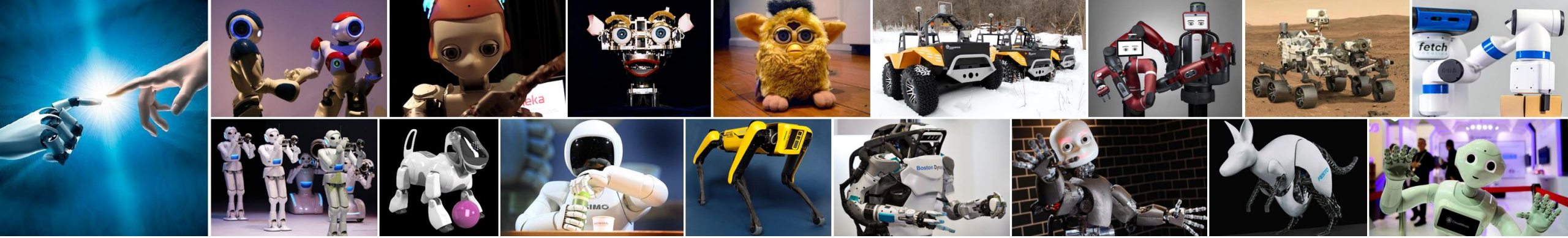


COMPSCI-603: Robotics

Kalman Filter for State Estimation

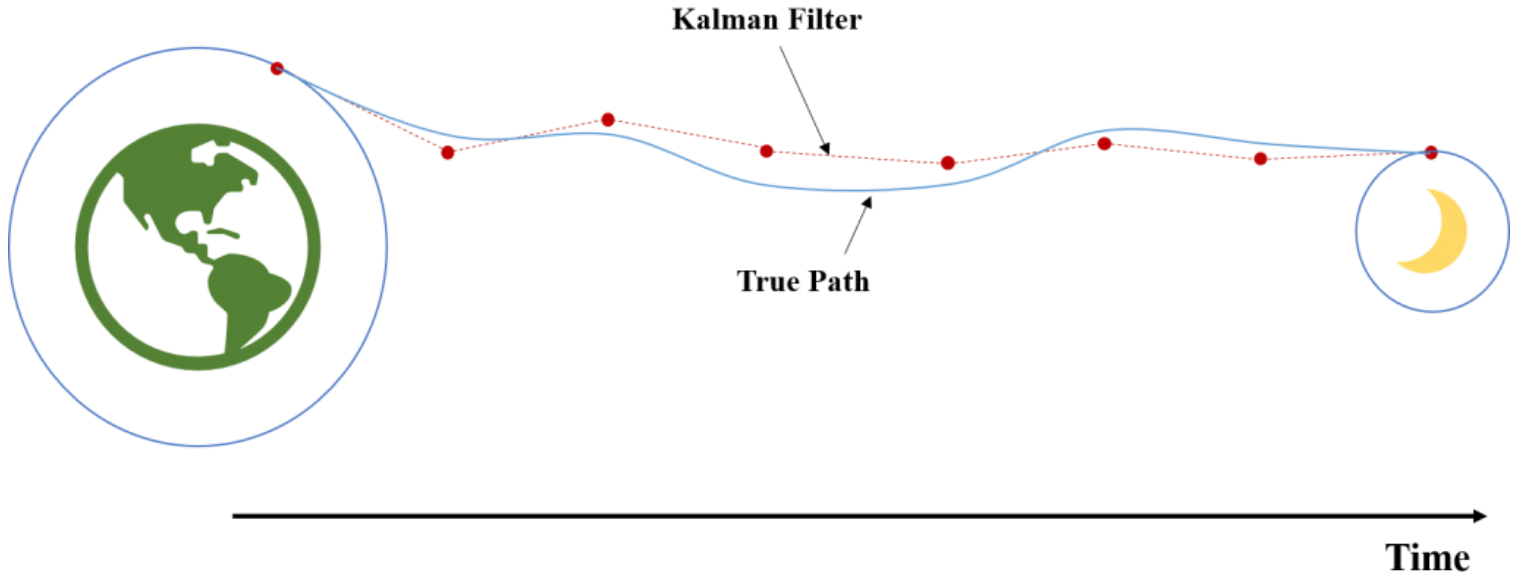
Partial slide courtesy by K. Kitani and C. Stachniss

Kalman Filter

Chapter 3.2, Sebastian Thrun, Wolfram Burgard and Dieter Fox.
“Probabilistic Robotics.” MIT Press. 2005.

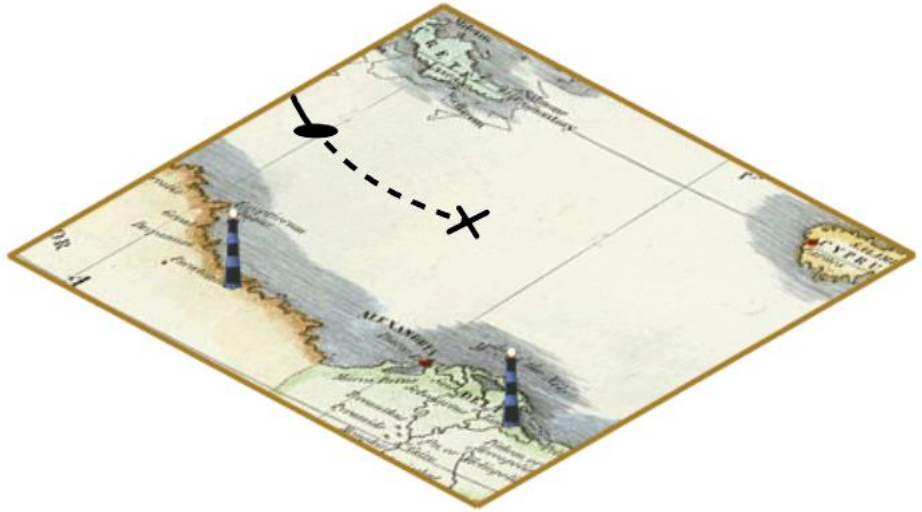
Kalman Filter

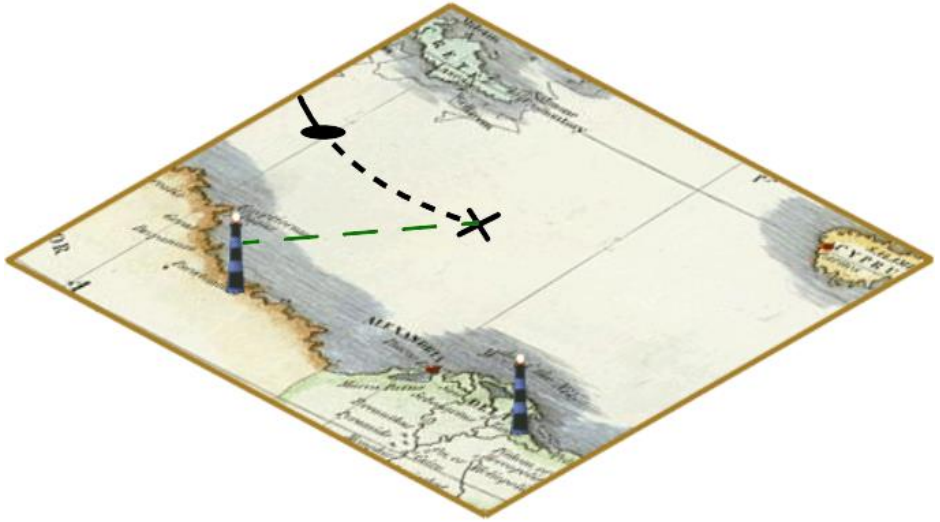
- Kalman filter is a Bayes filter for the *linear Gaussian* case.
- It performs recursive state estimation.
  - Prediction/estimation step exploits the motion/control/action.
  - Update/correction step exploits the observation/sensing/observation.

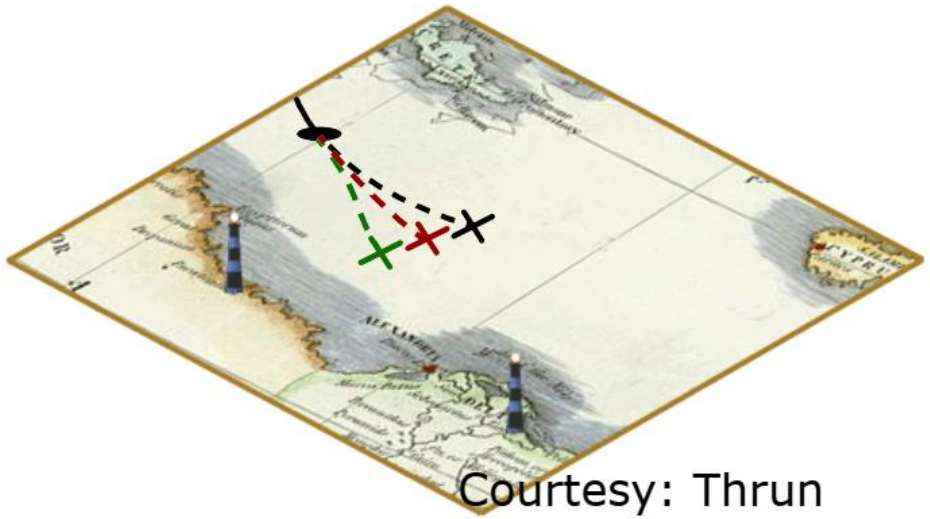


How NASA used the Kalman Filter in the Apollo Program: https://www.lancaster.ac.uk/stor-i-student-sites/jack-trainer/how-nasa-used-the-kalman-filter-in-the-apollo-program

Kalman Filter Example







Courtesy: Thrun

Bayes Filter and Kalman Filter

- Bayes filter is a mathematical tool for state estimation:
  - Prediction/estimation:

$$\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\ bel(x_{t-1})\, dx_{t-1}$$

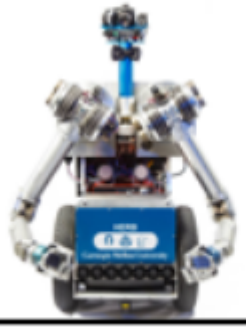
  - Correction/update:

$$bel(x_t) = \eta\, p(z_t \mid x_t)\ \overline{bel}(x_t)$$

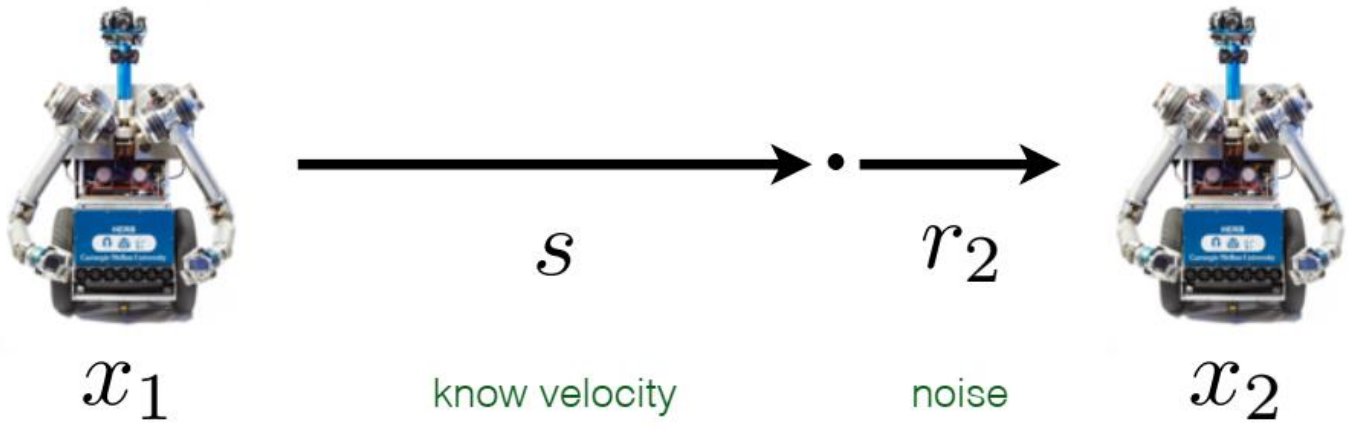
- Kalman filter is an estimator for the linear Gaussian case.
- It is an optimal solution for linear models with Gaussian distributions.

1D Example of Kalman Filter

x



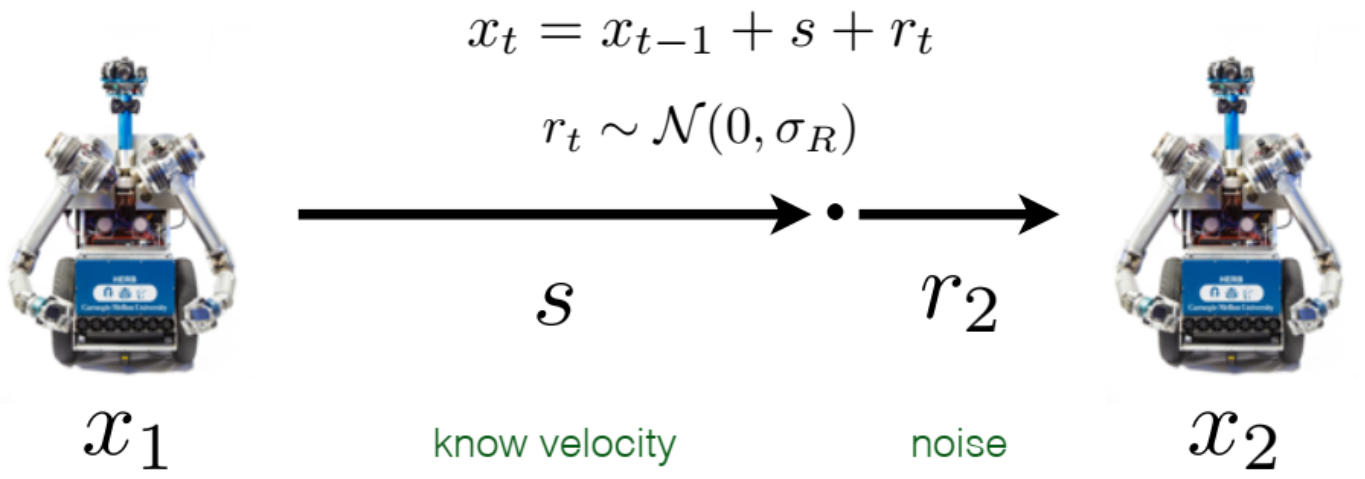
Motion Model



Motion $x_t = x_{t-1} + s + r_t$

$$r_t \sim \mathcal{N}(0, \sigma_R)$$

Motion Model

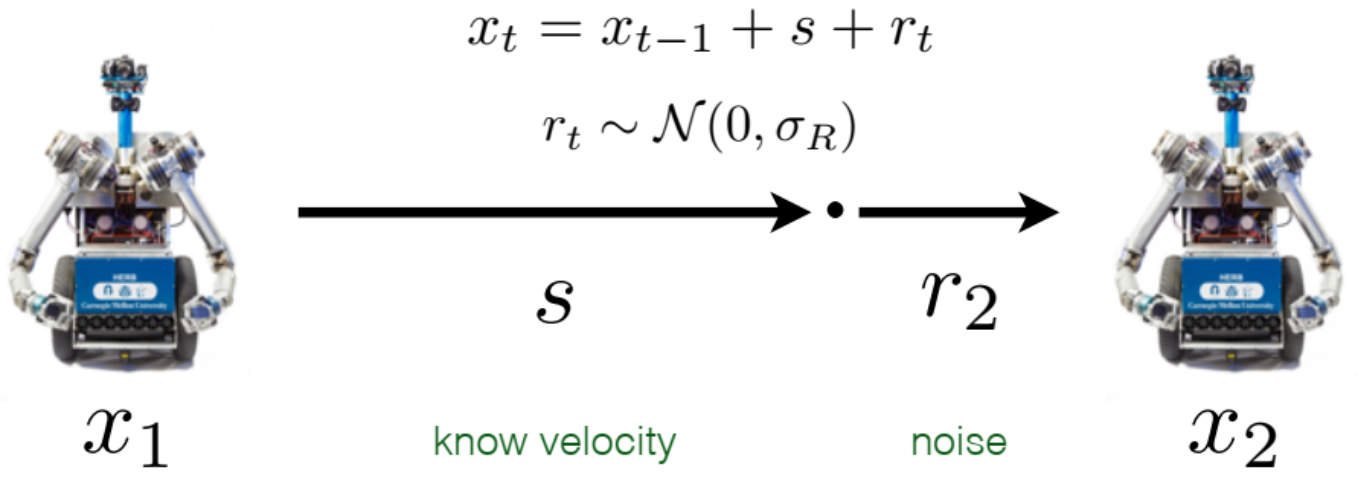


- How can we represent the motion model $P(x_t|x_{t-1})$?
- It is a linear Gaussian model.

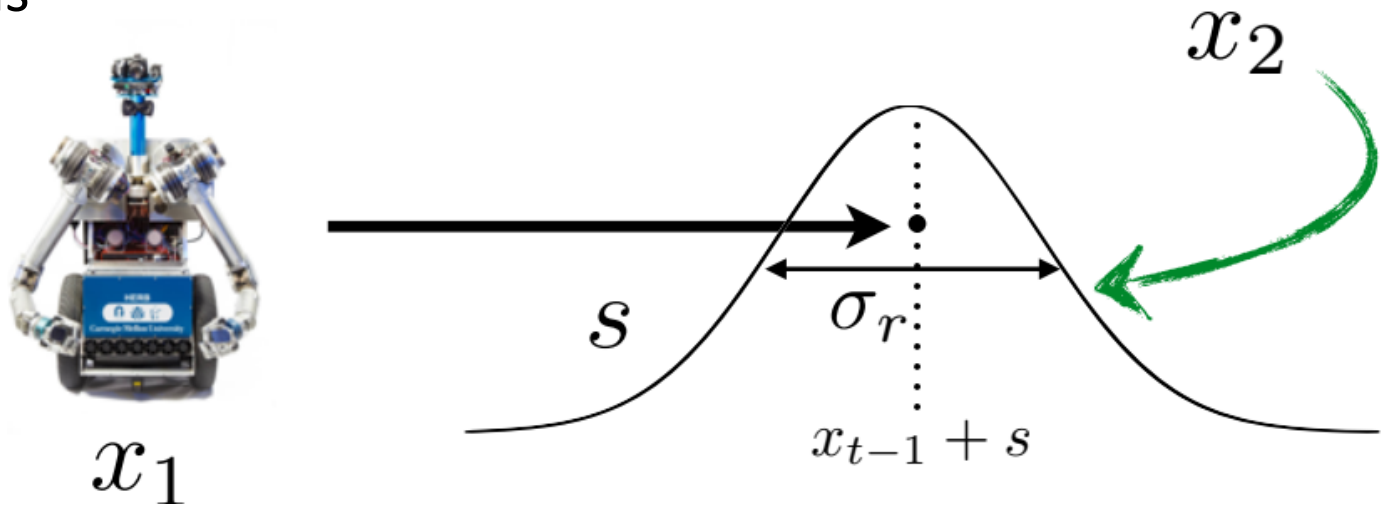
$$P(x_t|x_{t-1}) = \mathcal{N}(x_t; x_{t-1} + s, \sigma_r)$$

Mean Standard Deviation

Motion Model



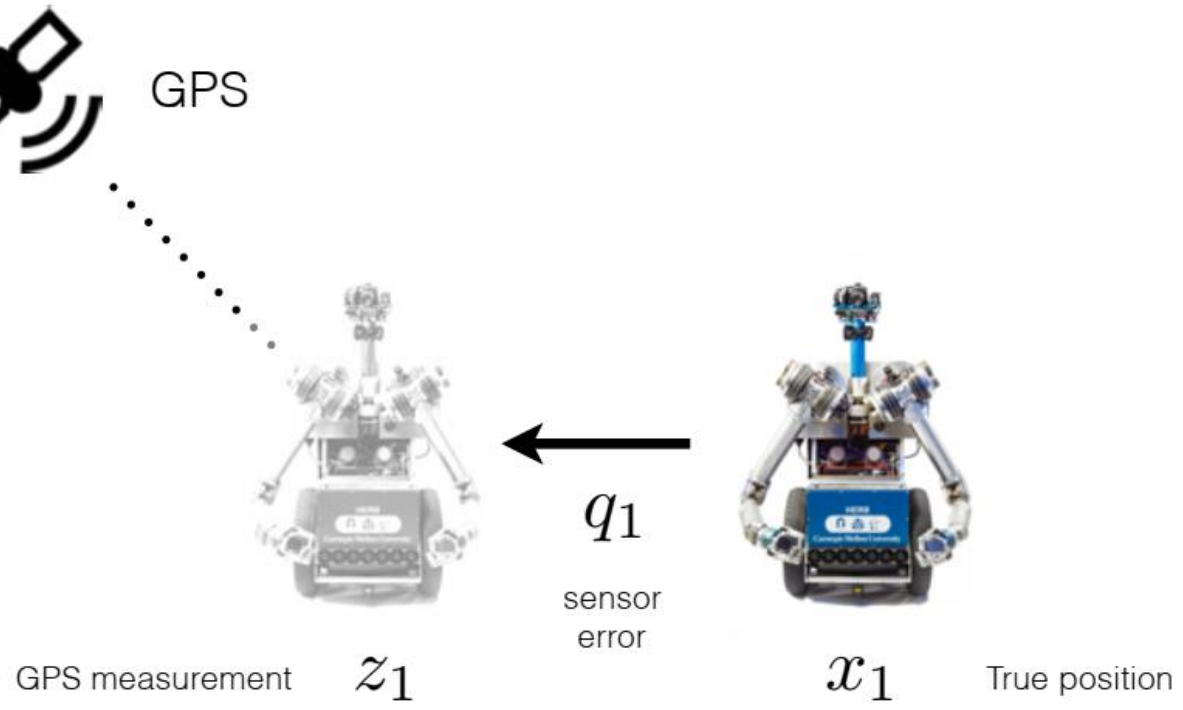
Visualization of this distribution from motion model:



Sensor Model

Sensing $z_t = x_t + q_t$

$$q_t \sim \mathcal{N}(0, \sigma_Q)$$

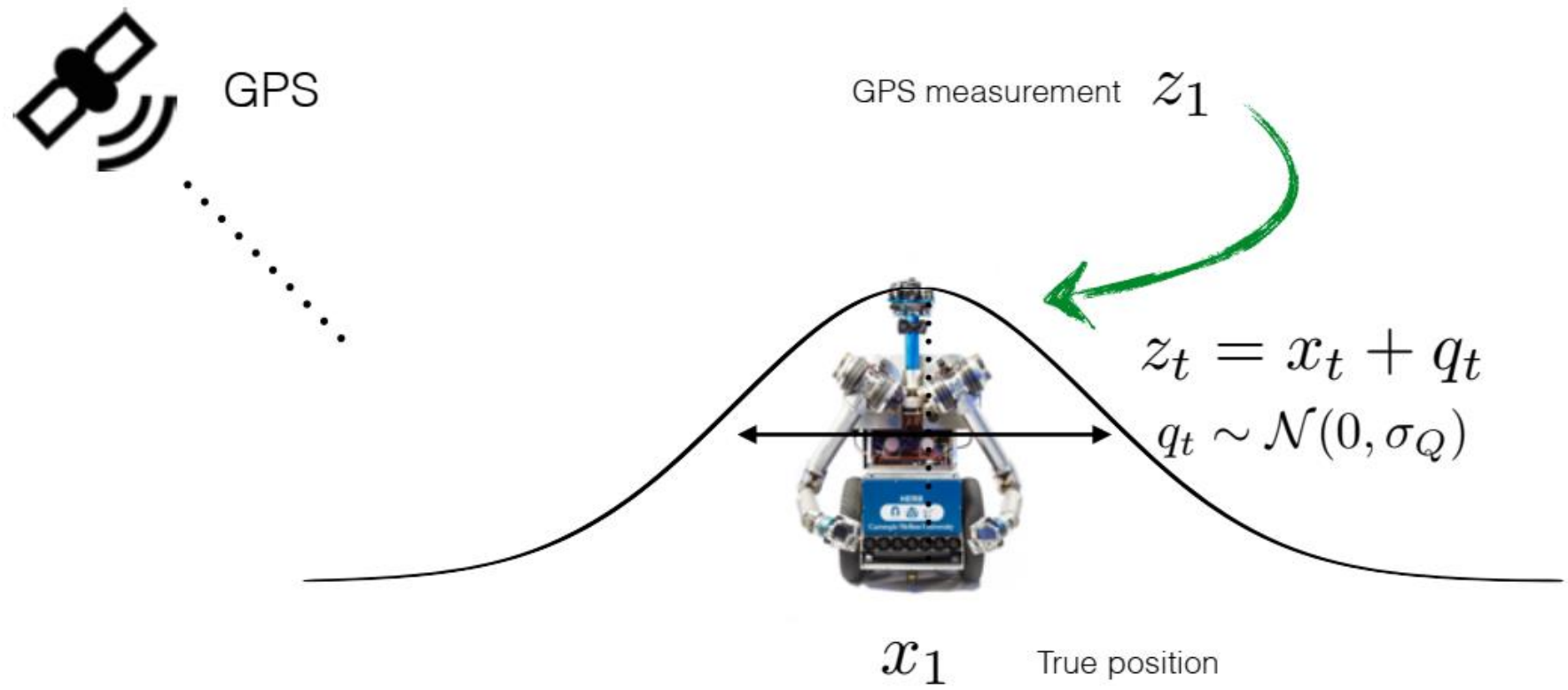


- How to represent the sensor / observation model?
- It is also a linear Gaussian model:

$$P(z_t|x_t) = \mathcal{N}(z_t; x_t, \sigma_Q)$$

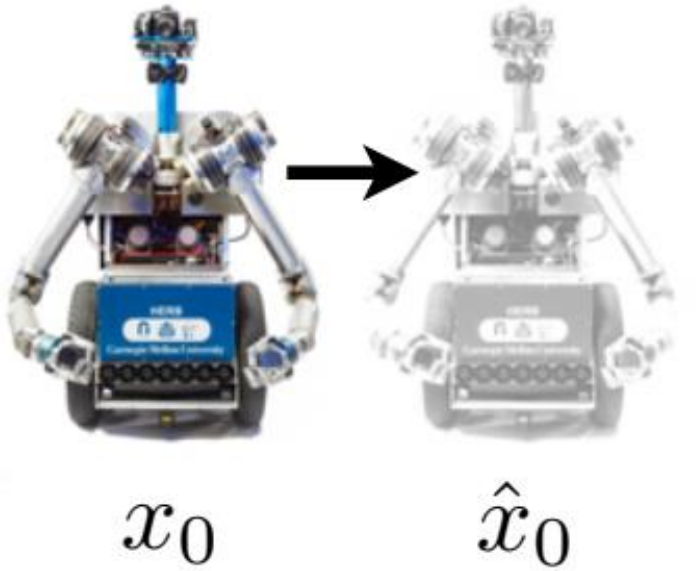
Sensor Model

- Visualization of the sensor model:



Prior State Distribution

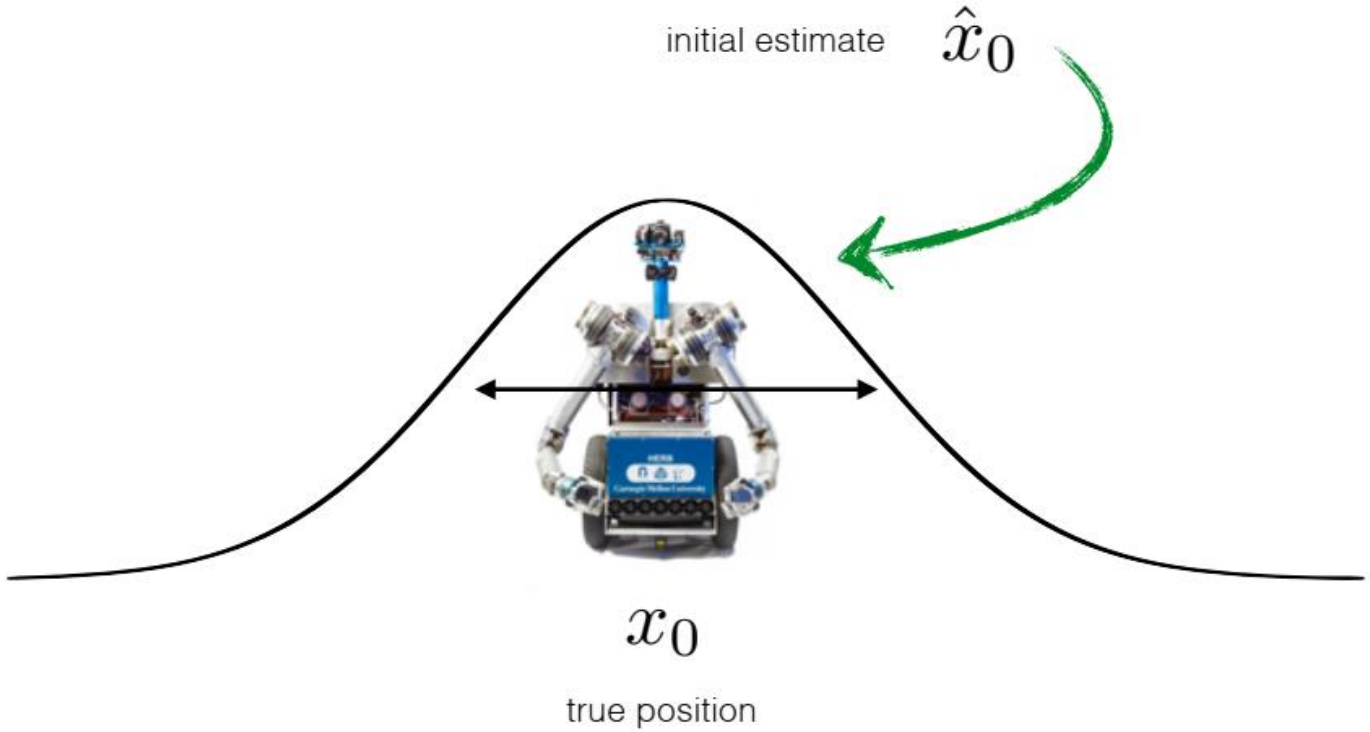
- Prior state distribution is assumed to be a linear Gaussian model:



x_0 true position

$\hat{x}_0$ initial estimate

initial estimate uncertainty σ_0



$$P(\hat{x}_0) = \mathcal{N}(\hat{x}_0; x_0, \sigma_0)$$

The 'cap' notation denotes 'estimate'

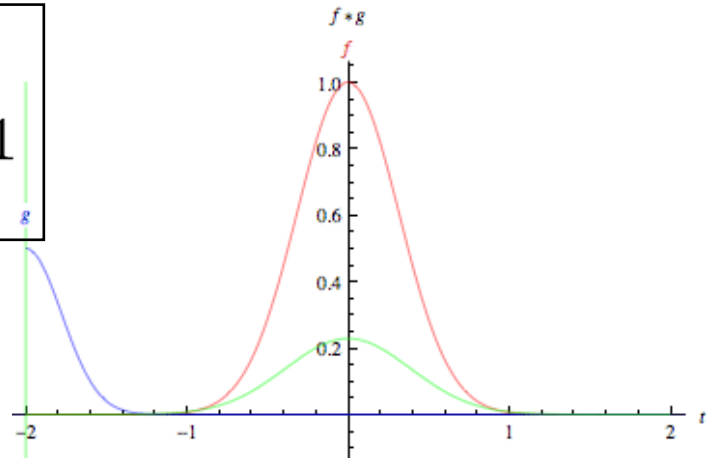
Prediction/Estimation

- How can we predict $\hat{x}_1$ given $\hat{x}_0$?
- Prediction/Estimation: We use the prediction step to estimate the belief using the motion model

$$\underbrace{\overline{bel}(x_t)}_{\hat{x}_1} = \underbrace{\int p(x_t \mid u_t, x_{t-1})\ bel(x_{t-1})\, dx_{t-1}}_{\hat{x}_0}$$

Mean of the new estimate: $\hat{x}_1 = \hat{x}_0 + s$

Variance of the new estimate: $\sigma_1^2 = \sigma_0^2 + \sigma_r^2$

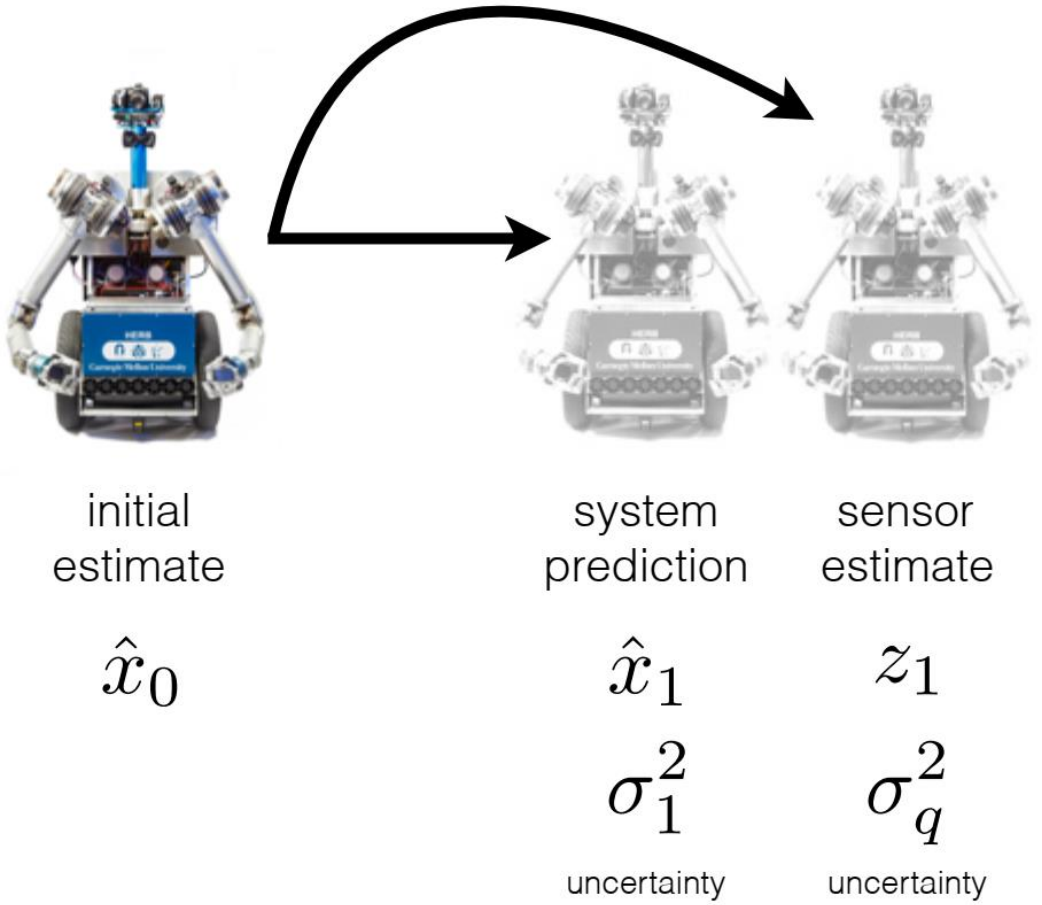


https://mathworld.wolfram.com/Convolution.html

Correction/Update

- How can we update the estimated belief?

- Correction/Update: We use the sensor model to update the estimated belief.
- Given the uncertainty (encoded by the variance) of the prediction and sensor estimate, which one should we trust more?
- How to merge the information?



Correction/Update

- Intuitively, the smaller variance means less uncertainty, so that we can trust it more.
- Thus, we want a weighted state estimate correction.
- Something like this:

system prediction σ_1^2

sensor estimate σ_q^2

$$\hat{x}_1^+ = \frac{\sigma_q^2}{\sigma_1^2 + \sigma_q^2}\hat{x}_1 + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_q^2}z_1$$

Correction/Update

- This happens naturally in the Bayes filtering (with Gaussians) framework:

$$bel(x_t) = \eta\, p(z_t \mid x_t)\ \overline{bel}(x_t)$$

mean: z_1 variance: σ_q

mean: $\hat{x}_1$ variance: σ_1

new mean:

$$\hat{x}_1^+ = \frac{\hat{x}_1\sigma_q^2 + z_1\sigma_1^2}{\sigma_q^2 + \sigma_1^2}$$

new variance:

$$\hat{\sigma}_1^{2+} = \frac{\sigma_q^2\sigma_1^2}{\sigma_q^2 + \sigma_1^2}$$

'plus' sign denotes post 'update' estimate (posterior)

Correction/Update

- As a recall from a math class…

$$x_1(t) \triangleq \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-\frac{(t-\mu_1)^2}{2\sigma_1^2}}$$

$$x_2(t) \triangleq \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-\frac{(t-\mu_2)^2}{2\sigma_2^2}}$$

$$\mu = \frac{\frac{\mu_1}{2\sigma_1^2} + \frac{\mu_2}{2\sigma_2^2}}{\frac{1}{2\sigma_1^2} + \frac{1}{2\sigma_2^2}} = \frac{\mu_1\sigma_2^2 + \mu_2\sigma_1^2}{\sigma_2^2 + \sigma_1^2}$$

$$\sigma^2 = \sigma_1^2 \| \sigma_2^2 \triangleq \frac{1}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}} = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2}.$$

Correction/Update

system prediction σ_1^2

sensor estimate σ_q^2

- With a little algebra, we get a weighted state estimate correction:

$$\hat{x}_1^+ = \frac{\hat{x}_1\sigma_q^2 + z_1\sigma_1^2}{\sigma_q^2 + \sigma_1^2} = \hat{x}_1 \frac{\sigma_q^2}{\sigma_q^2 + \sigma_1^2} + z_1 \frac{\sigma_1^2}{\sigma_q^2 + \sigma_1^2}$$

Kalman Gain

- With more algebra, we can rewrite the new mean and variance as:

$$\hat{x}_1^+ = \hat{x}_1 + \frac{\sigma_1^2}{\sigma_q^2 + \sigma_1^2}(z_1 - \hat{x}_1) = \hat{x}_1 + K(z_1 - \hat{x}_1)$$

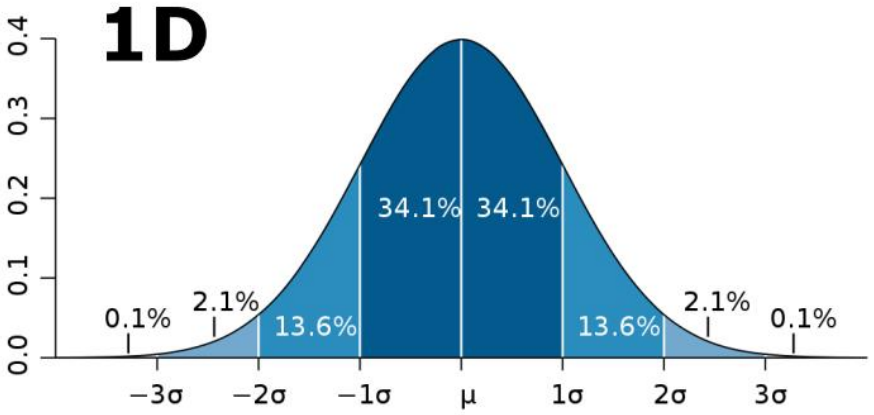
Kalman gain Innovation

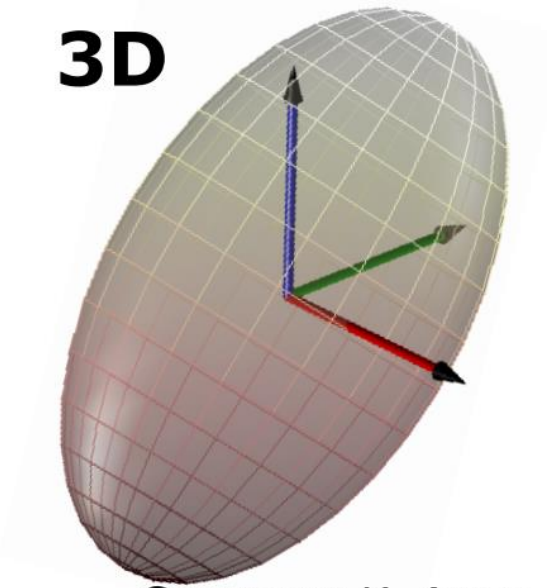
$$\sigma_1^+ = \frac{\sigma_1^2 \sigma_q^2}{\sigma_1^2 + \sigma_q^2} = \left(1 - \frac{\sigma_1^2}{\sigma_1^2 + \sigma_q^2}\right)\sigma_1^2 = (1 - K)\,\sigma_1^2$$

Moving to General Kalman Filter

- Everything is multi-variant Gaussian:

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu) \right)$$





Courtesy: K. Arras

Linear Models for Kalman Filter

- Kalman filter *assumes* linear models for motions and observations.

$$f(x) = A\,x + b$$

- Kalman filter *assumes* zero mean Gaussian noise in the linear models:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

$$z_t = C_t x_t + \delta_t$$

Linear Gaussian Motion Model

- We can represent the linear motion model

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

as a Gaussian probability distribution for the Bayes filter framework:

$$p(x_t \mid u_t, x_{t-1}) = \det(2\pi R_t)^{-\frac{1}{2}}$$
$$\exp\left(-\frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T R_t^{-1}(x_t - A_t x_{t-1} - B_t u_t)\right)$$

where R_t is covariance that encodes the noise of the motion.

Linear Gaussian Sensor Model

- We can also represent the linear sensor model

$$z_t = C_t x_t + \delta_t$$

as a Gaussian probability distribution for the Bayes filter framework:

$$p(z_t \mid x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(z_t - C_t x_t)^T Q_t^{-1}(z_t - C_t x_t)\right)$$

where Q_t is covariance that encodes the noise of the sensor.

General Kalman Filter Variables

- A_t: Matrix $(n \times n)$ that describes how the state evolves from $t-1$ to t without control or noise.
- B_t: Matrix $(n \times l)$ that describes how the control u_t changes the state from $t-1$ to t.
- C_t: Matrix $(k \times n)$ that describes how to map/project the state x_t to an observation z_t.
- ϵ_t and δ_t: Random variables representing the motion and sensor noise that are assumed to be independent and normally distributed with covariance R_t and Q_t respectively.

General Kalman Filter Algorithm

$$\text{KalmanFilter}(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$$

A_t: Matrix $(n \times n)$ that describes how the state evolves from $t-1$ to t without control or noise.

B_t: Matrix $(n \times l)$ that describes how the control u_t changes the state from $t-1$ to t.

C_t: Matrix $(k \times n)$ that describes how to map/project the state x_t to an observation z_t.

ϵ_t and δ_t : Random variables representing the motion and sensor noise that are assumed to be independent and normally distributed with covariance R_t and Q_t respectively.

General Kalman Filter Algorithm

$$\text{KalmanFilter}(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$$

prediction mean — motion — control — 'old' mean

$$\bar{\mu}_t = A_t \mu_{t-1} + B u_t$$

prediction covariance — 'old' covariance — Gaussian noise

$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^\top + R$$

A_t: Matrix $(n \times n)$ that describes how the state evolves from $t-1$ to t without control or noise.

B_t: Matrix $(n \times l)$ that describes how the control u_t changes the state from $t-1$ to t.

C_t: Matrix $(k \times n)$ that describes how to map/project the state x_t to an observation z_t.

ϵ_t and δ_t : Random variables representing the motion and sensor noise that are assumed to be independent and normally distributed with covariance R_t and Q_t respectively.

Prediction (Slide 23)

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

General Kalman Filter Algorithm

$$\text{KalmanFilter}(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$$

prediction mean — motion — control — 'old' mean

$$\bar{\mu}_t = A_t \mu_{t-1} + B u_t$$

prediction covariance — 'old' covariance — Gaussian noise

$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^\top + R$$

$$K_t = \bar{\Sigma}_t C_t^\top (C_t \bar{\Sigma}_t C_t^\top + Q_t)^{-1}$$

A_t: Matrix $(n \times n)$ that describes how the state evolves from $t-1$ to t without control or noise.

B_t: Matrix $(n \times l)$ that describes how the control u_t changes the state from $t-1$ to t.

C_t: Matrix $(k \times n)$ that describes how to map/project the state x_t to an observation z_t.

ϵ_t and δ_t : Random variables representing the motion and sensor noise that are assumed to be independent and normally distributed with covariance R_t and Q_t respectively.

Prediction (Slide 23)

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

Gain

(Slide 20, in 1D)

$$\hat{x}_1^+ = \hat{x}_1 + \frac{\sigma_1^2}{\sigma_q^2 + \sigma_1^2}(z_1 - \hat{x}_1) = \hat{x}_1 + K(z_1 - \hat{x}_1)$$

$$\sigma_1^+ = \frac{\sigma_1^2 \sigma_q^2}{\sigma_1^2 + \sigma_q^2} = \left(1 - \frac{\sigma_1^2}{\sigma_1^2 + \sigma_q^2}\right)\sigma_1^2 = (1-K)\,\sigma_1^2$$

General Kalman Filter Algorithm

A_t: Matrix $(n \times n)$ that describes how the state evolves from $t-1$ to t without control or noise.

B_t: Matrix $(n \times l)$ that describes how the control u_t changes the state from $t-1$ to t.

C_t: Matrix $(k \times n)$ that describes how to map/project the state x_t to an observation z_t.

ϵ_t and δ_t : Random variables representing the motion and sensor noise that are assumed to be independent and normally distributed with covariance R_t and Q_t respectively.

$$\text{KalmanFilter}(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$$

prediction mean

motion control

$$\bar{\mu}_t = A_t \mu_{t-1} + B u_t$$

'old' mean

prediction covariance

'old' covariance

$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^\top + R$$

Gaussian noise

$$K_t = \bar{\Sigma}_t C_t^\top (C_t \bar{\Sigma}_t C_t^\top + Q_t)^{-1}$$

observation model

update mean

$$\mu_t = \bar{\mu}_t + K_t(z_t - C_t \bar{\mu}_t)$$

update covariance

$$\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$$

Prediction (Slide 23)

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

Gain

$$\hat{x}_1^+ = \hat{x}_1 + \frac{\sigma_1^2}{\sigma_q^2 + \sigma_1^2}(z_1 - \hat{x}_1) = \hat{x}_1 + K(z_1 - \hat{x}_1)$$

(Slide 20, in 1D)

$$\sigma_1^+ = \frac{\sigma_1^2 \sigma_q^2}{\sigma_1^2 + \sigma_q^2} = \left(1 - \frac{\sigma_1^2}{\sigma_1^2 + \sigma_q^2}\right)\sigma_1^2 = (1-K)\,\sigma_1^2$$

Update (Slide 24)

$$z_t = C_t x_t + \delta_t$$

μ_t is the mean and Σ_t is the covariance of the distribution of x_t

2D Example of Kalman Filter

state

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}$$

measurement

$$\boldsymbol{z} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Constant position Motion Model

$$\boldsymbol{x}_t = A\boldsymbol{x}_{t-1} + B\boldsymbol{u}_t + \epsilon_t$$

system noise

$$\epsilon_t \sim \mathcal{N}(\mathbf{0}, R)$$

Constant position

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

$$B\boldsymbol{u} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$R = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_r^2 \end{bmatrix}$$

2D Example of Kalman Filter

state

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}$$

measurement

$$\boldsymbol{z} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Measurement Model

$$\boldsymbol{z}_t = C_t \boldsymbol{x}_t + \delta_t$$

$$C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \delta_t \sim \mathcal{N}(\mathbf{0}, Q) \qquad Q = \begin{bmatrix} \sigma_q^2 & 0 \\ 0 & \sigma_q^2 \end{bmatrix}$$

zero-mean measurement noise

2D Example of Kalman Filter

- To track the robot (with constant position):

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{motion model}$$

$$C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{observation model}$$

General Case

$$\begin{aligned}
\bar{\mu}_t &= A_t \mu_{t-1} + B u_t \\
\bar{\Sigma}_t &= A_t \Sigma_{t-1} A_t^\top + R \\
K_t &= \bar{\Sigma}_t C_t^\top (C_t \bar{\Sigma}_t C_t^\top + Q_t)^{-1} \\
\mu_t &= \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t) \\
\Sigma_t &= (I - K_t C_t) \bar{\Sigma}_t
\end{aligned}$$

Constant position Model

$$\begin{aligned}
\bar{\boldsymbol{x}}_t &= \boldsymbol{x}_{t-1} \\
\bar{\Sigma}_t &= \Sigma_{t-1} + R \\
K_t &= \bar{\Sigma}_t (\bar{\Sigma}_t + Q)^{-1} \\
\boldsymbol{x}_t &= \bar{\boldsymbol{x}}_t + K_t (z_t - \bar{\boldsymbol{x}}_t) \\
\Sigma_t &= (I - K_t) \bar{\Sigma}_t
\end{aligned}$$

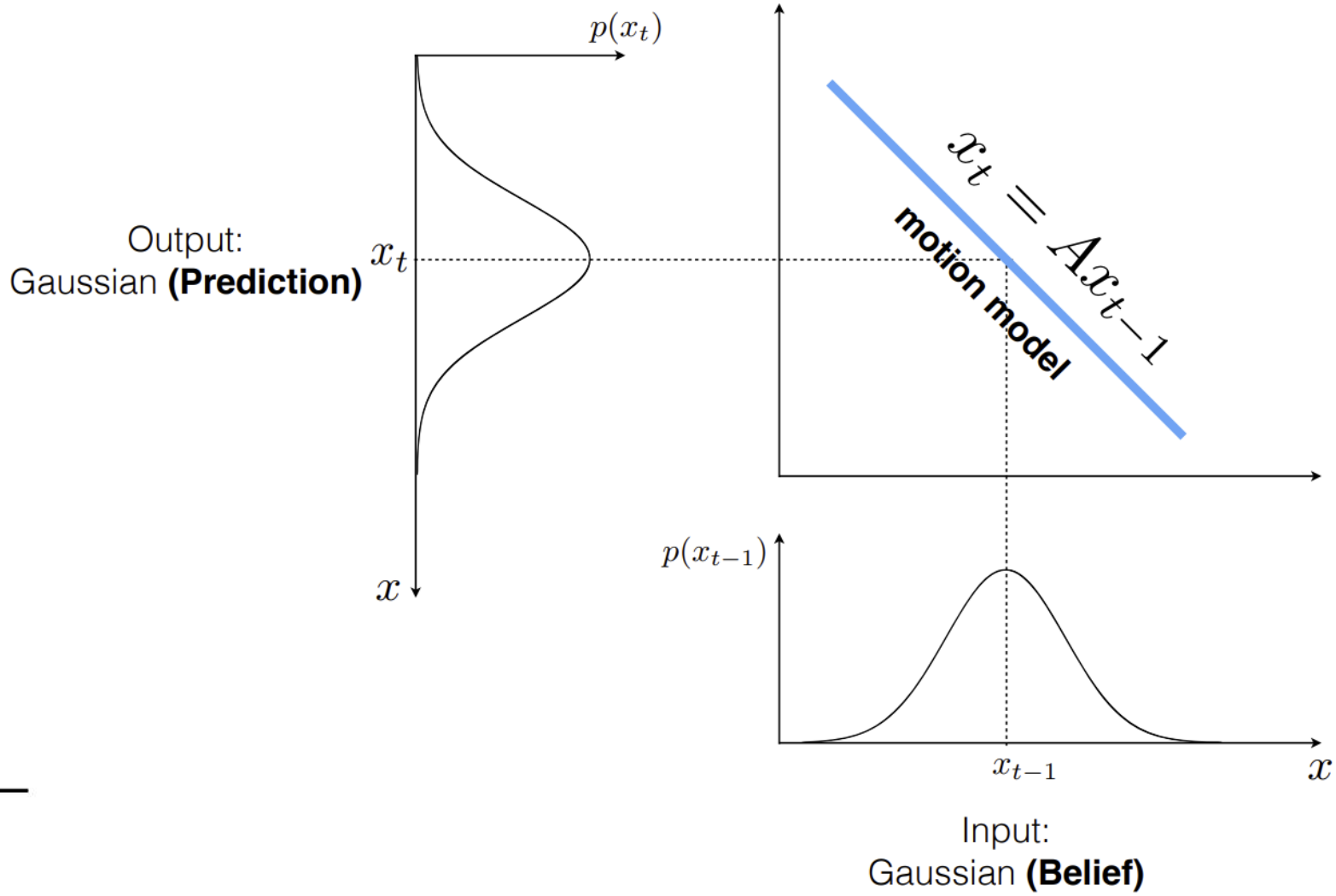
Extended Kalman Filter (EKF)

Chapter 3.3, Sebastian Thrun, Wolfram Burgard and Dieter Fox.
“Probabilistic Robotics.” MIT Press. 2005.

Motivation of EKF

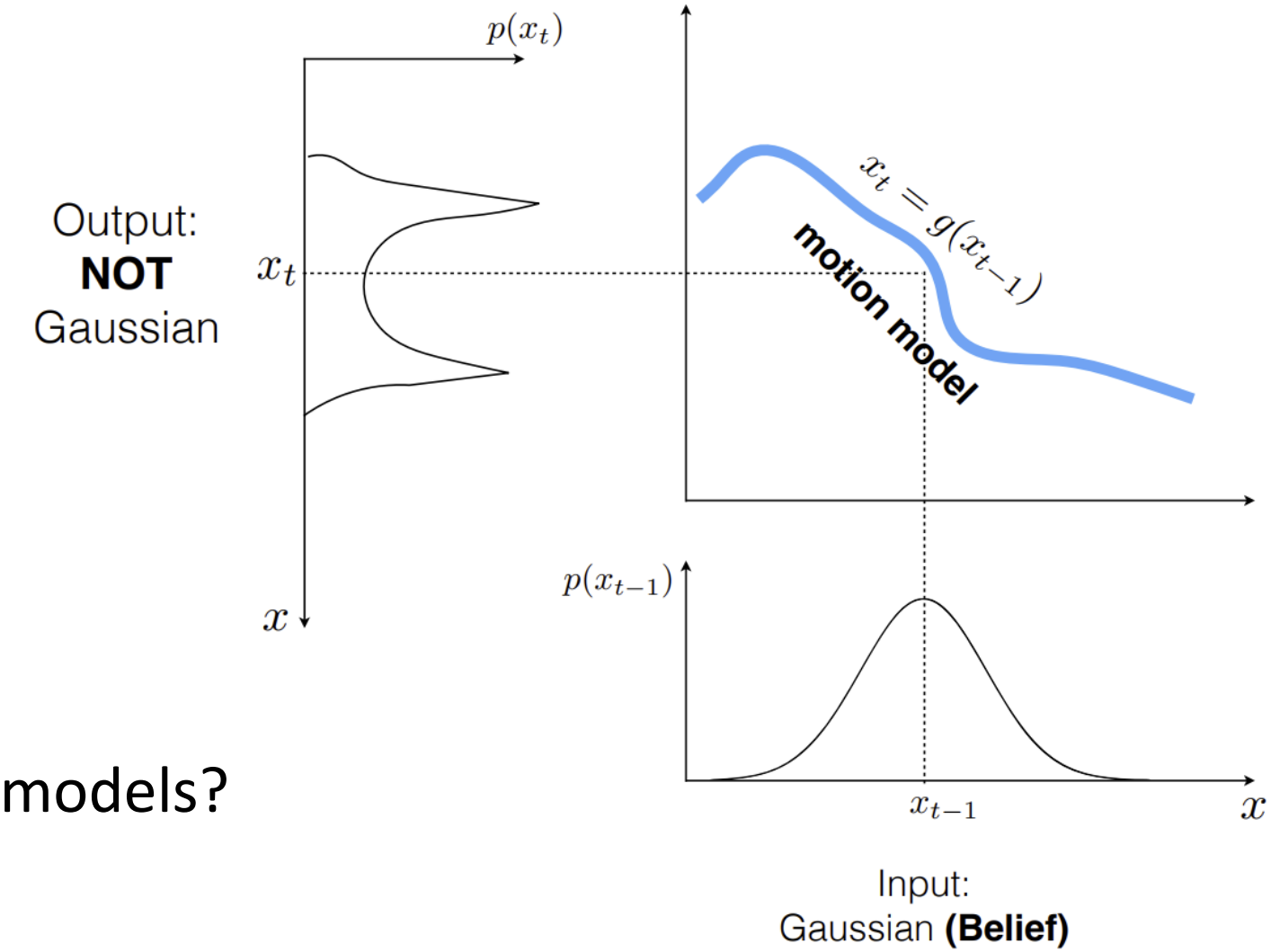
- Motion model of the basic Kalman filter must be linear.

$$x_t = Ax_{t-1} + Bu_t + \epsilon_t$$



Motivation

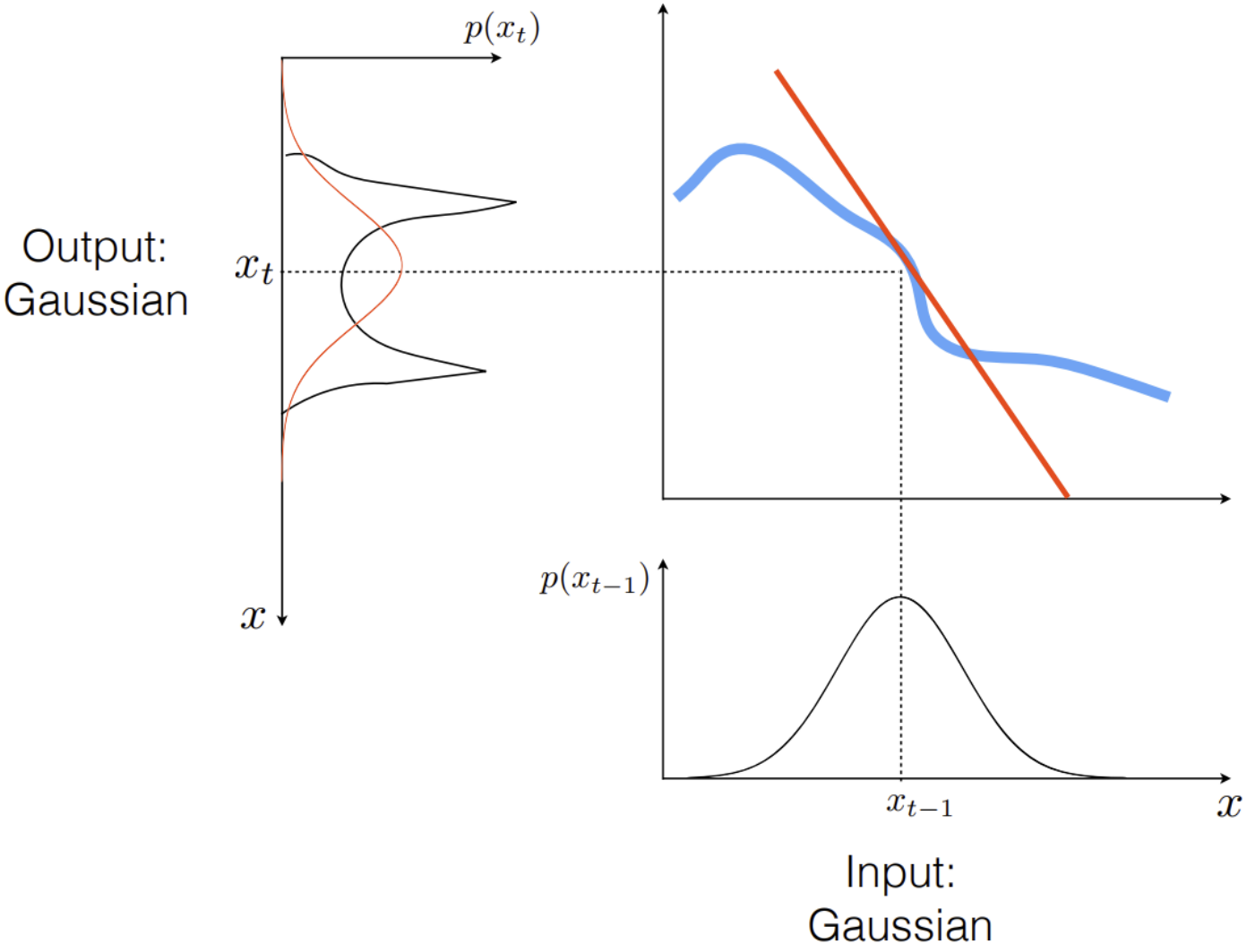
- But motion is not always linear; actually, in most cases, it is nonlinear.
- Can we use the Kalman Filter with nonlinear motion models?



- How to deal with non-linear models?

Extended Kalman Filter

- Uses local linearization (linear approximations) of model to keep the effectiveness of the KF framework.
- EKF does not assume linear models.
- It assumes Gaussian noise.



Extended Kalman Filter

Kalman Filter

linear motion model

$$x_t = Ax_{t-1} + Bu_t + \epsilon_t$$

linear sensor model

$$z_t = C_t x_t + \delta_t$$

Extended Kalman Filter

non-linear motion model

$$x_t = g(x_{t-1}, u_t) + \epsilon_t$$

non-linear sensor model

$$z_t = H(x_t) + \delta_t$$

Motion Model Linearization

$$g(x_{t-1}, u_t) \approx g(\mu_{t-1}, u_t) + \frac{\partial g(\mu_{t-1}, u_t)}{\partial x_{t-1}} (x_{t-1} - \mu_{t-1})$$

Taylor series expansion

$$\approx g(\mu_{t-1}, u_t) + \quad G_t \quad (x_{t-1} - \mu_{t-1})$$

Jacobian Matrix

'the rate of change in x'
'slope of the function'

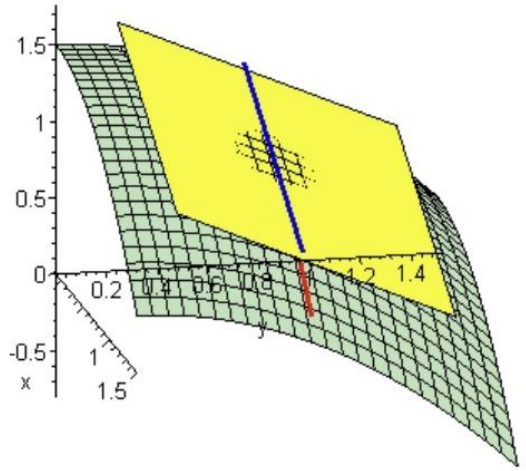
$$p(x_t \mid u_t, x_{t-1}) \approx \det(2\pi R_t)^{-\frac{1}{2}}$$

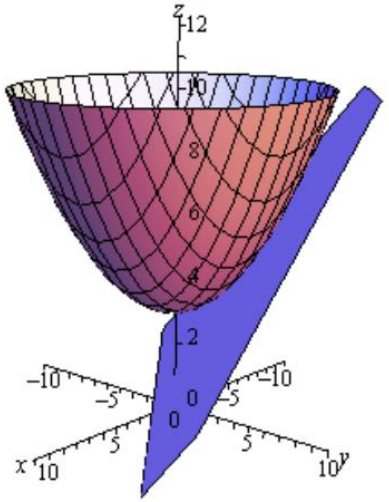
$$\exp\Big(-\frac{1}{2} (x_t - g(u_t, \mu_{t-1}) - G_t (x_{t-1} - \mu_{t-1}))^T$$

$$R_t^{-1} (x_t - \underbrace{g(u_t, \mu_{t-1}) - G_t (x_{t-1} - \mu_{t-1})}_{\text{linearized model}}) \Big)$$

Jacobian Visualization

- Jacobian is the orientation of the tangent plane to a multi-variant function at a given point, i.e., slope of the function.
- It generalizes the gradient (or tangent line) of a single-variant function.





Courtesy: K. Arras

Sensor Model Linearization

$$h(x_t) \approx h(\bar{\mu}_t) + \frac{\partial h(\bar{\mu}_t)}{\partial x_t}(x_{t-1} - \bar{\mu}_t)$$

Taylor series expansion

$$\approx h(\bar{\mu}_t) + \quad H_t \quad (x_t - \bar{\mu}_t)$$

Jacobian Matrix

'the rate of change in x'
'slope of the function'

$$p(z_t \mid x_t) = \det(2\pi Q_t)^{-\frac{1}{2}}$$

$$\exp\Big(-\frac{1}{2}\,(z_t - h(\bar{\mu}_t) - H_t\,(x_t - \bar{\mu}_t))^T$$

$$Q_t^{-1}\,(z_t - \underbrace{h(\bar{\mu}_t) - H_t\,(x_t - \bar{\mu}_t)}_{\text{linearized model}})\Big)$$

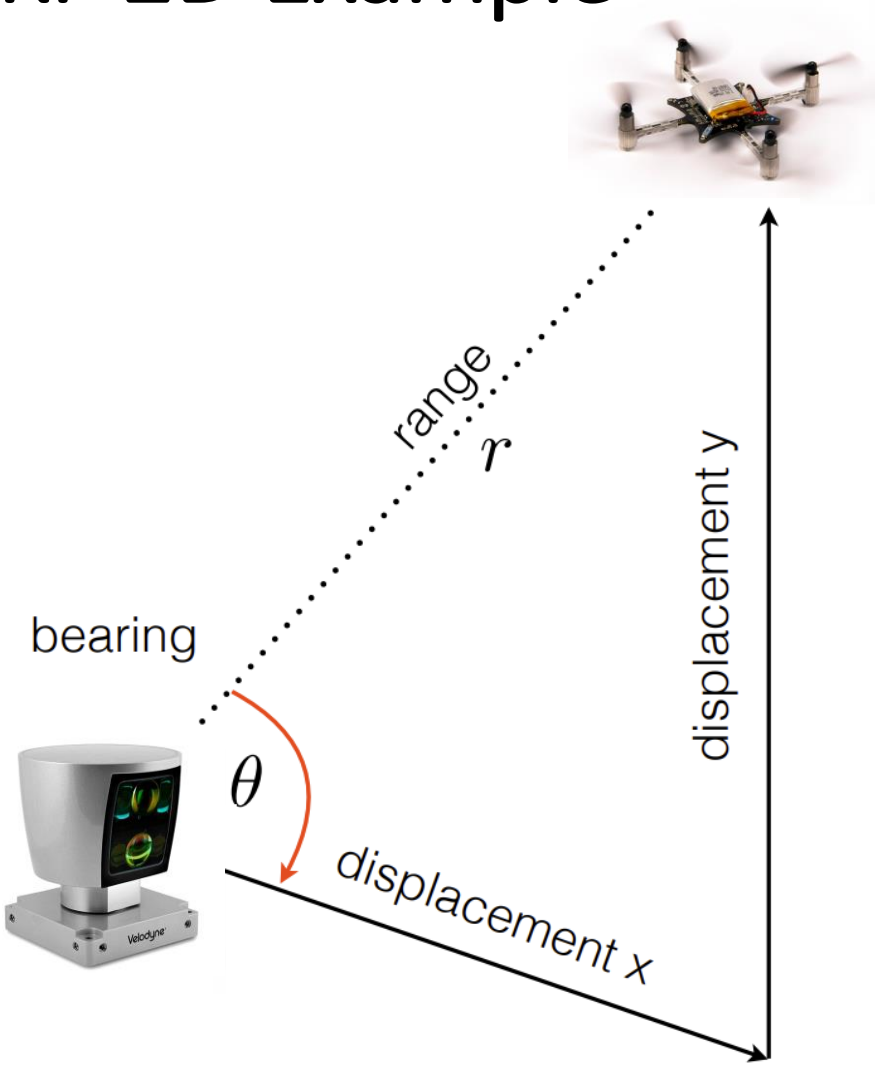
EKF Algorithm

Kalman Filter	Extended KF
$\bar{\mu}_t = A_t \mu_{t-1} + B u_t$	$\bar{\mu}_t = g(\mu_{t-1}, u_t)$
$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^\top + R$	$\bar{\Sigma}_t = G_t \bar{\Sigma}_{t-1} G_t^\top + R$
$K_t = \bar{\Sigma}_t C_t^\top (C_t \bar{\Sigma}_t C_t^\top + Q_t)^{-1}$	$K_t = \bar{\Sigma}_t H_t^\top \left(H_t \bar{\Sigma}_t H^\top + Q\right)^{-1}$
$\mu_t = \bar{\mu}_t + K_t(z_t - C_t \bar{\mu}_t)$	$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
$\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$	$\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$

https://www.youtube.com/watch?v=iTe6-lLp5iM

https://www.youtube.com/watch?v=NVf9uUJV7QY

EKF 2D Example



state: position-velocity

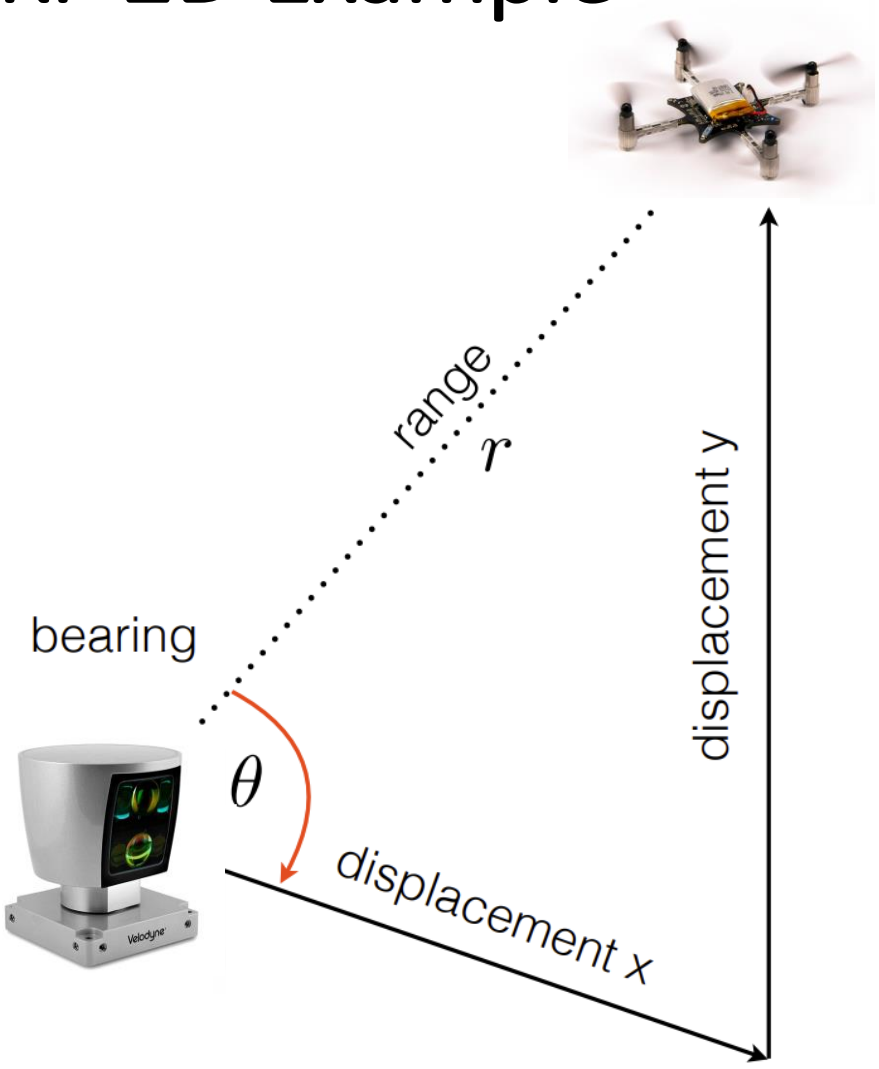
$$\boldsymbol{x} = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix} \begin{matrix} \text{position} \\ \text{velocity} \\ \text{position} \\ \text{velocity} \end{matrix}$$

constant velocity motion model

$$A = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

with additive Gaussian noise

EKF 2D Example



measurement: range-bearing

$$\boldsymbol{z} = \begin{bmatrix} r \\ \theta \end{bmatrix}$$

$$= \begin{bmatrix} \sqrt{x^2 + y^2} \\ \tan^{-1}(y/x) \end{bmatrix}$$

measurement model

Is the measurement model linear?

$$\boldsymbol{z} = h(r, \theta)$$

with additive Gaussian noise

non-linear!

What should we do?

EKF 2D Example

- Linearize the sensor/measurement model

$$\boldsymbol{z} = \begin{bmatrix} r \\ \theta \end{bmatrix} = \begin{bmatrix} \sqrt{x^2 + y^2} \\ \tan^{-1}(y/x) \end{bmatrix}$$

$$H = \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = ?$$

What is the Jacobian?

$$\bar{\mu}_t = A_t \mu_{t-1} + B u_t$$
$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^\top + R$$
$$K_t = \bar{\Sigma}_t H_t^\top \left(H_t \bar{\Sigma}_t H^\top + Q \right)^{-1}$$
$$\mu_t = \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t))$$
$$\Sigma_t = (I - K_t H_t) \bar{\Sigma}_t$$

- Linearize the observation/measurement model

$$H = \begin{bmatrix} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial \dot{x}} & \frac{\partial r}{\partial y} & \frac{\partial r}{\partial \dot{y}} \\ \\ \frac{\partial \theta}{\partial x} & \frac{\partial \theta}{\partial \dot{x}} & \frac{\partial \theta}{\partial y} & \frac{\partial \theta}{\partial \dot{y}} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ \\ -sin(\theta)/r & 0 & \cos(\theta)/r & 0 \end{bmatrix}$$

Problems with EKFs

- Taylor series expansion = poor approximation of non-linear functions, success of linearization depends on
  - Limited uncertainty and
  - Limited amount of local non-linearity
- Computing partial derivatives is a pain
- Cannot handle multi-modal (multi-hypothesis) distributions
- What's next?
  - Unscented Kalman Filter (how to better generalize to non-linear models)
  - Non-Gaussian noise Kalman Filter (how to generalize the Kalman Filter when noise distribution is Non-Gaussian)
  - Stability and Divergence (how to design a stable KF that does not diverge)