**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)

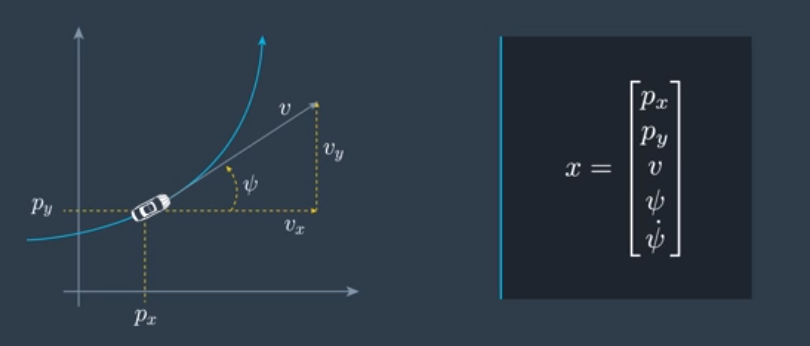
Each model makes different assumption about the objects motion, so the application determines which one we should use.

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.

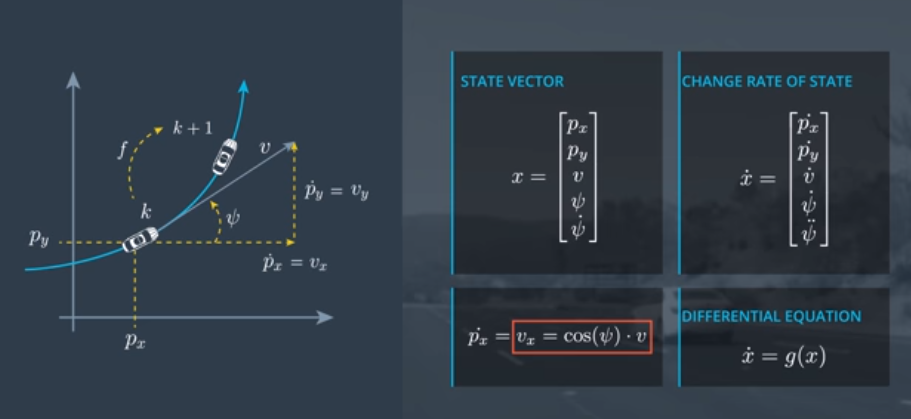


Note: When a car is going on a straight path, the psi\_dot is zero. When thinking about turning radius, it is the combination of both the turning angle, and the speed. If two vehicles have the same turning angle, the slower vehicle will have the smaller turning radius (make a complete circle in less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

**Change in Process Model**

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

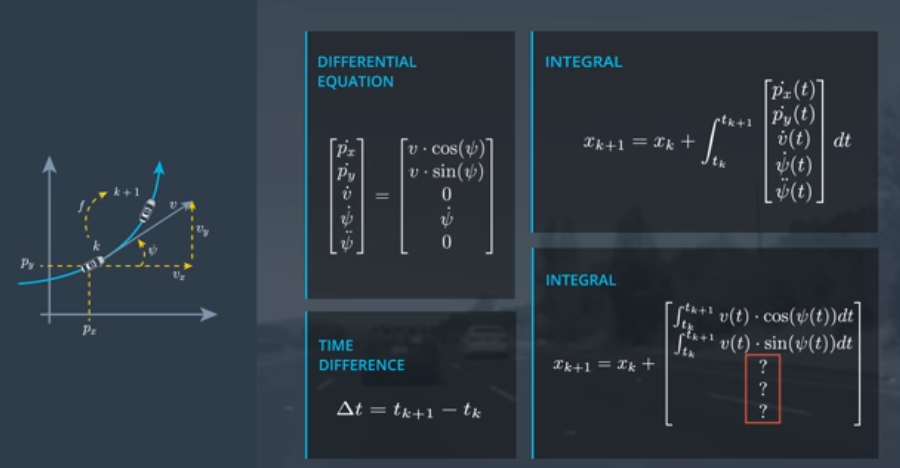
A technique we can use when trying to derive the equation that can take us from one state to the prediction of the state at the next time step is to look at the change rate of the state vector (derivative of the current state).



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

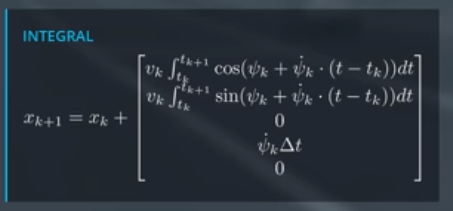
Now that we have the complete differential equation, we need to get from time step k to k+1. Looking at our time values as not only discrete steps but as continuous values, we can get from k to k+1 by using integrating our differential equation over the time difference.

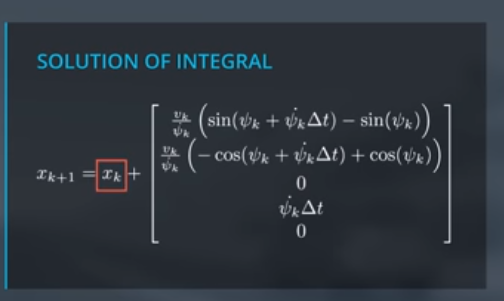
With the case of the CRTV, the integral can be solved pretty simply.



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.

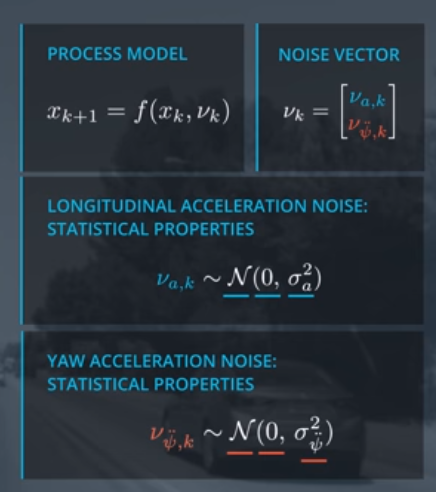
**CTRV Process Noise Vector**

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

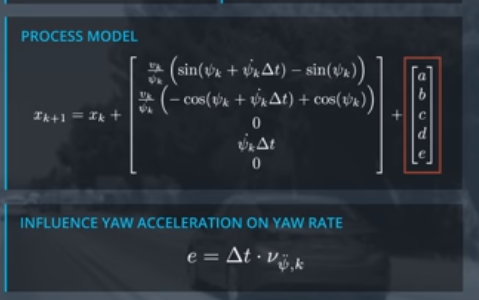
First Noise process is the Longitudinal Acceleration Noise, and the Second is Yaw Acceleration Noise.

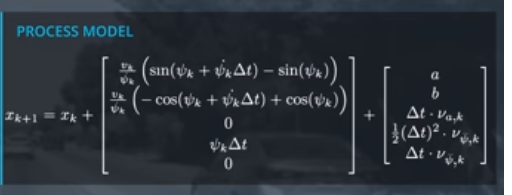
**Longitudinal Acceleration Noise:** Influences the Longitudinal Speed of the vehicle and randomly changes its value at every time step k. It is a normal distribution with 0 mean and a variance sigma\_A squared.

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



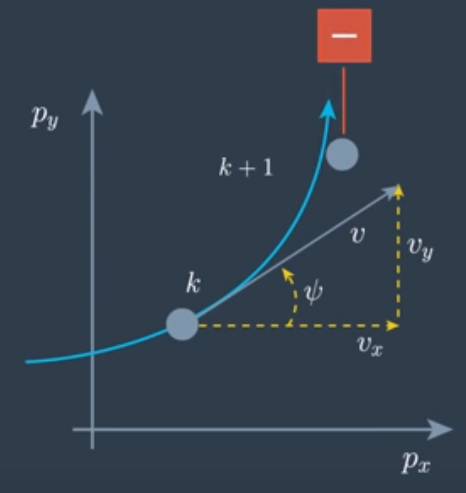
Under the CTRV model, and the assumption that the process noise from one time step to the next is the same, we can determine that the errors are:





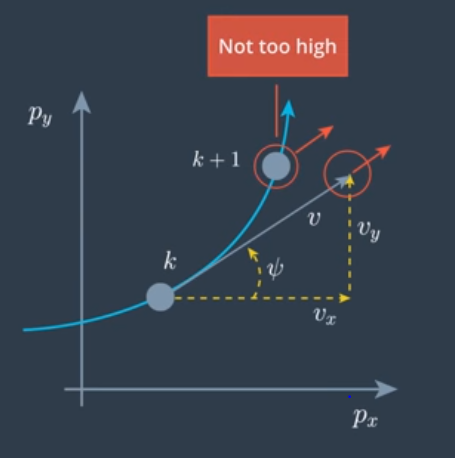
Process noise for position:

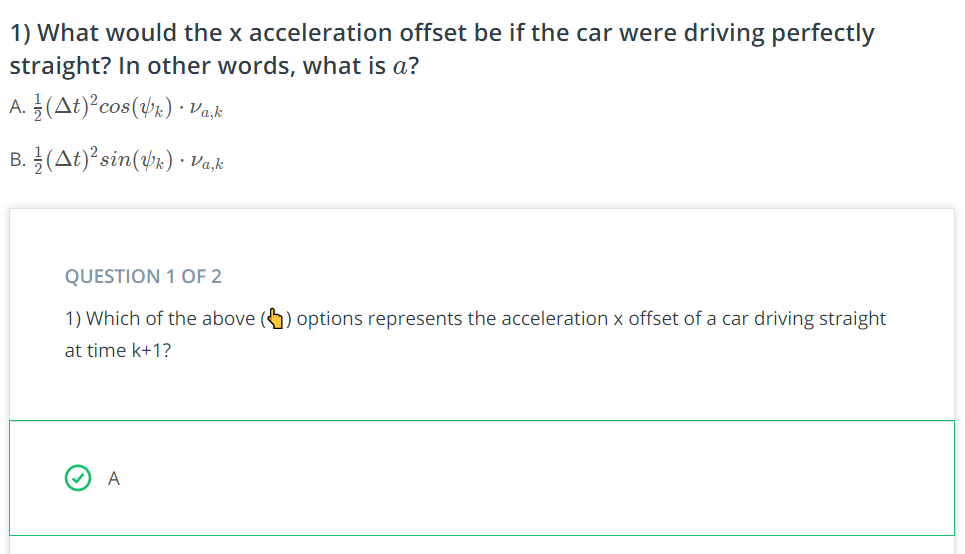
When thinking about a vehicles position as it is turning from one time step to the next, we can see that the yaw rate (phi\_dot) does effect the next position. The yaw rate effects the position, but for our application we can assume that because it has little effect overall on the position of a vehicle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

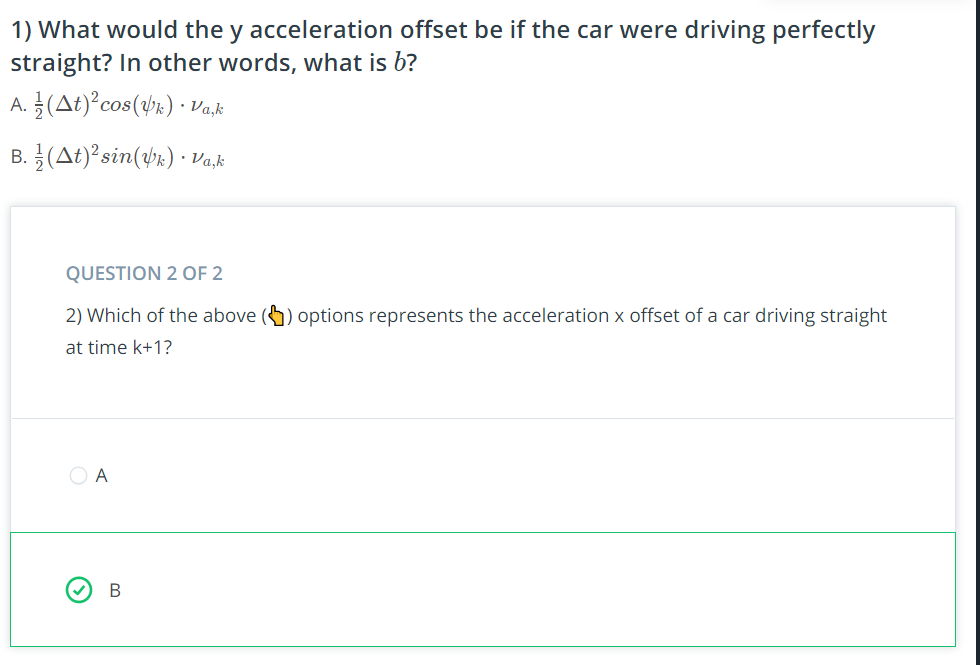


Calculating the effect of the Longitudinal Acceleration Noise on the position requires we go back to the differential equation that we started at, but this time make the assumption that instead of a constant velocity, we assume a constant acceleration and then solve the integral for that case.

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**

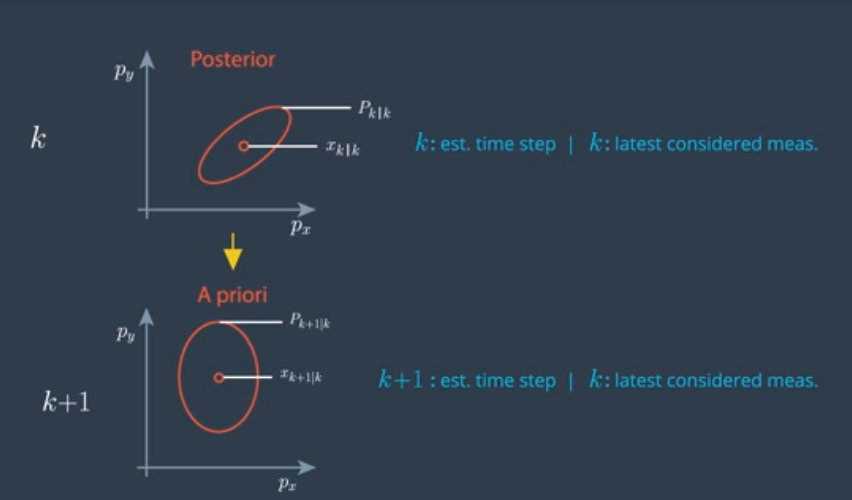
Prediction step and Measurement step in the same way as the Extended Kalman Filter, with new equations.

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.

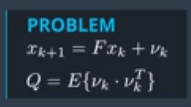
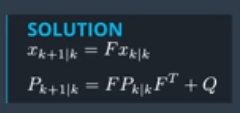
**What are the equation if we keep everything non-Linear?**



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.

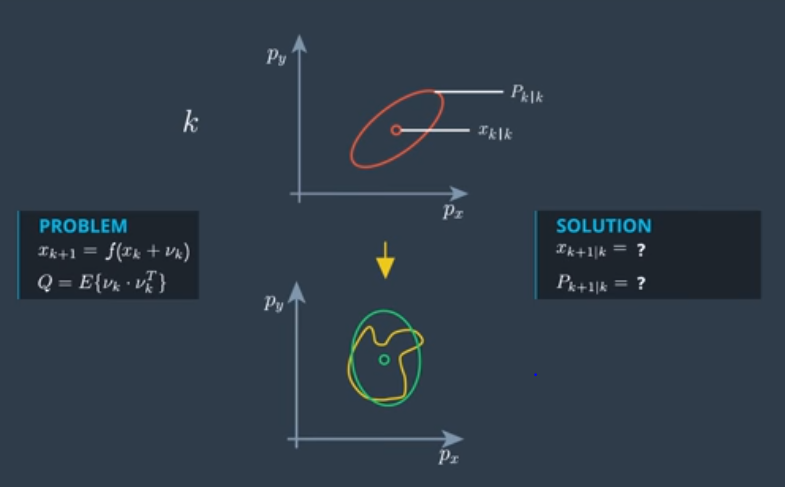
**Linear Process Model**

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**

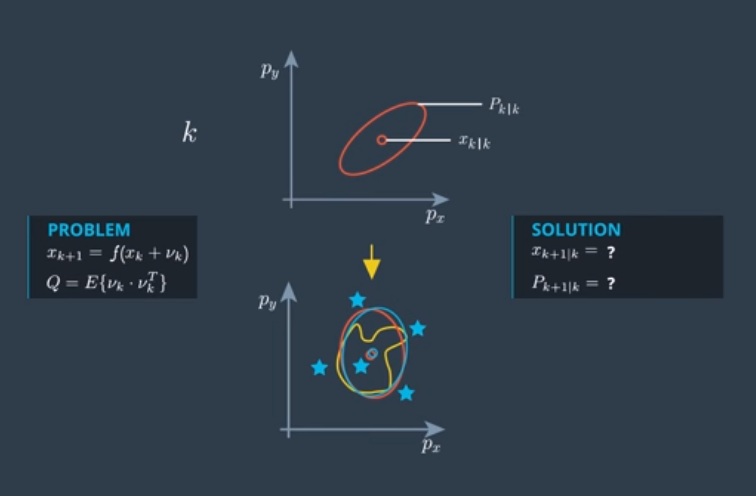
We can get an accurate normal distribution using **Sigma Points**.

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.

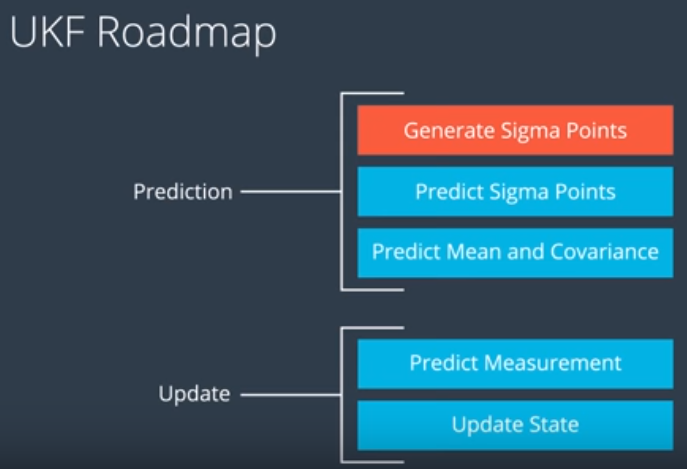
Once sigma points are chosen, they are inserted into the non-linear function F, which maps our sigma points somewhere in the predicted state space. All we need to do then, is to calculate the mean and variance of the group of sigma points, which gives a useful approximation of the actual probability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

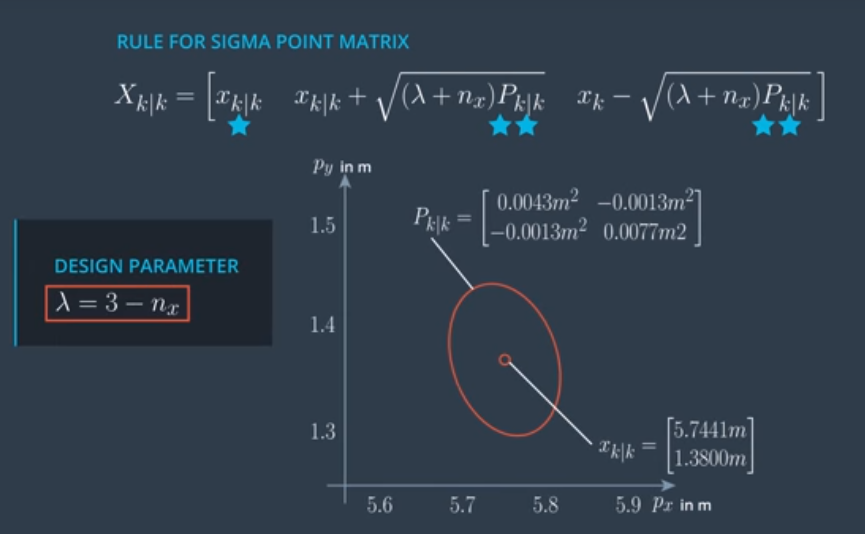
At the time we want to estimate Sigma Points, we have our posterior sate vector and covariance matrix, which represent the normal distribution.

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.

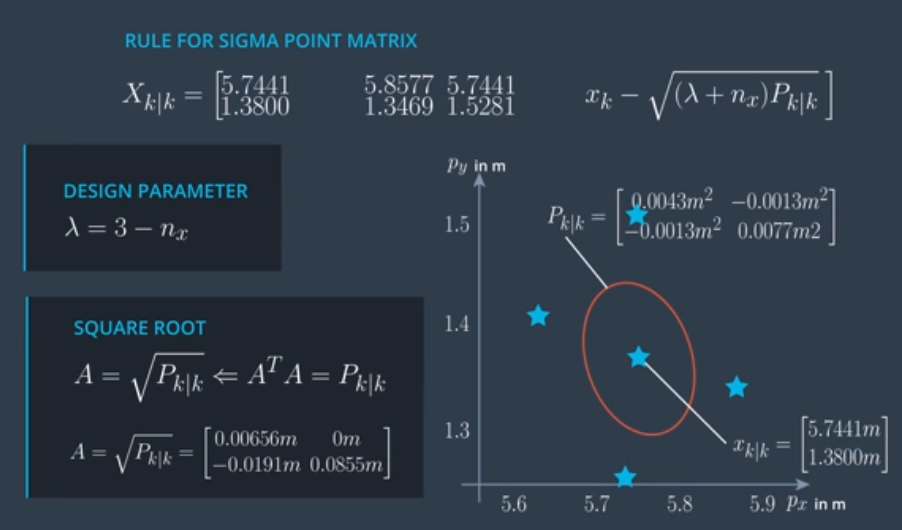
The first sigma point is the mean of the distribution, and the others are 2 points for every state dimension which are spread in different directions.

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

The calculation can be done with:



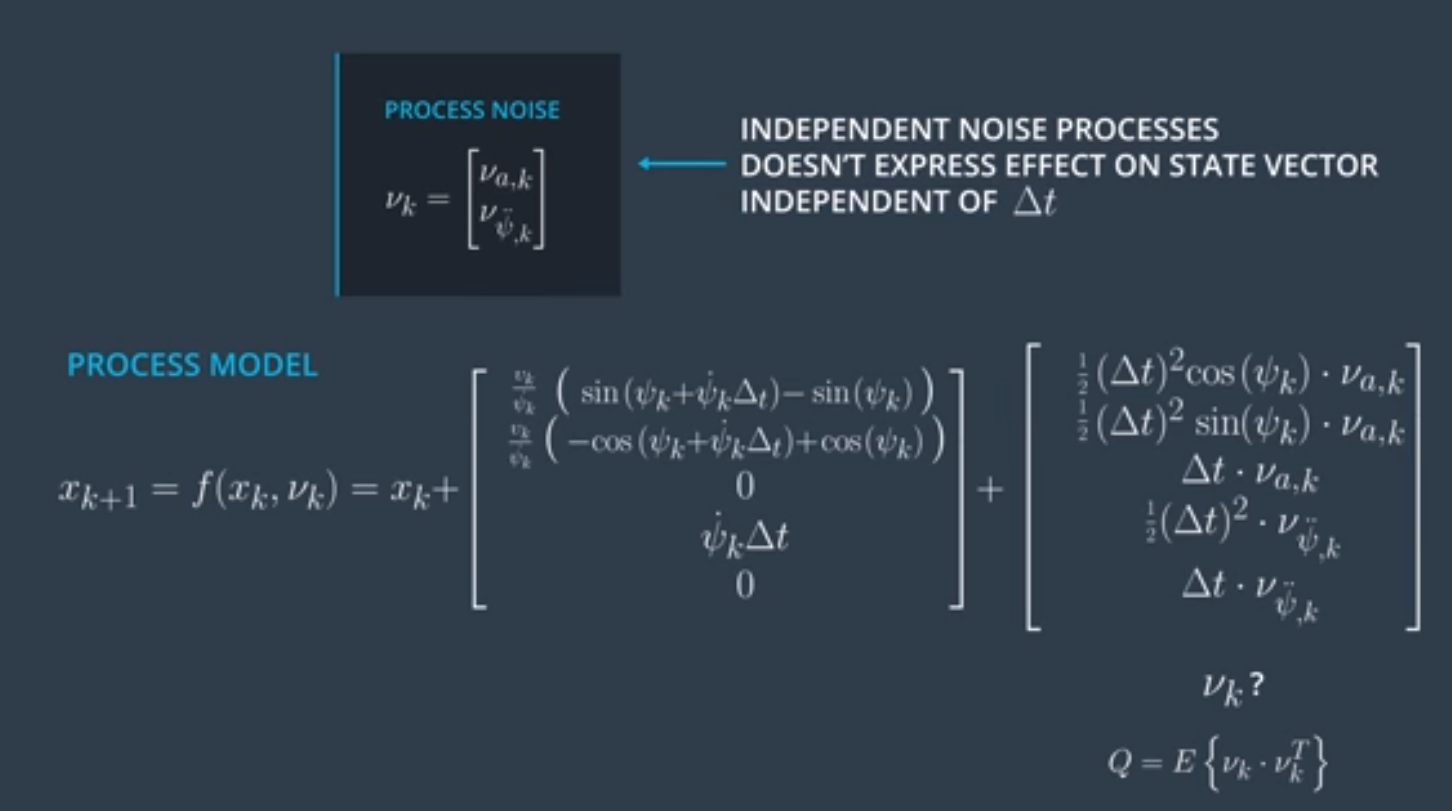
In order to get some sigma points for the variance, we need to be able to calculate the square root of a matrix, which can be done by finding a Matrix, A, that solves the equation below (Uses Cholesky Decomposition). This equation ensures the sigma points we use are in opposite directions because of 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.



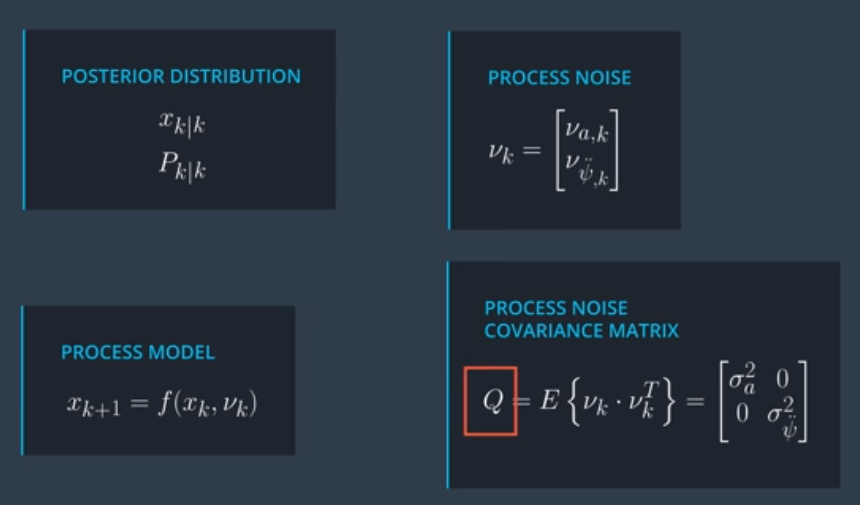
Now that we have our Sigma Points, we can use our Process Function to convert the points to our prediction state.

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

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

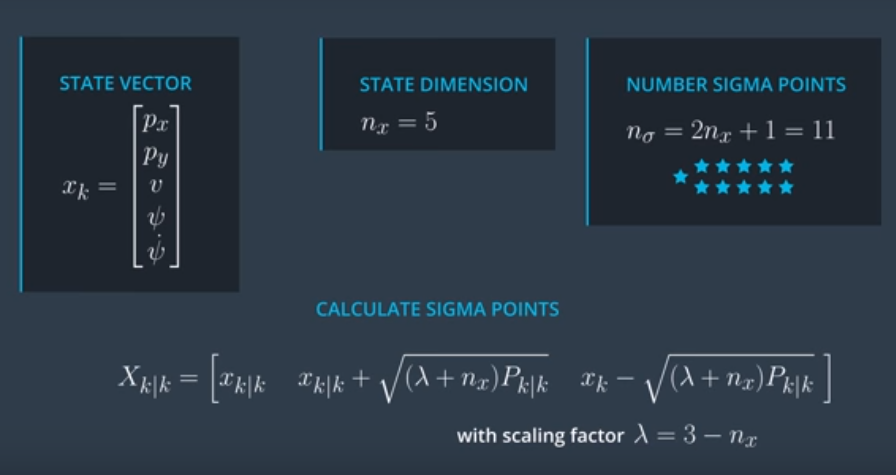


When talking about process noise, we are talking about the vector that lists all the sources of uncertainty (The vector at the top). In this case calculating the Process noise covariance matrix, Q, is much simpler.

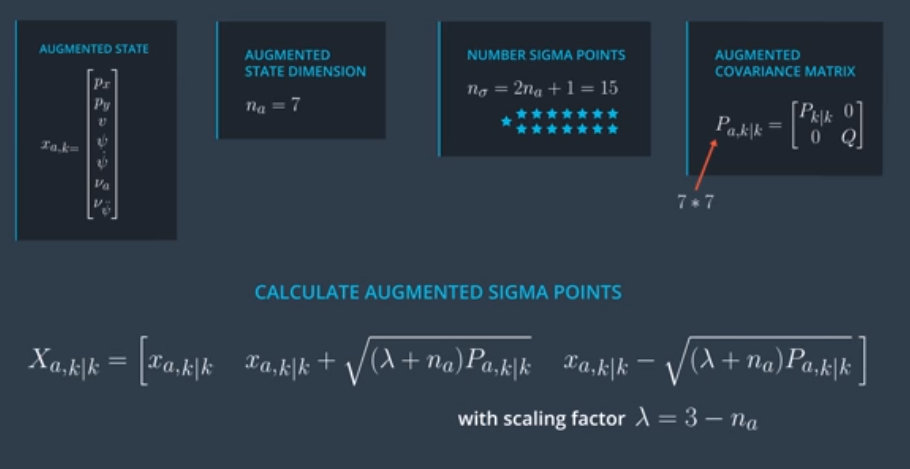


Now we need to consider how we represent the uncertainty of the covariance matrix with sigma points. The solution is called **Augmentation**.

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.

