UNSCENTED KALMAN FILTER ALGORITHM OVERVIEW

Given sensor data **z**, **t** (measurement vector, timestamp):

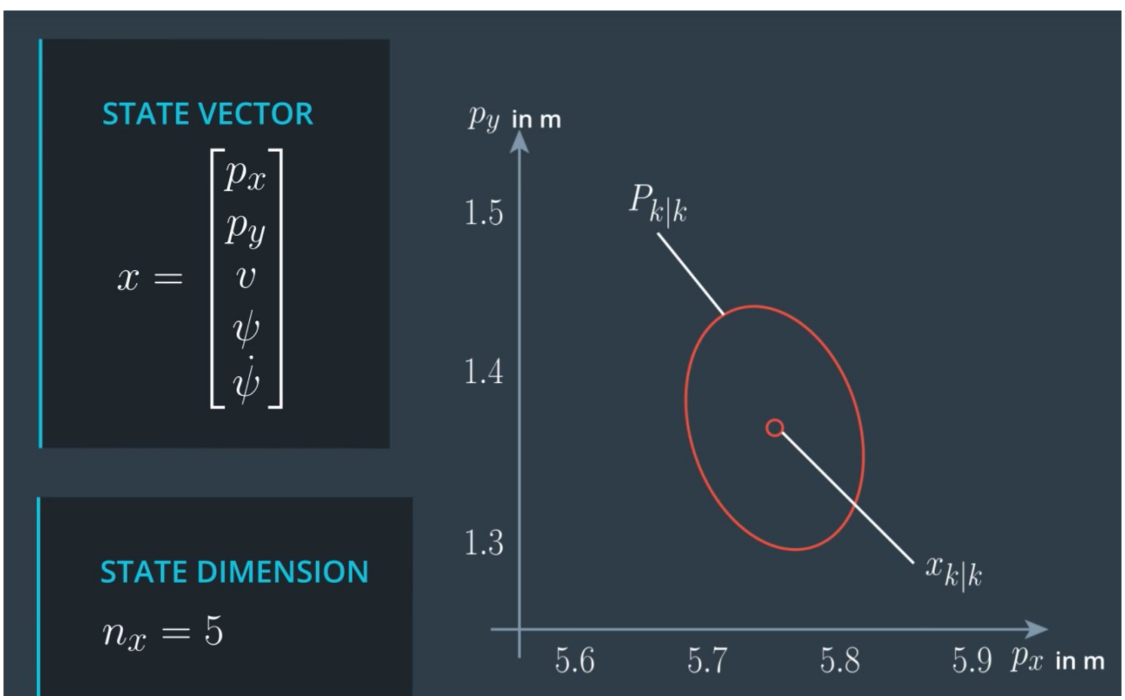
If 1st data:	initialize **x**, **P** (state vector, state covariance matrix)
Else:	– Compute **dt** (time difference) between new **t** and previous measurement's **t** – **update(z, dt)**

update(z, dt):

State Prediction using StatePredictor – Given: current **x**, current **P**, **dt** – Get: **predicted_x**, **predicted_P**, **predicted_sigma_x**
Measurement Prediction using MeasurementPredictor *(depends on sensor used)* – Given: **predicted_sigma_x**, **sensor_type** *(lidar or radar)* – Get: **predicted_sigma_z**, **predicted_z**, **S** *(measurement covariance)*
State Update using StateUpdater – Given: **z**, **S**, **predicted_P**, **predicted_x**, **predicted_z**, **predicted_sigma_x**, **predicted_sigma_z** – Get: updated **P**, updated **x** *(and **nis**)*

- Sigma matrices contain representative points in the gaussian distribution of a vector. Each column is a point.

- In the figure below, the point in center the mean state vector **x**, while the ellipse represents the state covariance matrix **P** which is assumed to be gaussian.



StatePredictor *1st Step:* – Given: current **x**, current **P** – Get: **augmented_sigma_x** *2nd Step:* – Given: **augmented_sigma_x**, **dt** – Get: **predicted_sigma_x** *3rd Step:* – Given: **predicted_sigma_x** – Get: **predicted_x** *4th Step:* – Given: **predicted_x**, **predicted_sigma_x** – Get: **predicted_P**	***1st step:*** – The **augmented_sigma_x** is a matrix that contains representative points of the gaussian distribution of state **x** – It's augmented because it includes **speed_noise_variance** and **yawrate_noise_variance** which are process noises – The computation of the sigma points consider the covariance of the process from the current P, the effect of this process covariance is scaled with a tuned parameter **lambda** ***2nd step:*** – The **predicted_sigma_x** is extrapolated from the time difference **dt** and **sigma_x** including the noise values stored at the **augmented_sigma_x** at that time ***3rd Step:*** – *The **predicted_x*** is the mean of all the sigma points stored at **predicted_sigma_x** scaled with appropriate weights. These weights come from the tuned parameter **lambda** ***4th Step*** – The **predicted_P** covariance is computed from the all the differences between the **predicted_x** and each **predicted_sigma_x** point also affected by the mentioned weights

MeasurementPredictor *1st Step:* – Given: **predicted_sigma_x** – Get: **predicted_sigma_z** *2nd Step:* – Given: **predicted_sigma_z** – Get: **predicted_z** *3rd Step:* – Given: **predicted_sigma_z** – Get: **S** (measurement covariance)	***1st step:*** – Just like when we map measurement vector **z** to state vector **x** *(and back)*, we just map each sigma point in **sigma_x** to **sigma_z**. In essence, we just transform the points in state space to measurement space. ***2nd step:*** – Analogous to the 3rd step of **StatePredictor**, we just get the mean of all sigma points scaled by the mentioned weights ***3rd step*** – We compute for the measurement covariance **S**, analogous to the 4th step of **StatePredictor**, note that the noise covariance **R** of the sensor is considered

StateUpdater *1st Step:* – Given: **predicted_z**, **predicted_x**, **predicted_sigma_z**, **predicted_sigma_x** – Get: **Tc** (cross correlation matrix) *2nd Step:* – Given: **predicted_x**, **predicted_P**, **z**, **predicted_z**, **S**, **Tc**, – Get update **P**, updated **x**, *(and **nis**)*	***1st step:*** – We compute the cross correlation between state **x** and measurement **z**, by considering the difference between state **x** and each sigma point in x-space as well as between measurement **z** and each sigma point in z-space . ***2nd step:*** – We get the kalman gain **K** from the cross correlation matrix **Tc** and measurement covariance matrix **S**, and use that difference between the actual measurement and predicted measurement to get the updated **P** and **x**, the **nis** can be computed here as well which can be used to gauge how well our filter is performing

PROCESS NOISE

$$\nu_k = \begin{bmatrix} \nu_{a,k} \\ \nu_{\ddot{\psi},k} \end{bmatrix}$$

INDEPENDENT NOISE PROCESSES
DOESN'T EXPRESS EFFECT ON STATE VECTOR
INDEPENDENT OF Δt

PROCESS MODEL

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

AUGMENTED STATE

$$x_{a,k} = \begin{bmatrix} p_x \\ p_y \\ v \\ \psi \\ \dot{\psi} \\ \nu_a \\ \nu_{\ddot{\psi}} \end{bmatrix}$$

AUGMENTED STATE DIMENSION

$$n_a = 7$$

NUMBER SIGMA POINTS

$$n_\sigma = 2n_a + 1 = 15$$

AUGMENTED COVARIANCE MATRIX

$$P_{a,k|k} = \begin{bmatrix} P_{k|k} & 0 \\ 0 & Q \end{bmatrix}$$

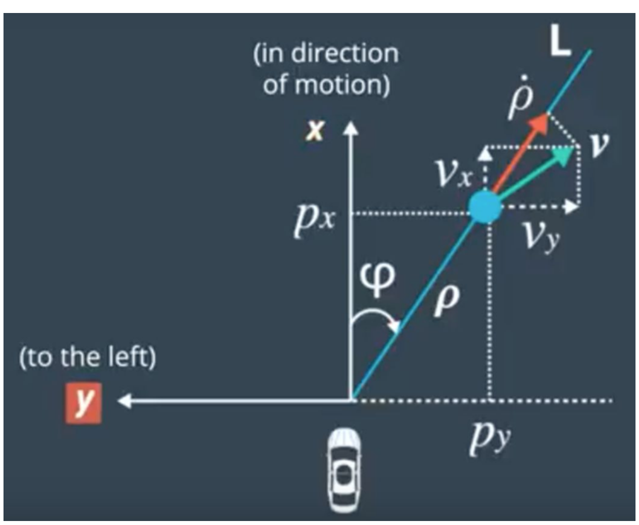
$7 * 7$

CALCULATE AUGMENTED SIGMA POINTS

$$X_{a,k|k} = \left[x_{a,k|k} \quad x_{a,k|k} + \sqrt{(\lambda + n_a) P_{a,k|k}} \quad x_{a,k|k} - \sqrt{(\lambda + n_a) P_{a,k|k}} \right]$$

with scaling factor $\lambda = 3 - n_a$

$$\nu_{a,k} \sim N(0, \sigma_a^2)$$
$$\nu_{\ddot{\psi},k} \sim N(0, \sigma_{\ddot{\psi}}^2)$$



PROCESS NOISE COVARIANCE MATRIX

$$Q = E\left\{\nu_k \cdot \nu_k^T\right\} = \boxed{\begin{bmatrix} \sigma_a^2 & 0 \\ 0 & \sigma_{\ddot{\psi}}^2 \end{bmatrix}}$$

Notice in figure in the left:

- rho_dot is not equal to v. rho_dot is a projection of v on line L
- The yaw corresponds to the direction of the object's (blue dot) movement, phi on the other hand corresponds to the sensor's (car's perspective) position