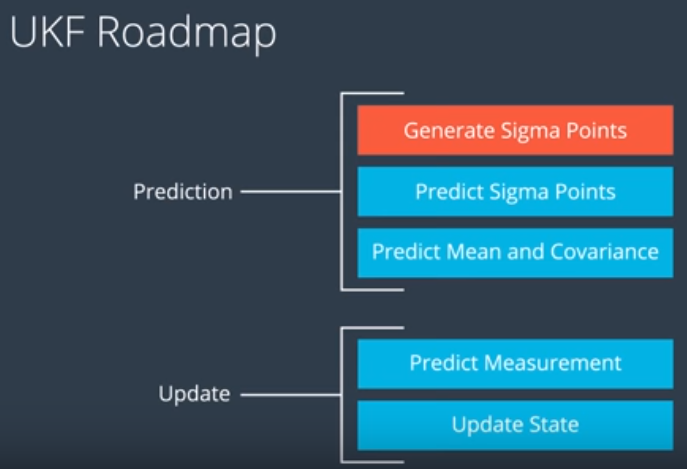
Once sigma points are chosen, they are inserted into the non-linear function F, which maps our sigma points somewhere in the predicted state space. All we need to do then, is to calculate the mean and variance of the group of sigma points, which gives a useful approximation of the actual probability distribution.



If F is linear, the sigma points approach gives the exact same solution as a normal Kalman Filter, but they are more expensive in terms of calculation time for linear problems.

**Process Chain:**

We will start with the prediction step, and then do the measurement step. The prediction step can be split into 3 parts: How to choose sigma points, How to predict Sigma Points (Insert into Process Function), and Calculate the Prediction mean and covariance from the Sigma Points.



UKF Generating Sigma Points:

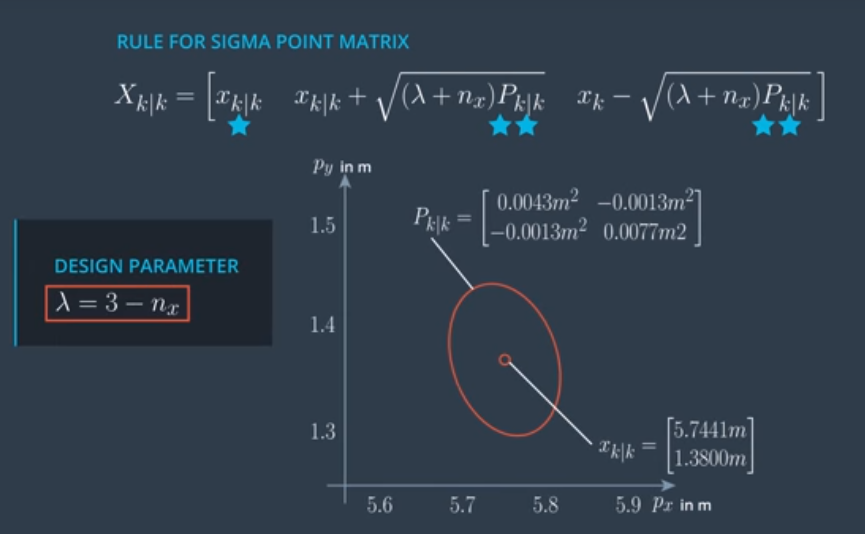
At the time we want to estimate Sigma Points, we have our posterior sate vector and covariance matrix, which represent the normal distribution.

The number of sigma points we need depends on the dimension of our state. In this case our state vector has a dimension of 5. The number of sigma points we need are 2\*Nx +1, which for our case is equal to 11.

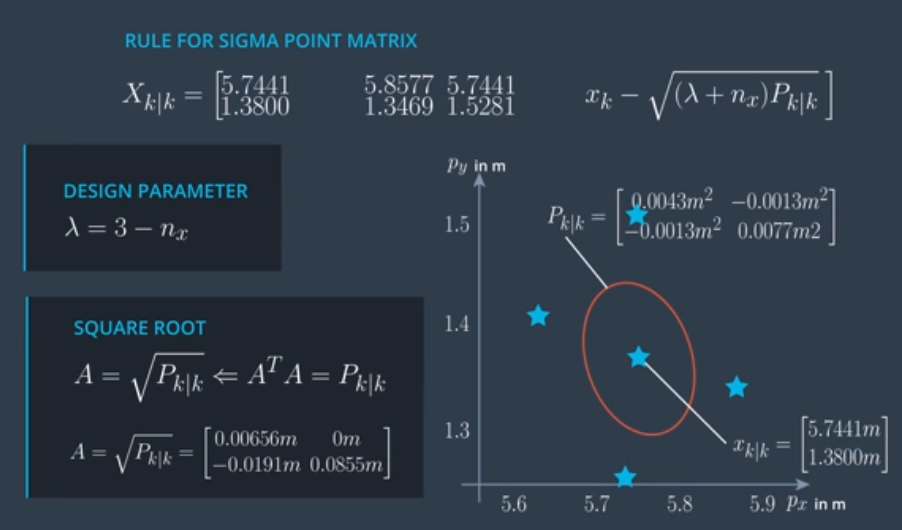
The first sigma point is the mean of the distribution, and the others are 2 points for every state dimension which are spread in different directions.

Assuming we have a state vector of only px and py. We need 5 sigma points.

The calculation can be done with:



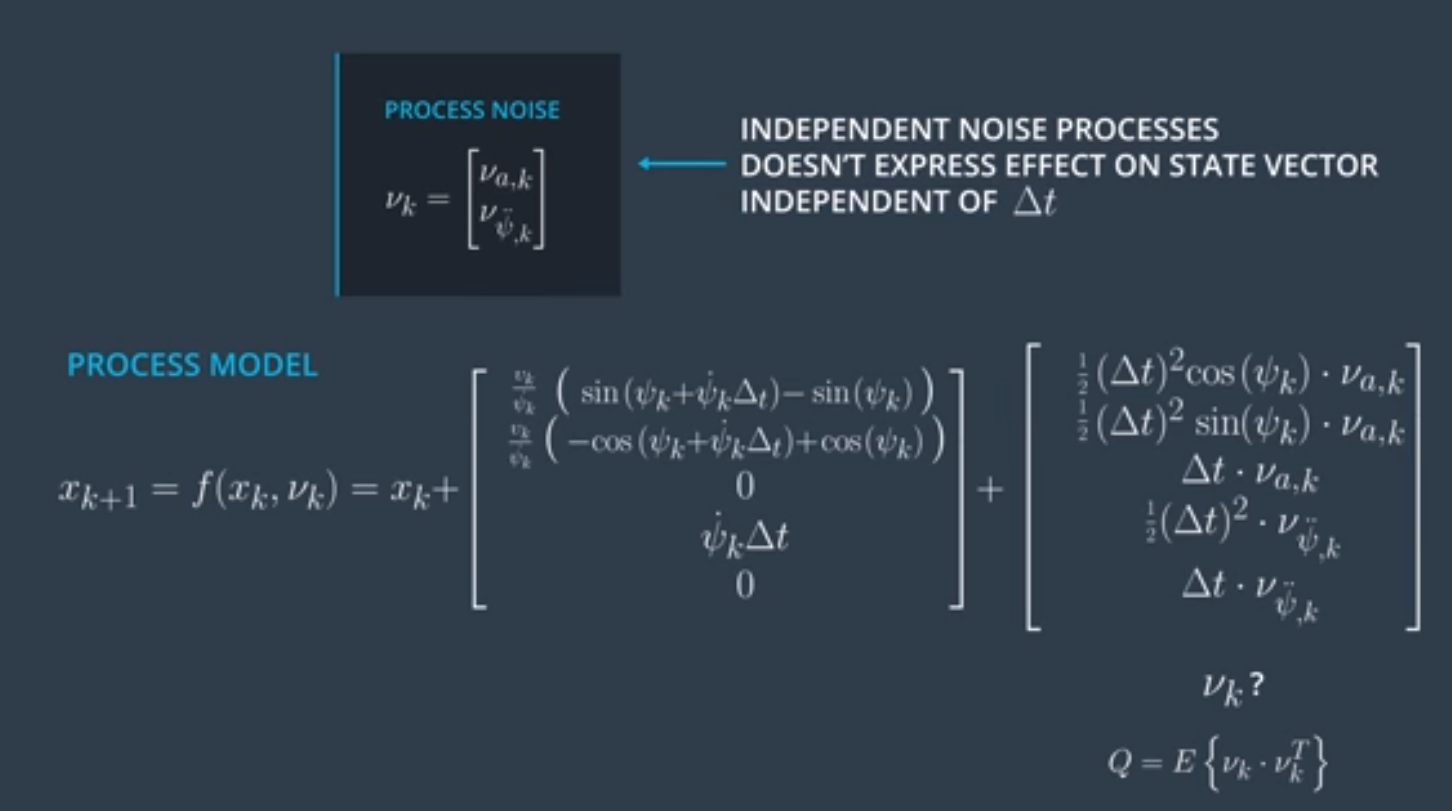
In order to get some sigma points for the variance, we need to be able to calculate the square root of a matrix, which can be done by finding a Matrix, A, that solves the equation below (Uses Cholesky Decomposition). This equation ensures the sigma points we use are in opposite directions because of the addition and subtraction from the mean of the same term. Lambda is a design parameter that is used to give us the distance from the mean our variance prediction should cover.



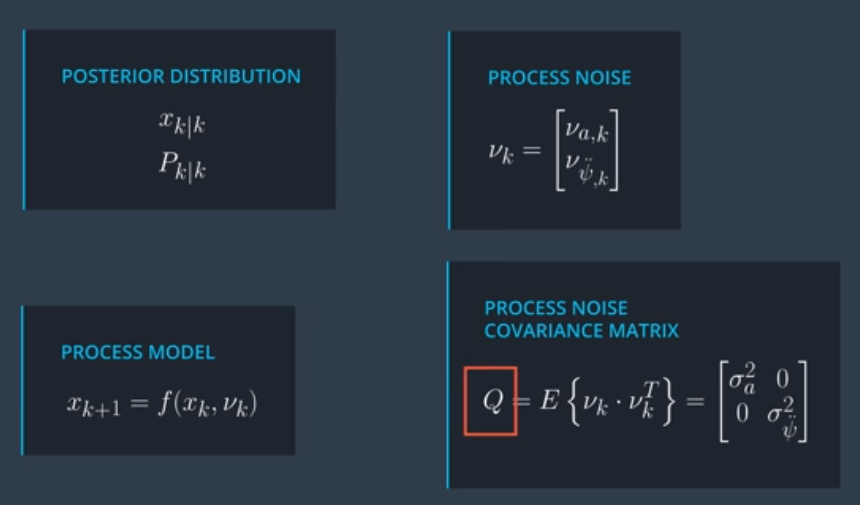
Now that we have our Sigma Points, we can use our Process Function to convert the points to our prediction state.

But our process function has the noise element, which is non-linear.

The UKF has an easy way to handle nonlinear process noise effects.

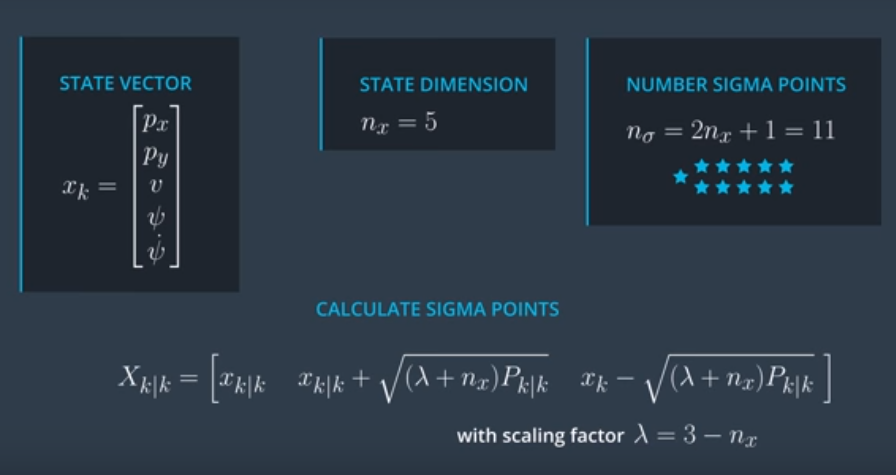


When talking about process noise, we are talking about the vector that lists all the sources of uncertainty (The vector at the top). In this case calculating the Process noise covariance matrix, Q, is much simpler.

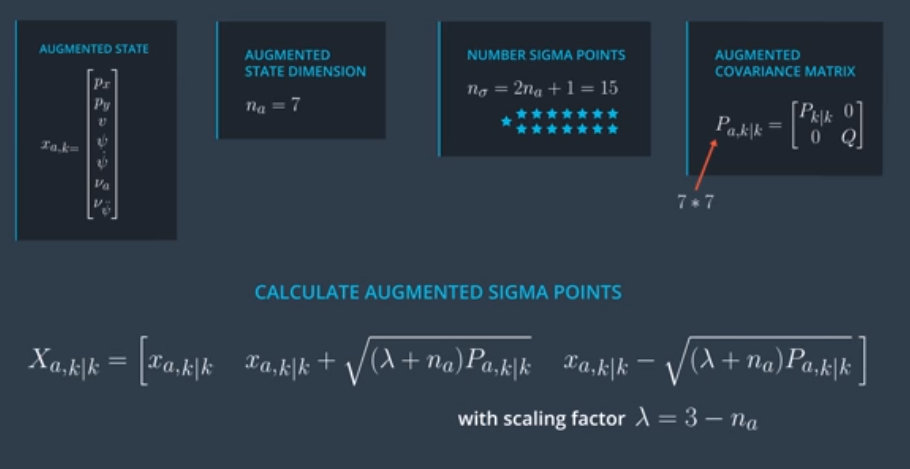


Now we need to consider how we represent the uncertainty of the covariance matrix with sigma points. The solution is called **Augmentation**.

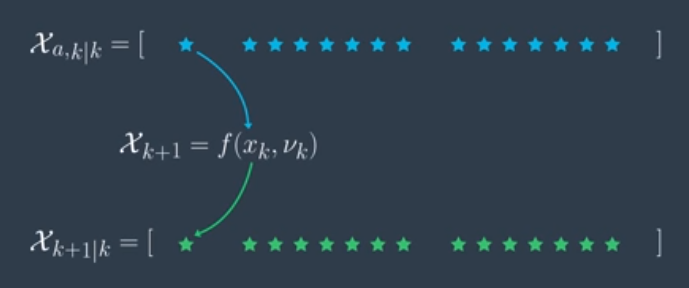
Without considering Q, this is how we got Sigma Points.

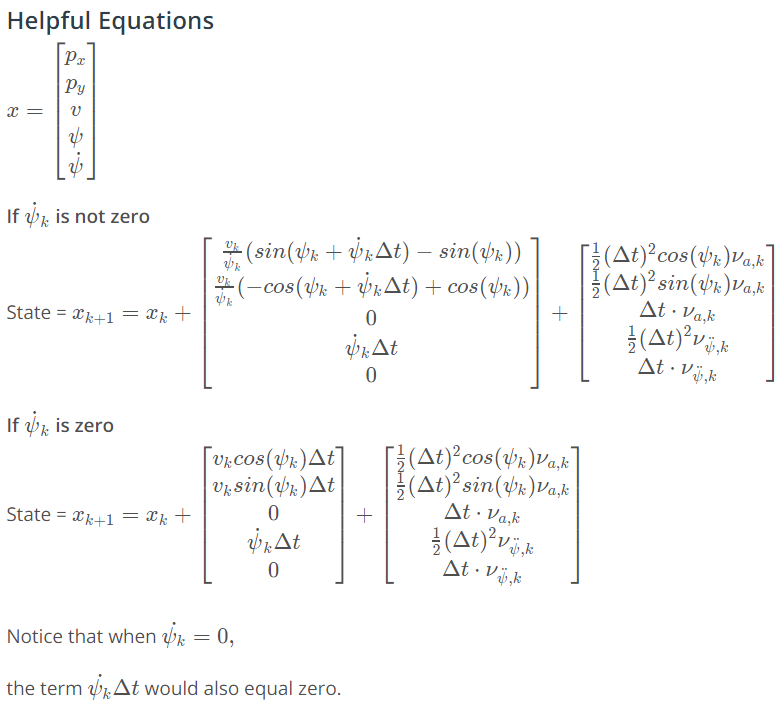


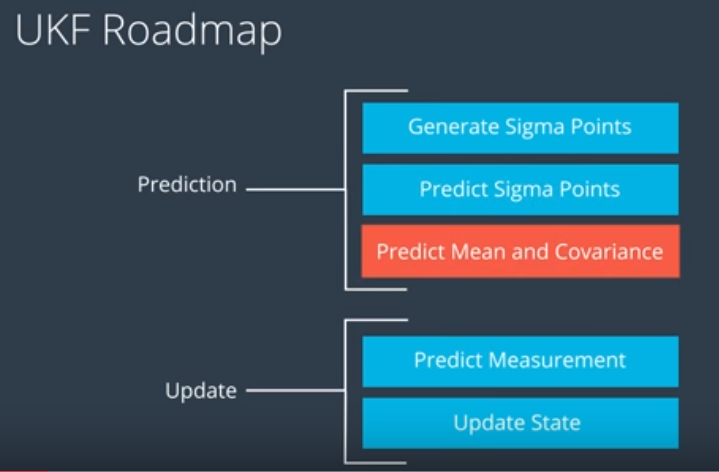
Now let’s look at the Augmented State. The uncertainty caused by the process noise that is added to our state vector are represented by the extra Sigma Points. We also include Q by expanding Augmenting the Covariance Matrix.



Now that we have our augmented Sigma Points, we can directly insert them into the Process Function to get our Predicted Sigma Points.



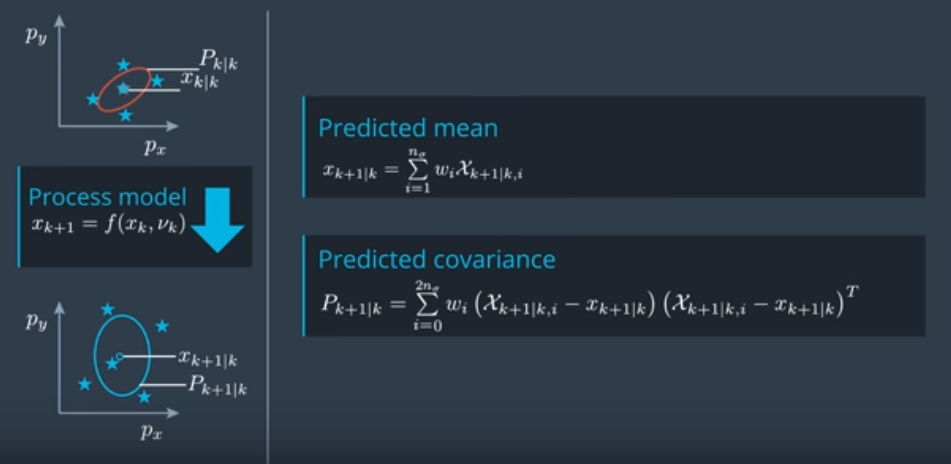




Now that we have generated sigma points and can predict them by using f(x,v), we can move on to Predicting the mean and covariance Matrix of the predicted state.

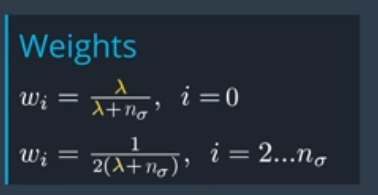
Now **How do we use the predicted sigma points to estimate the mean and the covariance of the predicted state**. This is like the opposite of trying to use our mean and covariance matrix to generate sigma points.

The general rule is given by these equations.



Calculating the mean of the distribution from the sigma points requires we add the columns together with some weighting value (index i is the column number).

The weights are calculated with:



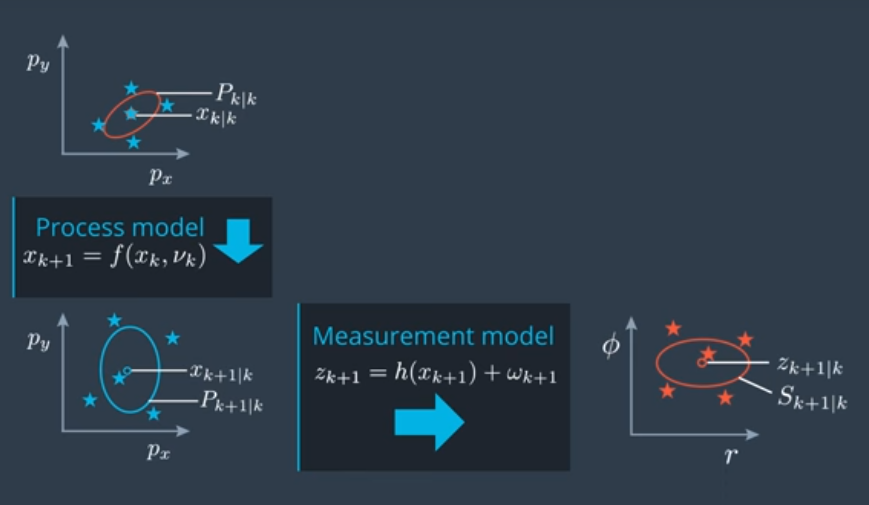
Because the spread of the points is described by lambda it is required when going back from the sigma points.

There are many ways the weights can be defined and can find many suggested ways when looking at the literature.

**Measurement Prediction (Update Step)**

Now that we have a predicted state, which needs to be transformed into the measurement space.

This is when you consider the type of sensors you are using and come up with a corresponding measurement model that can map your predicted state to the measurement space. H(x).



The problem here is that the measurement model h(x) is non-linear, and we run into the same issue as we did in the prediction step where we want the output of the function to stay as a Gaussian distribution.

To solve this we can use the same process as we did in the prediction step, where we evaluate sigma points and transform those points using the non-linear measurement model.

There are however 2 shortcuts that we can take during the measurement update step.

One shortcut is that instead of generating sigma points from the predicted mean and covariance matrix that we just acquired from sigma points we can just use the same predicted sigma points directly.

That way we can skip the sigma point generation and the augmentation step.

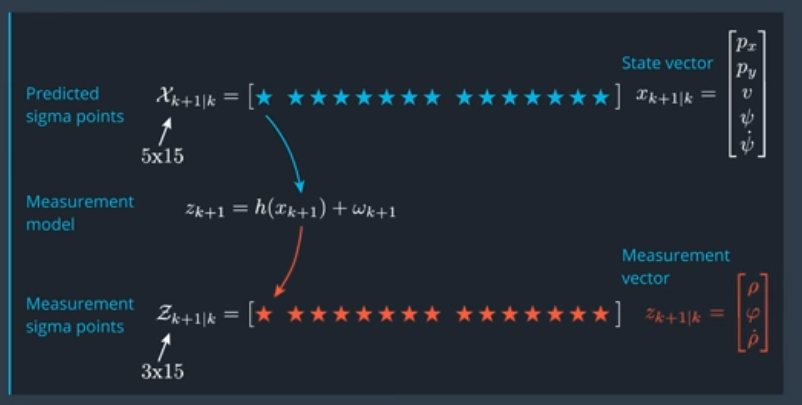
Why can we skip the augmentation step?

We first did the augmentation step because of the effect of the prediction noise being non-linear, however in the measurement mode, we see that the noise w just has an additive effect, meaning we don’t need to generate augmented sigma points.

There is a much easier method to consider the measurement noise that can be used because the effect it has on the measurement model is not non-linear.

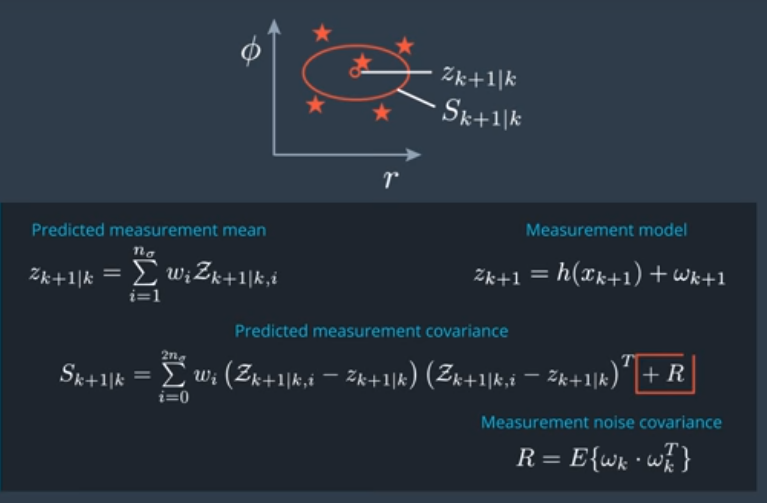
We should also keep the sigma points that are in the measurement space as they are useful.

For the example of the radar, the measurement space is 3 (row, phi, and row\_dot).



W can be set to 0 here because the measurement noise will be accounted for later.

We can transform our Measurement Sigma Points into a mean and covariance in the same way as the prediction step.

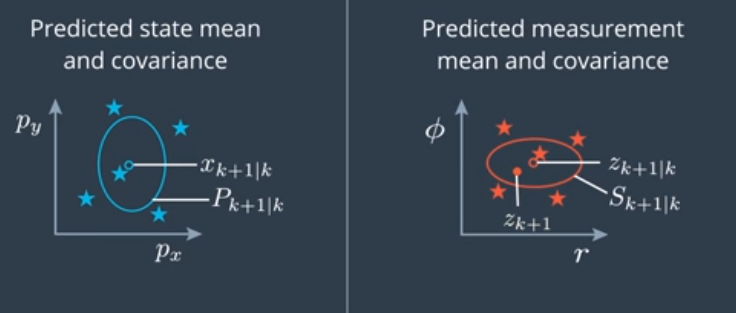


The only difference in generating a mean and covariance matrix from the sigma points is the addition of the measurement noise R. Again this method can only be used because the noise in the measurement model is additive, it has no non-linear effects.

Final step of the Update Step is to then use the measurement to update the State.

**Update State**

Now we have 2 pieces of information we need to update our state.

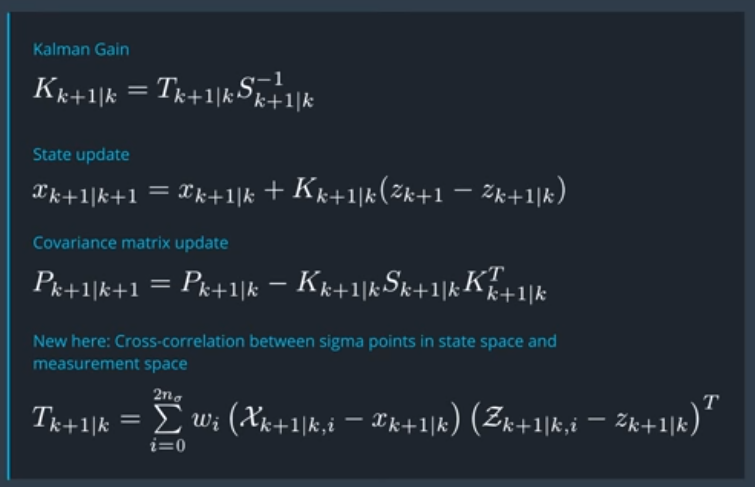


We have our Predicted state mean and covariance, as well as the predicted measurement mean and covariance.

The last bit of information we need is the actual measurement at zk+1

We used delta T in our prediction step to ensure our prediction of the state was at the right time step, we checked what sensor the data was coming from to ensure we used the correct measurement model, and now we need the actual measurement values.

The state update step is actually identical to the update step of a normal Kalman Filter, but with one change in how we calculate the Kalman Gain, and a newly required measurement of the cross-correlation between sigma points in the state space and the measurement space.



**Parameters and Consistency**

Now choosing noise parameters is **important step of not only the Unscented Kalman Filter, but to any Bayesian Filter**.

Our process and measurement models introduce several sources of uncertainty. The process model has vk and the measurement model has wk.

The noise values were also quantified with the Process noise covariance, and the Measurement Noise Covariance. These are the variance of the noise and determine how strong the noise is. But where do these numbers come from?

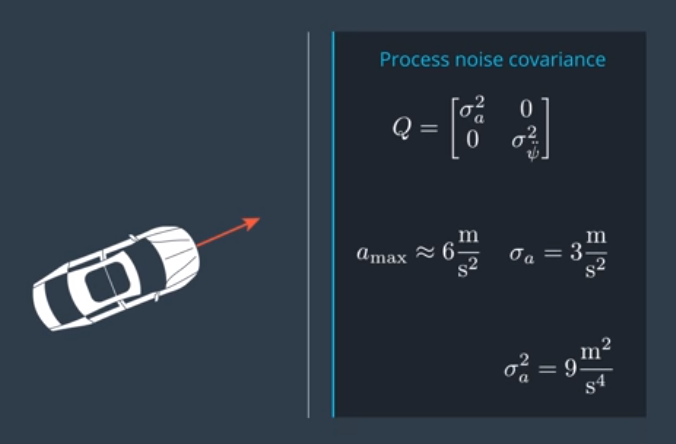
**Measurement Noise**: the noise describes how well the sensor can measure its values. We look into the sensors manual to determine how well it can function and determine the variance values that way.

Example: We see in the manual that the radar sensor can read its measurement of row with a standard deviation of 0.3m. The value can then be used to calculate the variance of row, and inserted into the Measurement Noise Covariance.

These are under the assumptions that the noise is white noise (Mean of 0) and is normally distributed which is not true with some sensors.

**Process Noise**: more difficult to choose, but drivers in traffic don’t move with a wide acceleration noise (constantly switching between gas and break).

A good rule of thumb when trying to determine values is to think about what the max value would be in your environment. With cars in an urban environment, you don’t usually see accelerations or breaks of 6 m/s2. The rule of thumb says to choose half of the maximum acceleration as you expect as process noise, meaning we would choose a standard deviation of 3 m/s2.



If the application needs to react fast on changes, a higher process noise is chosen.

If the application needs a smoother estimation, you choose a lower process noise.

Usually want to know if you set up noise parameters correctly, and for that purpose we run a consistency check on the filter.

**Constancy**:

At every time cycle we calculate the measurement mean zk+1|k and covariance Sk+1|k . We then receive the actual measurement zk+1. We can check to see if the measurement is actually within the error ellipses to ensure that the variance we chose had the measurement within its boundary.

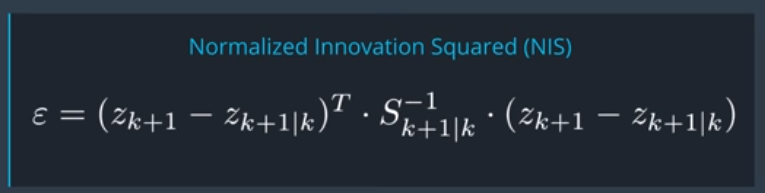
|  |  |
| --- | --- |
|  |  |

Having the actual measurement outside the error ellipses means you underestimated the uncertainty of the predicted measurement. Again these are stochastic processes, so it may just be coincidence for some time steps. However if the results keep looking like this, the estimates you are making are less precise than you think.

The opposite problem can also occur, we can overestimate the uncertainty of the system, meaning that the estimate is more precise than you think.

**A Filter is consistent if it provides a realistic estimation uncertainty**.

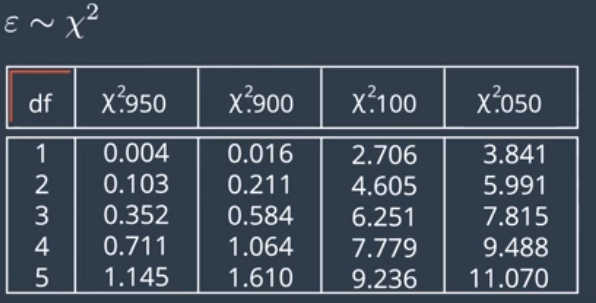
We can check the consistency of the filter easily using many different equations, with one being the Normalized Innovation Squared Check.



The result is just a scalar number, but we need to know what range we want our value to be in.

The NIS value follow a distribution called the **chi squared distribution**.

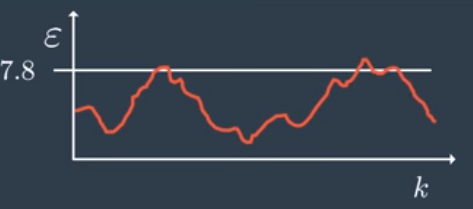
This table is a chi squared distribution for a given number of Degrees of Freedom (df).



With Radar measurements, we have a 3 dimensional measurement, meaning 3 degrees of freedom.

The First column represents that statistically in 95% of all cases, your NIS will be higher than 0.352 for 3 degrees of freedom. It also says that in 5% of cases, the NIS will be higher than 7.815.

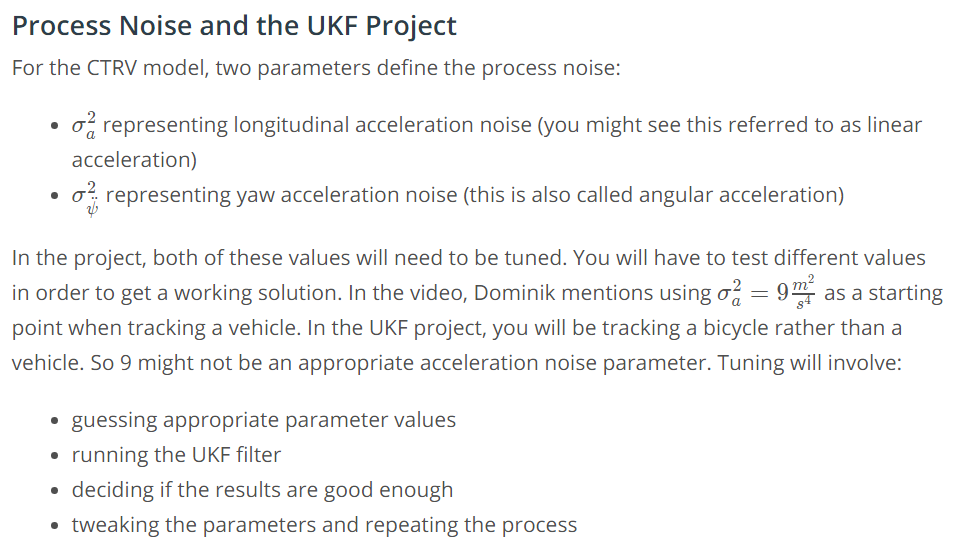
One thing we could always do is plot the 95% line, which is 7.8 (If 5% chance it is higher than 7.8, than there is 95% chance it is under), and plot the NIS value for each time step k.

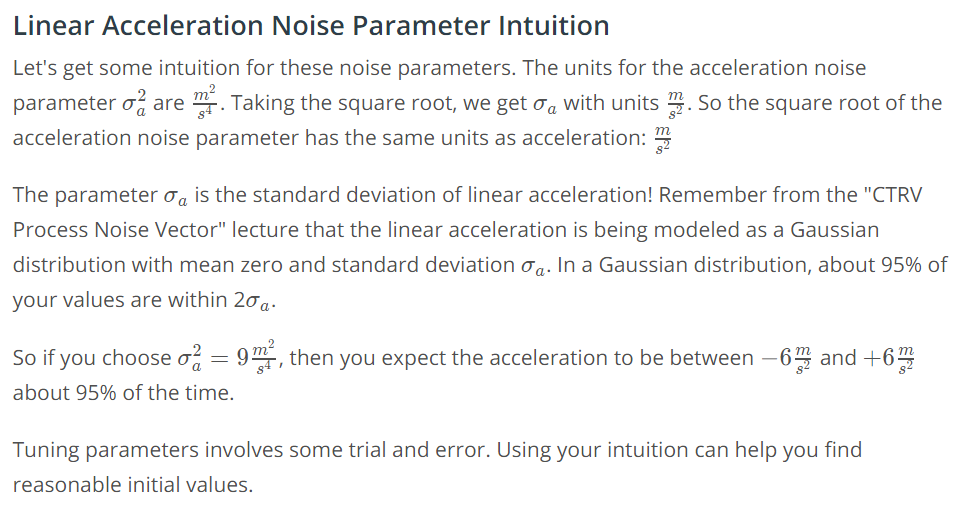


This is an Example of desired NIS values. Sometimes it does exceed the 7.8 line, but that is expected to happen. An undesired NIS plot is this:

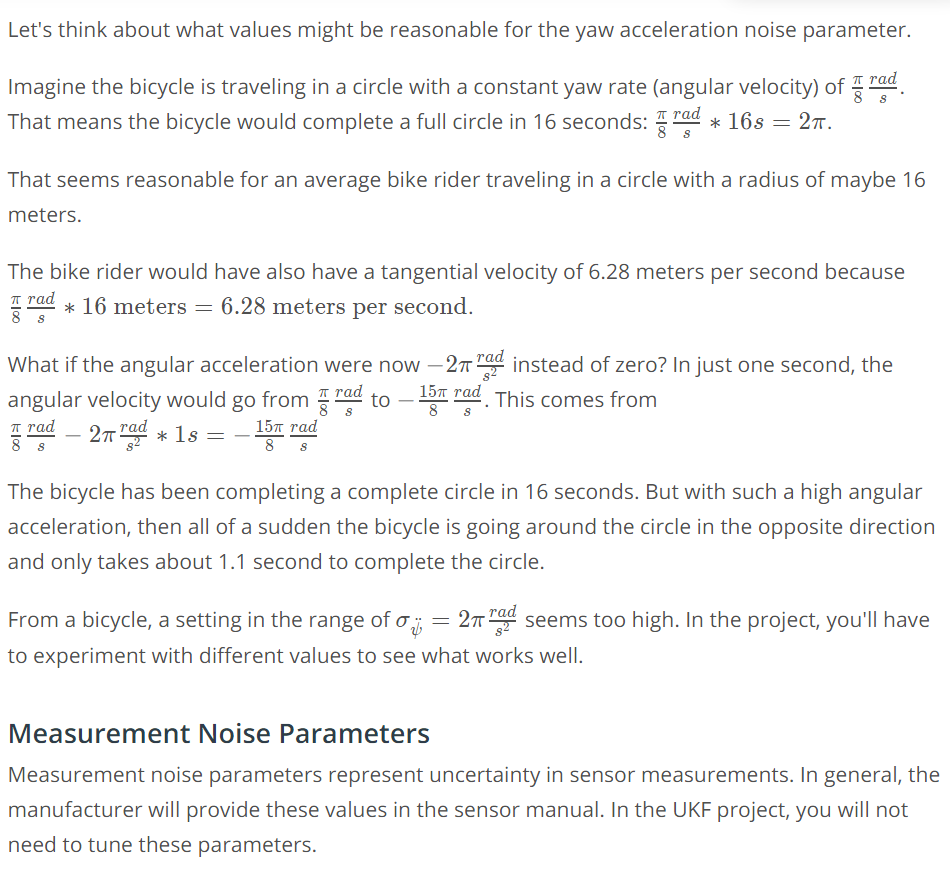
|  |  |
| --- | --- |
| Underestimating the Uncertainty: | Overestimating the uncertainty: |

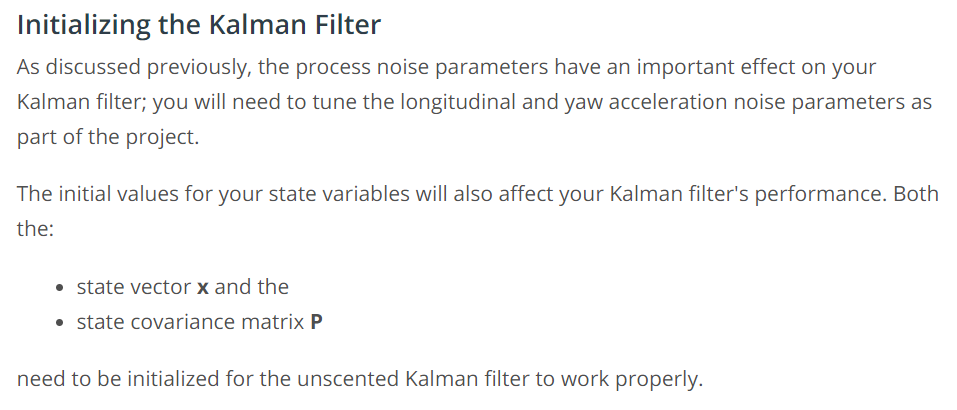
The unfortunate aspect of the NIS is that it doesn’t tell you where the mistake is coming from, but it is usually a good idea to change the process noise and try it again.

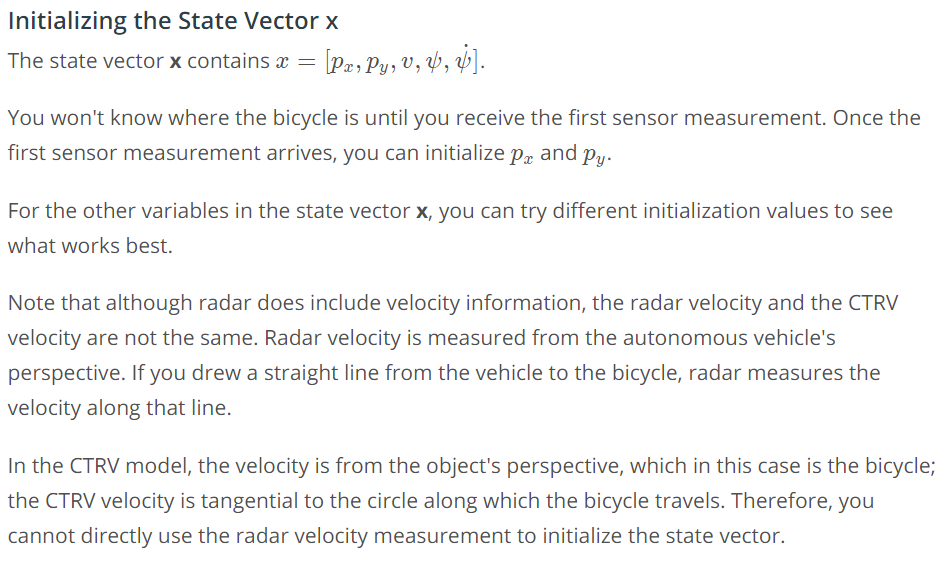


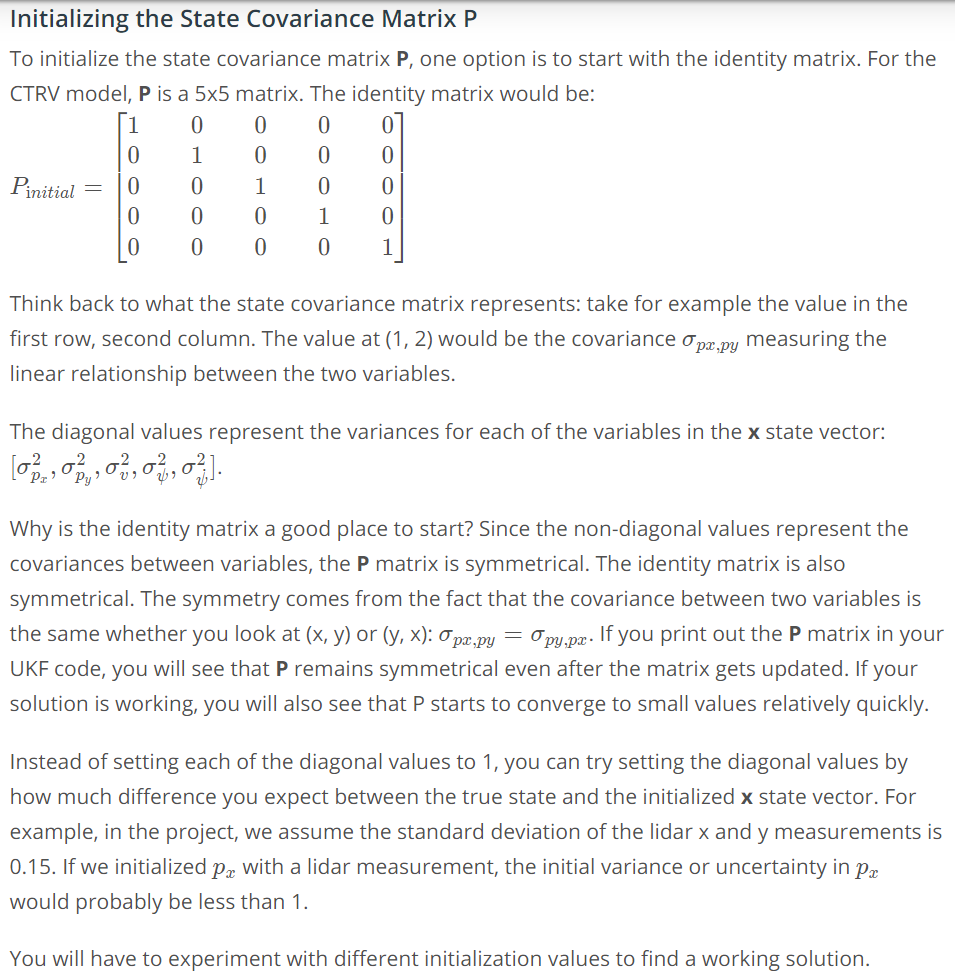


Note one standard deviation has 68% of the values, and 2 standard deviations has 95% of the values.



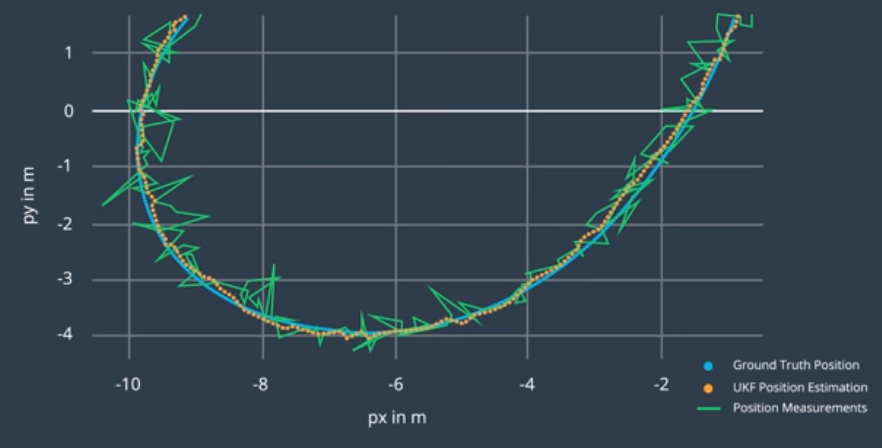




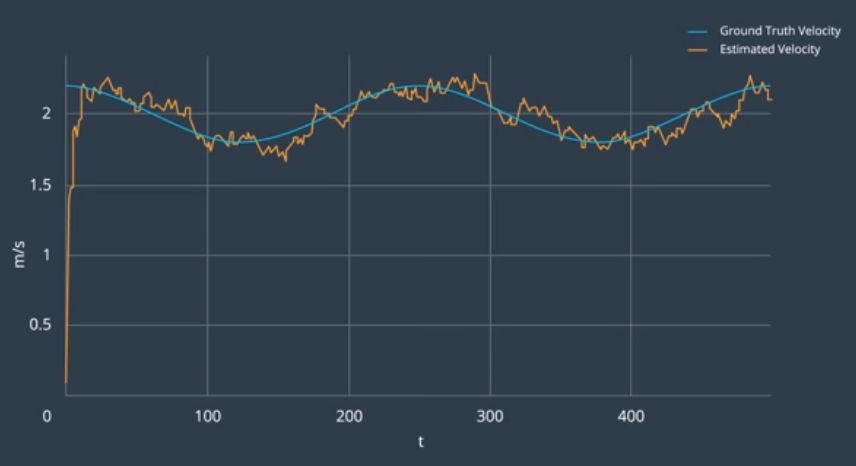


Expected UKF Values and Constancy Values.

Position Estimates:



Velocity Estimates:



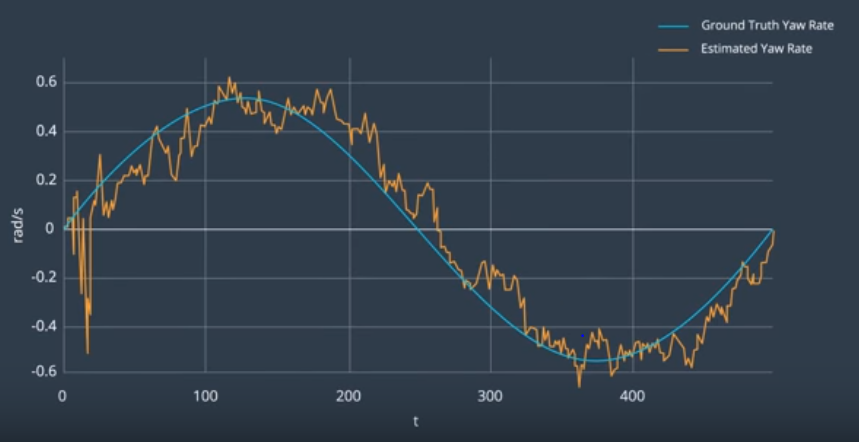
Velocity values converge much faster with the use of Radar Measurements then with only LIDAR.

Orientation Estiamte:



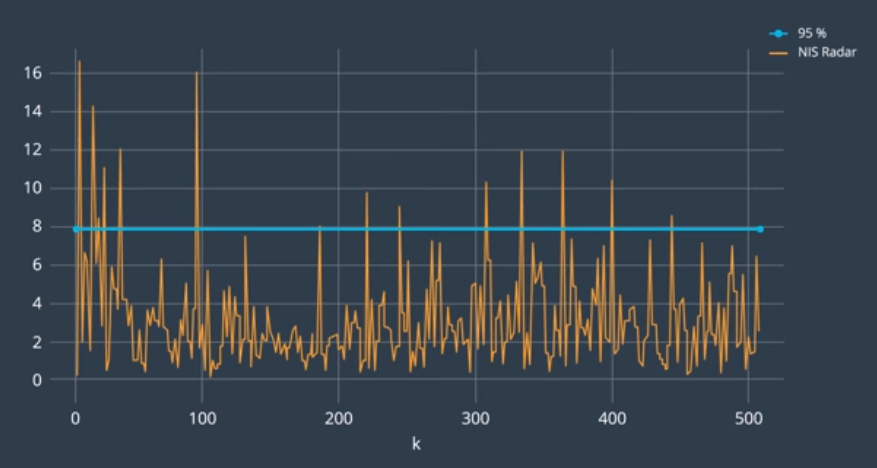
No sensor we used can directly observe the orientation, but we still get a precise estimate.

Yaw Rate Estimation (turning velocity):

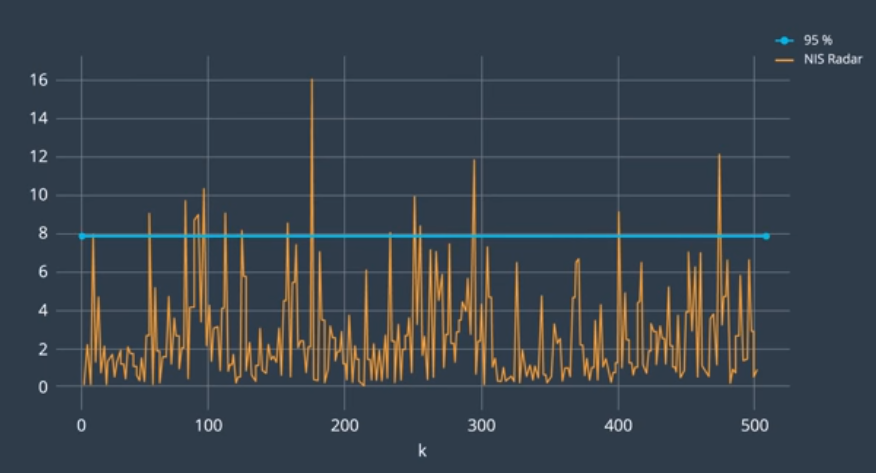


Yaw Rate of other vehicles is very important to know like for cars of vehicles changing lanes or turning left or right.

Constancy Of Filter with Radar Measurements (3 Degrees of Freedom):



Consistency of Filter with LIDAR Measurements (2 Degrees of Freedom):



Summary:

An Unscented Kalman Filter is an important measurement tool for self-driving cars.

1. You can take noisy measurement data as an input and provide a smooth position and velocity estimate of objects around you without introducing a delay.
2. We can gain estimations of things like the orientation and the yaw rate of other vehicles using sensors that can’t even directly observe these states.
3. The UKF also gives information on how precise the result is by providing a covariance matrix for every estimation, and we know that the covariance matrix is realistic if the UFK performs an Consistency check. The uncertainty of the estimation result is important because if a leading vehicle has an uncertain position measurement, we can make sure the self-driving car keeps a safe distance.