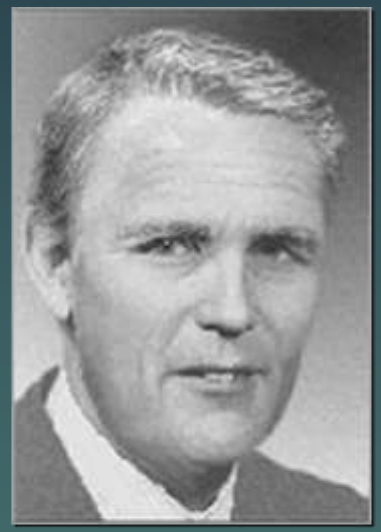
Kalman Filter and Extended Kalman Filter(EKF)

By
Weitao Xiong

Advisor:
Prof. Hongfei Xue

• Kalman Filter



Rudolf Emil Kalman

- Seminal paper by R.E.Kalman, 1960
- Set of mathematical equations
- Optimal estimator

  minimum mean square error
- Versatile
  - ✓ Estimation
  - ✓ Filtering
  - ✓ Prediction
  - ✓ Fusion

Kalman Filter: Start with an simple example

- We start a car in position X_(k-1) and drive it at a constant speed. We want to track where our car is at time k. It seems very simple because we know the formula is

$$x = \begin{bmatrix} p \\ v \end{bmatrix} \qquad p_k = p_{k-1} + \Delta t \cdot v_{k-1}$$

$$v_k = v_{k-1}$$

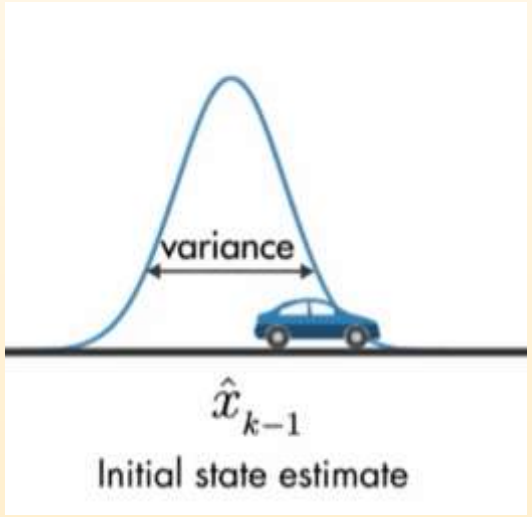
- Convert them to a matrix

$$x_k = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} x_{k-1}$$

- However, it is not perfect. In the real world, there exists many noise and uncertainty. So we can't just use an input that you get from the sensor to put it in a formula to get an actual result.

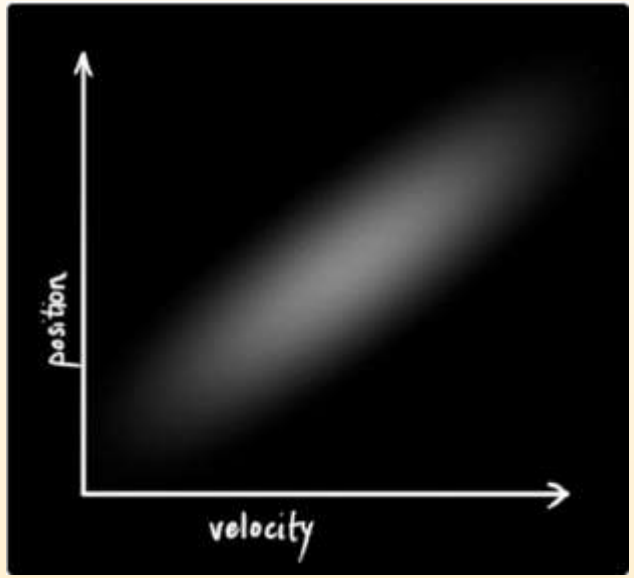
Internal Noise and uncertainty

- We can just assume that the velocity and position we get from the sensor are Gaussian distributed. Because when only the mean and variance are known, the Gaussian distribution is the most reasonable choice for modeling uncertainty because it makes the least assumptions about unknown factors.

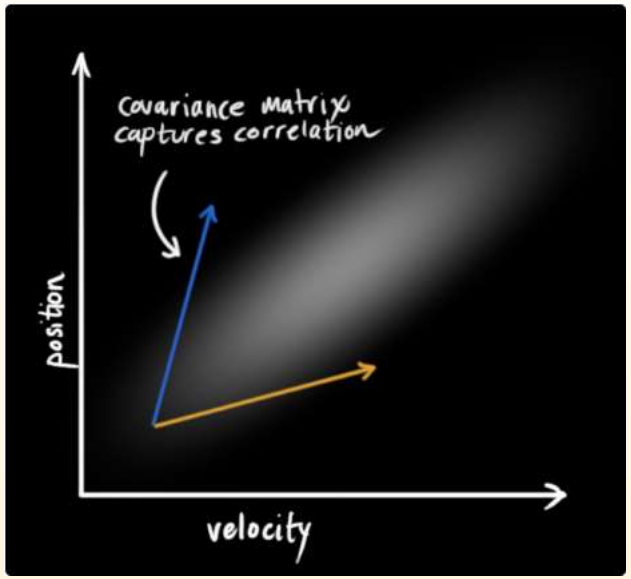


- For at the initial state k-1, the car position can be any number in the distribution. So we can see it is an estimated value.

$\hat{x}$
Estimated
Stated



- We know that these two variables are correlated, and might their relationship look like this.
- But How can we figure out this relationship?



- We can capture it by using the covariance. It represents your uncertainty about the state of the system

$$Cov(x, y)$$

- Covariance matrices P_k are often labeled “Σ”, so we can write

$$P_k = \sum = \begin{bmatrix} \sum_{pp} & \sum_{pv} \\ \sum_{vp} & \sum_{vv} \end{bmatrix}$$

- Next page we will talk about the four element in the matrix

P_k : Covariance matrices

p_k : Position at k time

- The first diagonal element represents the uncertainty of the position

$$\sum pp = Cov(p, p) = Var(p)$$

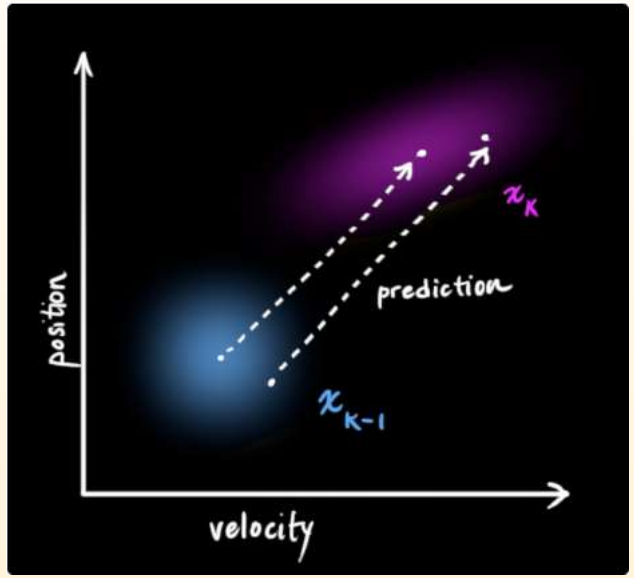
- The second diagonal element represents the uncertainty of the velocity

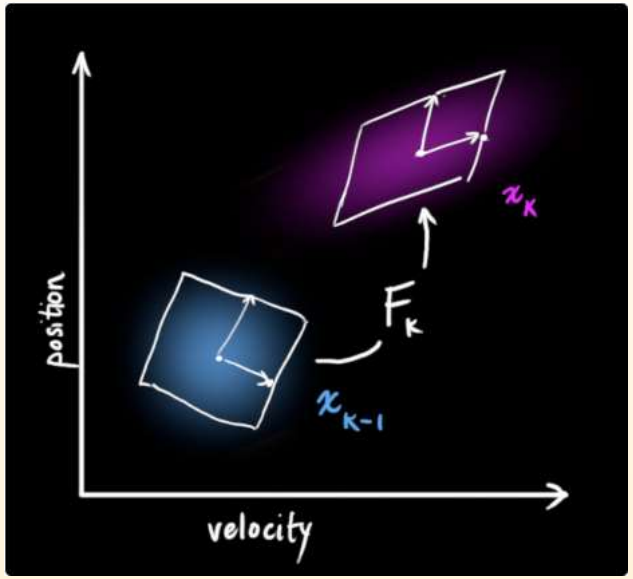
$$\sum vv = Cov(v, v) = Var(v)$$

- The rest of the two represents the correlation between noise in measurements to the speed and to the position of the car

$$\sum pv = \sum vp$$

- We can represent this prediction step with a matrix, F_k





$$\hat{x}_k = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \hat{x}_{k-1} = F_k \cdot \hat{x}_{k-1}$$

- We know the formula below

$$Cov(Ax) = A\sum A^T$$

- Then we can update our covariance matrix

$$\hat{x}_k = F_k\hat{x}_{k-1}$$
$$P_k = F_k P_{k-1} F_k^T$$

- However, it is not done yet, because there also exists **External influence** and **External uncertainty**

External Influence

- **External Influence** is the thing in the outside world that could be affecting or controlling the system.

$$\vec{u}_k$$

- For example, if the state models the motion of a train, the train operator might push on the throttle, causing the train to accelerate.
- Similarly, in our robot example, the navigation software might issue a command to turn the wheels or stop. If we know this additional information about what's going on in the world, we could stuff it into a vector called **u_k**, do something with it, and add it to our prediction as a correction.

- If we know the expected acceleration “a” due to the throttle setting or control commands. Then we can update our formula

$$p_k = p_{k-1} + \Delta t v_{k-1} + \frac{1}{2} a \Delta t^2$$

$$v_k = v_{k-1} + a\Delta t$$

- In matrix form:

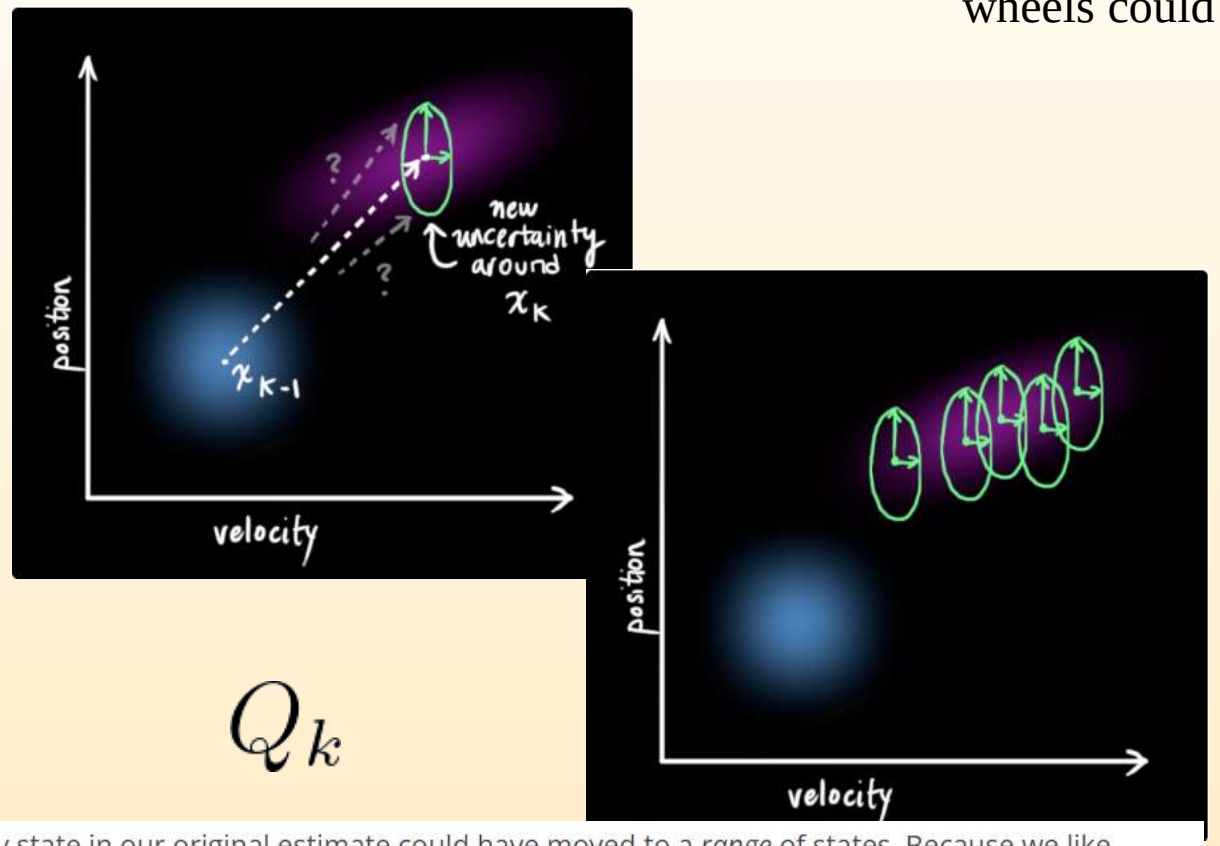
$$\hat{x}_k = F_k \hat{x}_{k-1} + \begin{bmatrix} \frac{\Delta t^2}{2} \\ \Delta t \end{bmatrix} a$$

$$= F_k \hat{x}_{k-1} + B_k u_k$$

B_k : is called the control matrix

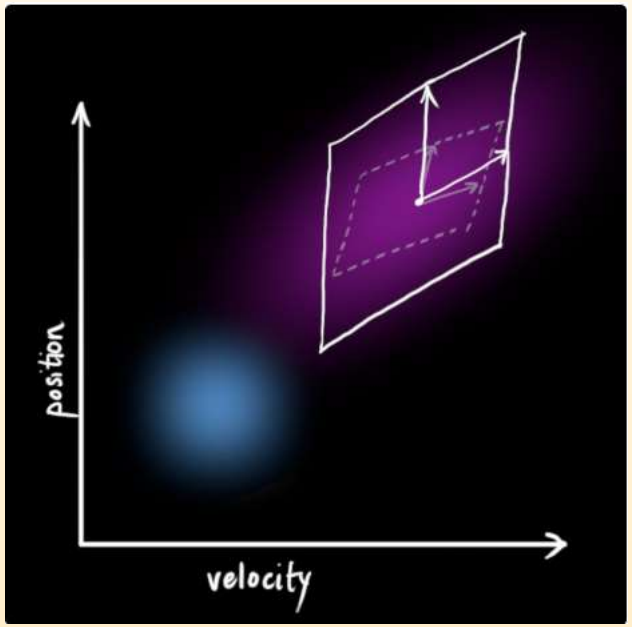
External Uncertainty

- **External Uncertainty** is something we can't track outside.
- For example, if we're tracking a quadcopter, it could be buffeted around by wind. If we're tracking a wheeled robot, the wheels could slip, or bumps on the ground could slow it down.



$$Q_k$$

Every state in our original estimate could have moved to a *range* of states. Because we like Gaussian blobs so much, we'll say that each point in $\hat{\mathbf{x}}_{k-1}$ is moved to somewhere inside a Gaussian blob with covariance $\mathbf{Q}_k$. Another way to say this is that we are treating the untracked influences as **noise** with covariance $\mathbf{Q}_k$.



This produces a new Gaussian blob, with a different covariance (but the same mean)

- In other words, the new best estimate is a prediction made from the previous best estimate, plus a correction for known external influences.

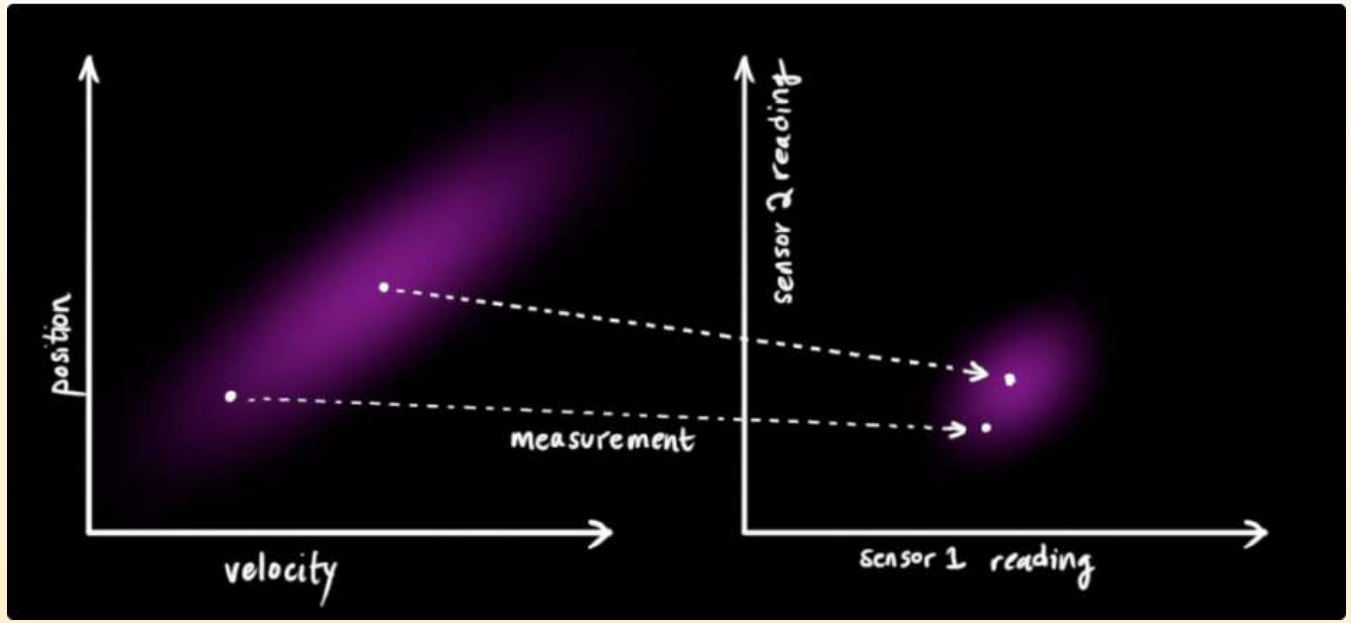
- And the new uncertainty is predicted from the old uncertainty, with some additional uncertainty from the environment.

$$\hat{x}_k = F_k \hat{x}_{k-1} + B_k \vec{u}_k$$

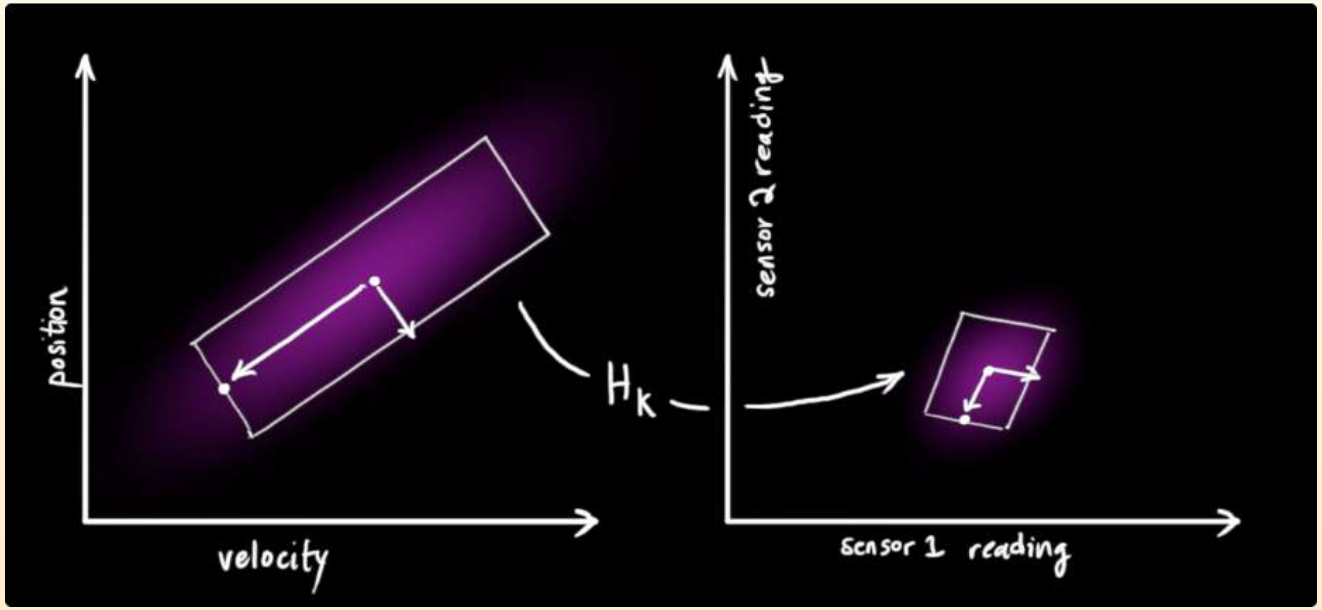
$$P_k = F_k P_{k-1} F_k^T + Q_k$$

Refining the estimate with measurements

- Each sensor tells us something indirect about the state—in other words, the sensors operate on a state and produce a set of readings.



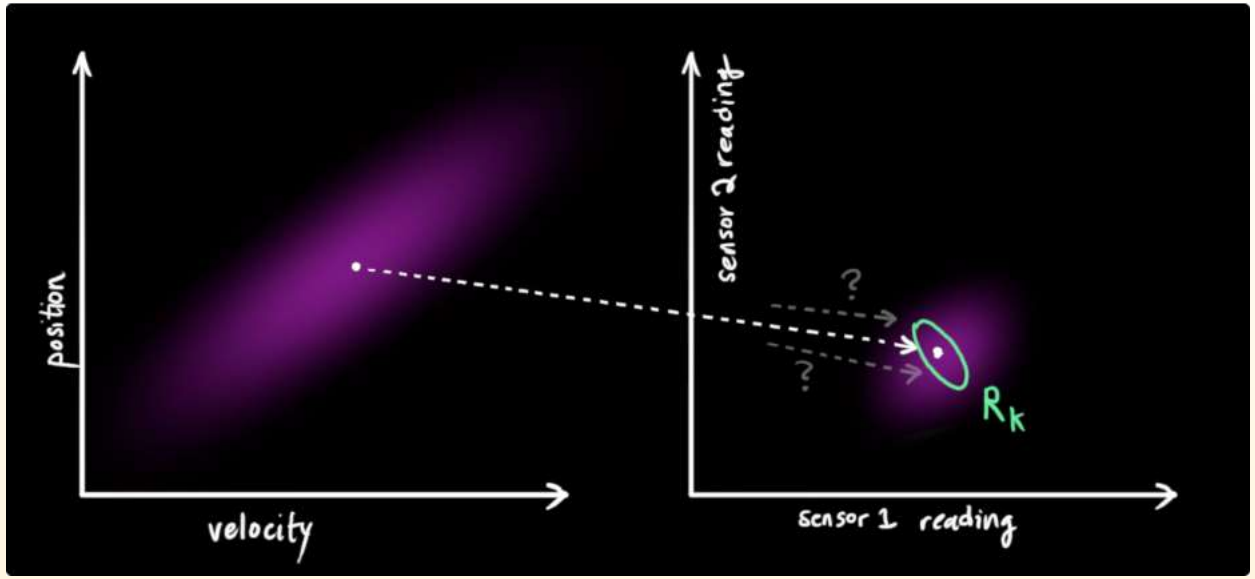
- Notice that the units and scale of the reading might not be the same as the units and scale of the state we're keeping track of.
- So we'll model the sensors with a matrix, H_k.



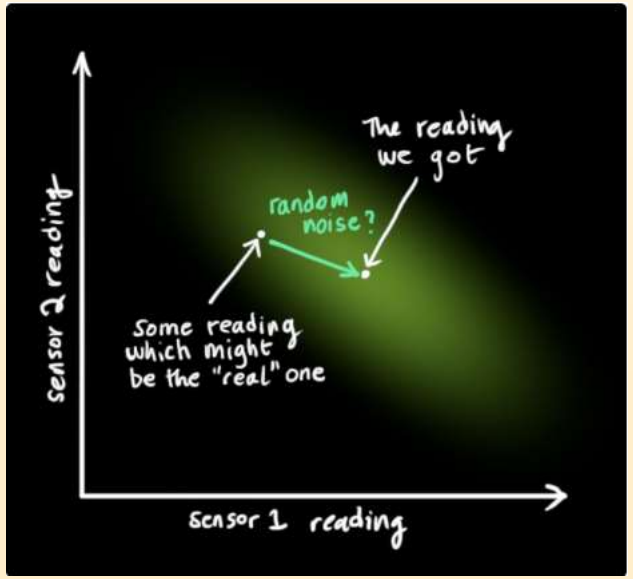
$$\vec{\mu}_{\text{expected}} = H_k \hat{x}_k$$

$$\Sigma_{\text{expected}} = H_k P_k H_k^T$$

The observed Measurement from sensor



- From each reading we observe, we might guess that our system was in a particular state. But because there is uncertainty, some states are more likely than others to have produced the reading we saw.

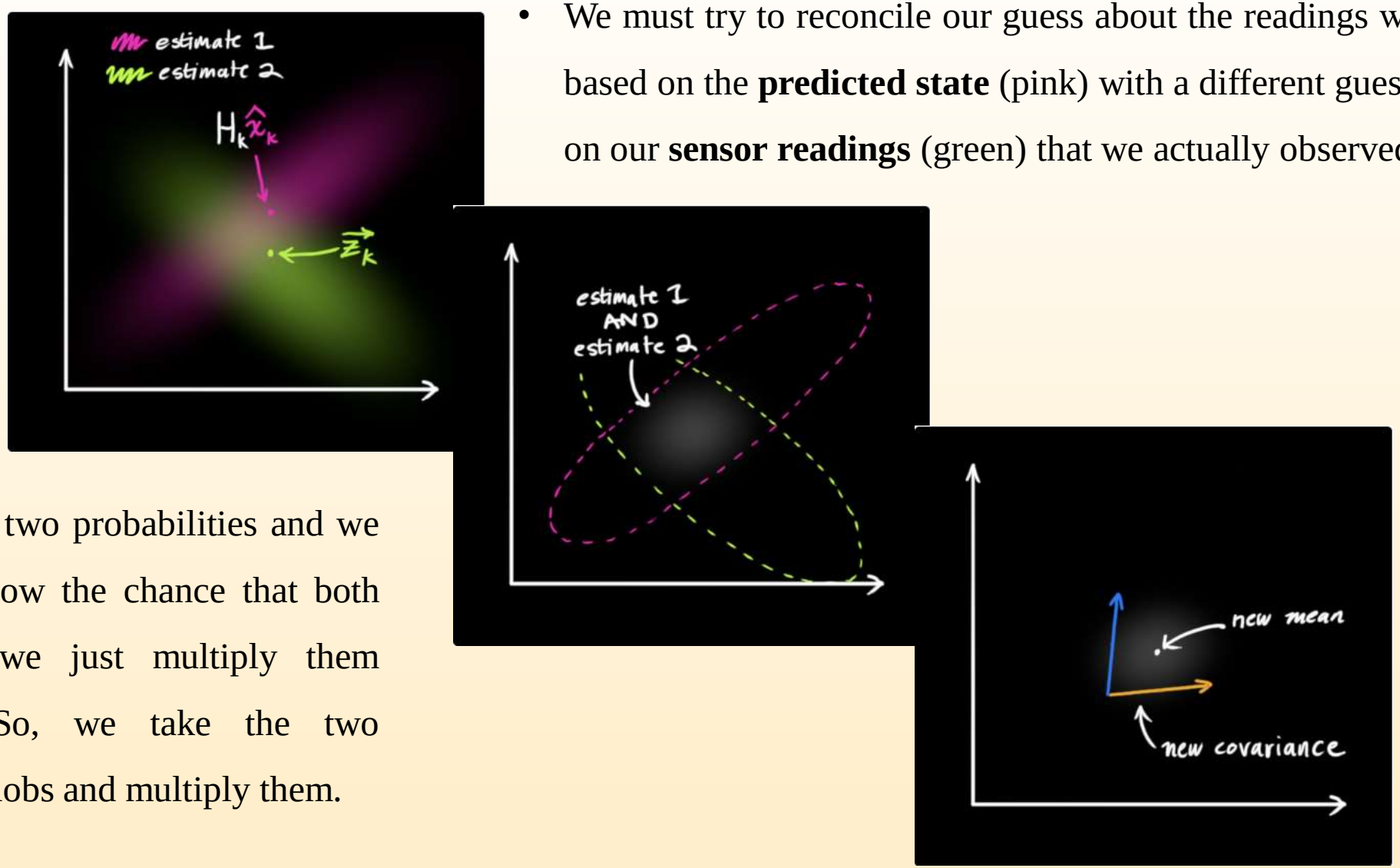


R_k

$\vec{z_k}$

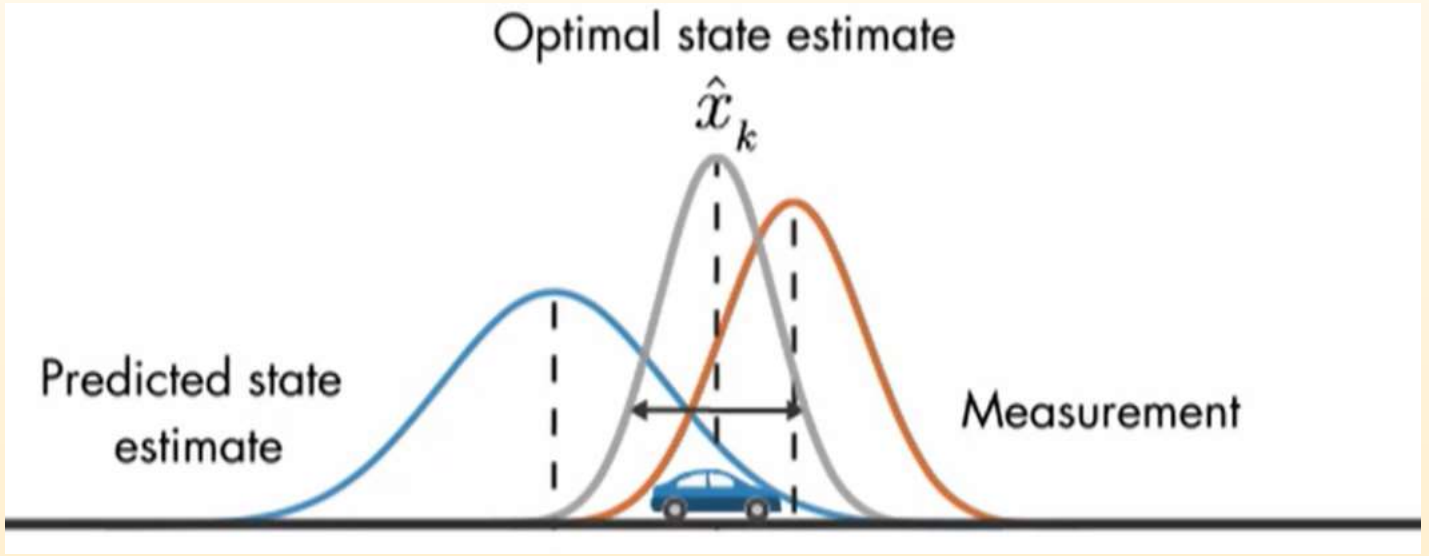
- We'll call the covariance of this uncertainty (i.e. of the sensor noise) R_k.
- The distribution has a mean equal to the reading we observed, which we'll call z_k.

- We must try to reconcile our guess about the readings we'd see based on the **predicted state** (pink) with a different guess based on our **sensor readings** (green) that we actually observed.



- If we have two probabilities and we want to know the chance that both are true, we just multiply them together. So, we take the two Gaussian blobs and multiply them.

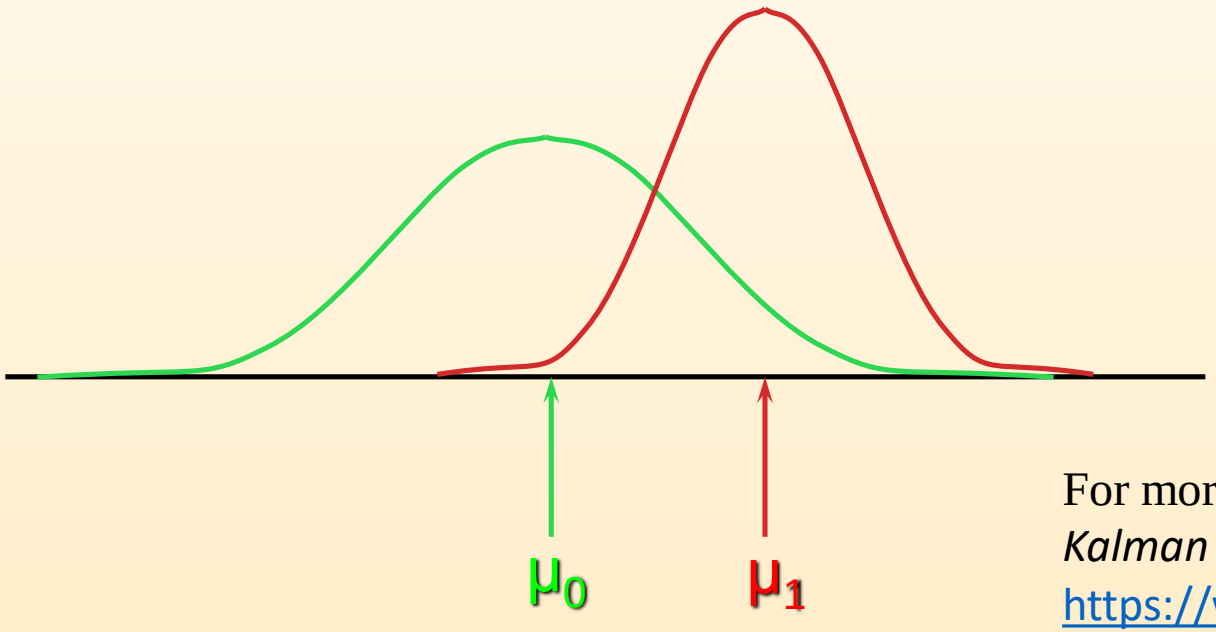
- We can get a new Optimal state estimate by combining two of them.



Combining Gaussians

- After we multiply two of the distributions, We can get a new one

$$\mathcal{N}(x, \mu_0, \sigma_0) \cdot \mathcal{N}(x, \mu_1, \sigma_1) = \mathcal{N}(x, \mu', \sigma')$$



$$\mu' = \mu_0 + \frac{\sigma_0^2}{\sigma_0^2 + \sigma_1^2}(\mu_1 - \mu_0)$$

$$\sigma'^2 = \frac{\sigma_0^2 \sigma_1^2}{\sigma_0^2 + \sigma_1^2}$$

For more detail about how to deduct the formula, you can see the paper *Kalman Filter: Multiplying Normal Distributions*
https://www.Norbert-freier.de/dateien/kalman_filter_multiplying_normal_distributions_norbert_freier_2013.pdf

- We can simplify as below：

$$k = \frac{\sigma_0^2}{\sigma_0^2 + \sigma_1^2}$$

$$\mu' = \mu_0 + k(\mu_1 - \mu_0)$$

$$\sigma'^2 = \sigma_0^2 - k\sigma_0^2$$

- Convert in matrix version:

$$K = \Sigma_0(\Sigma_0 + \Sigma_1)^{-1}$$

$$\vec{\mu}' = \vec{\mu}_0 + K(\vec{\mu}_1 - \vec{\mu}_0)$$

$$\Sigma' = \Sigma_0 - K\Sigma_0$$

K is a matrix called the **Kalman gain**

- Predicted Measurement:

$$(\mu_0, \Sigma_0) = (H_k\hat{x}_k, H_k P_k H_k^T)$$

- Observed Measurement:

$$(\mu_1, \Sigma_1) = (\vec{z}_k, R_k)$$

- Then we can convert these $\begin{cases} K = \Sigma_0(\Sigma_0 + \Sigma_1)^{-1} \\ \vec{\mu}' = \vec{\mu}_0 + K(\vec{\mu}_1 - \vec{\mu}_0) \\ \Sigma' = \Sigma_0 - K\Sigma_0 \end{cases}$ as below

$$K = H_k P_k H_k^T (H_k P_k H_k^T + R_k)^{-1}$$

$$H_k\hat{x}'_k = H_k\hat{x}_k + K(z_k - H_k\hat{x}_k)$$

$$H_k P'_k H_k^T = H_k P_k H_k^T - K H_k P_k H_k^T$$

- We can knock an H_k off the front of every term in $\begin{cases} K = H_k P_k H_k^T (H_k P_k H_k^T + R_k)^{-1} \\ H_k \hat{x}'_k = H_k \hat{x}_k + K(z_k - H_k \hat{x}_k) \\ H_k P'_k H_k^T = H_k P_k H_k^T - K H_k P_k H_k^T \end{cases}$

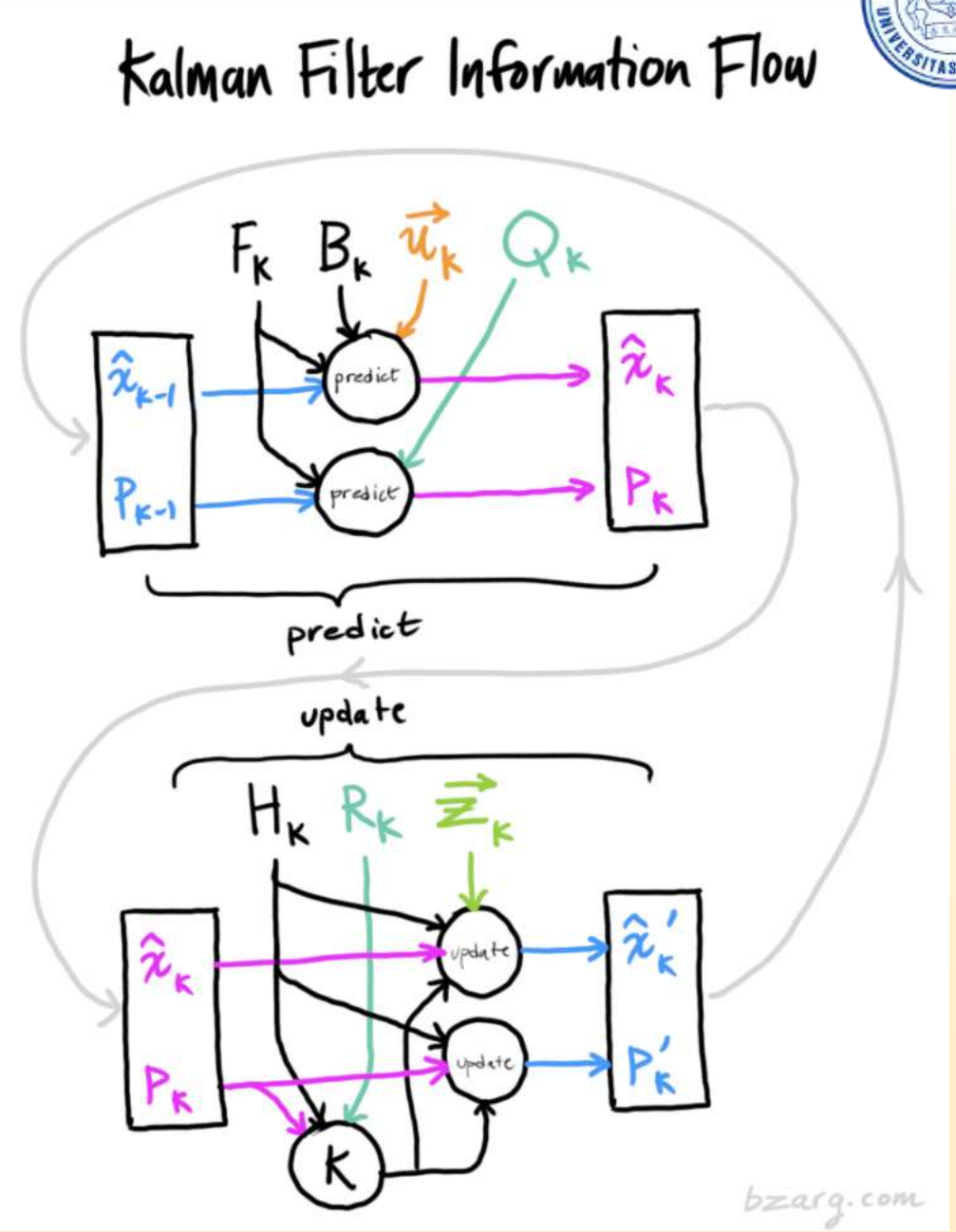
- And an H_k.T off the end of all terms in the equation for P_k'.

$$K' = P_k H_k^T (H_k P_k H_k^T + R_k)^{-1}$$

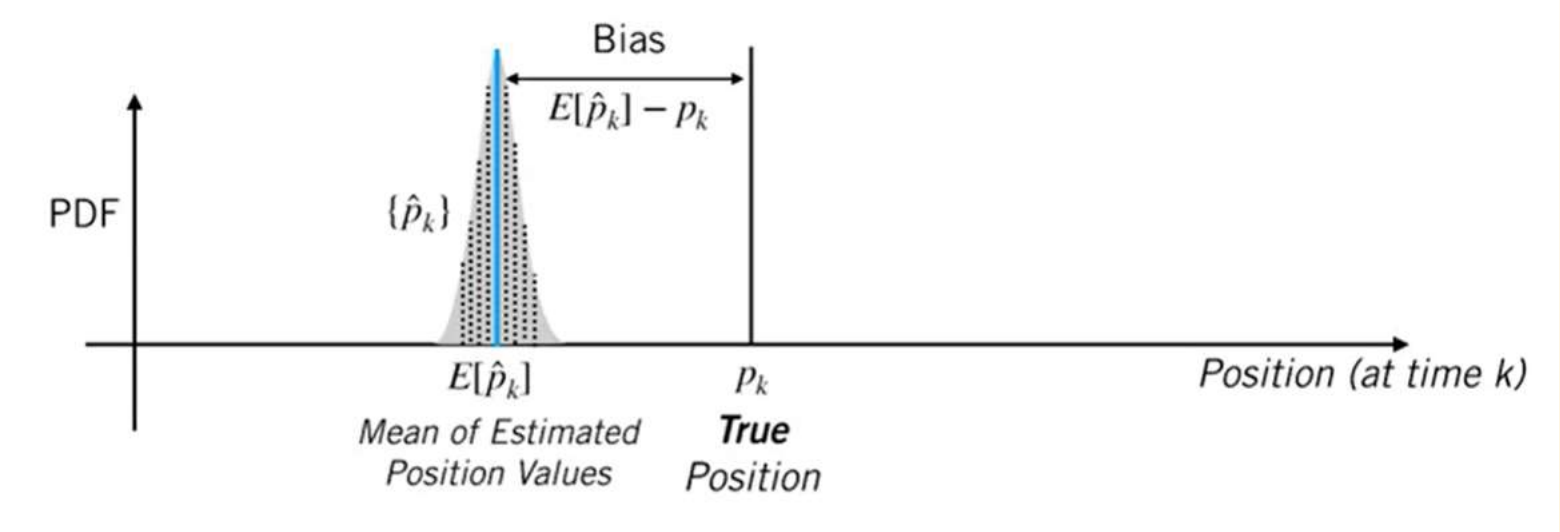
$$\hat{x}'_k = \hat{x}_k + K'(\vec{z}_k - H_k \hat{x}_k)$$

$$P'_k = P_k - K' H_k P_k$$

$\hat{x}'_k$ is our new best estimate, and we can go on and feed it (along with P'_k) back into another round of prediction or update as many times as we like.



Bias

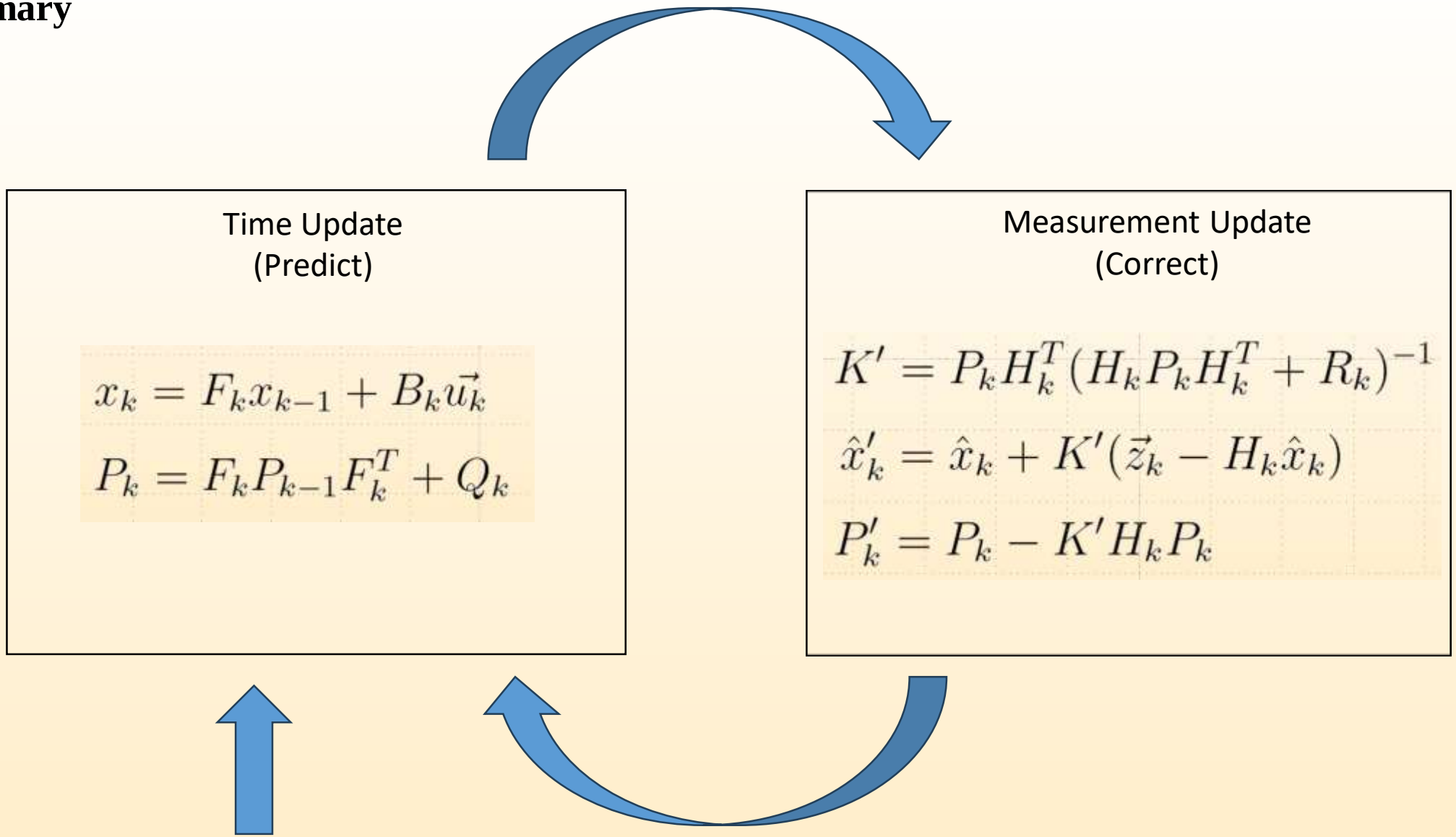


$$E[\hat{e}_k] = E[\hat{p}_k - p_k]$$

$$= E[\hat{p}_k] - p_k = 0$$

If close to 0, means our prediction is well!

Summary

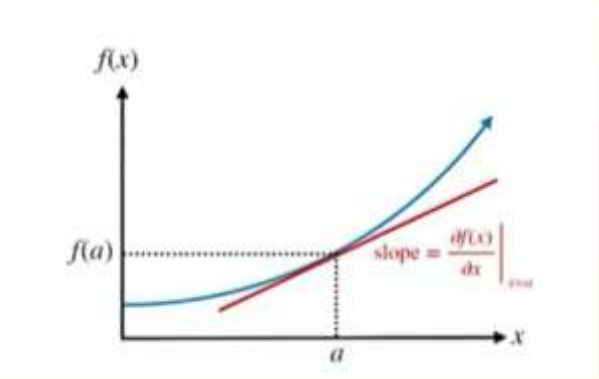


Input x_{k-1} and p_{k-1}

- Extended Kalman Filter (EKF)

Extended Kalman Filter (EKF)

- Hint: Linearization treatment of nonlinear systems



- Use Taylor Expansion

$$f(x) \approx f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots$$

Just focus on First-order terms

- Linearized motion model

$$x_k = f_{k-1}(x_{k-1}, u_{k-1}, w_{k-1})$$

$$\approx f_{k-1}(\hat{x}_{k-1}, u_{k-1}, 0) + \underbrace{\left.\frac{\partial f_{k-1}}{\partial x_{k-1}}\right|_{\hat{x}_{k-1}, u_{k-1}, 0}}_{F_{k-1}} (x_{k-1} - \hat{x}_{k-1}) + \underbrace{\left.\frac{\partial f_{k-1}}{\partial w_{k-1}}\right|_{\hat{x}_{k-1}, u_{k-1}, 0}}_{L_{k-1}} w_{k-1}$$

- Linearized measurement model

$$y_k = h_k(x_k, v_k)$$

$$\approx h_k(\hat{x}_k, 0) + \underbrace{\left.\frac{\partial h_k}{\partial x_k}\right|_{\hat{x}_k, 0}}_{H_k} (x_k - \hat{x}_k) + \underbrace{\left.\frac{\partial h_k}{\partial v_k}\right|_{\hat{x}_k, 0}}_{M_k} v_k$$

Jacobian Matrices

Jacobian Matrices

$$f : \mathbb{R}^n \to \mathbb{R}^m$$

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^{\mathrm{T}} f_1 \\ \vdots \\ \nabla^{\mathrm{T}} f_m \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

- For example:

$$X_k = \begin{bmatrix} x & y & \theta \end{bmatrix}^T \qquad H_k = \frac{\partial h_k}{\partial X_k} = \begin{bmatrix} \dfrac{\partial r}{\partial x_k} & \dfrac{\partial r}{\partial y_k} & \dfrac{\partial r}{\partial \theta_k} \\ \dfrac{\partial \phi}{\partial x_k} & \dfrac{\partial \phi}{\partial y_k} & \dfrac{\partial \phi}{\partial \theta_k} \end{bmatrix}$$

$$Y_k = \begin{bmatrix} r & \phi \end{bmatrix}^T$$

H_k M_k is Jacobian Matrices.

So the Kalman gain will change a little:

$$K_k = P_k \mathbf{H}_k^T \left(\mathbf{H}_k P_k \mathbf{H}_k^T + \mathbf{M}_k \mathbf{R}_k \mathbf{M}_k^T \right)^{-1}$$

- The rest part is the same as the Kalman Filter
- We could see more detail in the Project1

Background of Project 1

$$\mathbf{x}_k = \mathbf{x}_{k-1} + T \begin{bmatrix} \cos\theta_{k-1} & 0 \\ \sin\theta_{k-1} & 0 \\ 0 & 1 \end{bmatrix} \left(\begin{bmatrix} v_k \\ \omega_k \end{bmatrix} + \mathbf{w}_k \right), \quad \mathbf{w}_k = \mathcal{N}(0, Q)$$

- Time between status updates:

$$T$$

- Velocity Vector:

$$\begin{bmatrix} v_k \\ \omega_k \end{bmatrix}$$

- Noise:

$$\mathbf{w}_k$$

- Rotation Matrix:

$$\begin{bmatrix} \cos\theta_{k-1} & 0 \\ \sin\theta_{k-1} & 0 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos(\theta_{k-1}) & 0 \\ \sin(\theta_{k-1}) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_k \\ \omega_k \end{bmatrix} = \begin{bmatrix} v_k \cdot \cos(\theta_{k-1}) \\ v_k \cdot \sin(\theta_{k-1}) \\ \omega_k \end{bmatrix}$$

- This function relates the vehicle's current state—its position and orientation—to the range and bearing measurements obtained from LIDAR for a specific landmark.

$$y_k^l = \begin{bmatrix} \sqrt{(x_l - x_k - d\cos\theta_k)^2 + (y_l - y_k - d\sin\theta_k)^2} \\ \operatorname{atan2}(y_l - y_k - d\sin\theta_k, x_l - x_k - d\cos\theta_k) - \theta_k \end{bmatrix} + n_k^l, \quad n_k^l = \mathcal{N}(0, R)$$

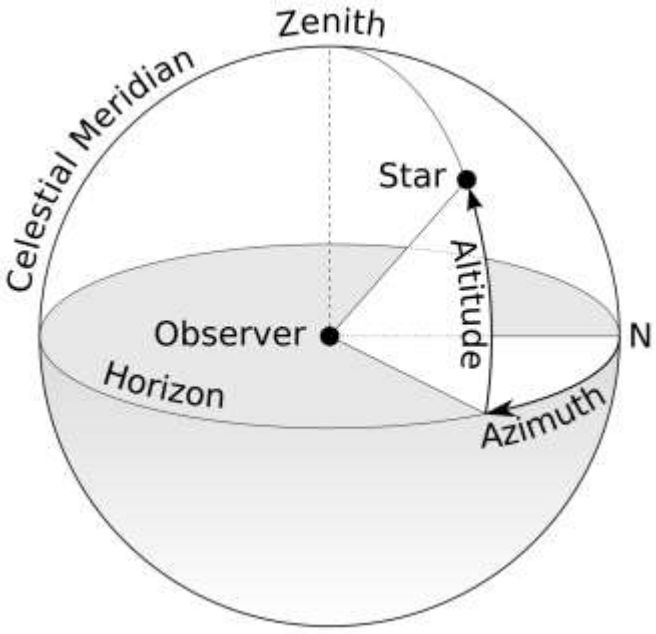
$$y_k^l = \begin{bmatrix} r \\ \phi \end{bmatrix}$$ Distance

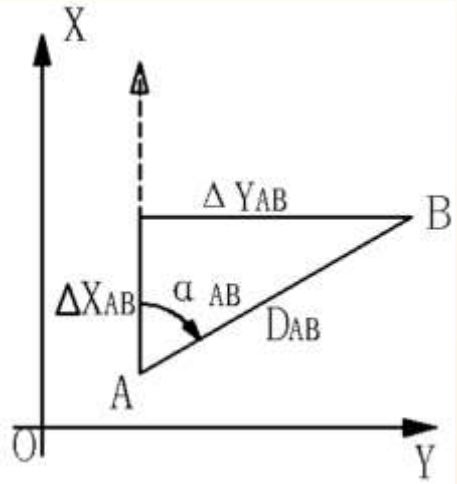
Azimuth

- x^l and y^l are the x and y coordinates of the landmark l.
- x_k and y_k are the x and y position coordinates of the vehicle at time step k.
- θ_k is the orientation of the vehicle at time step k.
- d is the known offset distance between the vehicle's center and the LIDAR sensor.
- n_k^l is the measurement noise with a mean of 0 and a covariance matrix R, representing the uncertainty in the measurements.

- "Landmark" refers to any distinct physical object or feature in the environment that can be easily identified and has a known location that doesn't change over time. Landmarks are crucial for localization because they serve as reference points; robots or vehicles can determine their own location and orientation by measuring their distance and angle relative to these landmarks.

- azimuth angle is the angular measurement in a spherical coordinate system that represents the horizontal angle from a cardinal direction, most commonly north.





- We know two points, we can get azimuth angle

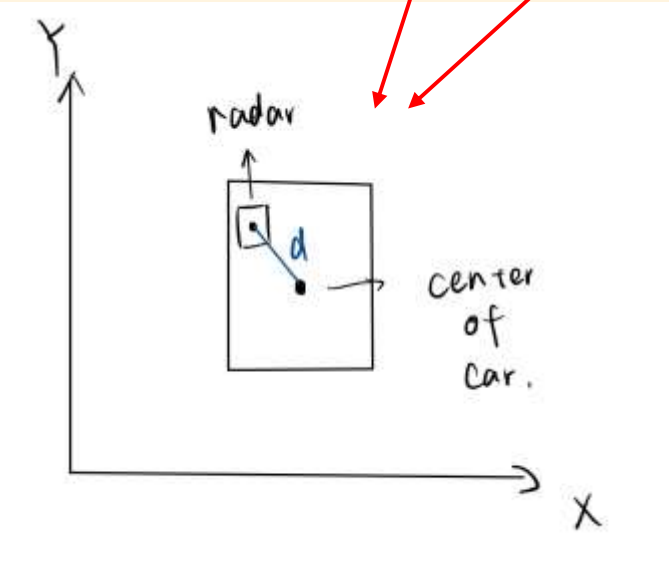
$$\alpha = \text{atan2}(X_2 - X_1, Y_2 - Y_1)$$

- We know one azimuth angle, we can get another point

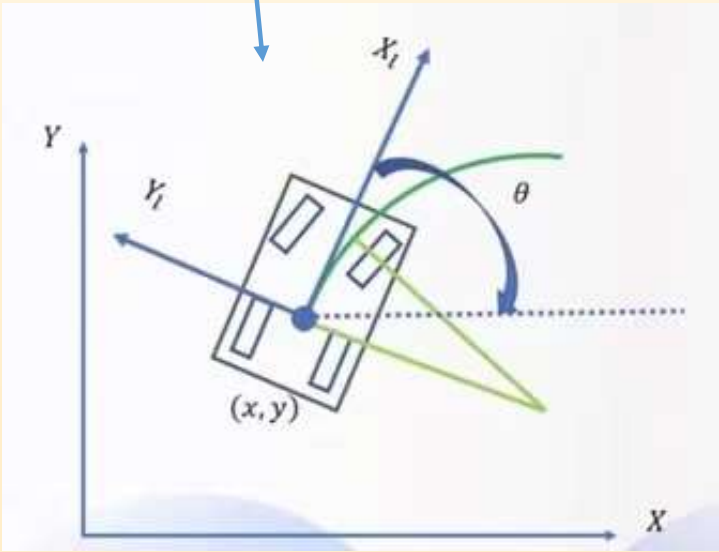
$$X_2 = X_1 + D\sin\alpha$$
$$Y_2 = Y_1 + D\cos\alpha$$

$$\text{atan2}(y_l - y_k - d\sin\theta_k, x_l - x_k - d\cos\theta_k) - \theta_k$$

- When the lidar is not at the geometric center of the vehicle, you must correct the lidar readings to ensure that they are accurate relative to the geometric center of the vehicle



- When you use the atan2 function, what you get is the angle of the vector pointing from the global coordinate system to the landmark. To convert this angle to an angle relative to the vehicle's current heading, you need to subtract the vehicle's heading angle θk from the result of atan2



Correction Step

First, let's implement the measurement update function, which takes an available landmark measurement l and updates the current state estimate $\tilde{\mathbf{x}}_k$. For each landmark measurement received at a given timestep k, you should implement the following steps:

- Compute the measurement model Jacobians at $\tilde{\mathbf{x}}_k$

$$\mathbf{y}_k^l = \mathbf{h}(\mathbf{x}_k, \mathbf{n}_k^l)$$

$$\mathbf{H}_k = \frac{\partial \mathbf{h}}{\partial \mathbf{x}_k}\Big|_{\tilde{\mathbf{x}}_k,0}, \quad \mathbf{M}_k = \frac{\partial \mathbf{h}}{\partial \mathbf{n}_k}\Big|_{\tilde{\mathbf{x}}_k,0}.$$

- Compute the Kalman Gain

$$\mathbf{K}_k = \tilde{\mathbf{P}}_k \mathbf{H}_k^T \left(\mathbf{H}_k \tilde{\mathbf{P}}_k \mathbf{H}_k^T + \mathbf{M}_k \mathbf{R}_k \mathbf{M}_k^T\right)^{-1}$$

- Correct the predicted state

$$\tilde{\mathbf{y}}_k^l = \mathbf{h}\left(\tilde{\mathbf{x}}_k, \mathbf{0}\right)$$
$$\hat{\mathbf{x}}_k = \tilde{\mathbf{x}}_k + \mathbf{K}_k\left(\mathbf{y}_k^l - \tilde{\mathbf{y}}_k^l\right)$$

Suppose there is no noise

- Correct the covariance

$$\hat{\mathbf{P}}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\tilde{\mathbf{P}}_k$$

$$H_k = \frac{\partial h_k}{\partial X_k} = \begin{bmatrix} \frac{\partial r}{\partial x_k} & \frac{\partial r}{\partial y_k} & \frac{\partial r}{\partial \theta_k} \\ \frac{\partial \phi}{\partial x_k} & \frac{\partial \phi}{\partial y_k} & \frac{\partial \phi}{\partial \theta_k} \end{bmatrix}$$

$$M_k = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Mk is an Identity Matrix. It is an Additive and independent Noise

Prediction Step

Now, implement the main filter loop, defining the prediction step of the EKF using the motion model provided:

$$\check{\mathbf{x}}_k = \mathbf{f}\left(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}, \mathbf{0}\right)$$

Suppose there is no noise

$$\check{\mathbf{P}}_k = \mathbf{F}_{k-1}\hat{\mathbf{P}}_{k-1}\mathbf{F}_{k-1}^T + \mathbf{L}_{k-1}\mathbf{Q}_{k-1}\mathbf{L}_{k-1}^T .$$

Where

$$\mathbf{F}_{k-1} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}_{k-1}}\right|_{\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k, 0}, \quad \mathbf{L}_{k-1} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{w}_k}\right|_{\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k, 0} .$$

$$\begin{aligned} x_k &= x_{k-1} + v_{k-1}\Delta t \cos(\theta_{k-1}) \\ y_k &= y_{k-1} + v_{k-1}\Delta t \sin(\theta_{k-1}) \\ \theta_k &= \theta_{k-1} + \omega_{k-1}\Delta t \end{aligned}$$

$$\mathbf{F}_{k-1} = \begin{bmatrix} \frac{\partial x_k}{\partial x_{k-1}} & \frac{\partial y_k}{\partial x_{k-1}} & \frac{\partial \theta_k}{\partial x_{k-1}} \\ \frac{\partial x_k}{\partial y_{k-1}} & \frac{\partial y_k}{\partial y_{k-1}} & \frac{\partial y_k}{\partial \theta_{k-1}} \\ \frac{\partial x_k}{\partial \theta_{k-1}} & \frac{\partial y_k}{\partial \theta_{k-1}} & \frac{\partial \theta_k}{\partial \theta_{k-1}} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -v\Delta t \sin(\theta) \\ 0 & 1 & v\Delta t \cos(\theta) \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{L}_{k-1} = \begin{bmatrix} \frac{\partial \Delta t \cos(\theta)\mathbf{w}_k}{\partial \mathbf{w}_k} & 0 \\ \frac{\partial \Delta t \sin(\theta)\mathbf{w}_k}{\partial \mathbf{w}_k} & 0 \\ 0 & \frac{\partial \Delta t \mathbf{w}_k}{\partial \mathbf{w}_k} \end{bmatrix} = \begin{bmatrix} \Delta t \cos(\theta) & 0 \\ \Delta t \sin(\theta) & 0 \\ 0 & \Delta t \end{bmatrix}$$

Reference

- Babb, T. (n.d.). How a Kalman filter works, in pictures. *Bzarg*. https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/

- "State Estimation and Localization for Self-Driving Cars" launched by the University of Toronto on Coursera. https://www.coursera.org/learn/state-estimation-localization-self-driving-cars/home/week/2

- *Understanding Kalman filters, part 3: Optimal State Estimator*. MATLAB. (n.d.). https://ww2.mathworks.cn/videos/understanding-kalman-filters-part-3-optimal-state-estimator--1490710645421.html

- STC Lecture series an introduction to the Kalman filter. (n.d.). https://www.cs.unc.edu/~welch/media/pdf/kalmanIntroSlides.pdf