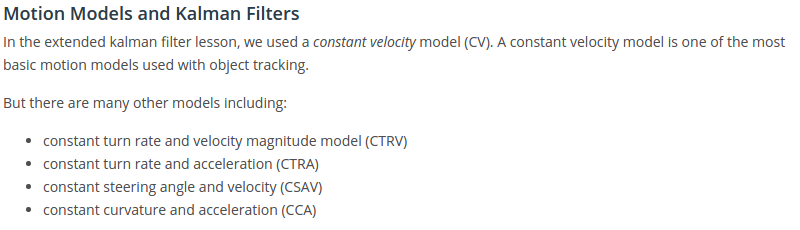
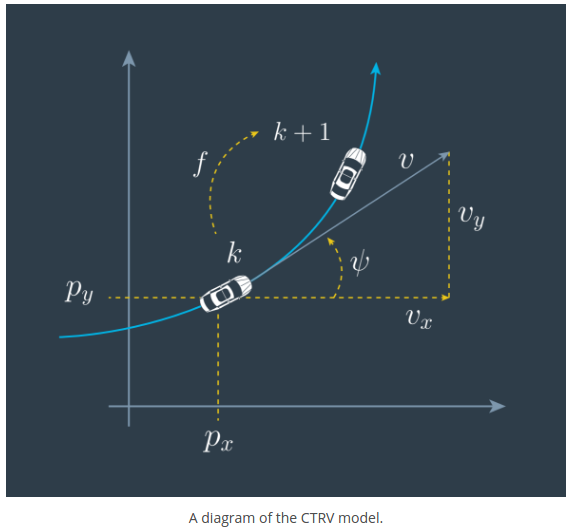
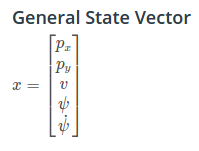
**LESSON 7.2-7.9: The CTRV Model**





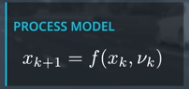
NOTE: in CTRV, we assume the yaw rate and velocity are constant !!

The CTRV model assumes a circular motion, described by 5 state parameters:

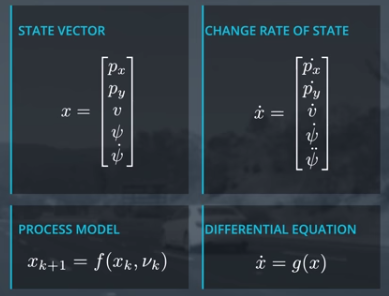


**Lesson 7.4: CTRV Differential Equation**

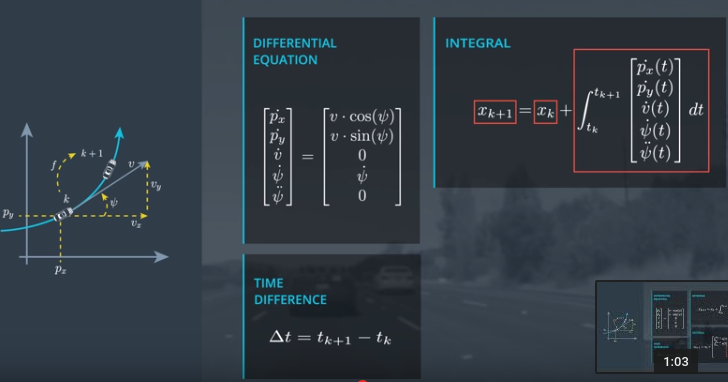
For CTRV model, **the process model *f***, which predicts where the car will be at time=k+1 if the state at time=k is known is given by:

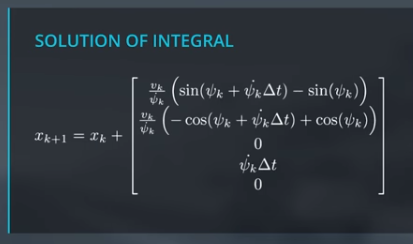
nu\_k = noise vector

This process model can be derived by first deriving the **change rate of state**:

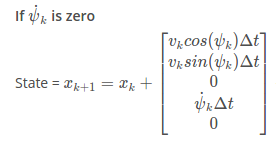


Then, integrate over time dt=t(k+1) – t(k): **(Lesson 7.5 & 7.6 & 7.7)**



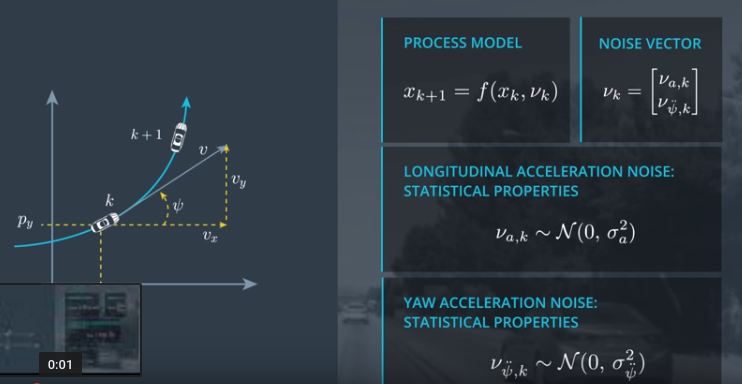


**NOTE 1**: Special case needed for psi\_dot(k) = 0.0:



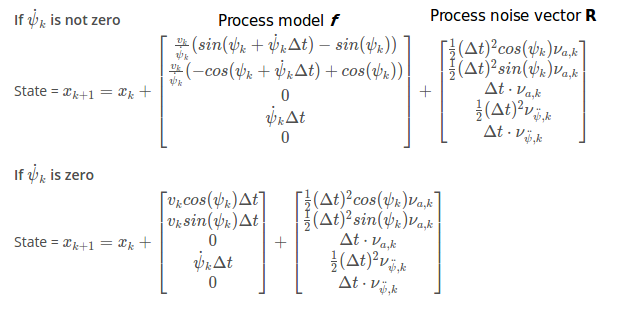
**NOTE 2**: Noise needs to be added to this process model to account for accelerations that we ignore in derivation of ***f***. See lesson 7.8.

**Lesson 7.8 – 7.9: CTRV Process Noise Vector R**

****

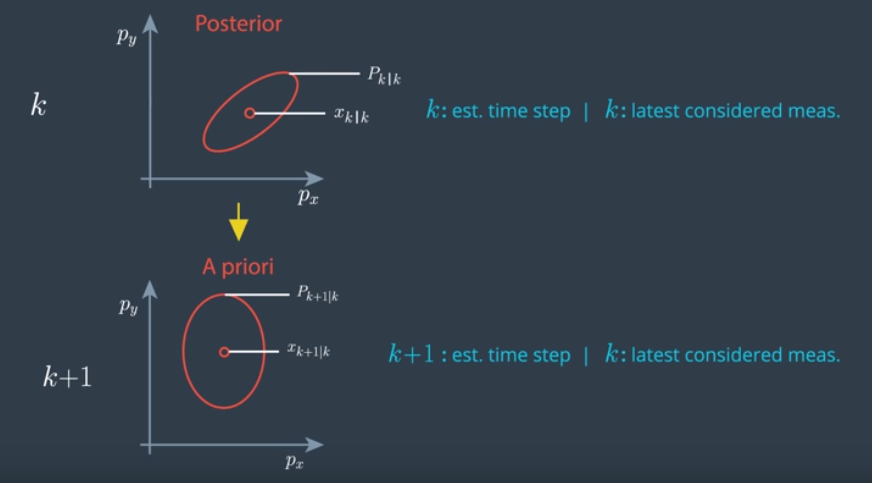
The acceleration noises are represented by a:

* Normal distribution
* With mean 0
* With variance sigma\_..^2

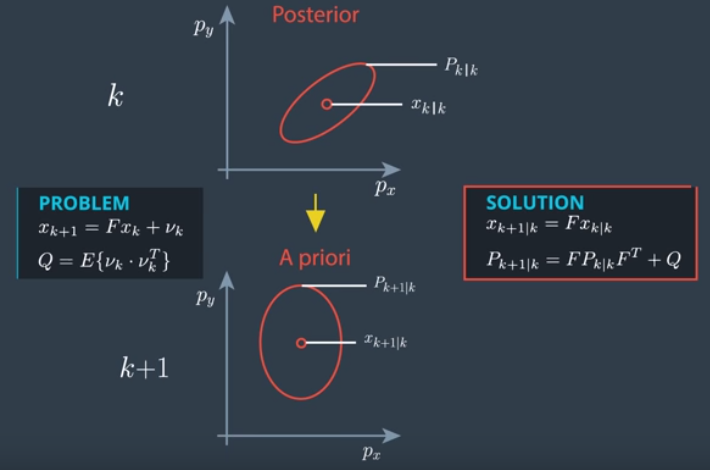
NOTE: These functions are not used directly, but through Sigma Points-→ 7.11-7.15

**Lesson 7.11-7.11: Predicting next state with unscented transformation**

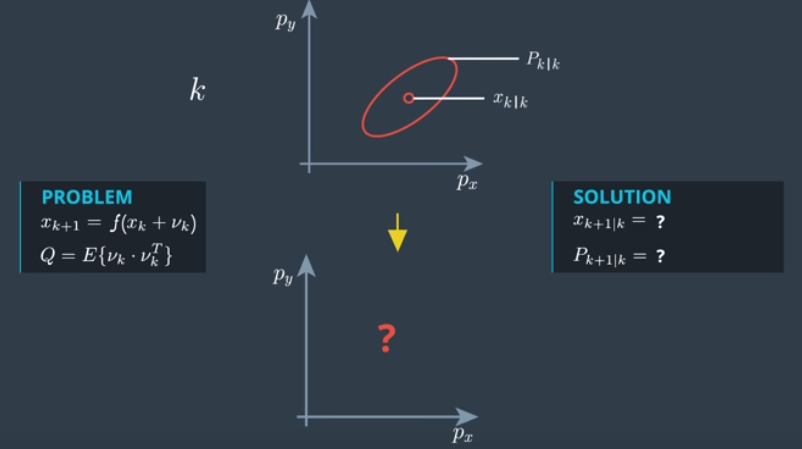
Having state mean and covariance at time k, predict it at time k+1:



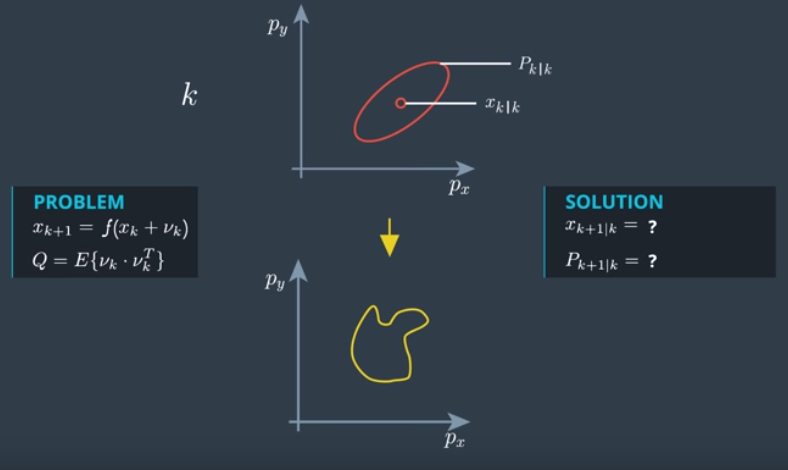
For a linear process model (KF & EKF), this was the (linear) problem & solution:n



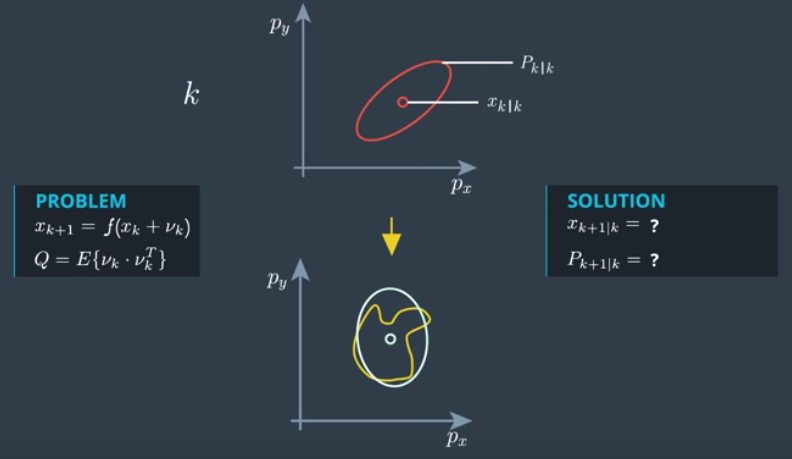
Prediction with a non-linear process model, with nonlinear function *f* as we just derived for the CTRV model, does not provide a normal distribution anymore, and cannot be calculated directly, but needs to be done numerically:



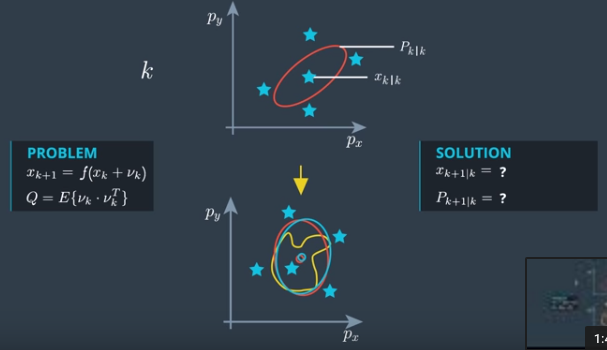
It can be done, using a particle filter that we will see later, during localization lessons, and it would look like this: (Note it is not a normal distribution any more!)



The unscented transformation (UKF) determines a normal distribution that is as close as possible to the actual distribution, so we can continue to work with a predicted state given by a mean and covariance.

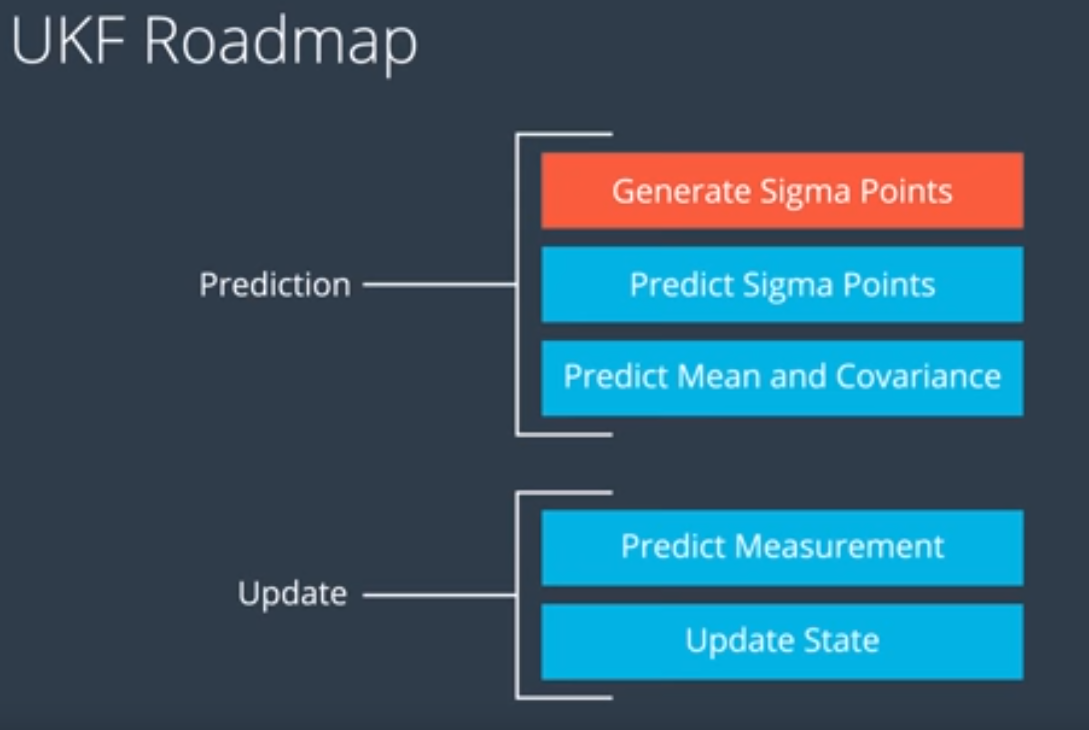


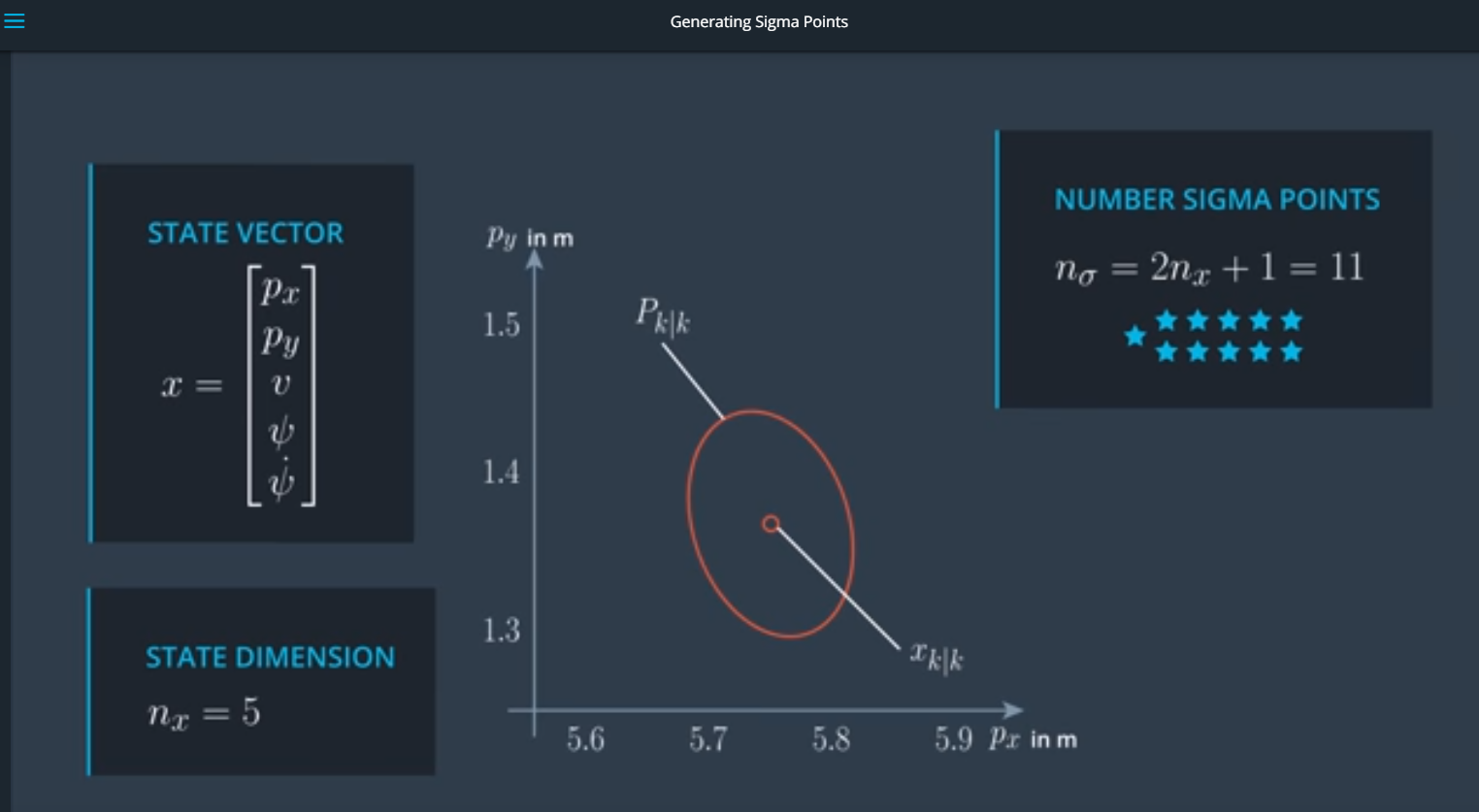
It is difficult to transfer the whole state space, but it is easy to do it for individual points. The unscented transformation does just that. It uses a number of **Sigma Points** at time k, calculates their predicted state at time k+1, and from that calculates the predicted mean & covariance at time k+1 as an approximation. These points are called Sigma Points, because they are chosen at the mean and in relation to the covariance:



NOTE: the Sigma Point approach could have been used for the linear process model case as well, and in that case the approximated state mean and covariance at time k+1 will be exactly the same as when doing the prediction directly! You would not do that though, because sigma points are more expensive than a direct, linear transformation.

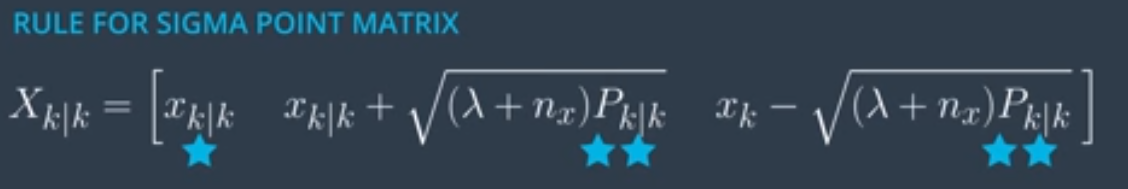
**LESSON 7.13-7.18: Generating Sigma Points**



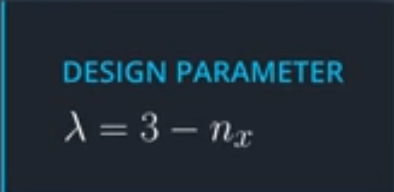


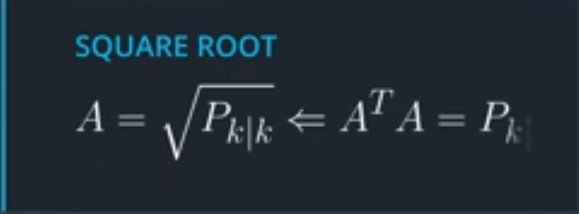


Each column is a sigma point !

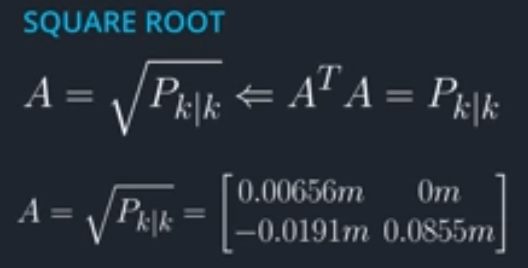
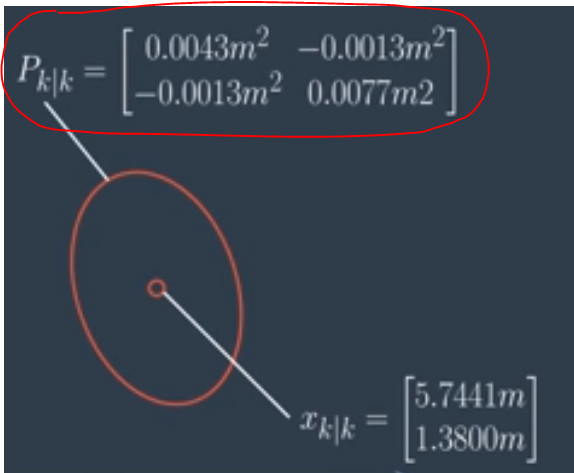


 = a design parameter!

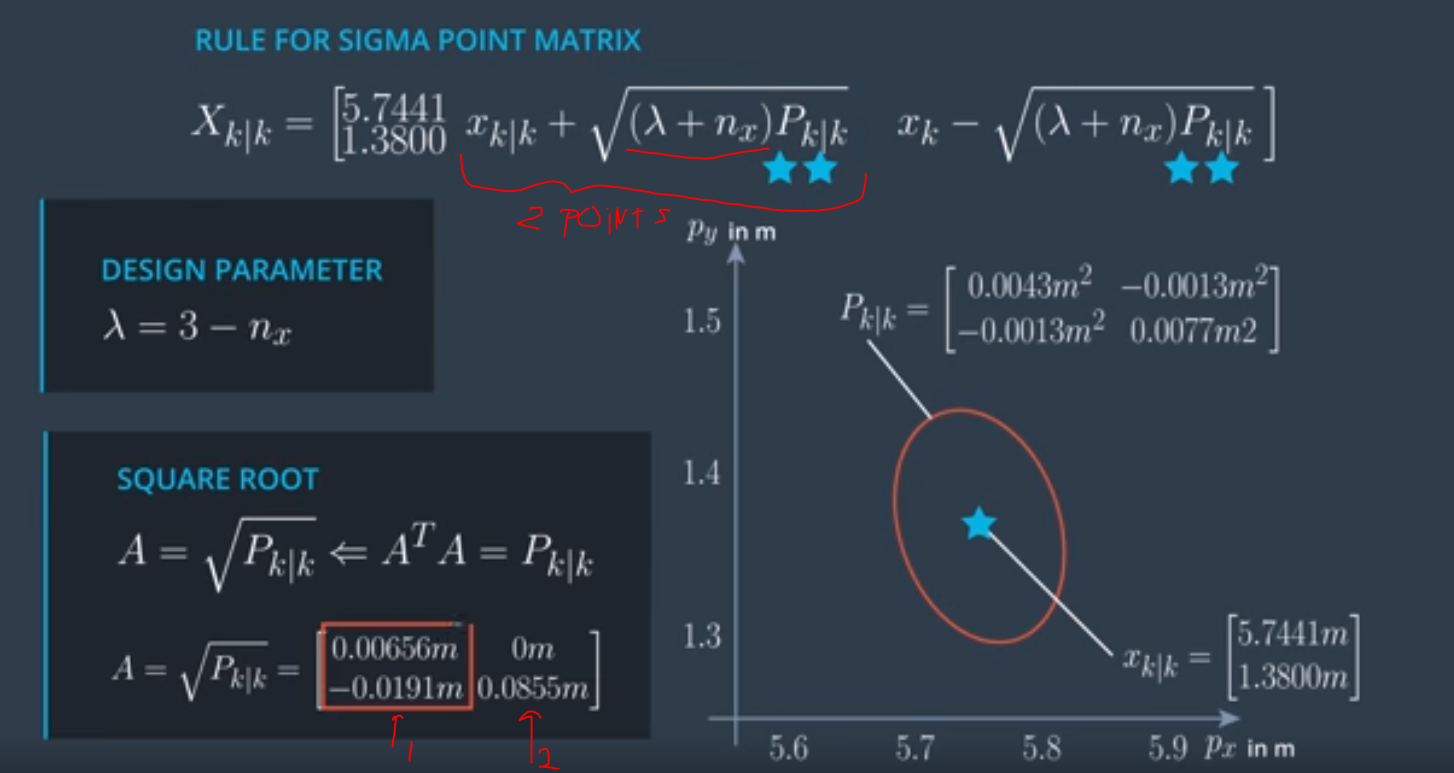
  Seems to give good results

 This is the square root of the co-variance matrix. This can be calculated with Cholesky decomposition, which results in a lower triangular matrix.

As example, to keep it simple, let’s use just (px, py), with symmetric 2x2 covariance matrix:



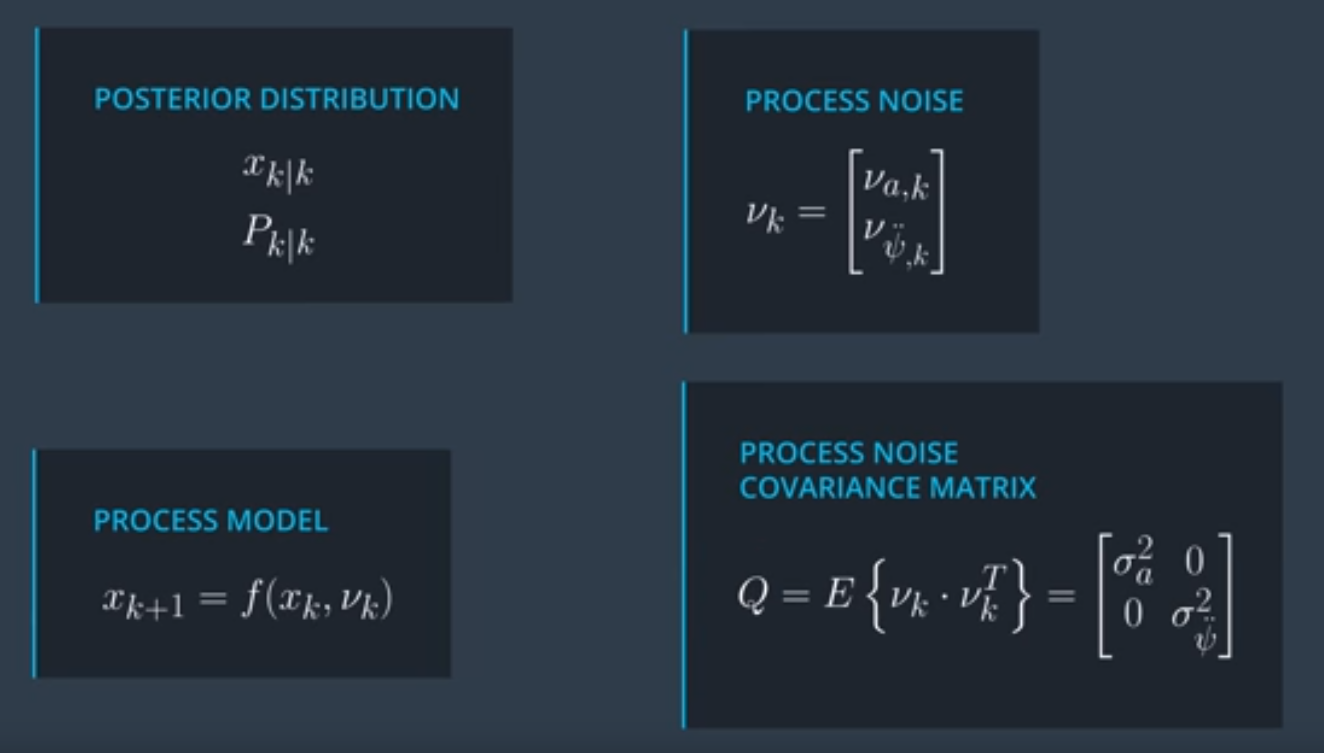
These values for A do not seem correct!

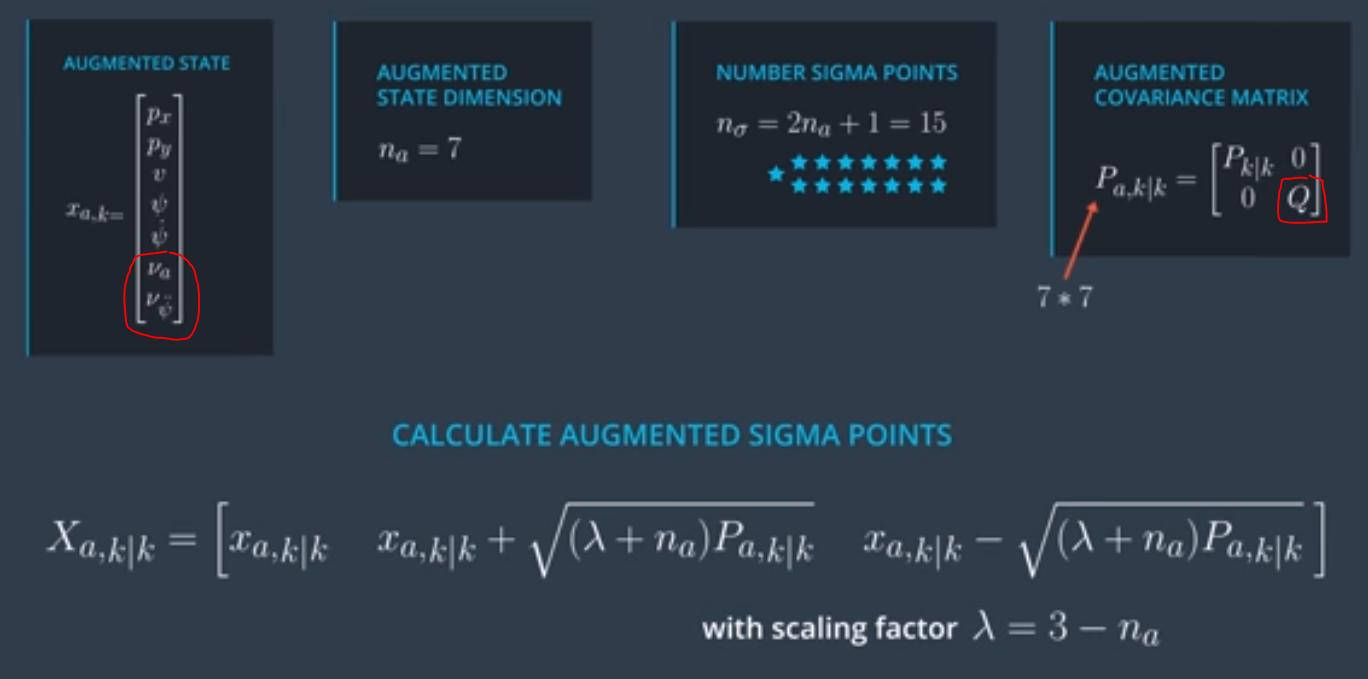




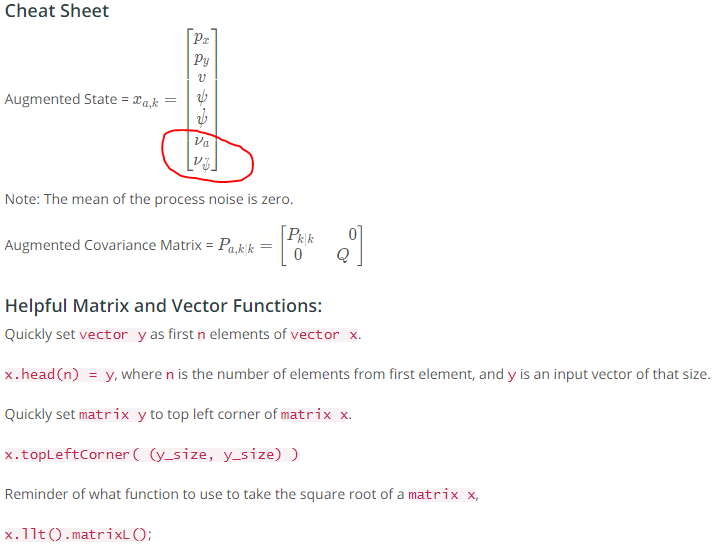
LESSON 7.16: UKF AUGMENTATION

IMPORTANT: See beginning of this lesson for explanation of Process Noise and Process Noise Covariance Matrix (Q), to distinguish between meaning in context of Regular/Extended Kalman filters vs Unscented Kalman Filters. It is actually much simpler for Unscented Kalman Filters.

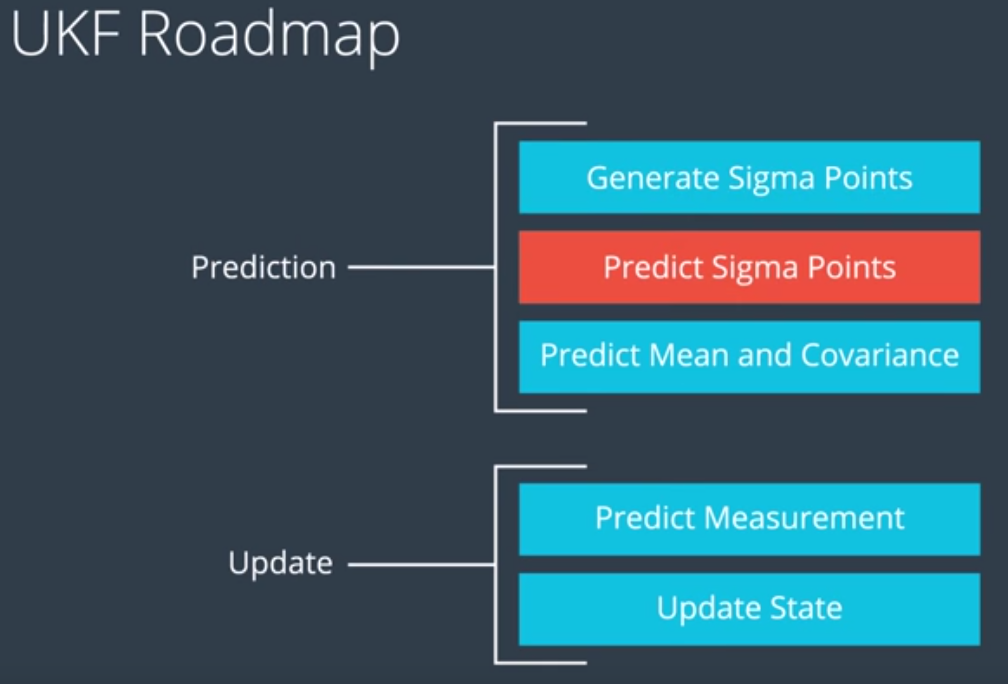


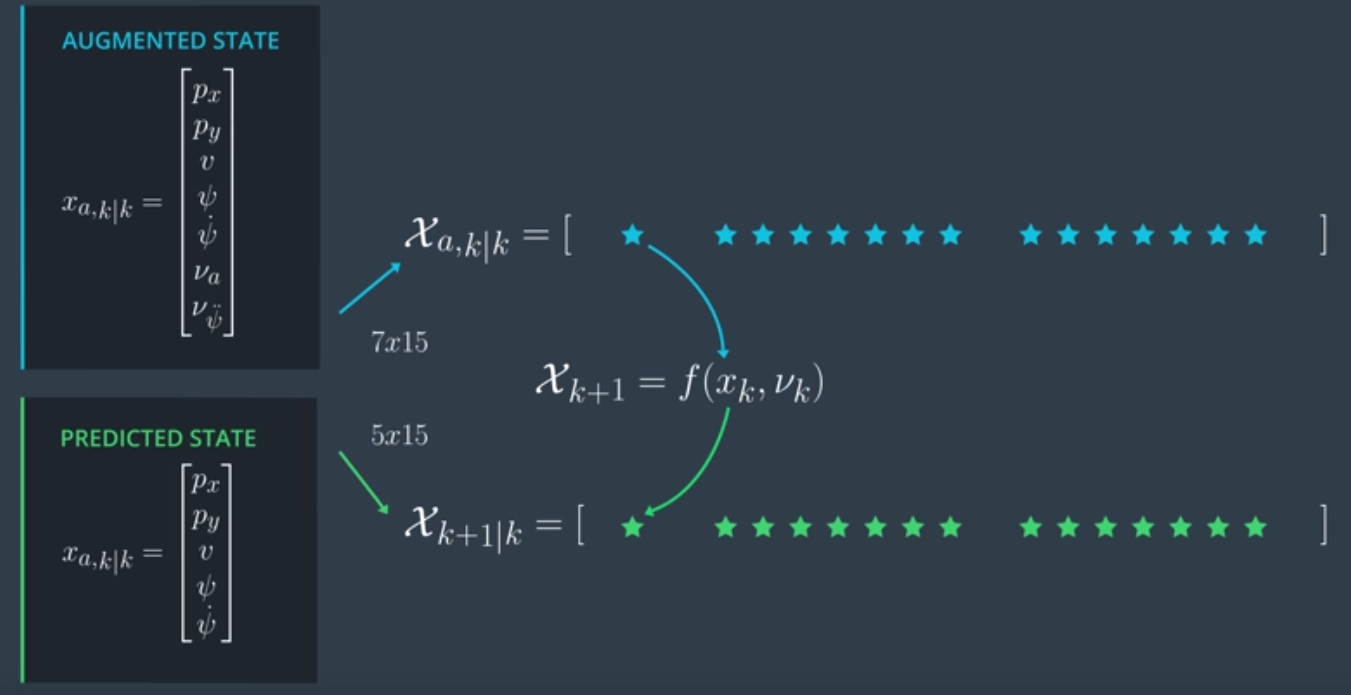


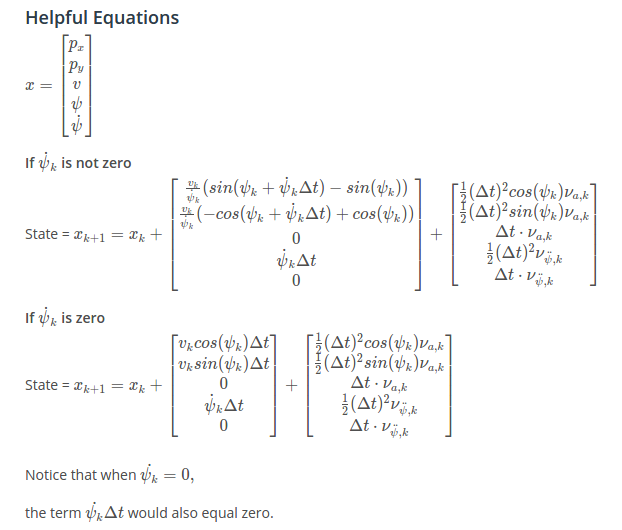
Lesson 7.17 – Cheat Sheet



**LESSON 7.19-7.21: Predict Sigma Points**



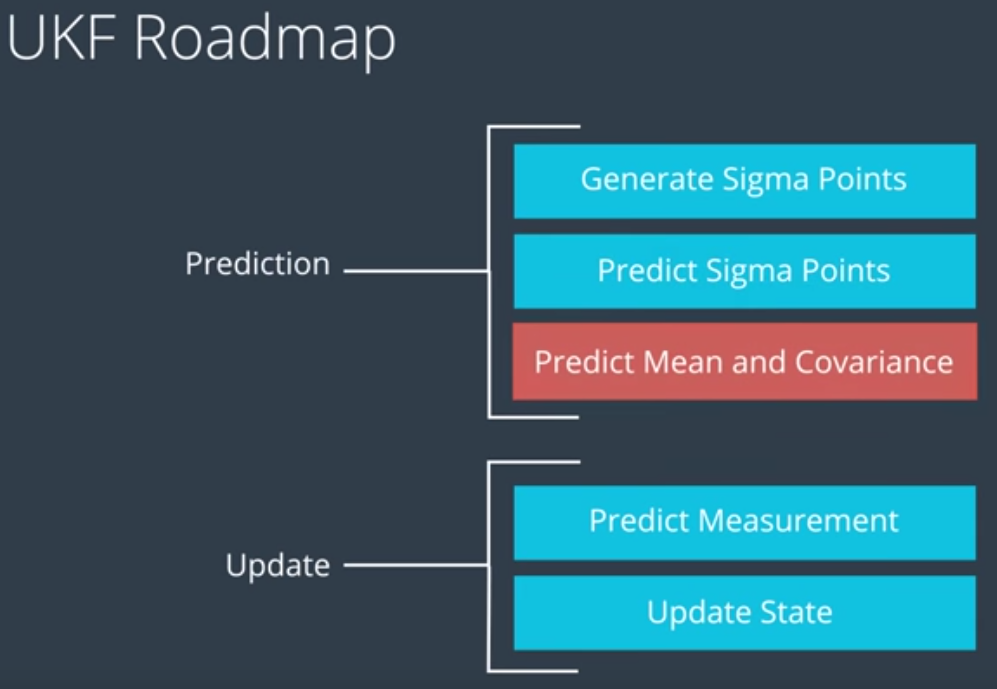


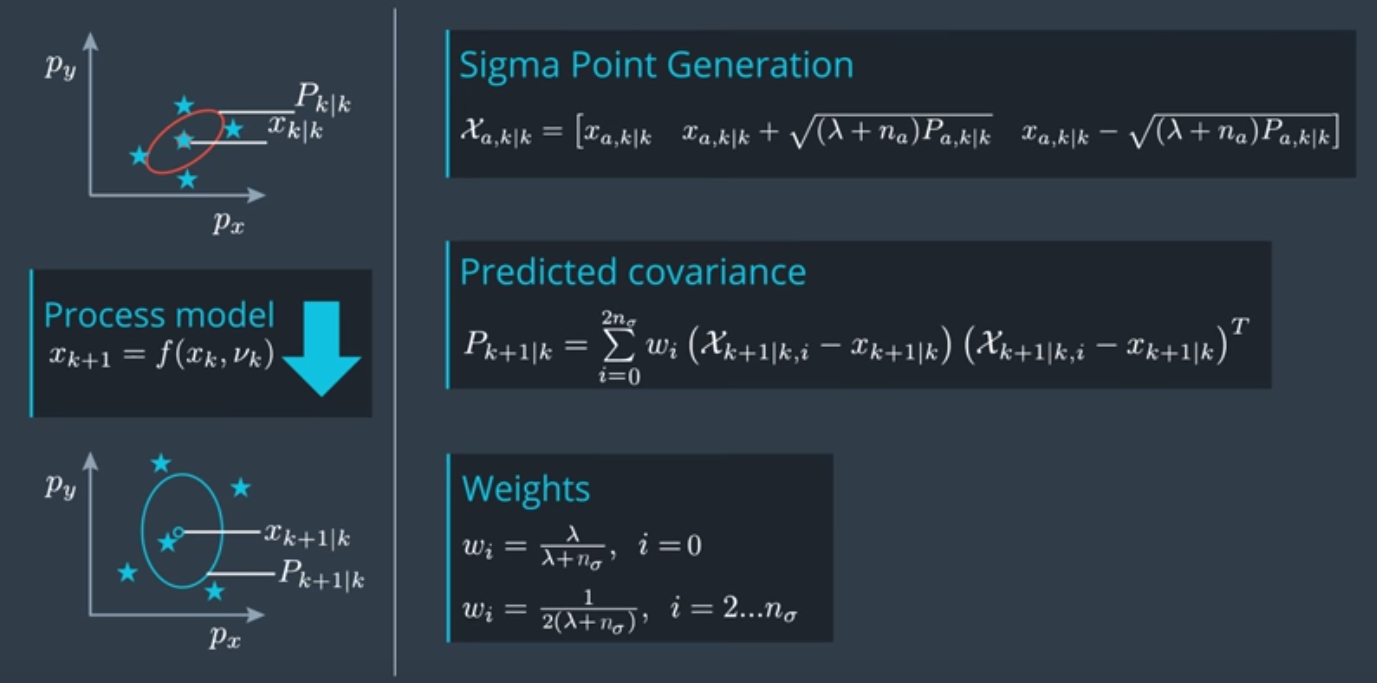


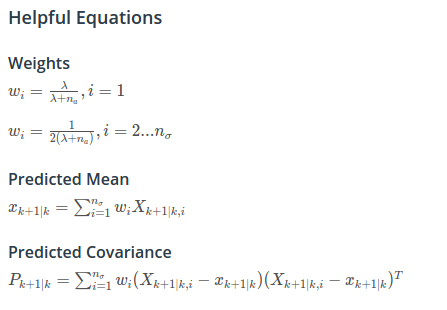
Process model ***f***

Process noise vector **R**

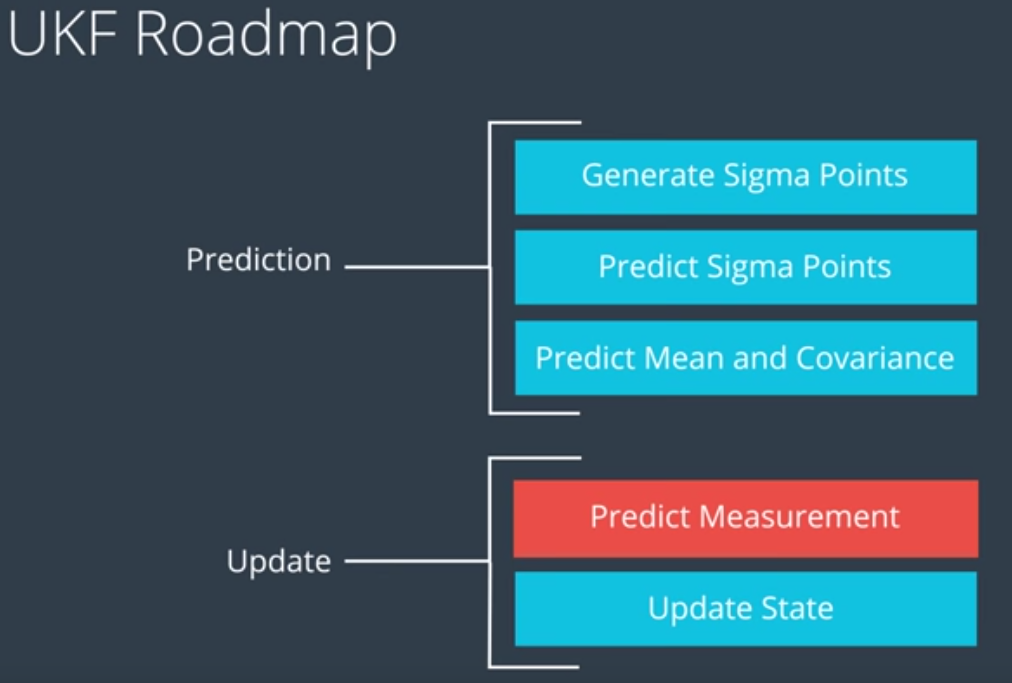
**LESSON 7.22-7.24: Predict Mean and Covariance**

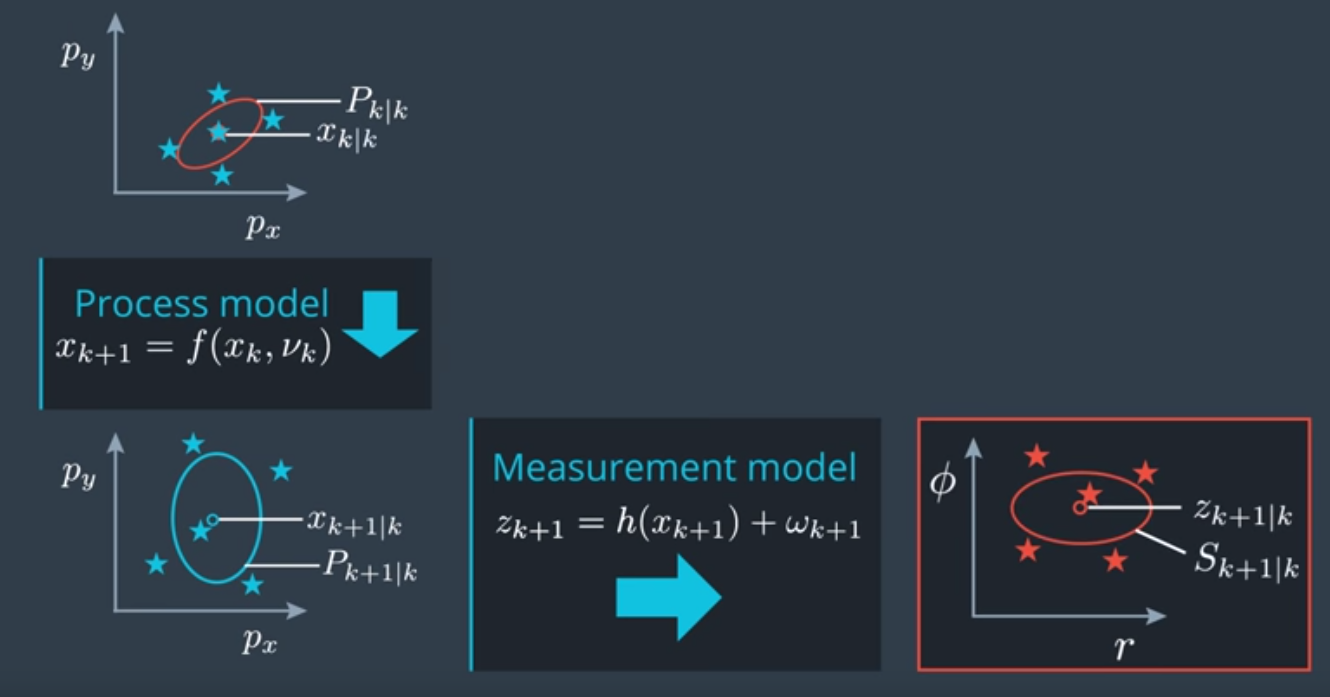






**LESSON 7.25-7.27: Predict Measurement**

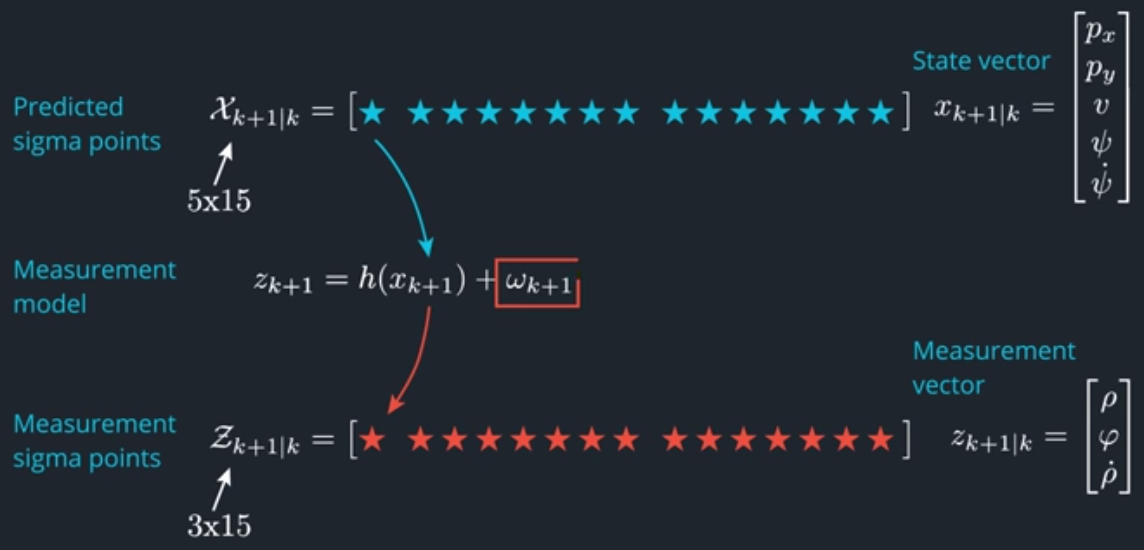
****

Use same **unscented transformation approach** as during prediction, to transform the predicted state into the measurement space.

* Re-use predicted sigma points to represent predicted state.
* Transform into measurement space to predict the expected measurement
* Skip augmentation step, because measurement noise (omega) is linear.

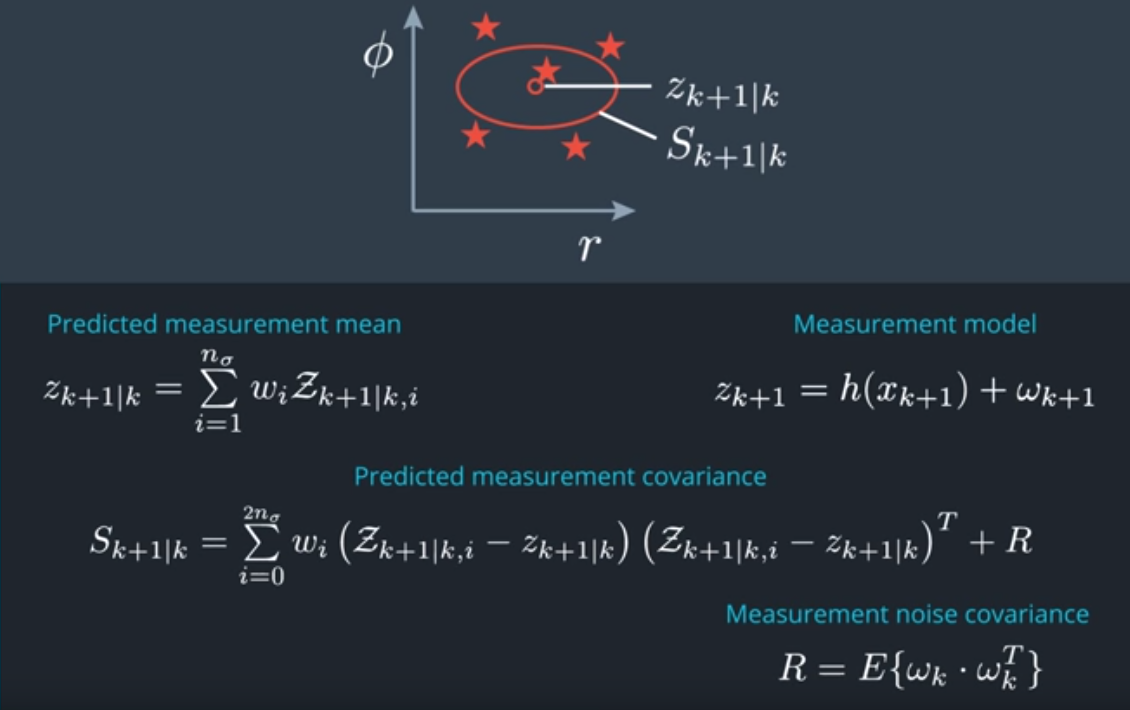
For Radar: measurement space is: rho, phi, rho\_dot

Omega = measurement noise. Just use zero for now. We will account for it later.



Predicted Sigma Points in Measurement Space

R = Measurement noise covariance, is done instead of augmentation!



mean

Covariance matrix

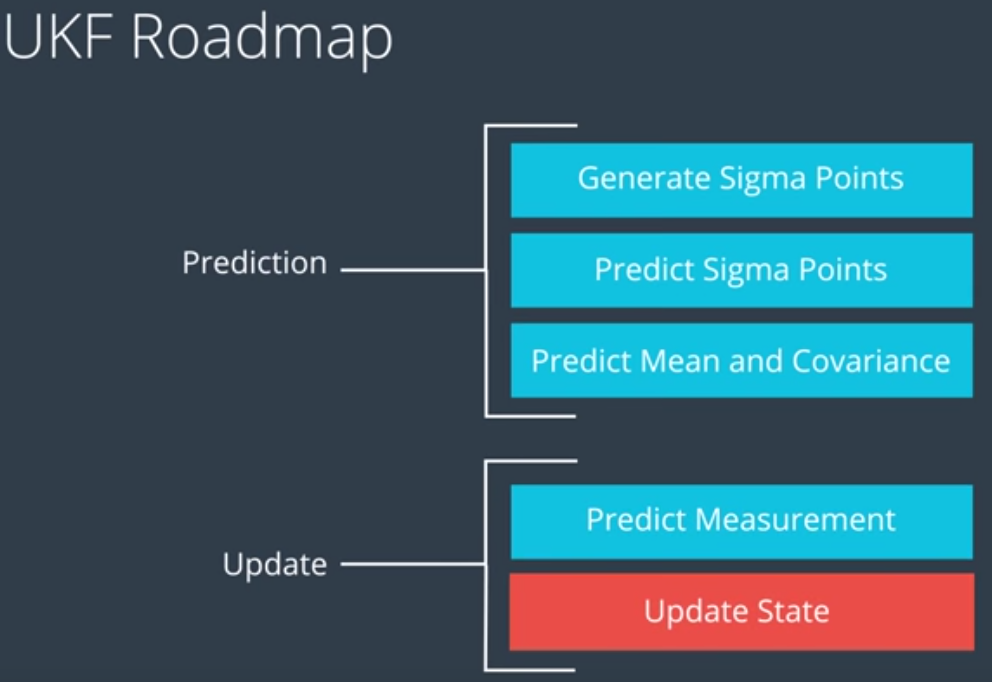
Predicted Sigma Points in Measurement Space

I

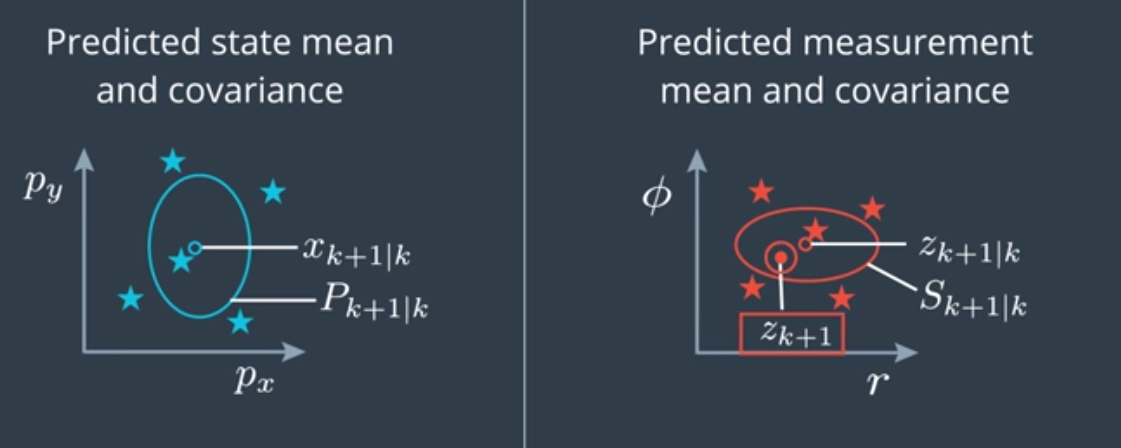


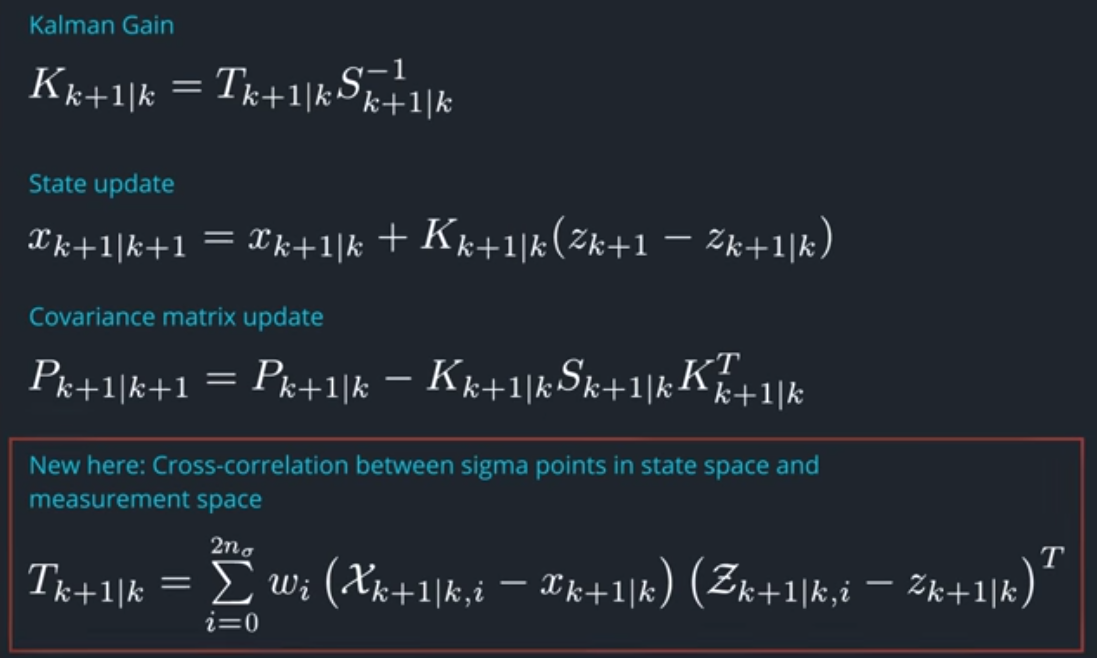
*h*

**LESSON 7.28-7.30: Update Step**



We now use the actual measurements (z\_k+1) during the update step:

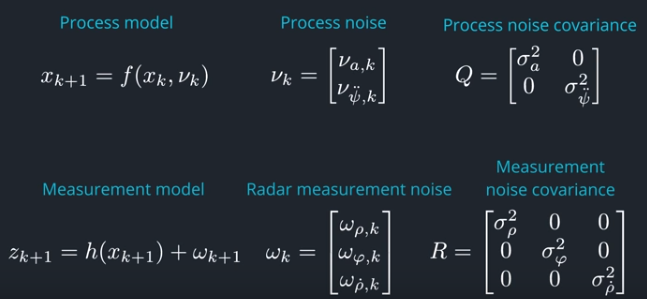






**LESSON 7.31: Noise Parameters**

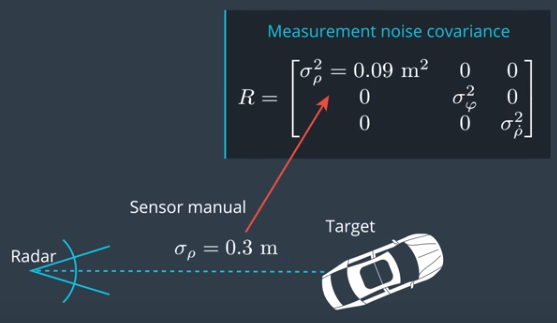
This is applicable not just to Kalman filters, but to any Bayesian filters.



The noise covariance matrices quantify how strong the noise is.

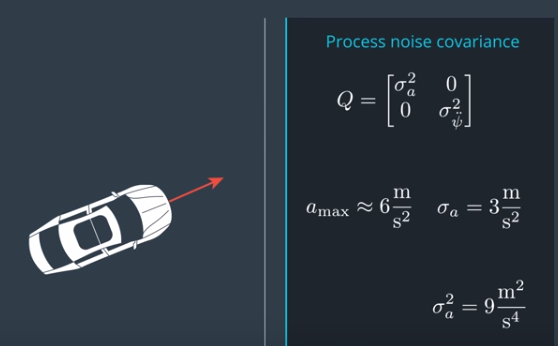
Question is where to get these values from?

**For measurement noise**, you can find the standard deviations in the manual of the sensor:

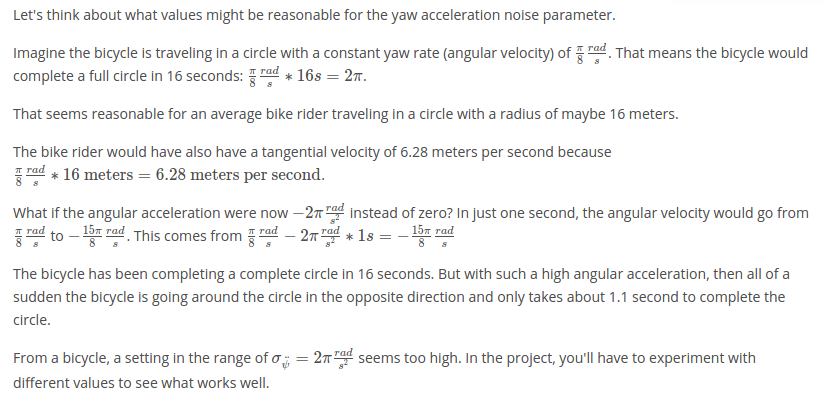


**For process noise**, it is more complicated, because for example, the accelerations that we ignore are causing this, and we do not know what the accelerations are. A rule of thumb is the following:

* Estimate the maximum acceleration you expect in your environment.
  + For example, in an urban environment this is 6 m/s2
* Then, use HALF of this maximum value as process noise

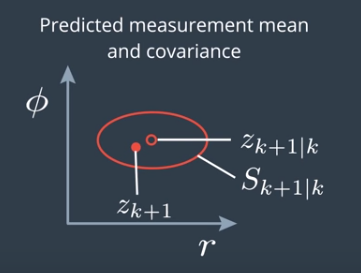


* If application requires it reacts fast to changes, then use a larger value.
* If application requires a smooth reaction, then use a smaller value.



**Always run a consistency check on the filter**

A filter is consistent if actual measurement falls within predicted measurement and its its noise covariance:



OK



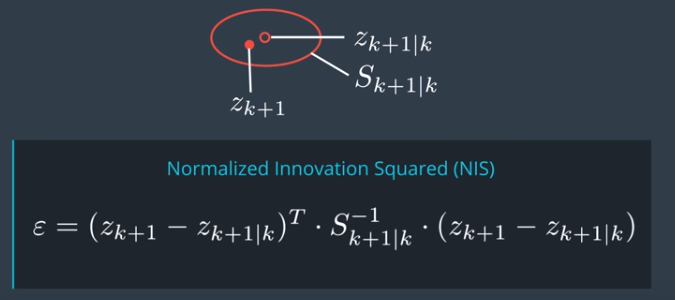
NOK

Filter over-estimates accuracy

NOK

Filter under-estimates accuracy

Always use Normalized Innovation Squared (NIS) the check consistency of a filter:



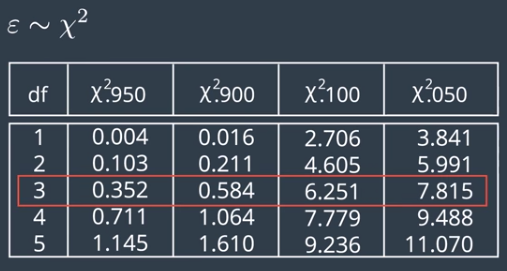
The ‘innovation’ is the difference between the predicted and actual measurement.

‘Normalized’ means it is put in relation to the covariance matrix S.

NIS is just a scalar number!

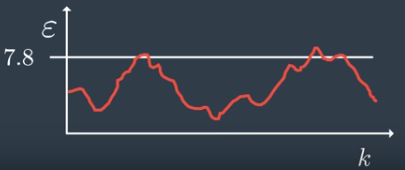
What should you look for? Depends on the **degrees of freedom (df)**

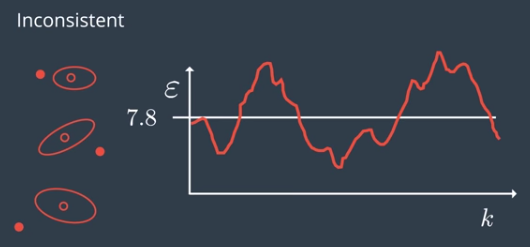
For radar measurement, we have 3 dimensions:



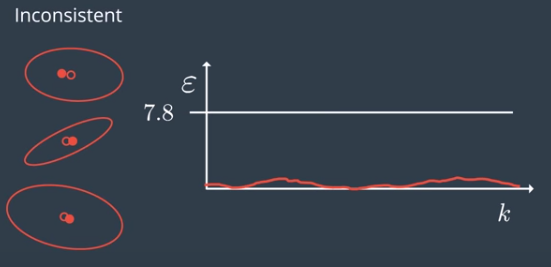
The 4th column of this table says that in 95% of cases, for df=3, the NIS will be 7.815 or higher

Use this information to plot the NIS for each timestep k, and check that it goes over that 95% value about 5% of the time:

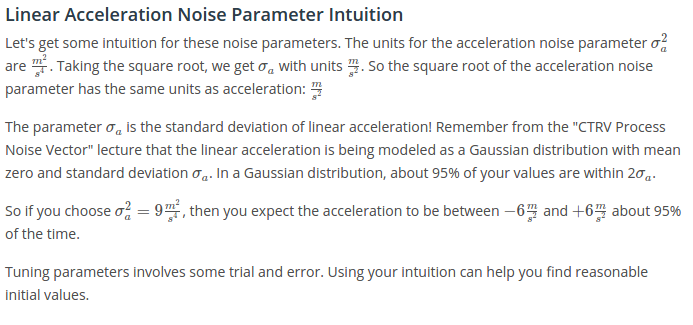
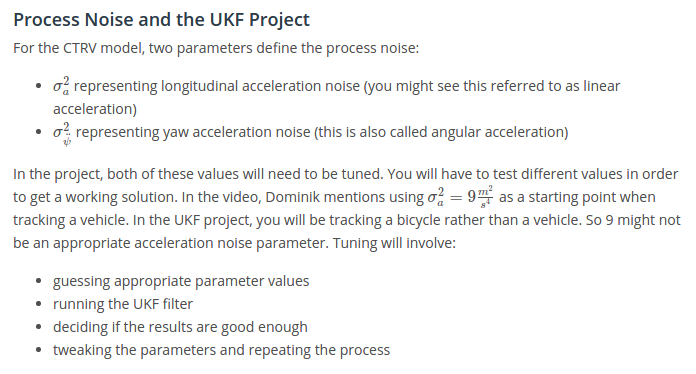
OK:

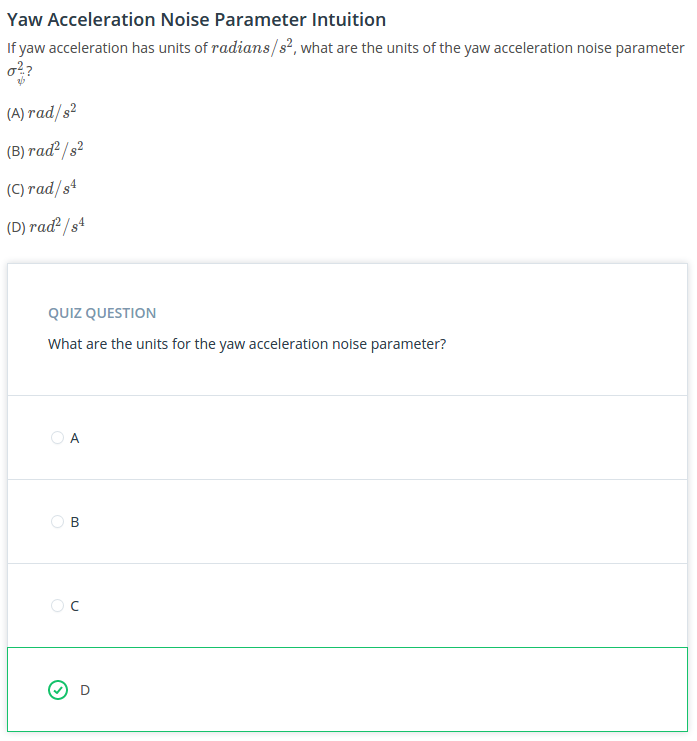
NOK:

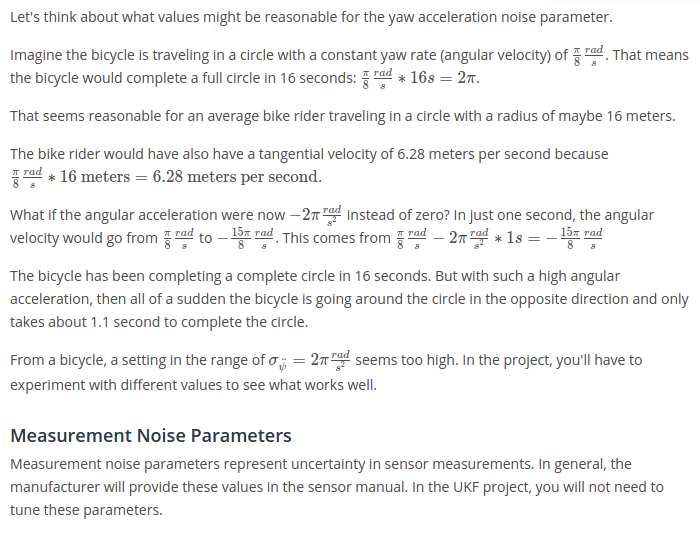
=> try increasing process noise. We are over-estimating accuracy

NOK:

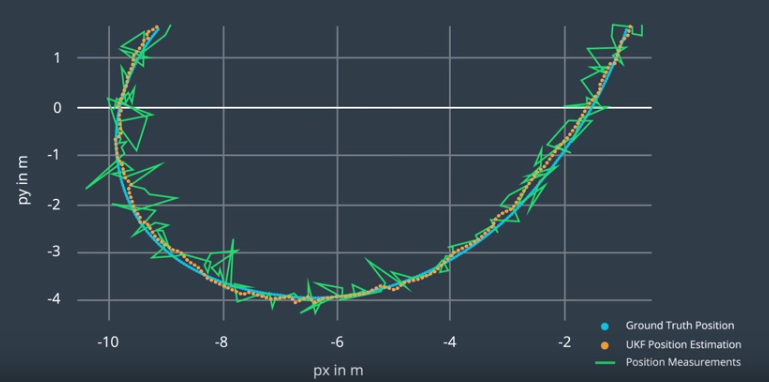
=> try decreasing process noise. We are under-estimating accuracy.



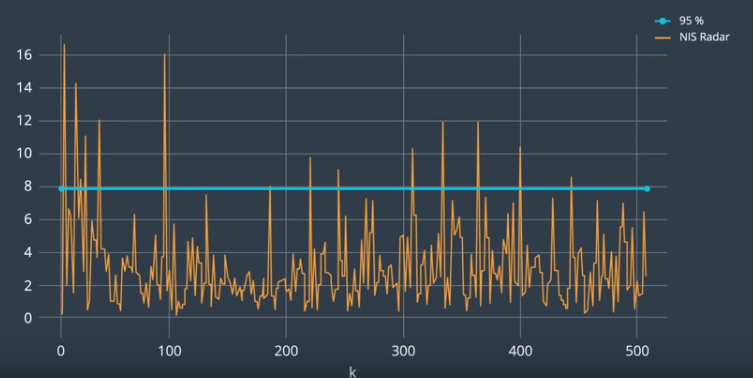




**Lesson 32: Project Results to achieve**



NIS FOR RADAR:



NIS FOR LIDAR (LASER):

Note that 95% line will be at different level, because it is a 2 df state:

→ 95% line is at 5.991 (See table above)

