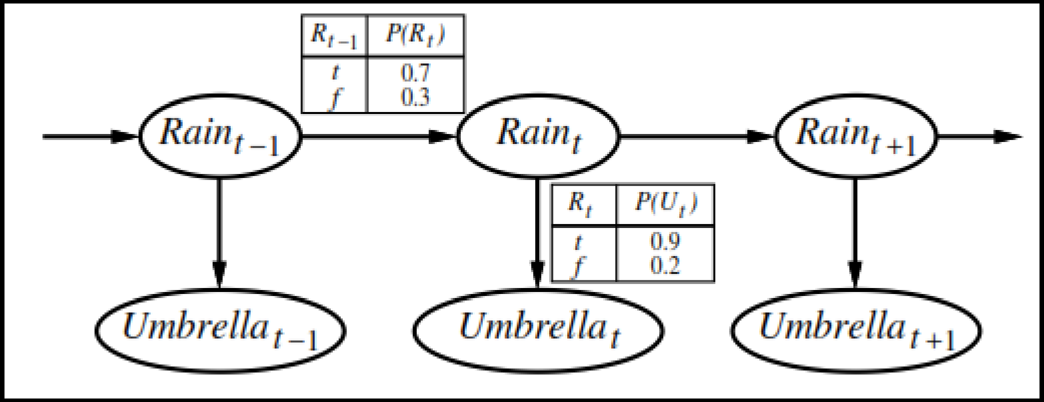
Bayes Filter and Kalman Filters

Using materials from probabilistic-robotics.org, AIMA book

Last time…

- We saw how to incorporate time into probabilistic reasoning (in the form of a Bayes net)
- We made the Markov Assumption to keep the inference manageable



- But we only considered cases where the state was discrete
- Today we will look at algorithms that handle continuous distributions

Outline

- Hidden Markov Models (HMMs) (review)
- Bayes filter
- Kalman Filter
- Extended Kalman Filter (EKF)

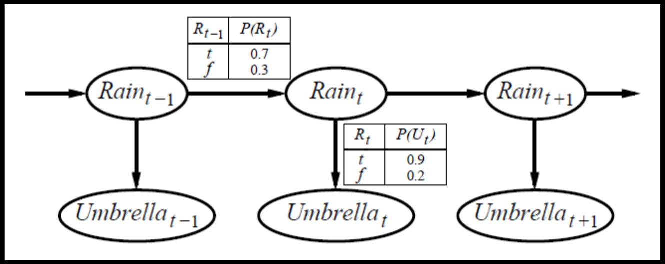
Inference Tasks

Filtering: $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$
    belief state—input to the decision process of a rational agent

Prediction: $\mathbf{P}(\mathbf{X}_{t+k}|\mathbf{e}_{1:t})$ for $k > 0$
    evaluation of possible action sequences;
    like filtering without the evidence

Smoothing: $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ for $0 \leq k < t$
    better estimate of past states, essential for learning

Most likely explanation: $\arg\max_{\mathbf{x}_{1:t}} P(\mathbf{x}_{1:t}|\mathbf{e}_{1:t})$
    speech recognition, decoding with a noisy channel



Hidden Markov Models (HMMs) (review)

- An HMM is a temporal probabilistic model in which the state is described by a **single discreet random variable**.

$\mathbf{X}_t$ is a single, discrete variable (usually $\mathbf{E}_t$ is too)
Domain of X_t is $\{1, \ldots, S\}$

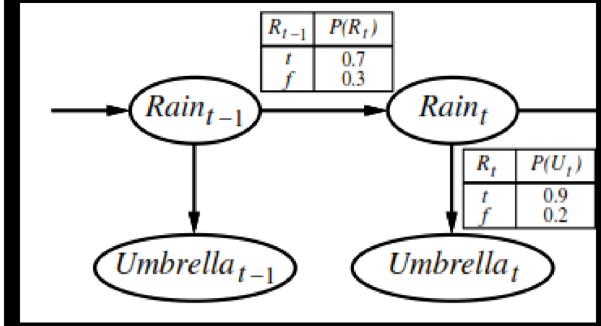
Transition matrix $\mathbf{T}_{ij} = P(X_t = j | X_{t-1} = i)$, e.g., $\begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$

Sensor matrix $\mathbf{O}_t$ for each time step, diagonal elements $P(e_t | X_t = i)$
e.g., with $U_1 = true$, $\mathbf{O}_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix}$

Forward and backward messages as column vectors:

$$\begin{aligned} \mathbf{f}_{1:t+1} &= \alpha \mathbf{O}_{t+1} \mathbf{T}^\top \mathbf{f}_{1:t} \\ \mathbf{b}_{k+1:t} &= \mathbf{T}\mathbf{O}_{k+1}\mathbf{b}_{k+2:t} \end{aligned}$$

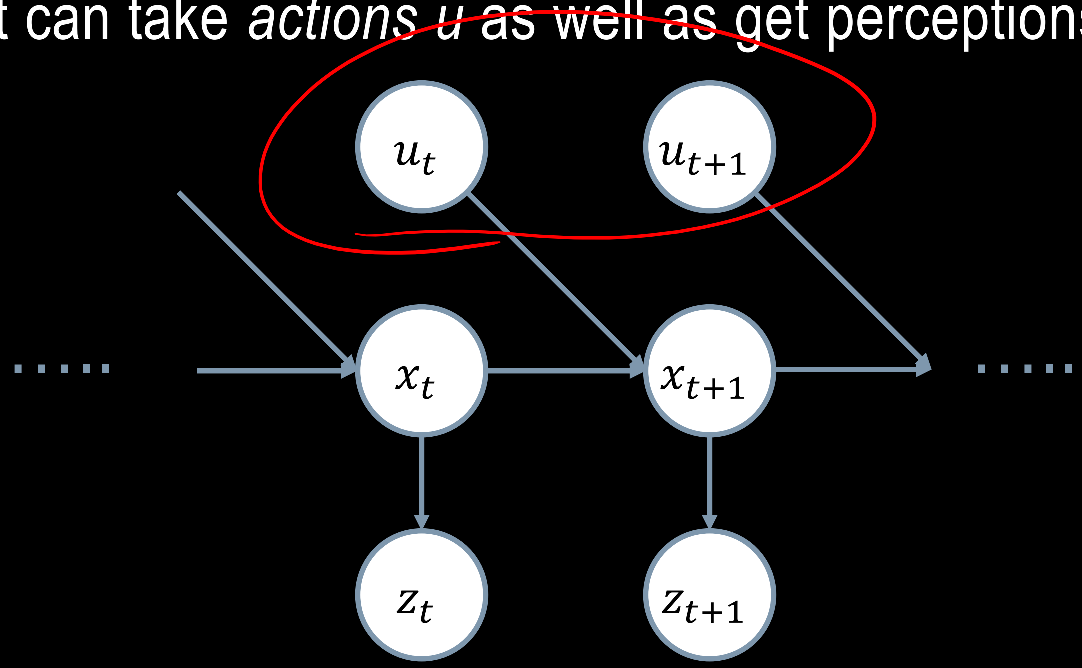
Forward-backward algorithm needs time $O(S^2 t)$ and space $O(St)$



Bayes Filter

HMM vs. Bayes Filter

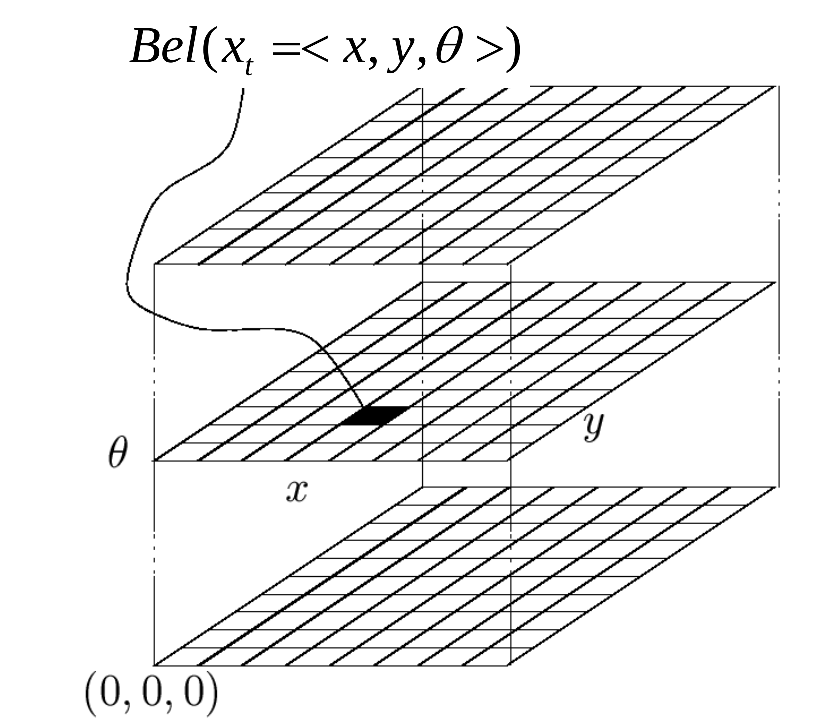
- In HMMs, the system is passive: just a stream of perception data
- A robot can take *actions u* as well as get perceptions z



- Robot state and perception data are usually multi-dimensional
- When the state is discrete, we can use a Bayes filter

Bayes Filter Belief

- The state must be discrete, so we usually use a grid to represent it
- Each grid cell contains the *belief* $Bel(x_t)$ (the probability that the true state of the system is x_t)
- E.g. for a mobile robot:



Discrete Bayes Filter Algorithm

- Given a piece of sensor or action data, update Bel(x) using this algorithm:

Very inefficient for large state spaces! There are heuristics which only update the belief locally

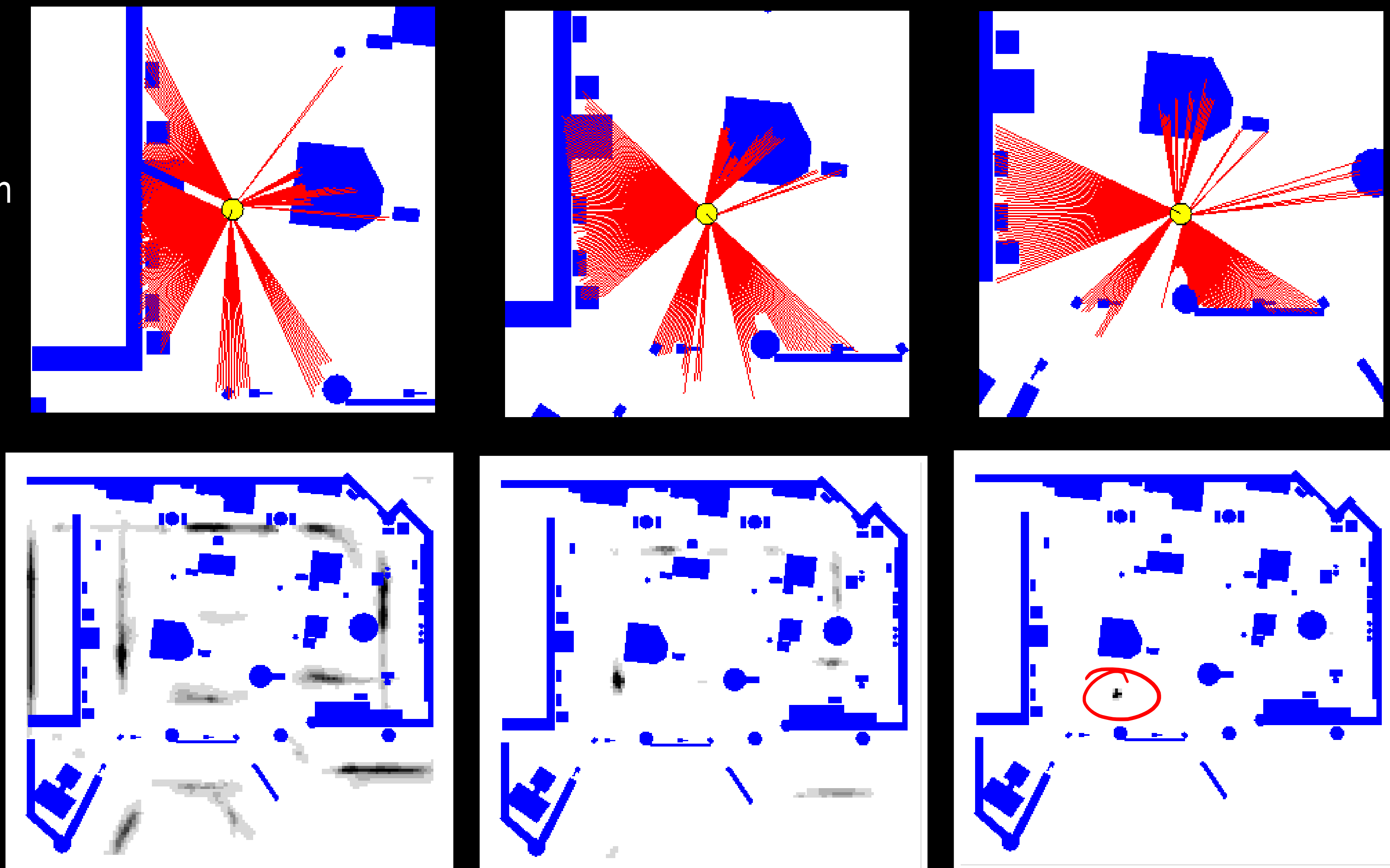
Algorithm **Discrete_Bayes_filter**(*Bel(x),d*):
η=0
If *d* is a perceptual data item *z* then
  For all *x* do
    $$Bel'(x) = P(z \mid x) Bel(x)$$
    $$\eta = \eta + Bel'(x)$$
  For all *x* do
    $$Bel'(x) = \eta^{-1} Bel'(x)$$
Else if *d* is an action data item *u* then
  For all *x* do
    $$Bel'(x) = \sum_{x'} P(x \mid u, x') \, Bel(x')$$
Return *Bel′(x)*

u x'

Grid-based Localization Example

Perception data

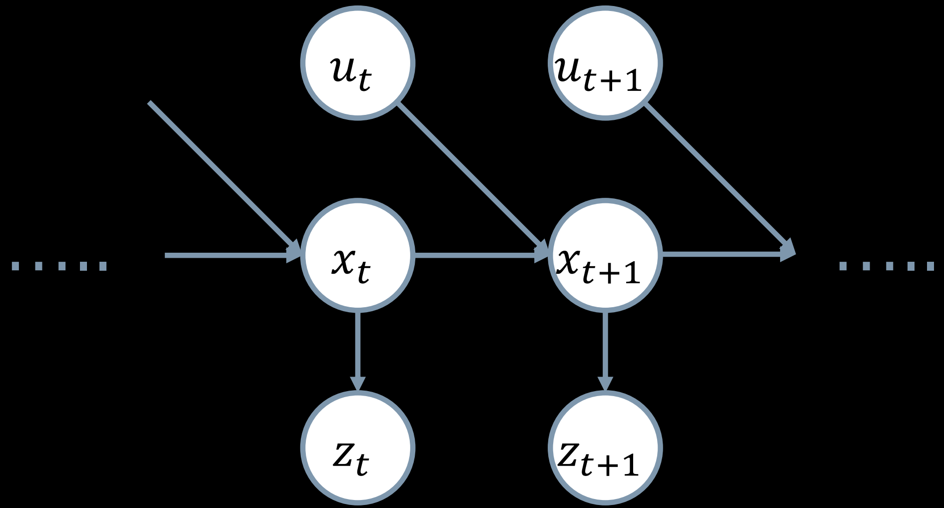
Belief



Kalman Filter

Kalman Filters

- The real world is not discrete! Need to consider **continuous** variables



- Kalman filters are used to track state of robots, chemical plants, planets, etc.
- Key Idea: Arbitrary continuous models are intractable, so represent everything with *Gaussians*
  - Gaussian prior, linear Gaussian transition model and sensor model

Gaussians

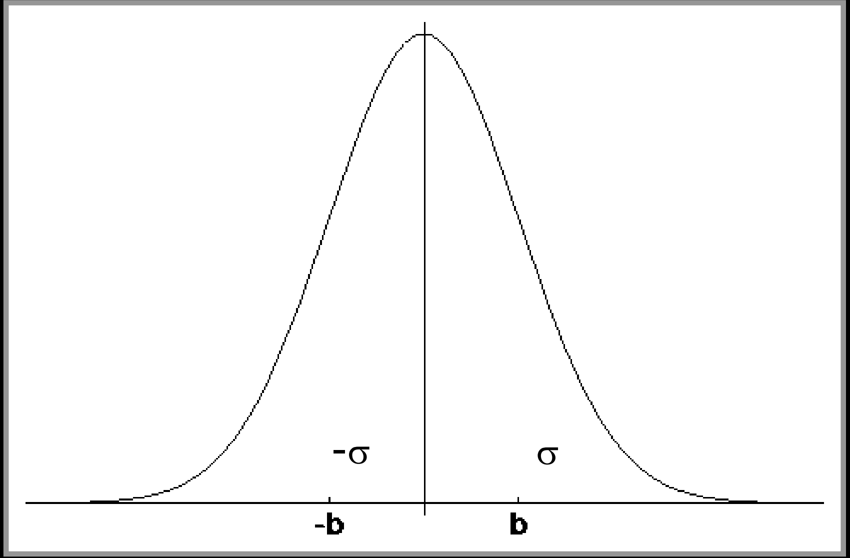
- Univariate

$$p(x) \sim N(\mu, \sigma^2):$$

variance

mean

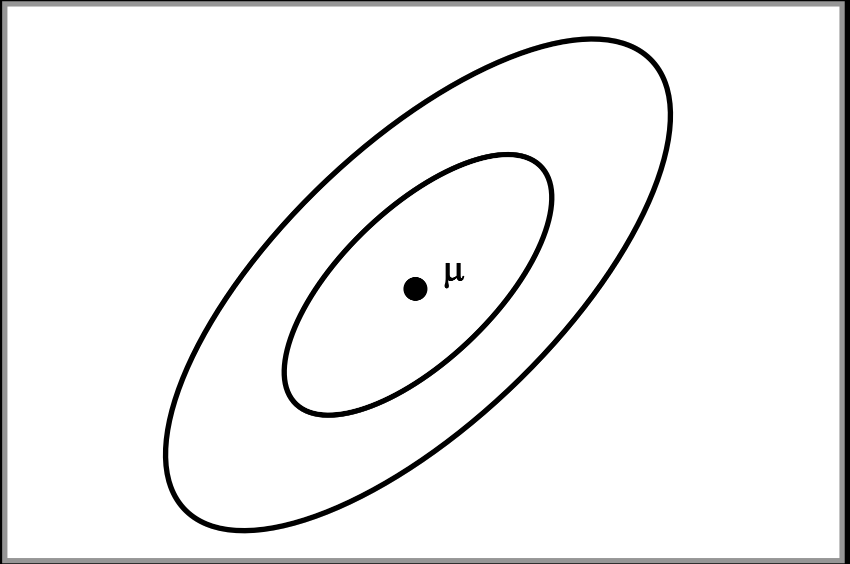
$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}$$



- Multivariate

$$p(\mathbf{x}) \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma}):$$

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^t \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}$$



Multivariate Gaussians

$$\left.\begin{array}{r} X \sim N(\mu, \Sigma) \\ Y = AX + \underset{\text{matrix}}{A}\!\!\!\!\! X + B \end{array}\right\} \Rightarrow \quad Y \sim N(A\mu + B, A\Sigma A^T)$$

$B \to$ vector

$A \to$ matrix

$$\left.\begin{array}{l} X_1 \sim N(\mu_1, \Sigma_1) \\ X_2 \sim N(\mu_2, \Sigma_2) \end{array}\right\} \Rightarrow p(X_1) \cdot p(X_2) \sim N\left(\frac{\Sigma_2}{\Sigma_1 + \Sigma_2}\mu_1 + \frac{\Sigma_1}{\Sigma_1 + \Sigma_2}\mu_2, \quad \frac{1}{\Sigma_1^{-1} + \Sigma_2^{-1}} \right)$$

We stay in the "Gaussian world" as long as we start with Gaussians and perform only linear transformations.

Kalman Filter

Estimates the state x of a discrete-time controlled process that is governed by the linear stochastic difference equation

$$\boxed{\underbrace{x_t}_{\text{Current state}} = A_t \underbrace{x_{t-1}}_{\text{Previous state}} + B_t \underbrace{u_t}_{\text{Action}} + \underbrace{\varepsilon_t}_{\text{Actuation noise}}}$$

with a sensor measurement

$$\boxed{\underbrace{z_t}_{\text{Current measurement}} = C_t \underbrace{x_t}_{\text{Current state}} + \underbrace{\delta_t}_{\text{Sensor noise}}}$$

Example for GPS sensor:

$$z_t = \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} pos_x \\ vel_x \end{bmatrix}}_{x} + \delta_t$$

Components of a Kalman Filter

A_t Matrix (n x n) that describes how the state changes from $t\text{-}1$ to t without controls or noise.

B_t Matrix (n x $|u|$) that describes how the control u_t changes the state from $t\text{-}1$ to t.

C_t Matrix (k x n) that describes how to map the state x_t to an observation z_t.

ε_t

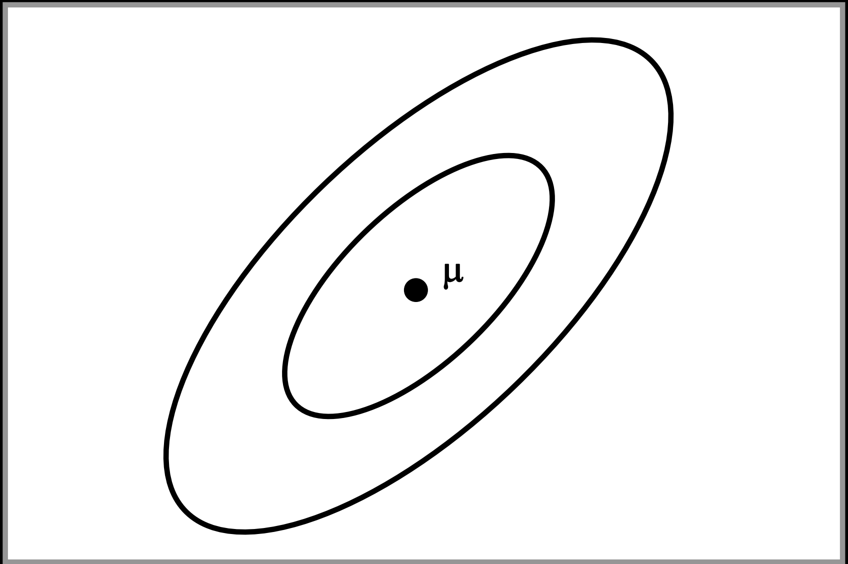
δ_t Random variables representing the process and measurement noise that are assumed to be independent and normally distributed with covariance R_t and Q_t respectively.

Kalman Filter State Estimate

- The Kalman filter computes a Gaussian probability distribution of the state given the prior distribution and a sensor measurement

$$p(\mathbf{x}) \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma}):$$

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^t \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})}$$



- A Gaussian is represented by μ (a vector) and Σ (a matrix), so all we need to do is track these two variables

Kalman Filter Algorithm

Algorithm **Kalman_filter**(μ_{t-1}, Σ_{t-1}, u_t, z_t):

Use dynamics (A_t and B_t) to predict what will happen

1. Prediction:

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

$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$$

process noise

2. Correction:

Use sensor measurement z_t to correct prediction

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

sensor noise

This is called the Kalman gain

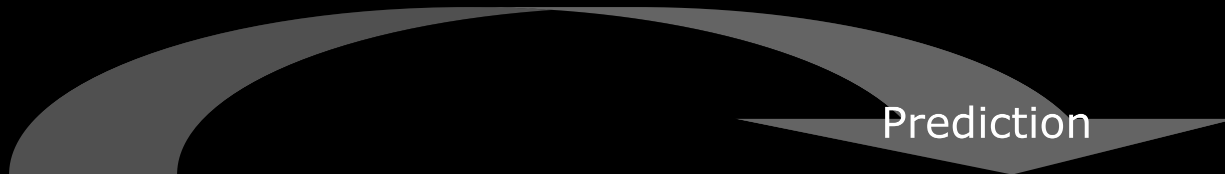
$$\mu_t = \bar{\mu}_t + K_t (z_t - \underbrace{C_t \bar{\mu}_t}_{\text{expected sensor measurement}})$$

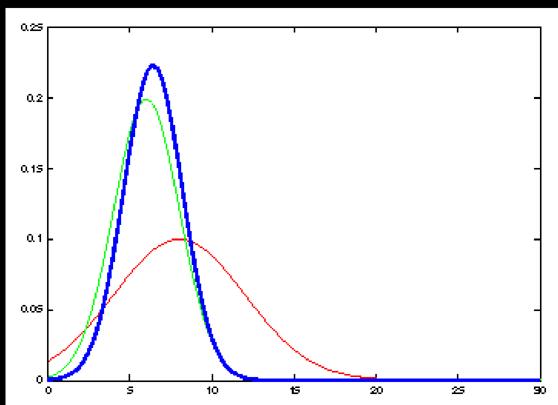
$$\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$$

Return μ_t, Σ_t

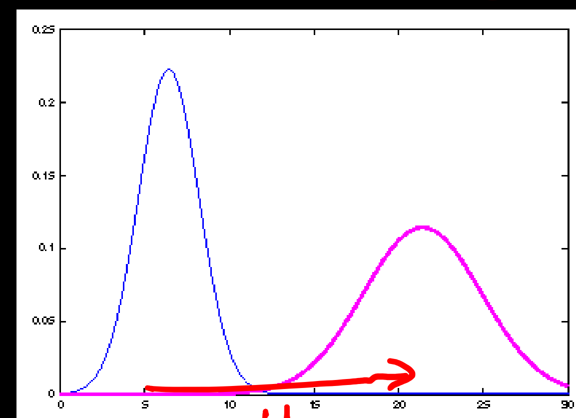
See here for derivation. Note that they use different notation!

The Prediction-Correction-Cycle



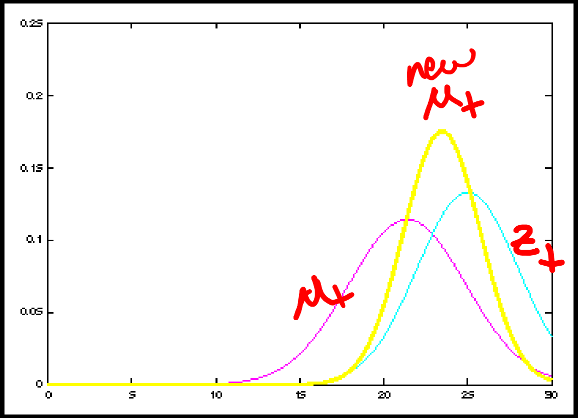


$$\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$$
$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$$

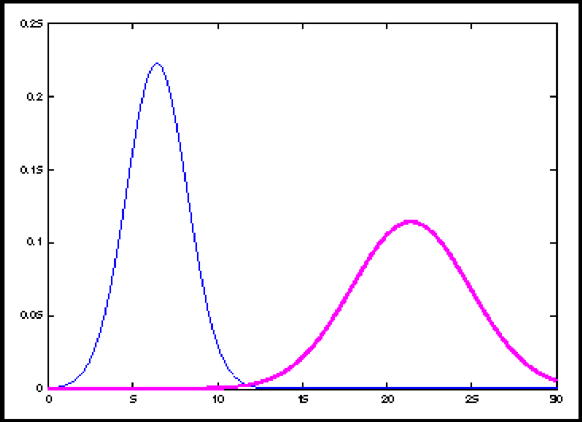


u_t

The Prediction-Correction-Cycle



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



Correction

Prediction-Correction Cycle

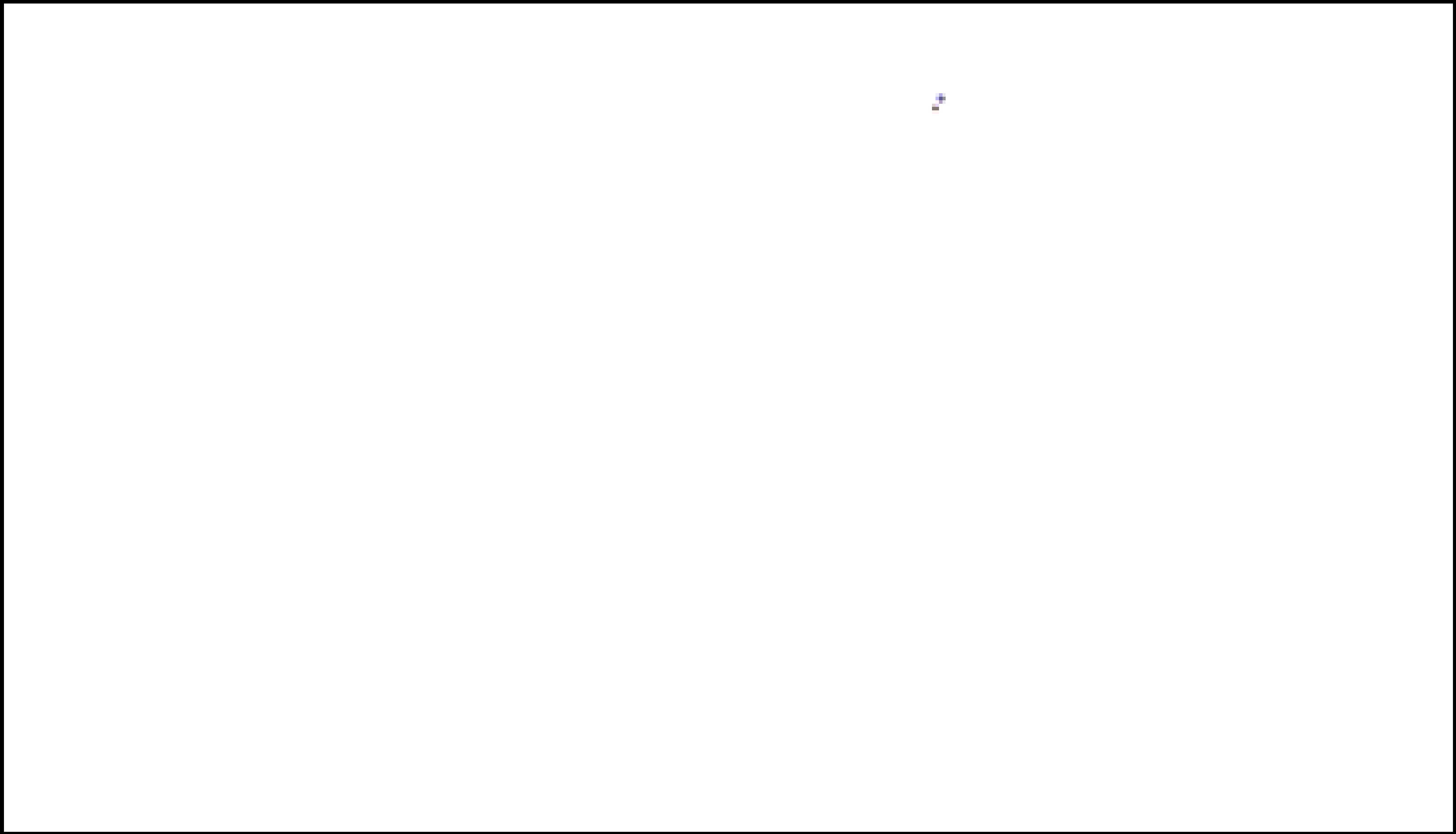
Prediction

$$\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$$
$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$$

Correction

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

Kalman Filter Localization Example



Blue line: actual trajectory
Blue dots: GPS measurements
Red line: estimated states

by Keyan Ghazi-Zahedi
https://www.youtube.com/watch?v=ZYexI6_zUMkby

Kalman Filter Highlights

- Highly efficient: Polynomial in measurement dimensionality *k* and state dimensionality *n*:

$$O(k^{2.376} + n^2)$$

- Optimal for linear Gaussian systems!
  (known R and Q)

- But, most robotics systems are nonlinear! ☹

BREAK

Extended Kalman Filter (EKF)

Nonlinear dynamic systems

- Most robotics problems involve nonlinear dynamics and sensors

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

$$\boxed{z_t = h(x_t)}$$

The EKF trick

- Can't deal with non-linear functions directly
- But, if the change is small, we can use a *local linear approximation*
- How? Compute the Jacobians of *g* and *h*!

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

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

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

$$z_t = h(x_t) \approx h(\bar{\mu}_t) + \frac{\partial h(\bar{\mu}_t)}{\partial x_t}(x_t - \bar{\mu}_t)$$

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

$$H_t = \frac{\partial h(\bar{\mu}_t)}{\partial x_t}$$

Algorithm **Extended_Kalman_filter**(μ_{t-1}, Σ_{t-1}, u_t, z_t):

1. Prediction:

$$\bar{\mu}_t = g(u_t, \mu_{t-1})$$

Jacobian for g

$$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$$

2. Correction:

Jacobian for h

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

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

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

Return μ_t, Σ_t

Kalman Filter

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

$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$$

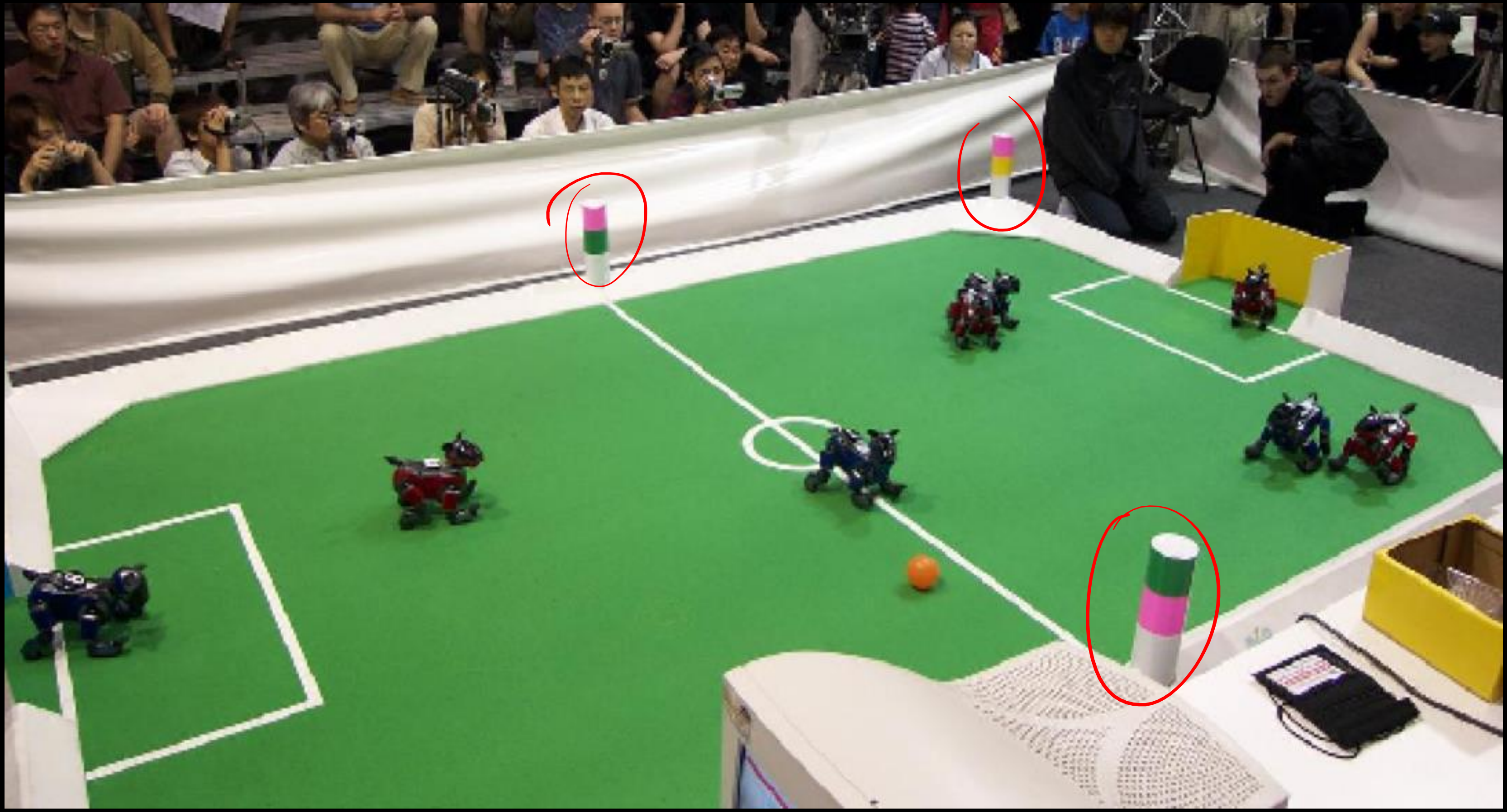
$$K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$$

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

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

$$H_t = \frac{\partial h(\bar{\mu}_t)}{\partial x_t} \qquad G_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}}$$

Example: Beacon-based Robot Localization



Example Motion Model

- State is $x_t = (x_t, y_t, \theta_t)$
- Command is rotation, translation, rotation

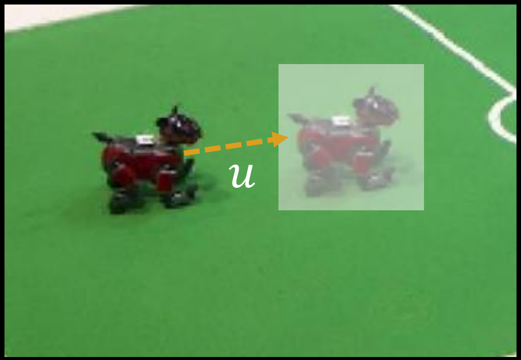
$$u_t = (\delta_{rot_1}, \delta_{trans}, \delta_{rot_2})$$

- Actual motion is $(\tilde{\delta}_{rot_1}, \tilde{\delta}_{trans}, \tilde{\delta}_{rot_2})$, a noisy version of the command

- Motion model g is:

$$x_{t+1} = x_t + \tilde{\delta}_{trans}\cos(\theta_t + \tilde{\delta}_{rot_1})$$
$$y_{t+1} = y_t + \tilde{\delta}_{trans}\sin(\theta_t + \tilde{\delta}_{rot_1})$$
$$\theta_{t+1} = \theta_t + \tilde{\delta}_{rot_1} + \tilde{\delta}_{rot_2}$$

(modulo 2π)



Not linear!

Example sensor model

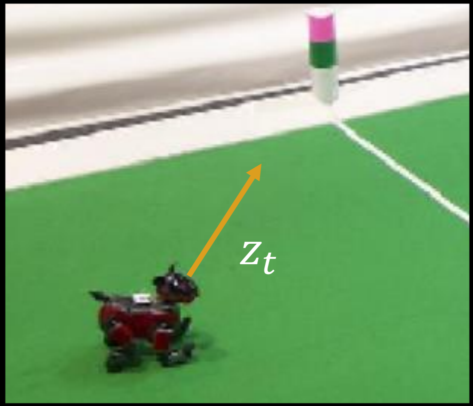
- The map is known
  - Beacons are at known positions
- Sensor reports noisy bearing $\tilde{\theta}$ and exact landmark ID L

discrete

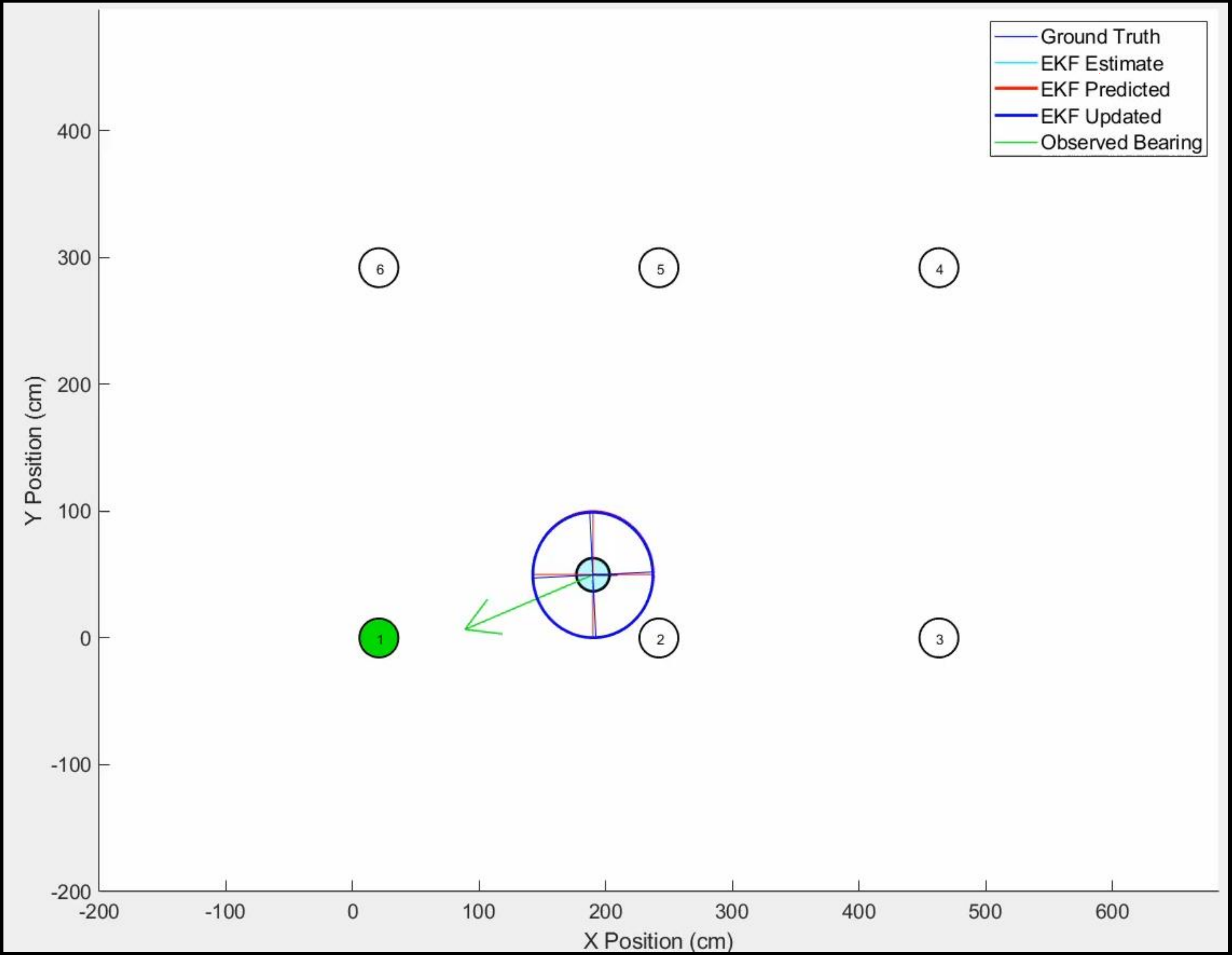
  - Only one beacon is observed at one time

$$z_t = \begin{pmatrix} \tilde{\theta} \\ L \end{pmatrix} = \begin{pmatrix} \mathrm{atan2}(y_{rob} - y_L, x_{rob} - x_L) \\ L \end{pmatrix}$$

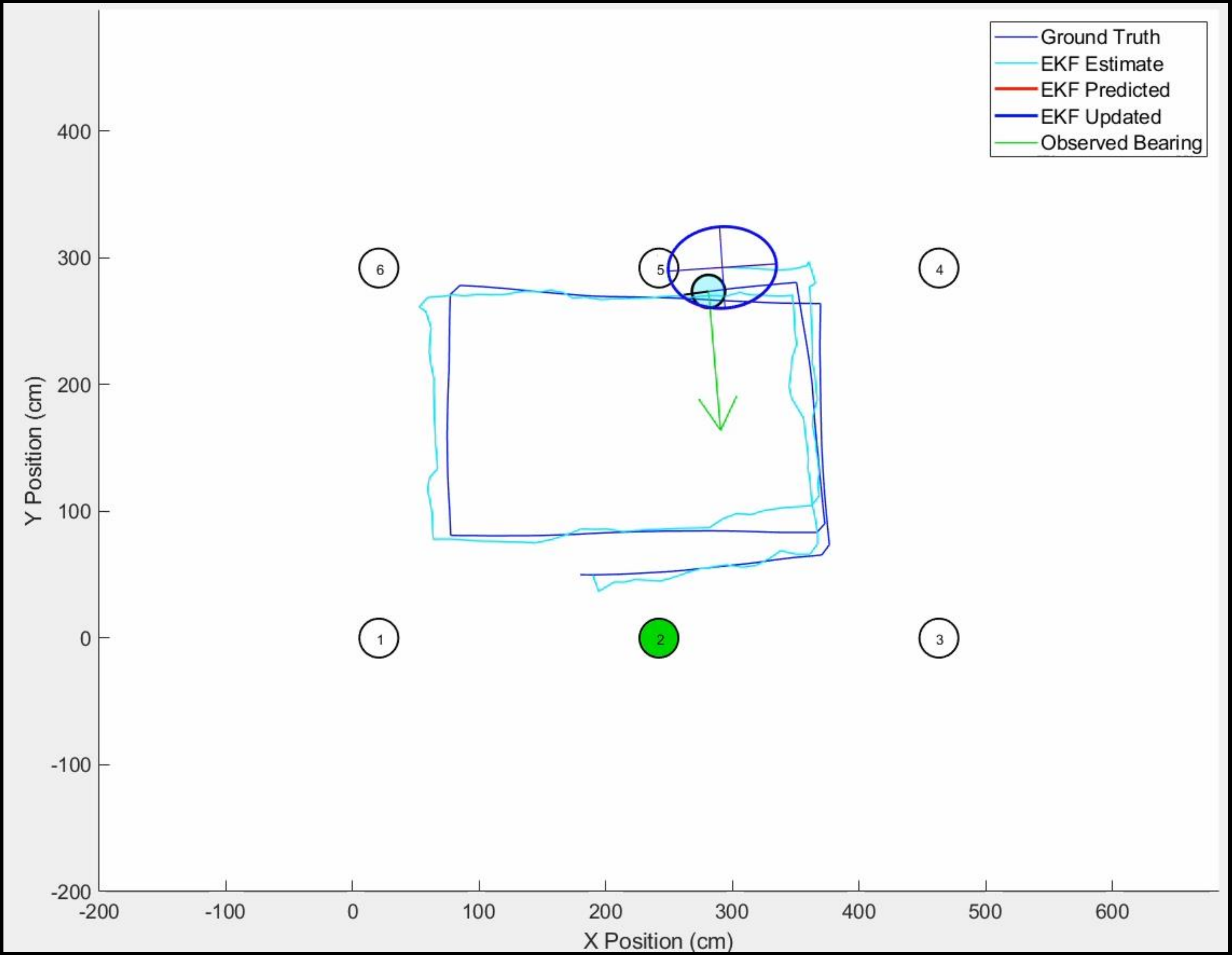
Not linear!



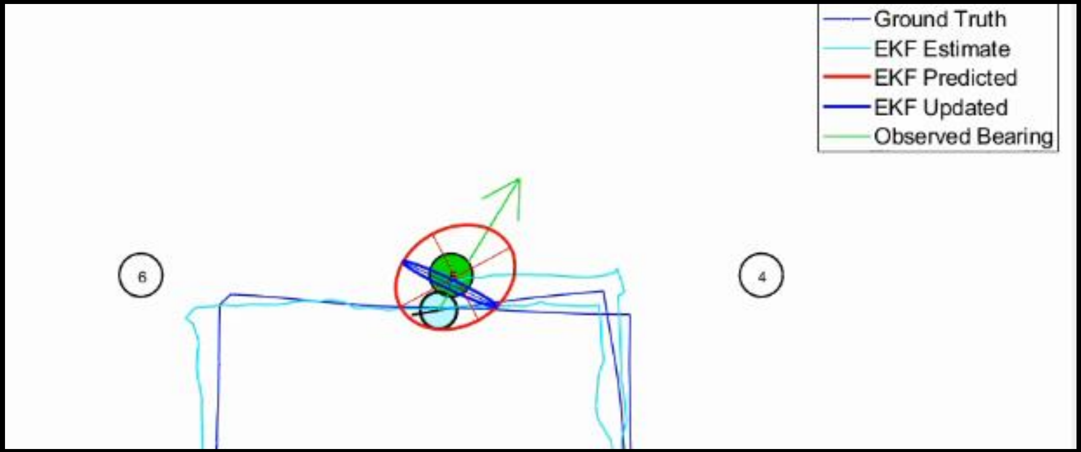
EKF Localization Example



What happened here?



EKF Inaccuracy



- Robot is close to beacon, so small changes in the position cause large changes in the measurement
- Observation Jacobian H_t has large terms
  - Causes filter to become confident in position
- But, measurement error is large: -1.86 radians!
  - Causes filter to be wrong about position due to non-linearities

EKF Highlights

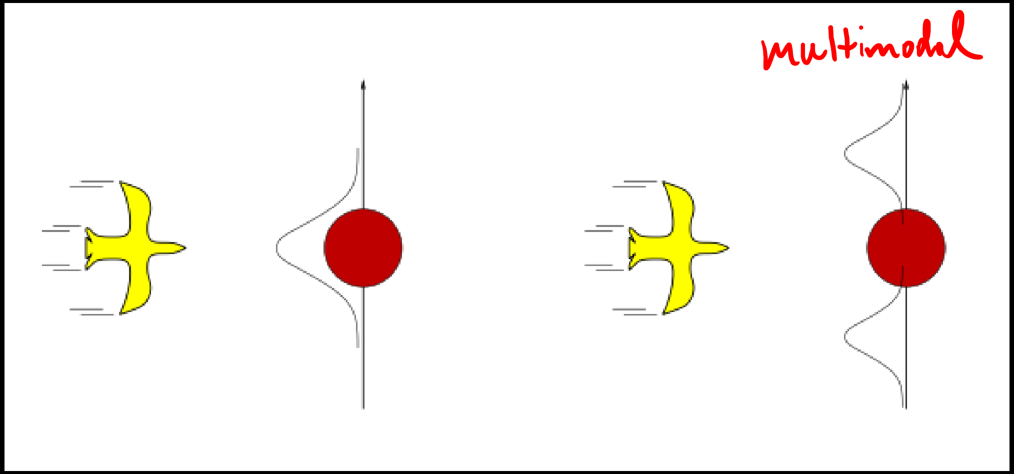
- Highly efficient: Polynomial in measurement dimensionality k and state dimensionality n:

$$O(k^{2.376} + n^2)$$

- Not optimal!
- Can diverge if nonlinearities are large!
- In practice, works surprisingly well even when assumptions are violated

Overall Kalman Filter Limitations

- Greatest strength is the greatest weakness
  - Everything must be a Gaussian!
- Cannot be used if transition is non-linear:



Summary

- HMMs are Bayes nets that make the Markov assumption for a single **discrete** random variable
- Bayes Filter tracks a discrete state
  - Computationally intractable in high dimensions
- Kalman Filters track **continuous** random state variables for linear systems
  - Can have multiple state and sensor variables
  - Assume state and measurement distributions are Gaussian
    - Fast updates at the expense of generality
- Extended Kalman Filter (EKF) is for nonlinear systems
  - Still using Gaussian state and measurement distributions
  - Use Jacobian of motion model and observation model functions
  - Works well when function is close to linear locally

Homework

- Particle Filters, Sec. 2.1