Mathematic Formula Note of Unscented Kalman Filter with CTRV model

2017, Sep 19

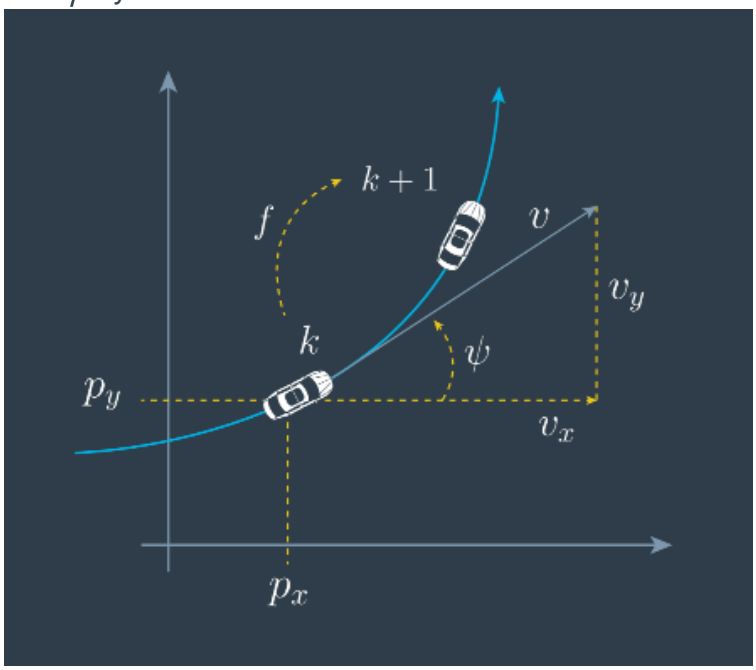
CTRV Model (constant turn rate and velocity magnitude model)

Here we work with a moving object of interest under CTRV nonlinear motion model which assumes the object can move straight, but they can also move with a constant turn rate and a const velocity magnitude.

1. The State Vector of CTRV model:

$$x = [\, p_x \quad p_y \quad v \quad \psi \quad \dot{\psi} \,]^T$$

- v : speed which describes the magnitude
- ψ : yaw angle which describes the orientation
- $\dot{\psi}$: yaw rate



the picture above is from Udacity Self-Driving Nanodegree project under CC BY-NC-ND 4.0.

2. Change rate of state:

$$\dot{x} = [\dot{p_x} \quad \dot{p_y} \quad \dot{v} \quad \dot{\psi} \quad \ddot{\psi}]^T$$

$$= [v \cdot cos(\psi) \quad v \cdot sin(\psi) \quad 0 \quad \dot{\psi} \quad 0]^T$$

3. Time difference: $\Delta t = t_{k+1} - t_k$

4. Process model (predicts the state at time step k+1):

$$x_{k+1} = f(x_k, \nu_k) = x_k + \begin{bmatrix} \frac{v_k}{\dot{\psi}_k}(sin(\psi_k + \dot{\psi}_k \Delta t) - sin(\psi_k)) \\ \frac{v_k}{\dot{\psi}_k}(-cos(\psi_k + \dot{\psi}_k \Delta t) + cos(\psi_k)) \\ 0 \\ \dot{\psi}_k \Delta t \\ 0 \end{bmatrix} + \begin{bmatrix} \frac{1}{2}(\Delta t)^2 cos(\psi_k) \cdot \nu_{a,k} \\ \frac{1}{2}(\Delta t)^2 sin(\psi_k) \cdot \nu_{a,k} \\ \Delta t \cdot \nu_{a,k} \\ \frac{1}{2}(\Delta t)^2 \cdot \nu_{\ddot{\psi},k} \\ \Delta t \cdot \nu_{\ddot{\psi},k} \end{bmatrix}$$

Note: We should be careful when $\dot{\psi}_k$ = 0, to avoid divison by 0
(This is at the situation when the yaw angle is not change, the car is going straight)

if $\dot{\psi}_k$ is zero

$$x_{k+1} = x_k + \begin{bmatrix} v_k cos(\psi_k) \Delta t \\ v_k sin(\psi_k) \Delta t \\ 0 \\ \dot{\psi}_k \Delta t \\ 0 \end{bmatrix} + \begin{bmatrix} \frac{1}{2}(\Delta t)^2 cos(\psi_k) \cdot \nu_{a,k} \\ \frac{1}{2}(\Delta t)^2 sin(\psi_k) \cdot \nu_{a,k} \\ \Delta t \cdot \nu_{a,k} \\ \frac{1}{2}(\Delta t)^2 \cdot \nu_{\ddot{\psi},k} \\ \Delta t \cdot \nu_{\ddot{\psi},k} \end{bmatrix}$$

- consider only deterministic part:

$$\begin{aligned} x_{k+1} &= f(x_k) \\ &= x_k + \int_{t_k}^{t_{k+1}} \begin{bmatrix} v(t) \cdot cos(\psi(t)) & v(t) \cdot sin(\psi(t)) & 0 & \dot{\psi} & 0 \end{bmatrix}^T dt \\ &= x_k + \begin{bmatrix} \int_{t_k}^{t_{k+1}} v(t) \cdot cos(\psi(t)) dt \\ \int_{t_k}^{t_{k+1}} v(t) \cdot sin(\psi(t)) dt \\ 0 \\ \dot{\psi}_k \Delta t \\ 0 \end{bmatrix} = x_k + \begin{bmatrix} v_k \int_{t_k}^{t_{k+1}} cos(\psi_k + \dot{\psi}_k \cdot (t - t_k)) dt \\ v_k \int_{t_k}^{t_{k+1}} sin(\psi_k + \dot{\psi}_k \cdot (t - t_k)) dt \\ 0 \\ \dot{\psi}_k \Delta t \\ 0 \end{bmatrix} \\ &= x_k + \begin{bmatrix} \frac{vk}{\dot{\psi}_k}(sin(\psi_k + \dot{\psi}_k \Delta t) - sin(\psi_k)) \\ \frac{vk}{\dot{\psi}_k}(-cos(\psi_k + \dot{\psi}_k \Delta t) + cos(\psi_k)) \\ 0 \\ \dot{\psi}_k \Delta t \\ 0 \end{bmatrix} \end{aligned}$$

- stochastic part (noise vector): $\nu_k = [\nu_{a,k} \quad \nu_{\ddot{\psi},k}]^T$

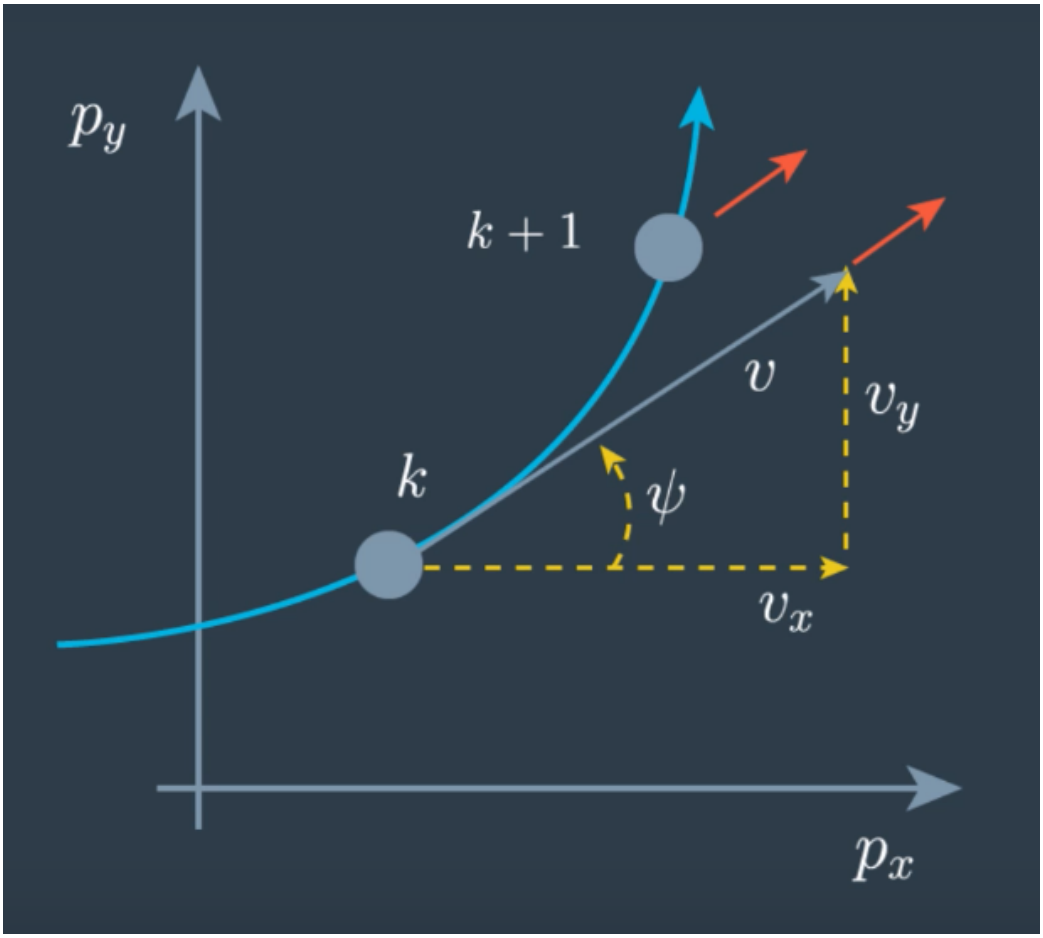
  - longitudinal acceleration noise: $\nu_{a,k} \sim \mathcal{N}(0, \sigma_a^2)$

- yaw acceleration noise: $\nu_{\ddot{\psi},k} \sim \mathcal{N}(0, \sigma_{\ddot{\psi}}^2)$

- Assume that both longitudinal and yaw acceleration noise are constant between k and k+1 step:

  (also we assume that the car wre driving prefectly straight, so we could calculate x accelertation and y acceletration accordingly.)
  (this approximation will be okay as long as the yaw rate is not too high!)

$$f(\nu_k) = \begin{bmatrix} \frac{1}{2}(\Delta t)^2 cos(\psi_k) \cdot \nu_{a,k} \\ \frac{1}{2}(\Delta t)^2 sin(\psi_k) \cdot \nu_{a,k} \\ \Delta t \cdot \nu_{a,k} \\ \frac{1}{2}(\Delta t)^2 \cdot \nu_{\ddot{\psi},k} \\ \Delta t \cdot \nu_{\ddot{\psi},k} \end{bmatrix}$$



the picture above is from Udacity Self-Driving Nanodegree project under CC BY-NC-ND 4.0.

UKF (Unscented Kalman Filter)

What makes UKF and EKF (Extended Kalman Filter) different is the method they uses to tackle with non-linear motion model and measurement model.

- EKF uses the Jacobian matrix to linearize to non-linear functions.

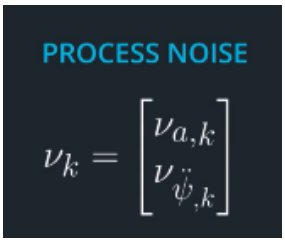
- UKF takes representative points of the whole distribution called **simga points** from a Gaussian distribution, and put them into the non-linear function which is called **unscented transformation**. And it will come out the **corresponded simga points** in the predicted or measurement state space. Then we can calculate the mean vector and covariance matrix from these corresponded simga points to get the Gaussian distribution of the predicted or measurement state space.

We can split the unscented prediction stage into three parts. 1. generate simga points from the last updated state and covariance matrix. 2. insert them into the process function or measurement function to do unscented transformation. 3. calculate the predicted mean and covariance matrix from the predicted or measurement sigma points.

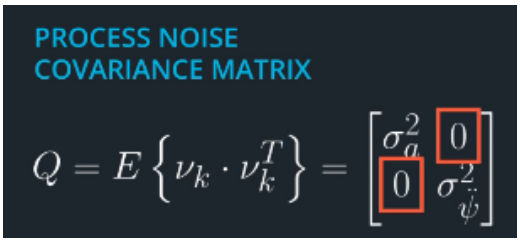
Prediction Stage

1. Generate Simga Points

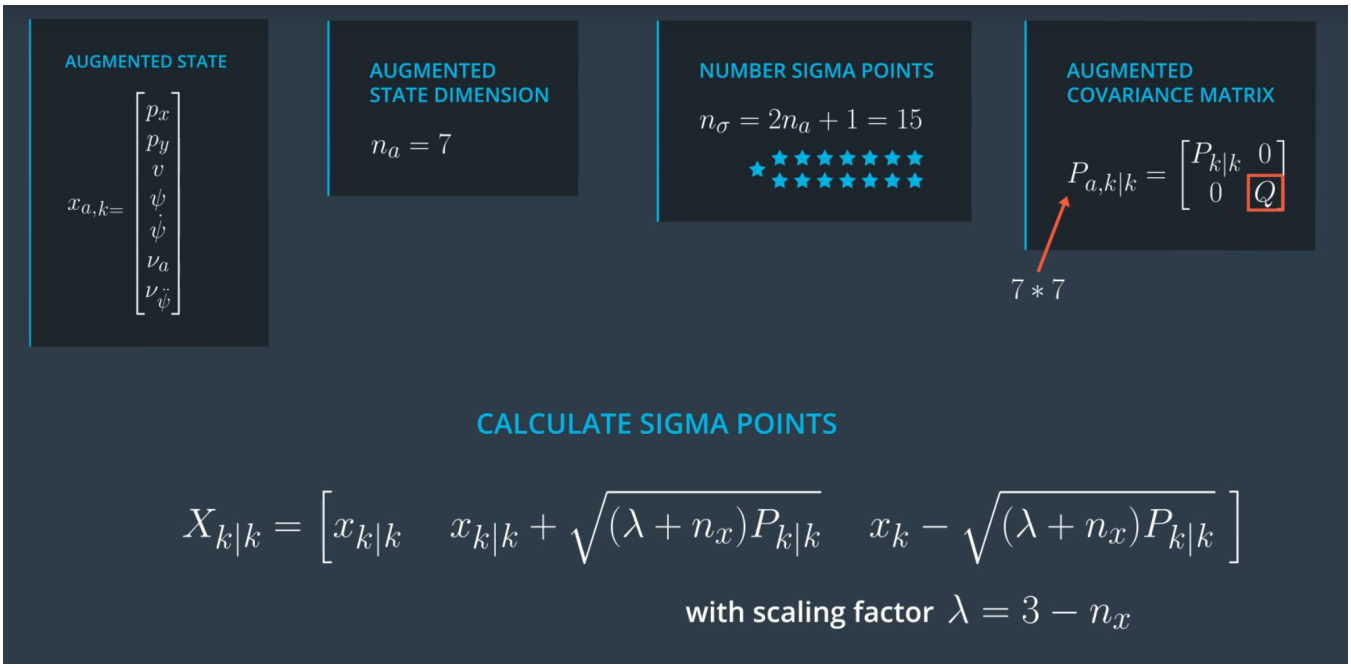
At the first step of prediction stage, we generate simga points from the last updated state and covariance matrix. With CTRV model, we have state dimension $n_x = 5$. However, we should also consider the process noise vector, which has two-dimension, because it also has a non-linear effect. $n_{aug} = 5 + 2 = 7$. We will choose $2n_{aug} + 1$ sigma points.



the picture above is from Udacity Self-Driving Nanodegree project under CC BY-NC-ND 4.0.



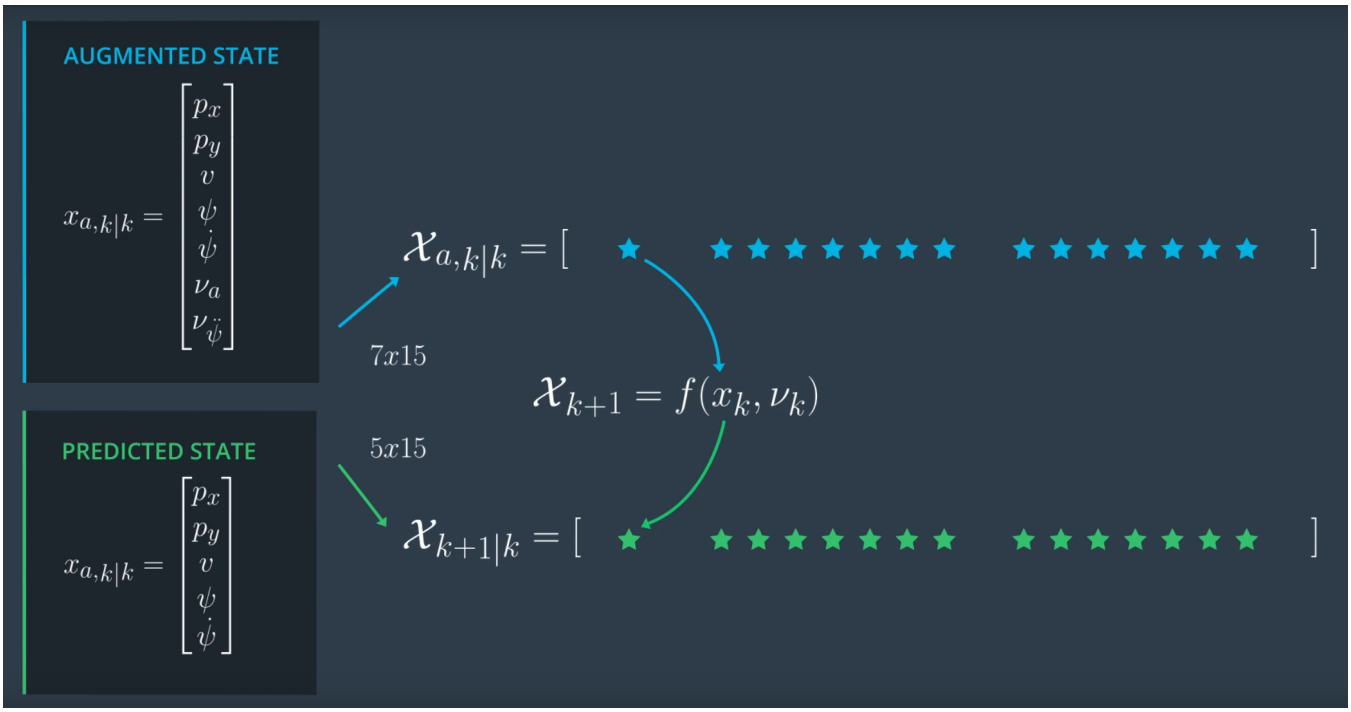
the picture above is from Udacity Self-Driving Nanodegree project under CC BY-NC-ND 4.0.



the picture above is from Udacity Self-Driving Nanodegree project under CC BY-NC-ND 4.0.

2. Prediction Step

In second step, we simply insert every sigma point into the process model of CTRV to get the prdicted simga points.



the picture above is from Udacity Self-Driving Nanodegree project under CC BY-NC-ND 4.0.

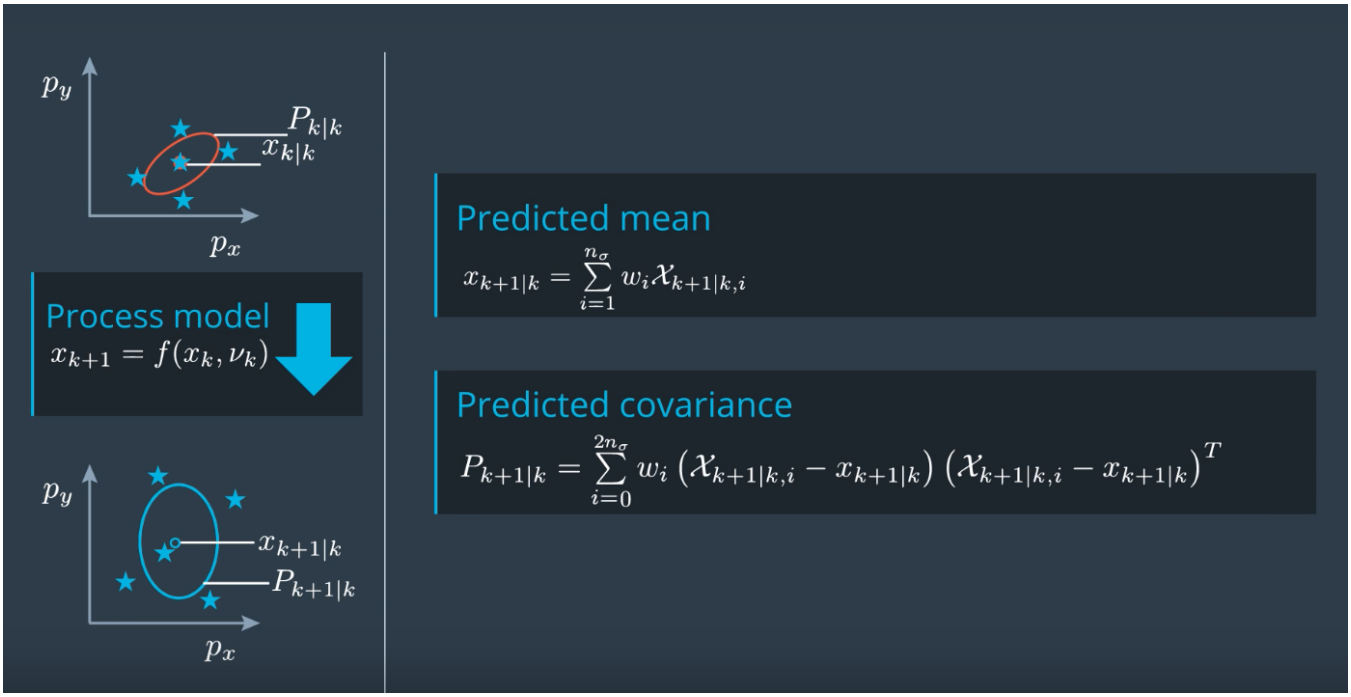
3. Calculate Mean and Covariance from the Predicted Sigma Points

In this step, we calculate predicted state mean vector and predicted state covariance matrix from the predicted sigma points. When generate simga points, we use lambda to get spreading value

from mean. Now we consider to do the inverse calculation. (There are several ways to calculate weights, we just stick to the following one.)

- Weights:

$$\omega_i = \frac{\lambda}{\lambda+n_{aug}} \quad , \quad i = 0$$
$$\omega_i = \frac{1}{2(\lambda+n_{aug})} \quad , \quad i = 1 \ldots 2 * n_{aug}$$



the picture above is from Udacity Self-Driving Nanodegree project under CC BY-NC-ND 4.0.

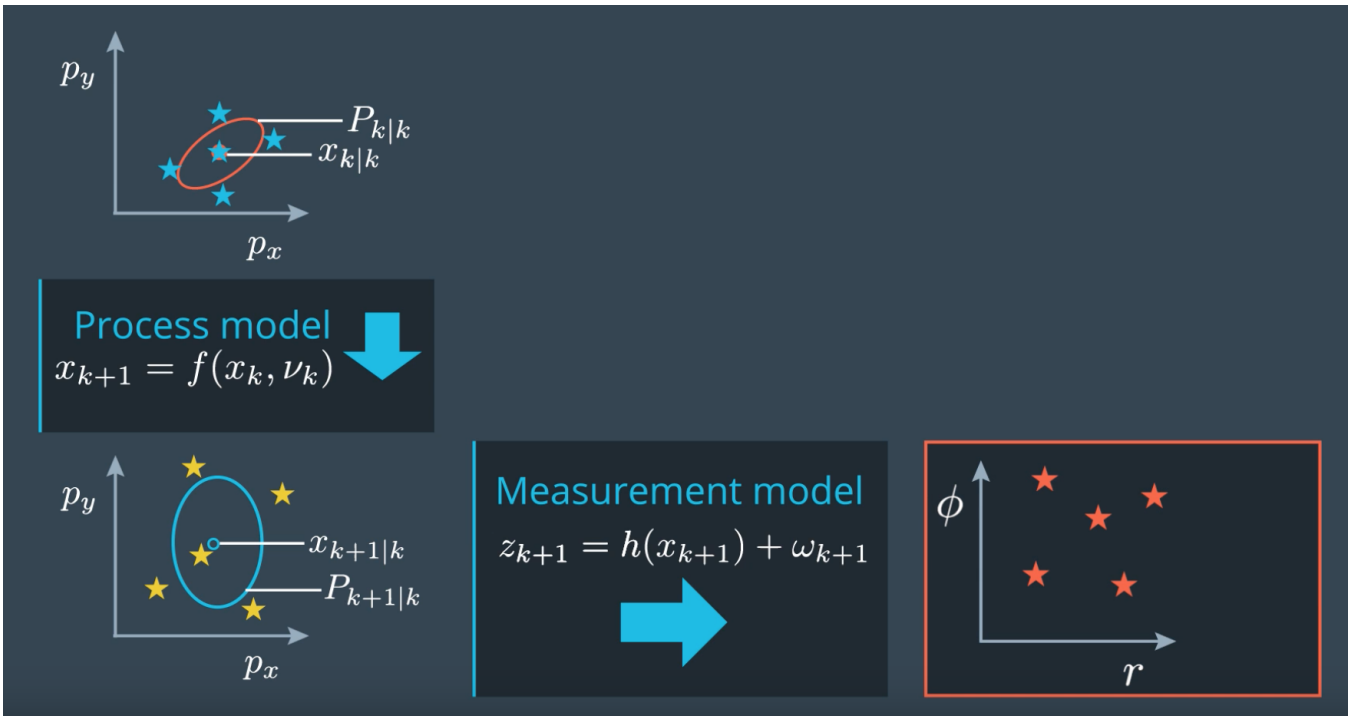
Note: There is an error in picture above, the predicted mean formula should be
$x_{k+1|k} = \Sigma_{i=0}^{2n_{aug}} \omega_i \cdot \chi_{k+1|k,i}$

Measurement Stage – Take Radar sensor data for instance

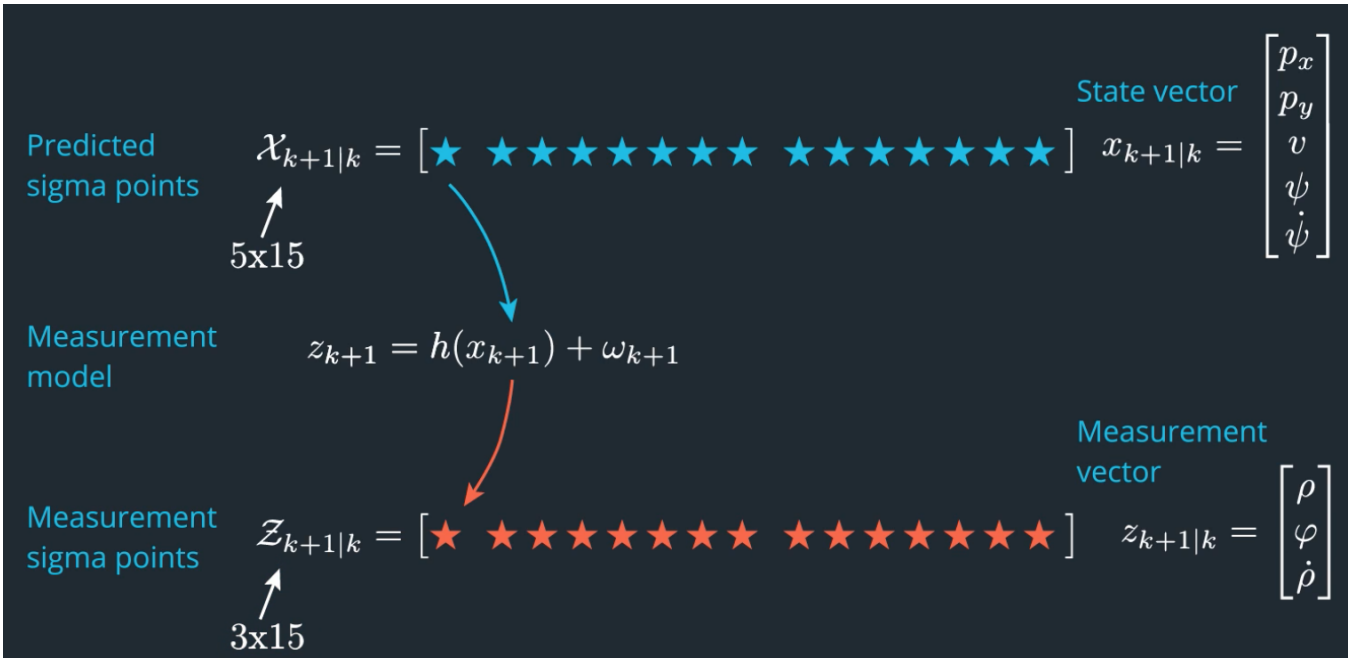
1. Generate Measurement Simga Points and Calculate Its Mean and Covariance.

Like what we do in Prediction Stage, measurement model is also non-linear that we need to put several sigma points into measurement function. But here we could have two shortcut.

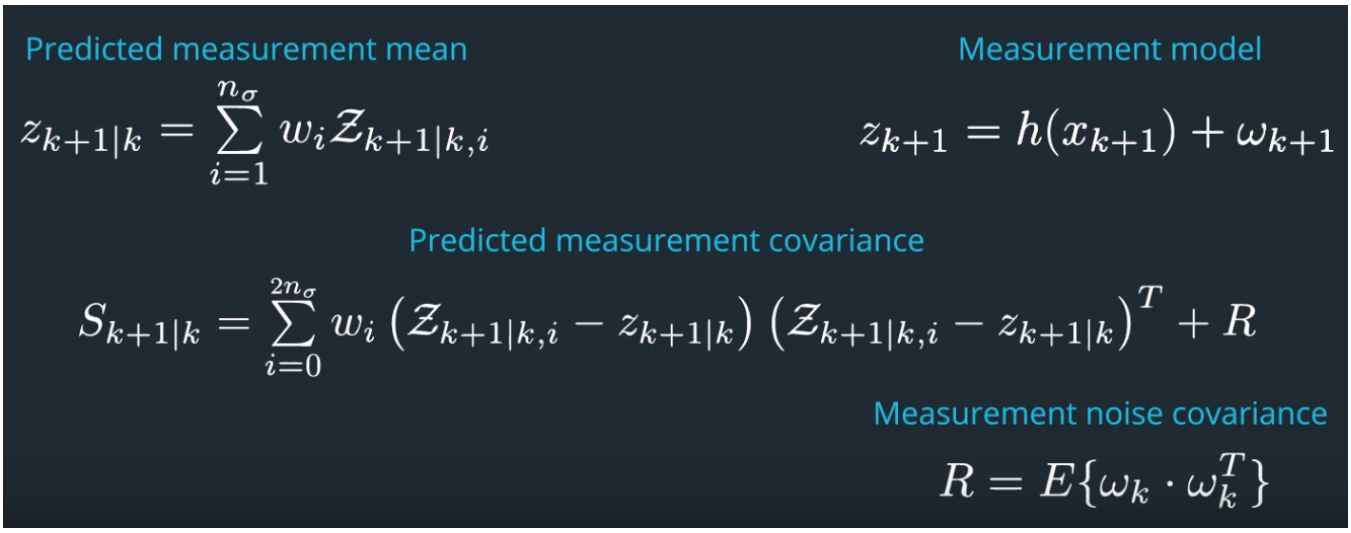
- First, we could directly put the predicted sigma points generated from Prediction Step into the measurement model.
- Second, we don't have to augment the predicted sigma points with measurement noise vector, because it has no non-linear effect in our measurement model.



the picture above is from Udacity Self-Driving Nanodegree project under CC BY-NC-ND 4.0.



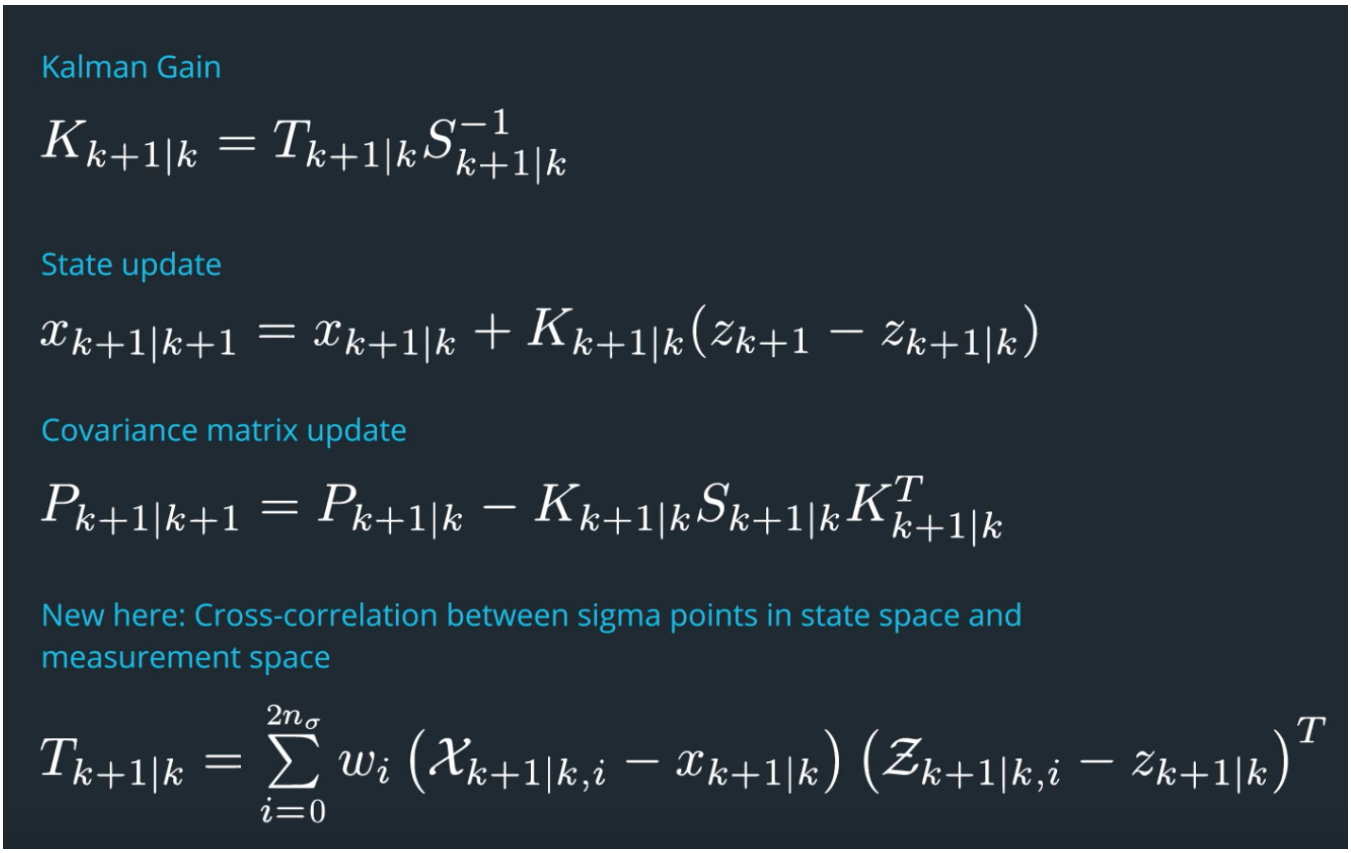
the picture above is from Udacity Self-Driving Nanodegree project under CC BY-NC-ND 4.0.



the picture above is from Udacity Self-Driving Nanodegree project under CC BY-NC-ND 4.0.

Note: There is also an error in predicted measurement mean formula, the predicted mean formula should be $z_{k+1|k} = \Sigma_{i=0}^{2n_{aug}} \omega_i \cdot Z_{k+1|k,i}$

2. Update the State and Covariance Matrix with Radar Data



the picture above is from Udacity Self-Driving Nanodegree project under CC BY-NC-ND 4.0.

---

1. Previous

← Previous
How to Deploy A Hugo Blog onto Github