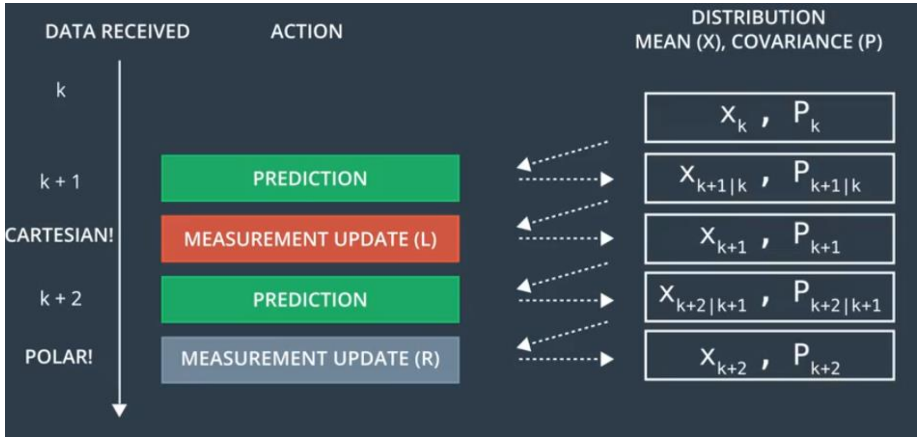
Extended Kalman Filter

Nacho Sañudo
University of Modena and Reggio Emilia
Ignacio.sanudoolmedo@unimore.it

Credits to Udacity self-driving course

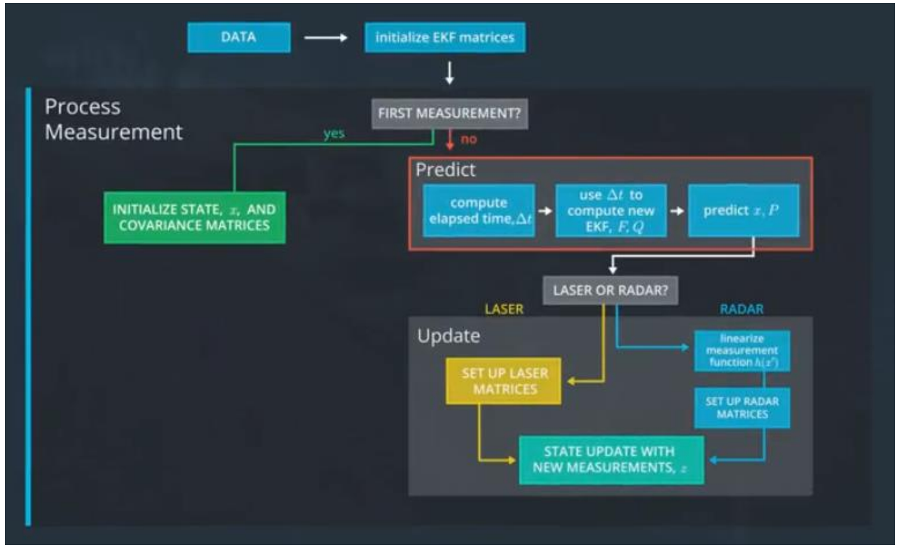
Extended Kalman Filter

- In Kalman filter, we assume that prediction and update steps are **linear functions**
  - Most real-world problems involve nonlinear functions (like angles, sins…)
  - If you feed a Gaussian with a Nonlinear function, then the output is not a Gaussian. Nonlinear functions lead to Non-Gaussian Distributions
- E.G: In the real-world, we receieve the measurements from different sensors that work differently (LiDAR measurement in Cartesian while radar in polar coordinate system)



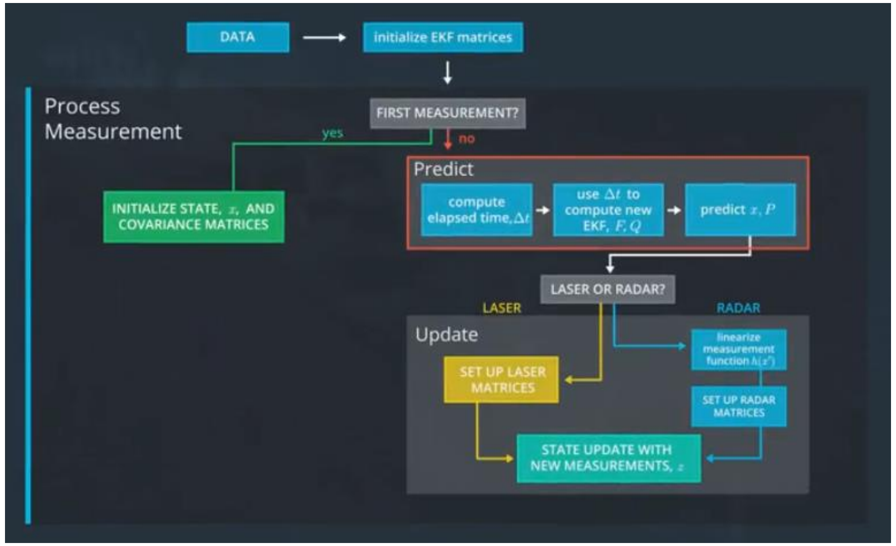
EKF – Processing flow

1. We recieve measurements from the sensors (L/R)
   - If it is the first measurement, we initialize the covariance matrix (**P’**)
2. We **predict** the object’s position and velocity using the motion model and information of the precedent step
3. When we receive a sensor measurement, we **update** the object’s position giving more importance to the predicted location or to the measured one (using Kalman Gain)
4. We iterate from 2 to 3



EKF – Processing flow

- The information provided by the sensors is used to estimate the state (**2D position & 2D velocity**)
  - **LiDAR**: meas_px, meas_py, timestamp, gt_px, gt_py, gt_vx, gt_vy
  - **Radar**: meas_rho, meas_phi, meas_rho_dot, timestamp, gt_px, gt_py, gt_vx, gt_vy
- Each time we recieve a new measurement the estimation function is triggered
  - **Prediction** (we predict the object state and each covariance)
  - **Update** (depends on sensor type)
    - LiDAR we can use the **standard Kalman Filter** (because the data provided is in cartesian coordinates)
    - **Radar measurements involve a non linear measurement function (polar coordinates)**
- **The belief about the object's position and velocity is updated each time a measurement is recieved**



$$x' = Fx + \nu$$

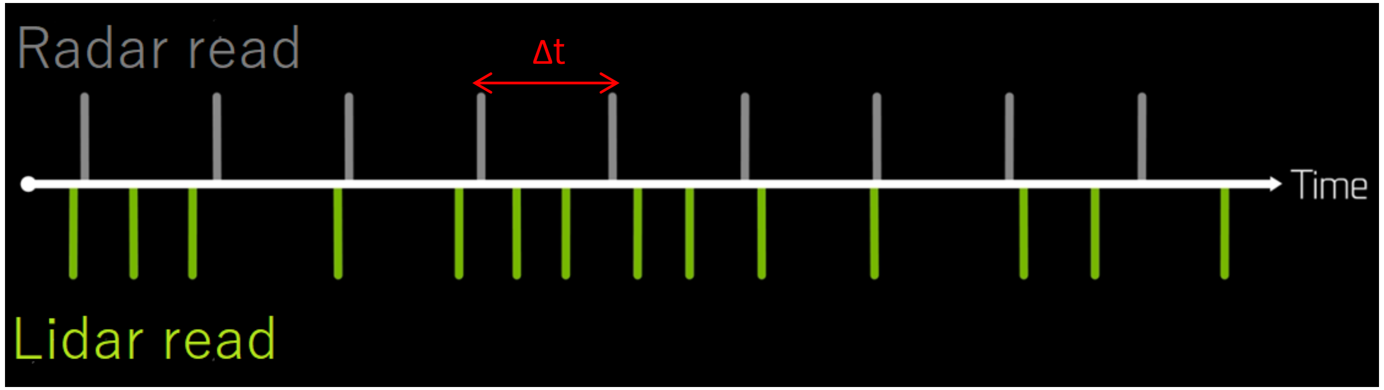
$$P' = FPF^T + Q$$

Prediction

- **Objective: Model how the object evolves over time**
  - We **predict** the state of the object over time (***x'***)
  - Model the **uncertainty** of the object over time (***P'***)
  - In the prediction process we have a deterministic and a stocastich part
    - We are now modelling the process and motion noise ***<u>Q and v</u>***
- **Glossary**
  - **x** is the mean state vector, it contains information about the object's position and velocity
    - [px,py,vx,vy]
  - **F** is the Transition Matrix, represents how the system evolves over time considering the time steps and constant velocities
  - ***v*** is the motion noise
  - **P** is the state covariance matrix, it contains the uncertainty of the object's position and velocity
  - **Q** is the Process Covariance Matrix. It represents the uncertainty in the object's position when predicting location. It is a covariance matrix associated with the noise in states

Prediction

- Measurement read rate is not consistent
  - Moreover, we have multiple sensors
- First step in prediction is to compute the elapsed time between measurements (***Δ*t**)
  - **The more *Δ*t the more uncertain our position and velocity will be**
    - **This will be reflected in the covariance matrix P**



$$x' = Fx + \nu$$

- The object is going at a constant velocity

Prediction

- We define the state vector x and linear motion model x' (how the system evolver over time)
  - x = $[p_x, p_y, v_x, v_y]$
  - x'= $\begin{cases} p'_x = p_x + v_x \Delta t + V_{px} \\ p'_y = p_x + v_x \Delta t + V_{py} \\ v'_x = v_x + V_{vx} \\ v'_y = v_y + V_{vy} \end{cases}$

- The velocity is **constant** in our model
  - However, we can model the uncertainty of the velocity (i.e., acceleration of the object) in the noise variable stocastich part
    - $(\boldsymbol{V_{px}}, \boldsymbol{V_{py}})$
  - x'= $\begin{cases} p'_x = p_x + v_x \Delta t + \frac{a_x \Delta t^2}{2} \\ p'_y = p_x + v_x \Delta t + \frac{a_y \Delta t^2}{2} \\ v'_x = v_x + a_x \Delta t \\ v'_y = v_y + a_y \Delta t \end{cases}$

$$x' = Fx + \nu$$

$$x' = \begin{cases} p'_x = p_x + v_x \Delta t + V_{px} \\ p'_y = p_x + v_x \Delta t + V_{py} \\ v'_x = v_x + V_{vx} \\ v'_y = v_y + V_{vy} \end{cases}$$

Prediction

- If we represent everything as a matrix

$$\underbrace{\begin{pmatrix} p'_x \\ p'_y \\ v'_x \\ v'_y \end{pmatrix}}_{x'} = \underbrace{\begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{F} \underbrace{\begin{pmatrix} p_x \\ p_y \\ v_x \\ v_y \end{pmatrix}}_{x} + \underbrace{\begin{pmatrix} \frac{a_x \Delta t^2}{2} \\ \frac{a_y \Delta t^2}{2} \\ a_x \Delta t \\ a_y \Delta t \end{pmatrix}}_{V}$$

- **Prediction: x'=F'*x+*V***
  - **Remind: F** is the Transition Matrix; it represents how the system evolves over time considering the time steps and constant velocities
  - We will not use ***V*** in the assignment

$$P' = \underbrace{FPF^T}_{\text{Deterministic}} + \underbrace{Q}_{\text{Stochastic}}$$

Prediction

- **P** is the covariance matrix**,** holds the current uncertainty of the position and velocity
- The **covariance Q** is proportional to the velocity
  - The higher the velocity the bigger the uncertainty
- **We introduce to the model a random acceleration**
  - **Assume that this is a random value that models the stochastic part of the object**

  - $$\boldsymbol{Q} = \begin{pmatrix} \frac{\Delta t^4}{4}\sigma_{ax}^2 & 0 & \frac{\Delta t^3}{2}\sigma_{ax}^2 & 0 \\ 0 & \frac{\Delta t^4}{4}\sigma_{ay}^2 & 0 & \frac{\Delta t^3}{2}\sigma_{ay}^2 \\ \frac{\Delta t^3}{2}\sigma_{ax}^2 & 0 & \Delta t^2\sigma_{ax}^2 & 0 \\ 0 & \frac{\Delta t^3}{2}\sigma_{ay}^2 & 0 & \Delta t^2\sigma_{ay}^2 \end{pmatrix}$$

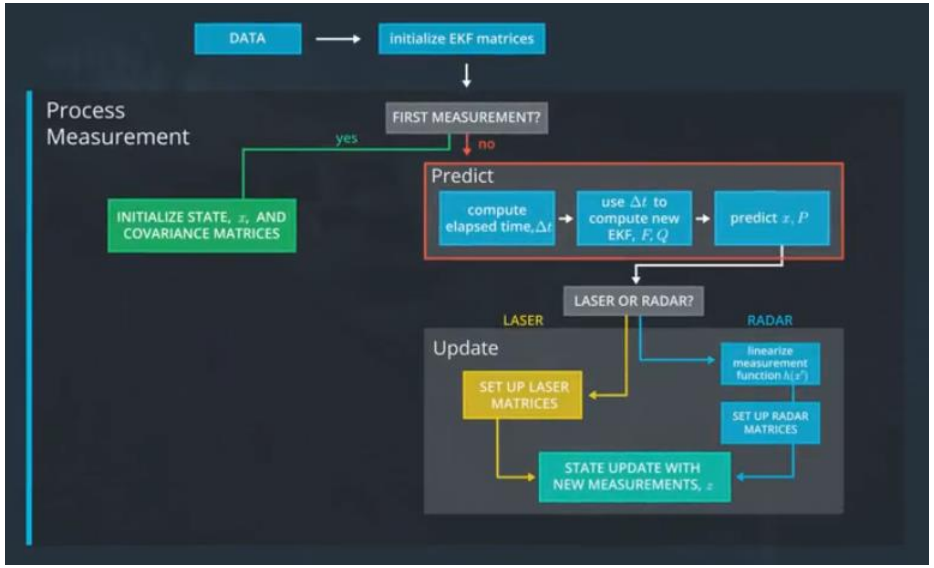
Initial P value

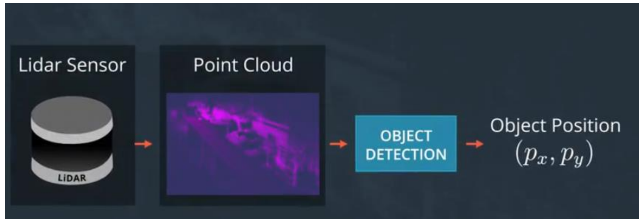
$$\boldsymbol{P} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1000 & 0 \\ 0 & 0 & 0 & 1000 \end{pmatrix}$$

$\boldsymbol{\sigma}$ *represents the noise*

EKF - Update

- We will use two different sensors:
  - **LiDAR**
  - **Radar**
- Each sensor has its own update scheme
  - **LiDAR (KF)**
  - **Radar (EKF)**





KF – Update (LiDAR)

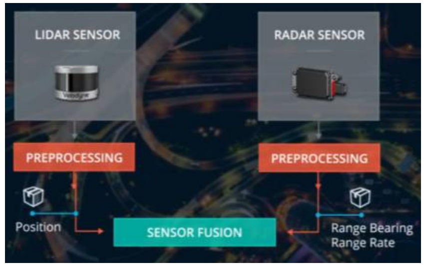
- For simplicity we assume that the object detection component gives us the position (px,py) of the object
- **z** is the measurement vector (actual measured value that is coming from the sensor) | z = $[p_x, p_y]$ | (p'_x is the predicted position)
- **y** is the difference between the measured value and actual value
  - The product of H with x', allows us to compare the prediction with the sensor measurement
- **H** is the matrix that projects the belief of the object's current state into the measurement space of the sensor
  - **We need to compare the measurement with the prediction. H is the matrix that maps the 4d estimation into 2d**
  - **we get rid of the velocity information from the state variable H since the lidar sensor only measures position**
- **R** is the covariance matrix of the measurement noise (value given by the manufacturer)
- **S** is the total error
- **P** is the covariance matrix**,** holds the current **uncertainty** of the states

$$\boldsymbol{y} = \boldsymbol{z} - \boldsymbol{H}\boldsymbol{x}'$$
$$\boldsymbol{S} = \boldsymbol{H}\boldsymbol{P}'\boldsymbol{H}^T + \boldsymbol{R}$$
$$\boldsymbol{K} = \boldsymbol{P}'\boldsymbol{H}^T\boldsymbol{S}^{-1}$$
$$\boldsymbol{x} = \boldsymbol{x}' + \boldsymbol{K}\boldsymbol{y}$$
$$\boldsymbol{P} = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{H})\boldsymbol{P}'$$

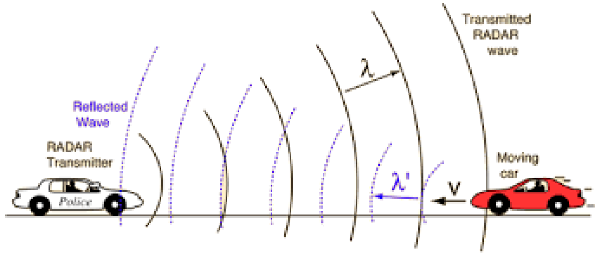
$$\begin{pmatrix} \rho_x \\ \rho_y \end{pmatrix} = H \begin{pmatrix} p'_x \\ p'_y \\ v'_x \\ v'_y \end{pmatrix} \qquad z = \begin{pmatrix} \rho_x \\ \rho_y \end{pmatrix}$$

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

EKF – Update (Radar)



- **LiDARs cannot produce velocity information**
  - We will use the radars (polar information) to update our predictions which are in cartesian coordinates!
- Using the doppler effect the radar can measure the radial velocity of an object
  - Radial velocity is the component of velocity moving towards or away from the sensor
- Let's combine the information of both sensors!



EKF – Update (Radar)

$$R = \begin{pmatrix} \sigma_\rho^2 & 0 & 0 \\ 0 & \sigma_\varphi^2 & 0 \\ 0 & 0 & \sigma_{\dot\rho}^2 \end{pmatrix}$$

- **z** is the measurement in polar coordinates
  - $z = [p, \emptyset, \dot{p}]$
- **x'** is the predicted value
- **y** is the difference between the measured value and actual value
- **h(x')** is the function that specifies the mapping between our predicted values in Cartesian coordinates and Polar coordinates
- **R** is the measurement noise
- **K** is the Kalman gain
- **H_j** is the Jacobian Matrix

$$y = z - h(x')$$
$$S = H_j P' H_j^T + R$$
$$K = P' H_j^T S^{-1}$$
$$x = x' + K^* y$$
$$P = (I - K H_j) P'$$

$$h(x') = \begin{pmatrix} \rho \\ \phi \\ \dot\rho \end{pmatrix} = \begin{pmatrix} \sqrt{p_x'^2 + p_y'^2} \\ \arctan(p_y'/p_x') \\ \frac{p_x' v_x' + p_y' v_y'}{\sqrt{p_x'^2 + p_y'^2}} \end{pmatrix}$$

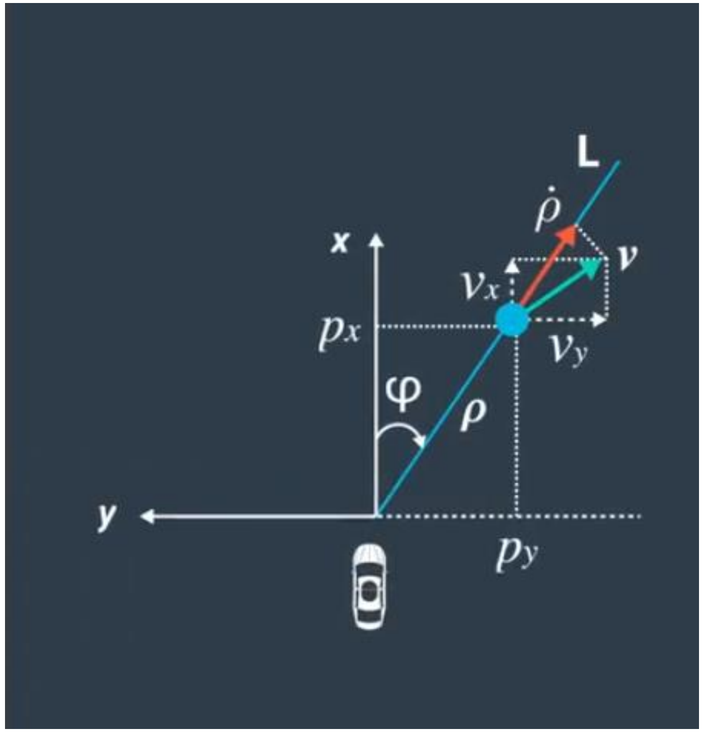
$y = z - h(x')$

EKF – Update (Radar)

- Radar information:
  - **Range** (ρ rho): distance from origin
  - **Bearing** (ϕ, phi): angle betwee rho and the x axis
  - **Radial velocity** (rho dot): velocity towards the line L (change rate)
  - The x axis always points in the vehicles direction (y axis to the left)
- Before, **we had a function H to map the information of the LiDAR to cartesian information**
- **Objective**: map the predicted information into the measurement model

$$\cancel{\begin{pmatrix} p_x \\ p_y \end{pmatrix} = H \begin{pmatrix} p'_x \\ p'_y \\ v'_x \\ v'_y \end{pmatrix}}$$

$$\begin{pmatrix} p \\ \emptyset \\ \dot{p} \end{pmatrix} \overset{?}{\rightarrow} \begin{pmatrix} p'_x \\ p'_y \\ v'_x \\ v'_y \end{pmatrix}$$



EKF – Update (Radar)

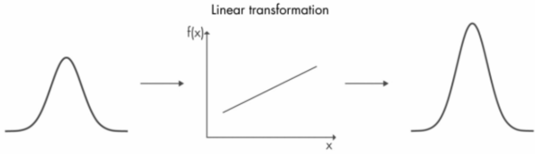
$$\cancel{\begin{pmatrix} p_x \\ p_y \end{pmatrix} = H \begin{pmatrix} p'_x \\ p'_y \\ v'_x \\ v'_y \end{pmatrix}}$$

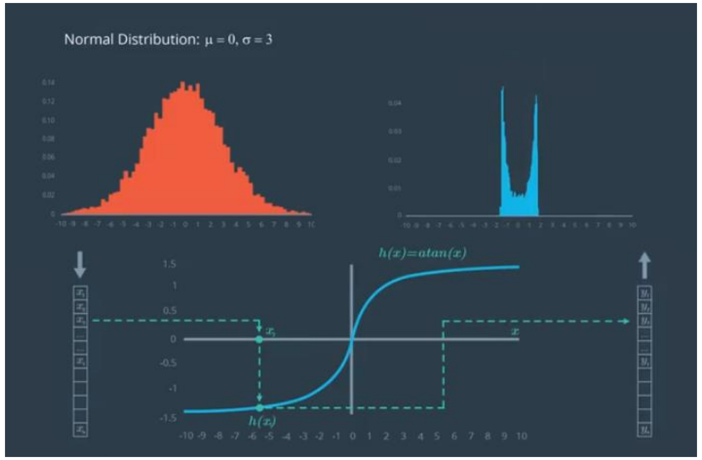
- The measurement information is represented in polar coordinates (***z***)
- We don't have a matrix **H** that maps the state vector ***x*** into Polar Coordinates
- **To compute "*y*" we need to convert from <u>cartesian coordinates to polar</u>**
  - $\cancel{\boldsymbol{y} = \boldsymbol{z} - \boldsymbol{H}\boldsymbol{x}'}$ → $\boldsymbol{y} = \boldsymbol{z} - \boldsymbol{h}(\boldsymbol{x}')$
- **h(x')** specifies how the predicted position and speed can be related to $[p, \emptyset, \dot{p}]$
  - This mapping is required because we are predicting in Cartesian coordinates, but our measurement (***z***) is in Polar Coordinates
  - $$h(x') = \begin{pmatrix} \sqrt{p'^2_x + p'^2_y} \\ \arctan(p'_y, p'_x) \\ \frac{P'_x v'_x + p'_y v'_y}{\sqrt{p'^2_x + p'^2_y}} \end{pmatrix}$$

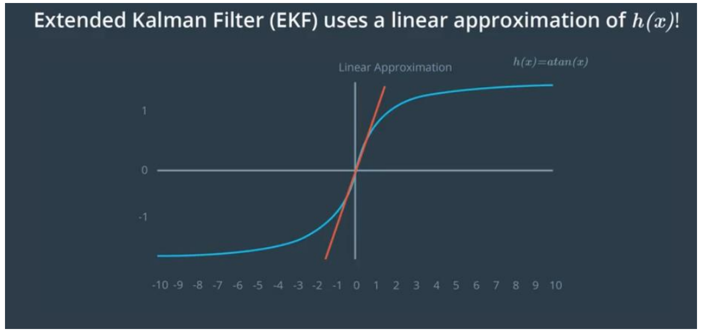
• The predicated state x (that is represented by a Gaussian) is mapped to a nonlinear function (radar info.) → **the resulting function will not be a gaussian function**

EKF – Update (Radar) - Nonlinearity

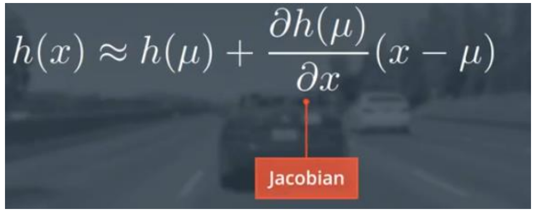
- <u>**If you feed a Gaussian with a Nonlinear function, then the resulting function will not be a Gaussian**</u>
- <u>**Linearize h(x')**</u>
  - Approximate the measurement function h(x') by a linear function which is tangent to h(x) at the mean location of the original Gaussian
- The EKF uses the first order Taylor expansion method to obtain linear approximation of the polar coordinate measurements
  - We first evaluate the nonlinear function h(x') at the mean location (mu) which is the best estimate of our predicted distribution and then we extrapolate a line with slope around mu
  - This slope is given by the first derivative of h(x)



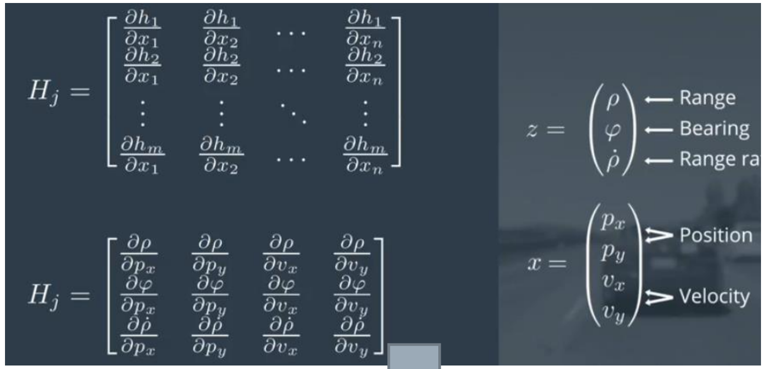




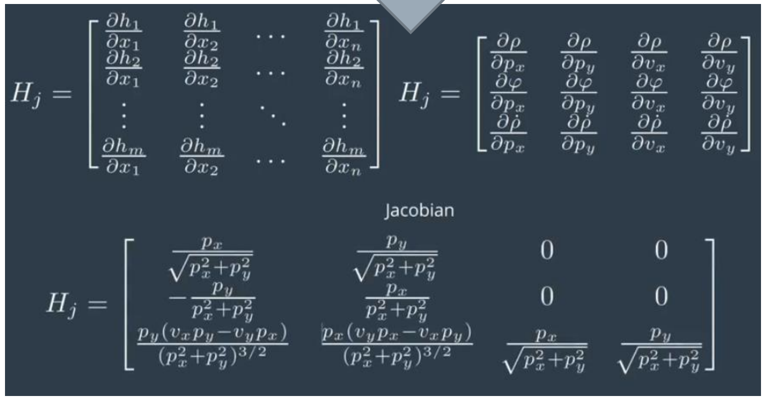
EKF – Update (Radar) - Jacobian



- The derivative of h(x) is called **Jacobian** with respect to x and it contains all the partial derivatives
- We first evaluate the nonlinear function h(mu) which is the best estimation of our predicted distribution
  - then we extrapolate a line with slope. This slope is given by the first derivative of (h(mu))
- In a nutshell we change H with $\mathbf{H_j}$



Computing the derivates...



EKF – Update (Radar)

$$R = \begin{pmatrix} \sigma_\rho^2 & 0 & 0 \\ 0 & \sigma_\varphi^2 & 0 \\ 0 & 0 & \sigma_{\dot\rho}^2 \end{pmatrix}$$

- **z** is the measurement in polar coordinates
  - $z = [p, \emptyset, \dot{p}]$
- **x’** is the predicted value
- **y** is the difference between the measured value and actual value
- **h(x’)** is the function that specifies the mapping between our predicted values in Cartesian coordinates and Polar coordinates
- **R** is the measurement noise
- **K** is the Kalman gain
- **H_j** is the Jacobian Matrix
- **P** is the covariance matrix**,** holds the current uncertainty of the states

$$y = z - h(x')$$
$$S = H_j P' H_j^T + R$$
$$K = P' H_j^T S^{-1}$$
$$x = x' + K * y$$
$$P = (I - K H_j) P'$$

$$h(x') = \begin{pmatrix} \rho \\ \phi \\ \dot\rho \end{pmatrix} = \begin{pmatrix} \sqrt{p_x'^2 + p_y'^2} \\ \arctan(p_y'/p_x') \\ \frac{p_x' v_x' + p_y' v_y'}{\sqrt{p_x'^2 + p_y'^2}} \end{pmatrix}$$

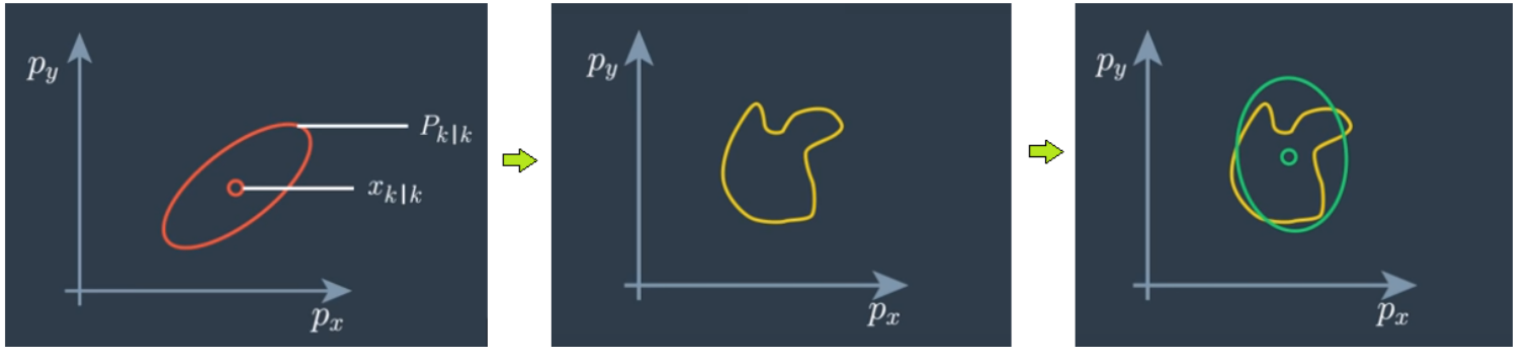
EKF – Root mean squared error

- We might want to check how good our method is with respect to the ground truth
- We can use the RMSE to measure the deviation of the estimate state from the true state
- The lower the error the higher the accuracy of our method

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}\left(x_t^{est} - x_t^{gt}\right)^2}$$

Unscented Kalman Filter (UKF) – Bonus track

- When the state transition and observation models—that is, the predict and update functions are highly nonlinear, the extended Kalman filter can give particularly poor performance
- The unscented Kalman filter (UKF) uses a deterministic sampling technique known as the unscented transformation (UT) to pick a minimal set of sample points (called sigma points) around the mean
  - Instead of linearizing the transformation we approximate the non-linear gaussian into a gaussian



Source: Wikipedia & Udacity