

COM SCI 188

Intro to Robotics

Lecture 9

Yuchen Cui

Winter 2025

Agenda

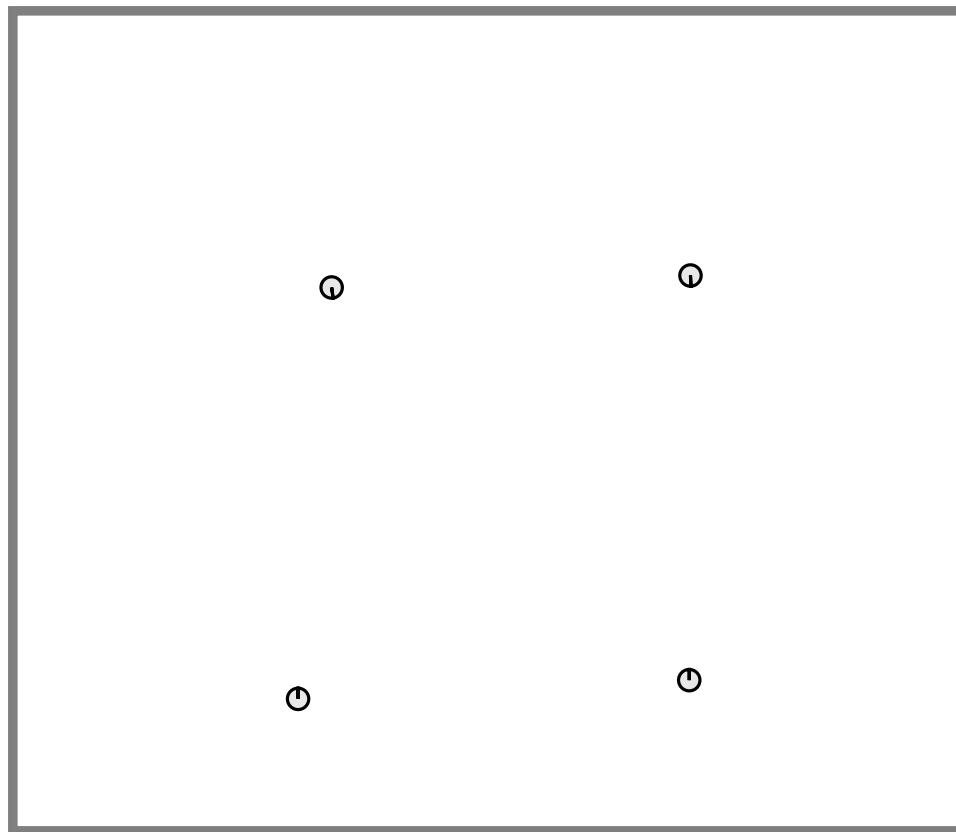
- Announcements
- Recap: Particle Filter
- SLAM (PR 10)

Announcements

- Office hour this week moves to Thursday 3-4pm
- Coding Assignment 2 due Friday
- Problem Set 3 out
- Midterm Exam (Week 7): Feb 19th
 - You are allowed to bring 1-page double-sided hand-written note
 - No calculators

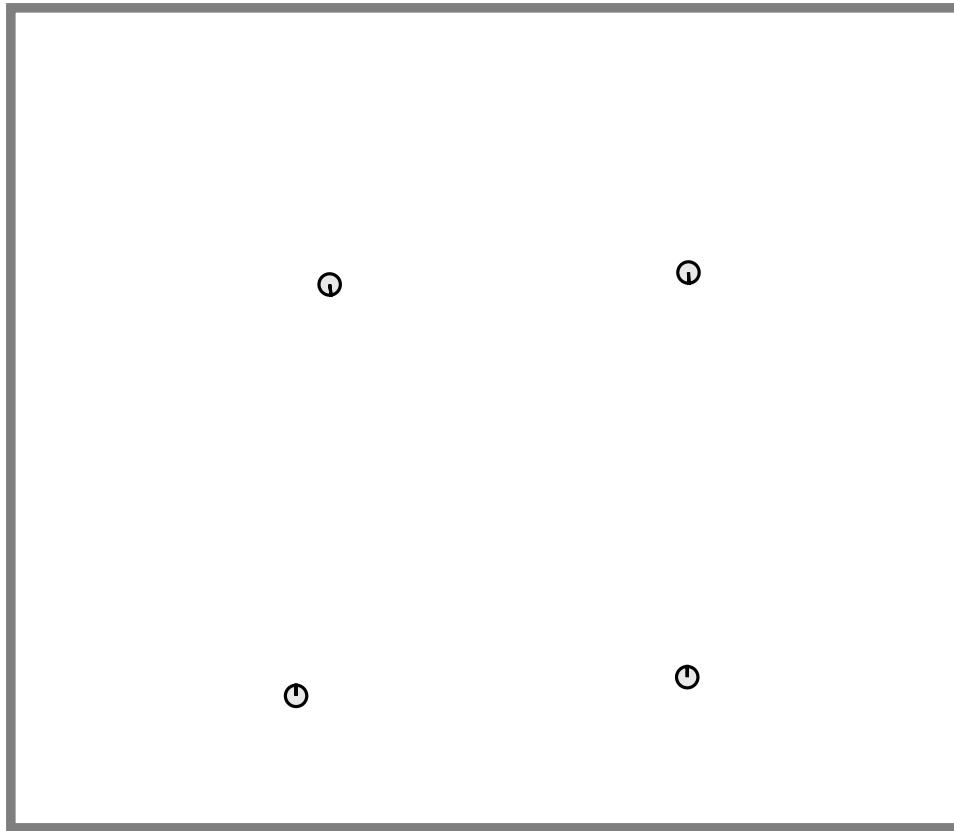
Recap: Particle Filter for Localization -> Roomba asks where am I?

Initialize: Create N particles distributed randomly around the map/initial guess, each with equal probability.



Recap: Particle Filter for Localization

Move: When a new **movement command (d, θ)** is issued: Go forward 3 ft. → $a = (3, 0)$

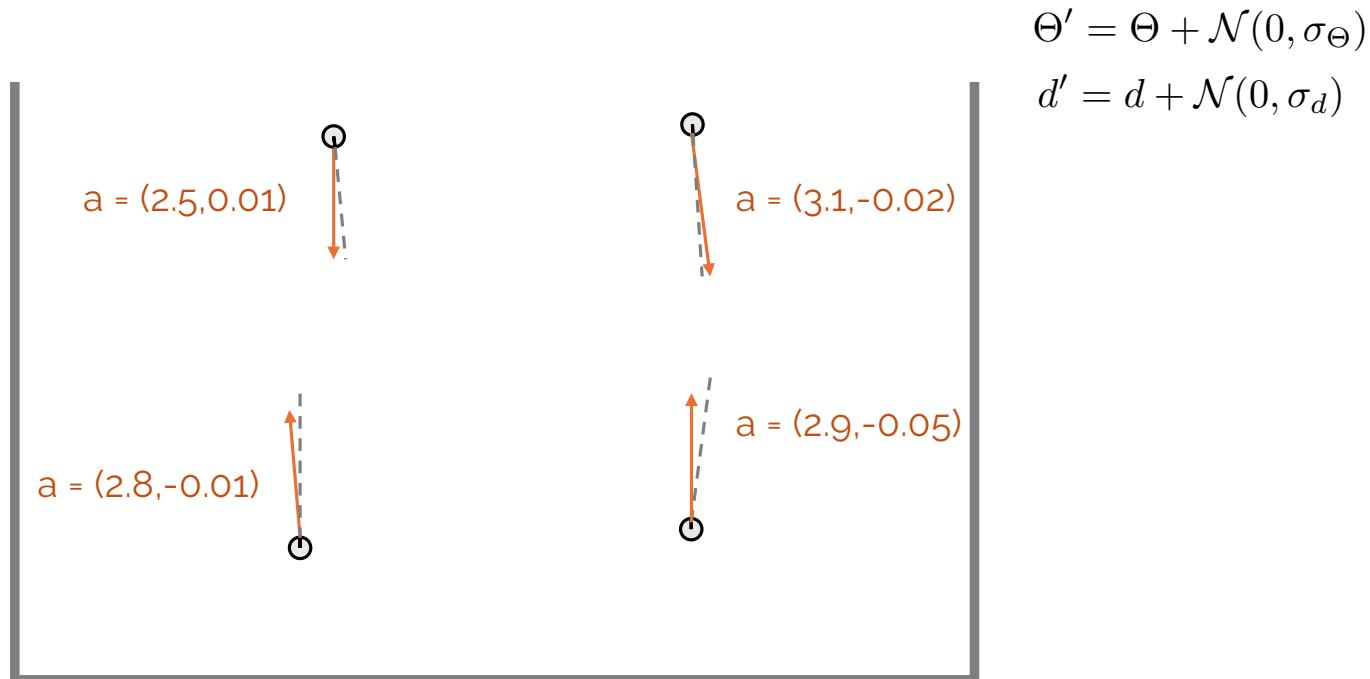


Recap: Particle Filter for Localization

Move: When a new **movement command** (d, θ) is issued: Go forward 3 ft. → $a = (3,0)$

For each particle "p" in "Particles":

Generate a random movement consisting of d' and θ' according **movement model**



Recap: Particle Filter for Localization

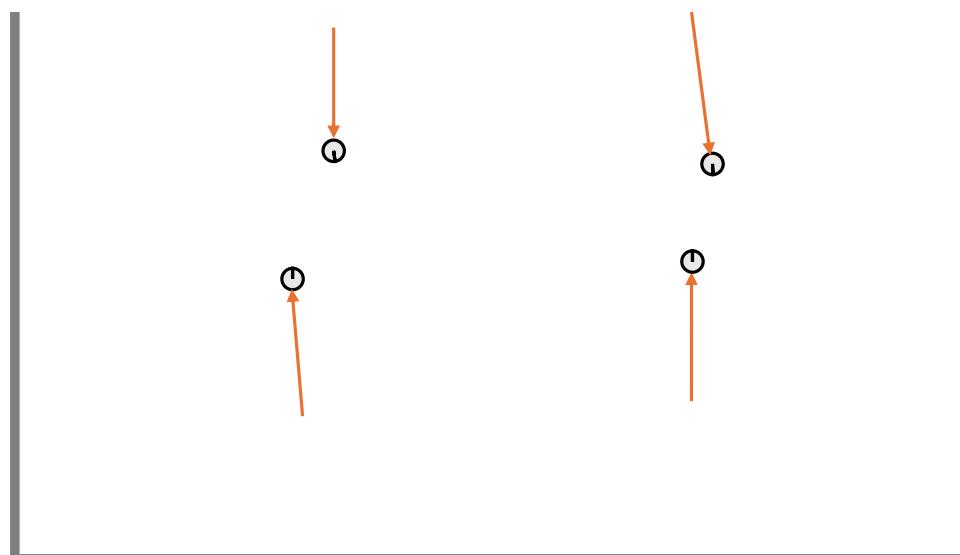
Move: When a new **movement command (d, θ)** is issued: Go forward 3 ft. → $a = (3,0)$

For each particle "p" in "Particles":

Generate a random movement consisting of d' and Θ' according **movement model**

Update the current position of "p" according to the **motion model** $x_{t+1} = x_t + d' \cos \Theta'$

$$y_{t+1} = y_t + d' \sin \Theta'$$

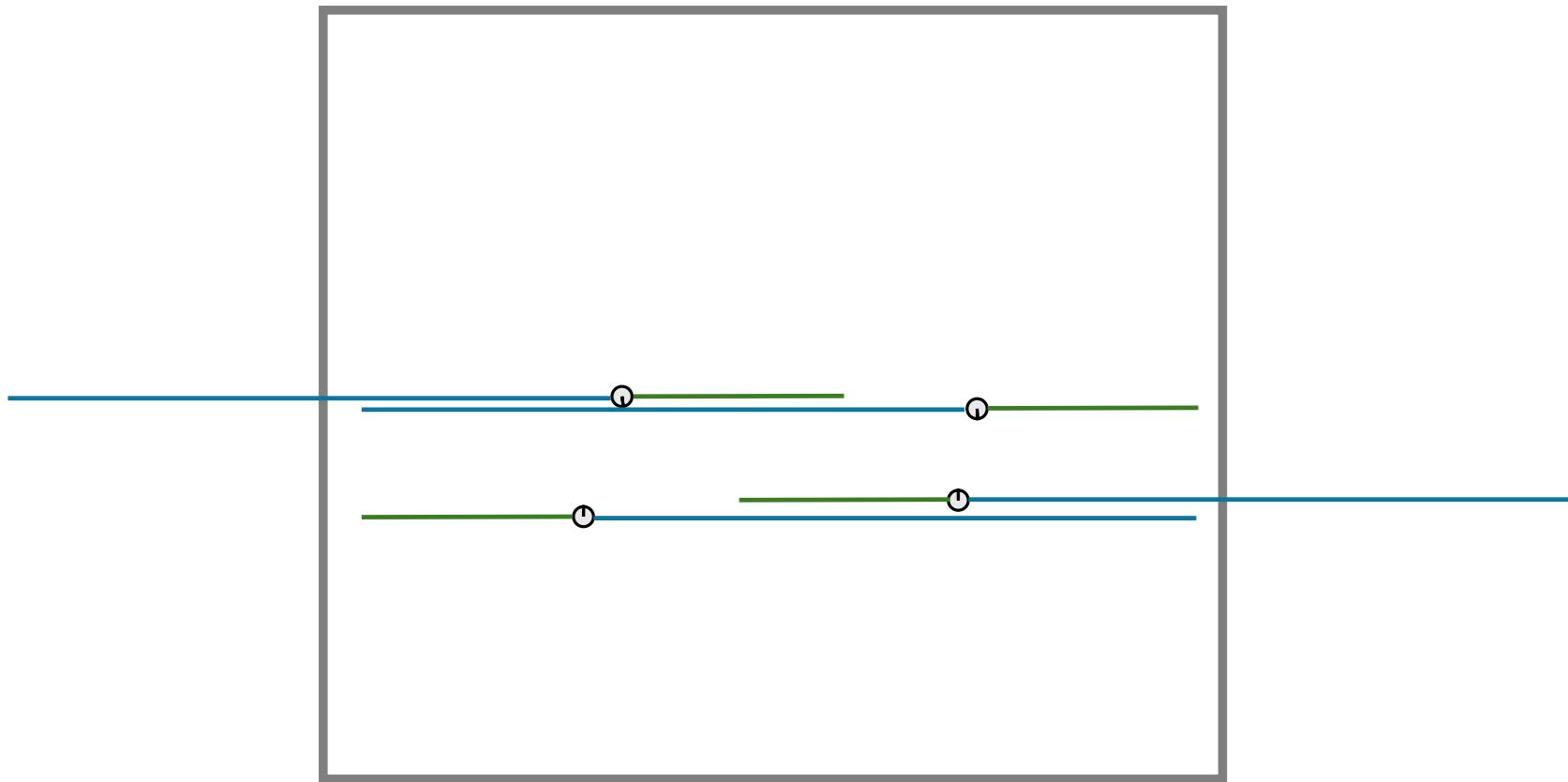


Make bounds checking to ensure that our particles are not ghosting through walls

Recap: Particle Filter for Localization

Move: When a new **movement command (d, θ)** is issued: Go forward 3 ft. → $a = (3, 0)$

Update: When a new **sensor measurement ("z")** is received: Left = 1m Right = 2.5m



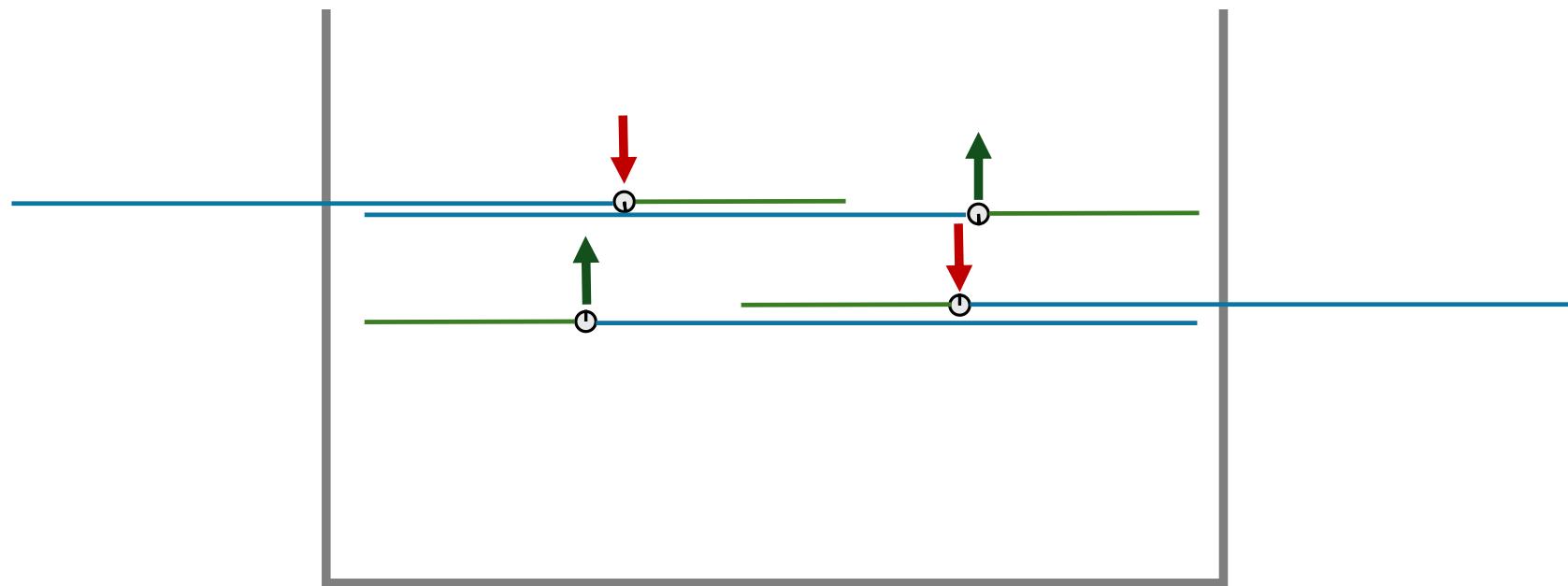
Recap: Particle Filter for Localization

Move: When a new **movement command (d, θ)** is issued: Go forward 3 ft. → $a = (3, 0)$

Update: When a new **sensor measurement ("z")** is received: Left = 1m Right = 2.5m

For each particle "p" in "Particles":

- Compute the **posterior probability** of each particle: $P("p", \text{location} | "z")$
- $P("p", \text{location} | "z") = P("z" | "p", \text{location}) P("p", \text{location}) / P("z")$

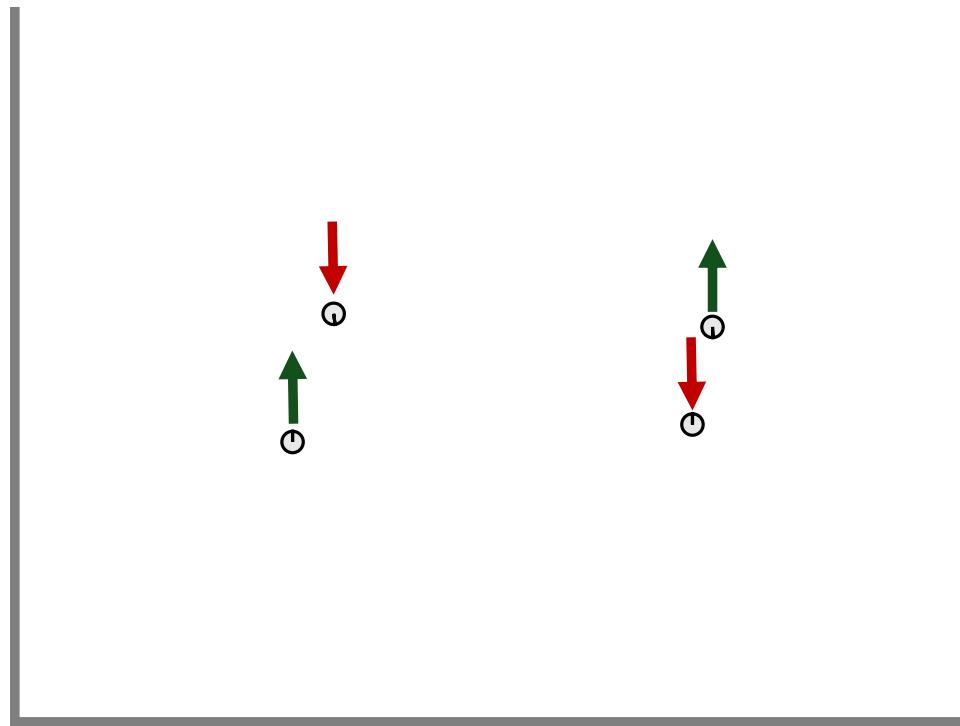


Recap: Particle Filter for Localization

Move: When a new **movement command (d, θ)** is issued: Go forward 3 ft. → $a = (3, 0)$

Update: When a new **sensor measurement ("z")** is received: Left = 1m Right = 2.5m

Resample the particles: "Particles" = resampleParticles("Particles")
resample with probability making copies proportional to the current posterior

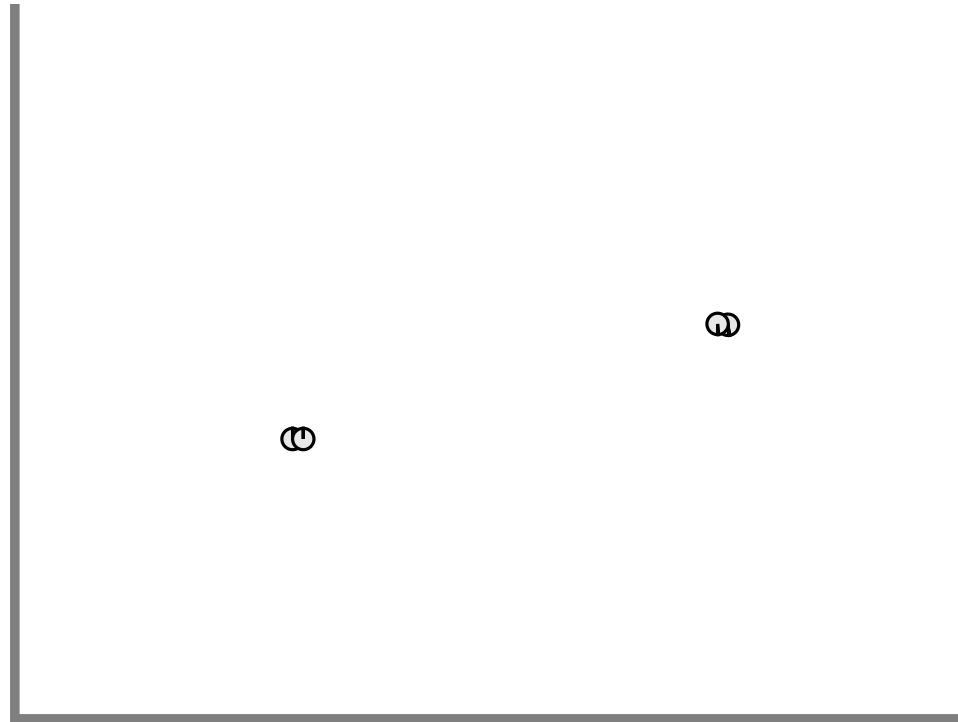


Recap: Particle Filter for Localization

Move: When a new **movement command (d, θ)** is issued: Go forward 3 ft. → $a = (3, 0)$

Update: When a new **sensor measurement ("z")** is received: Left = 1m Right = 2.5m

Resample the particles: "Particles" = resampleParticles("Particles")
resample with probability making copies proportional to the current posterior



SLAM: Simultaneous localization and mapping

Slides adapted from USC CSCI 445: Introduction to robotics
Credit: Erdem Biyik, Heather Culbertson

SLAM: Simultaneous localization and mapping

Localization: inferring location given a map

Mapping: inferring a map given exact locations

SLAM: learning a map and locating the robot simultaneously

What is the SLAM problem?

Given:

- A moving agent (roomba, drone, phone, self-driving car)
- Sensors (camera, LiDAR, sonar, IMU, wheel encoders)

Estimate **simultaneously**:

- The agent's **trajectory** over time
- A **map** of the environment
- while both are **initially unknown** and measurements are **noisy**

SLAM applications

SLAM is central to a range of indoor, outdoor, in-air and underwater applications for both manned and autonomous vehicles.

Examples:

- At home: vacuum cleaner, lawn mower
- Entertainment: ARKit / ARCore
- Air: surveillance with unmanned air vehicles
- Underwater: reef monitoring
- Underground: exploration of mines
- Space: terrain mapping for localization



The SLAM Problem

SLAM is usually framed as a **state estimation problem**:

$$p(\text{trajectory}, \text{map} \mid \text{sensor measurements})$$

- SLAM is considered a fundamental problem for robots to become truly autonomous
- Large variety of different SLAM approaches have been developed
- The majority uses probabilistic concepts
- History of SLAM dates to the mid-eighties

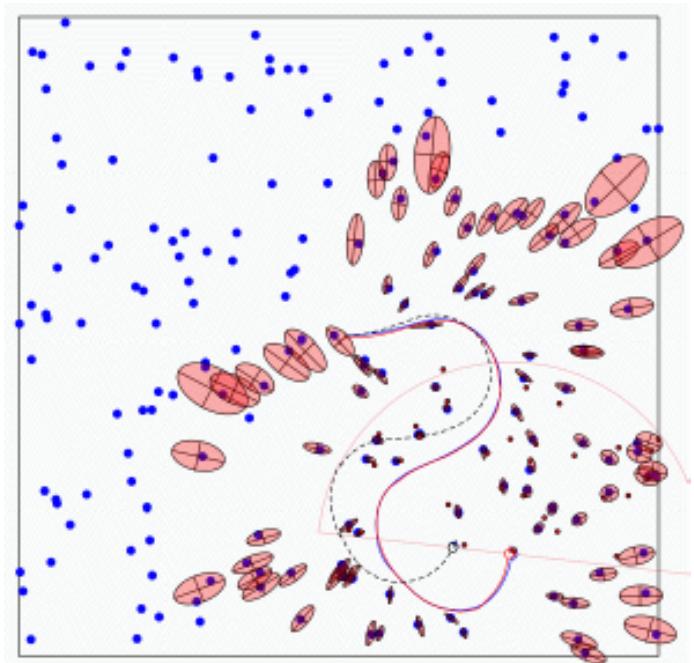
Feature-based SLAM

Given:

- The robot's controls $\mathbf{U}_{1:k} = \{u_1, u_2, \dots, u_k\}$
- Relative observations $\mathbf{Z}_{1:k} = \{z_1, z_2, \dots, z_k\}$

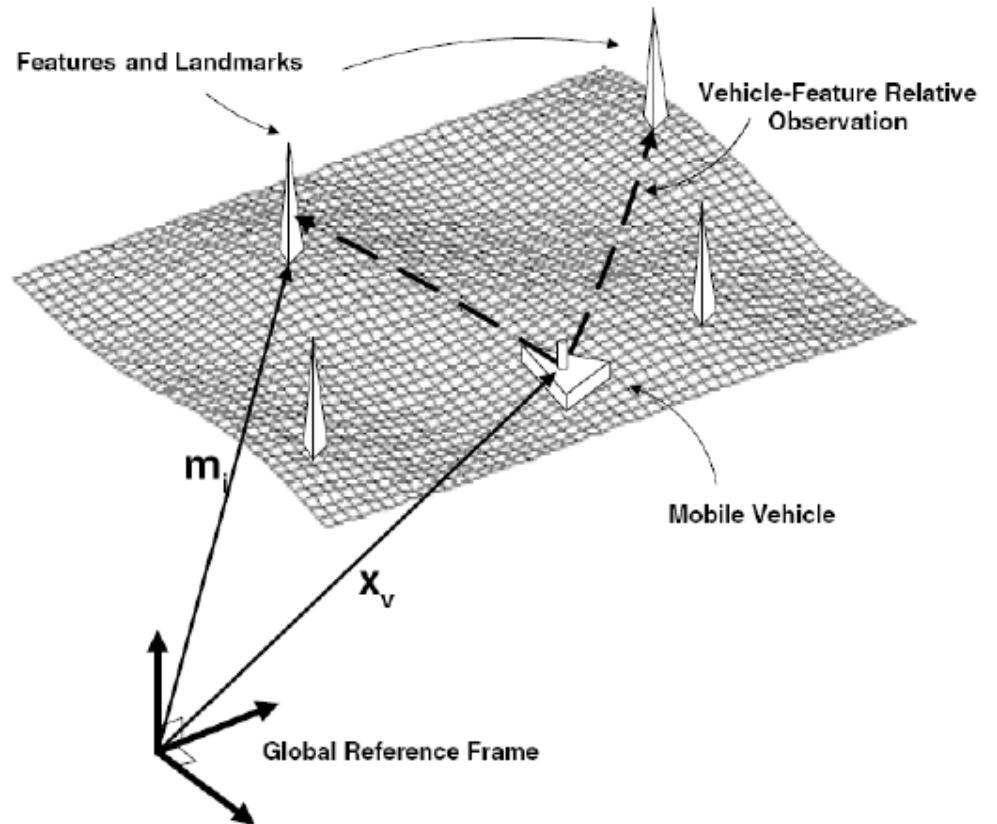
Wanted:

- Map of features $\mathbf{m} = \{m_1, m_2, \dots, m_n\}$
- Path of the robot $\mathbf{X}_{1:k} = \{x_1, x_2, \dots, x_k\}$



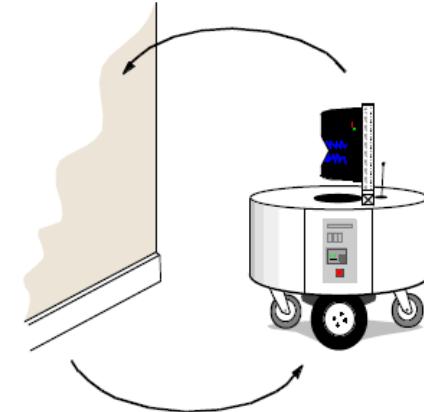
Feature-based SLAM

- Output:
 - **Absolute** robot poses
 - **Absolute** landmark positions
- Given:
 - **Relative** measurements of landmarks



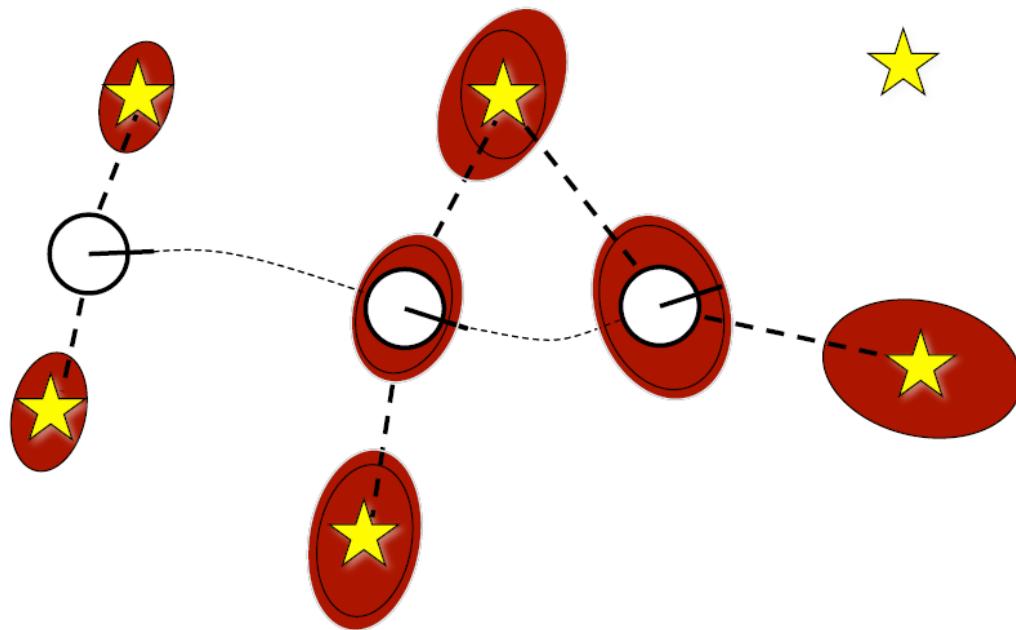
Why SLAM is fundamentally hard?

- A **chicken-and-egg problem!!!!**
- **Uncertainty compounds**
 - Sensor noise
 - Motion error (wheel slip, drift)
- **Data association**
 - Is this landmark I see now the *same* one I saw earlier?
- **Loop closure**
 - Recognizing you've returned to a previously visited place
 - Small errors accumulate → big map distortions unless corrected



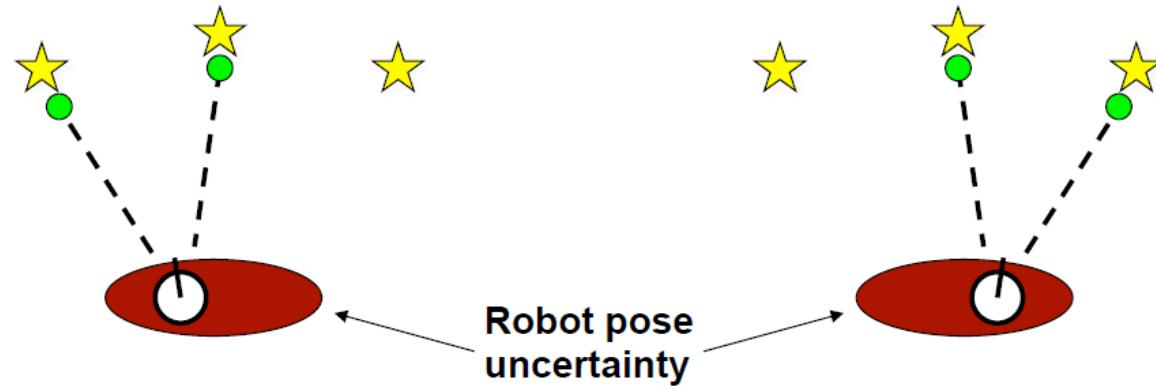
Why is SLAM a hard problem?

- Robot path and map are both unknown
- Errors in map and pose estimates correlated



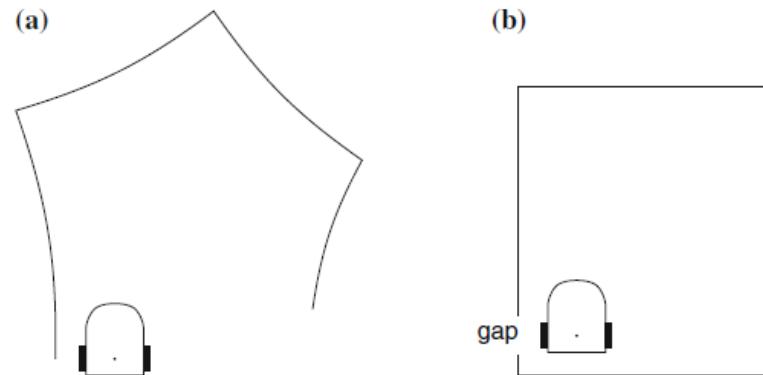
Why is SLAM a hard problem?

- The mapping between observations and landmarks is unknown
- Picking wrong data associations can have catastrophic consequences (divergence)



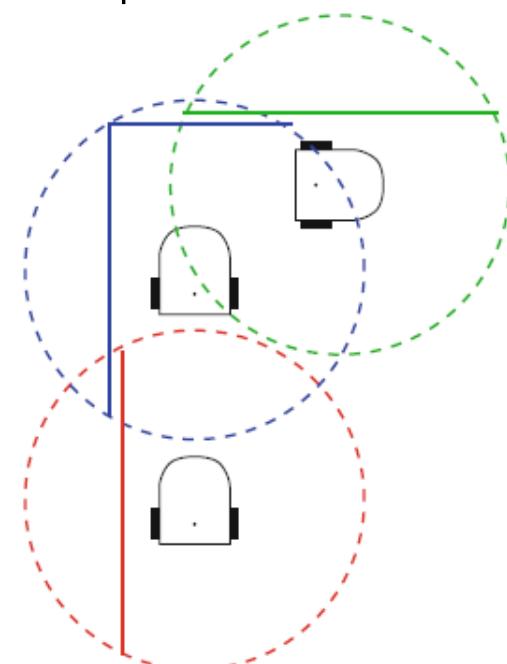
Knowledge about environment

- Cannot rely on odometry alone
- Need knowledge about environment or its structure
 - i.e., walls are straight, corners are 90°
- Can be used to close the loop



Overlapping sensor measurements

- Regular features, seen from multiple angles at different points in time
- Can be used to update localization and correct maps
- Basis of SLAM algorithm



Intuition

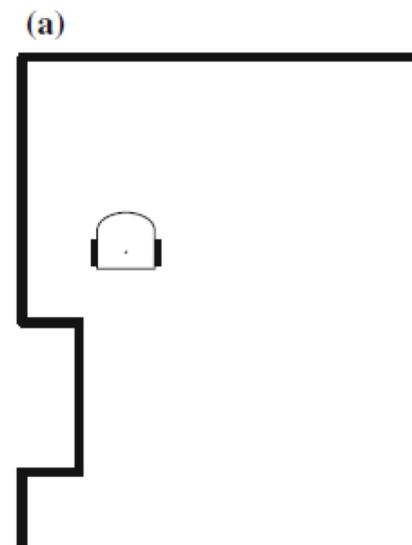
SLAM is about turning noisy motion and perception into a globally consistent map by **exploiting repeated observations** over time.

"I don't know where I am, but I'll figure it out as long as I keep recognizing stuff."

SLAM - Derivation

Divide environment into a grid

- Each cell is labeled as free or obstacle
 - Unknown/unexplored cells shown as ?



(b)

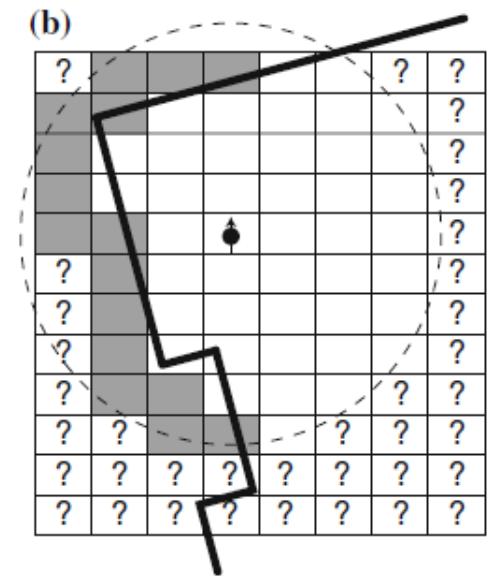
?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?
?						?	?
?						?	?
?						?	?
?				●		?	?
?						?	?
?						?	?
?	?					?	?
?	?					?	?
?						?	?
?						?	?

SLAM - Derivation

- Inconsistencies in actuators and effectors mean that the robot does not perfectly execute each motion command
- (a) is intended perception
- (b) is actual perception

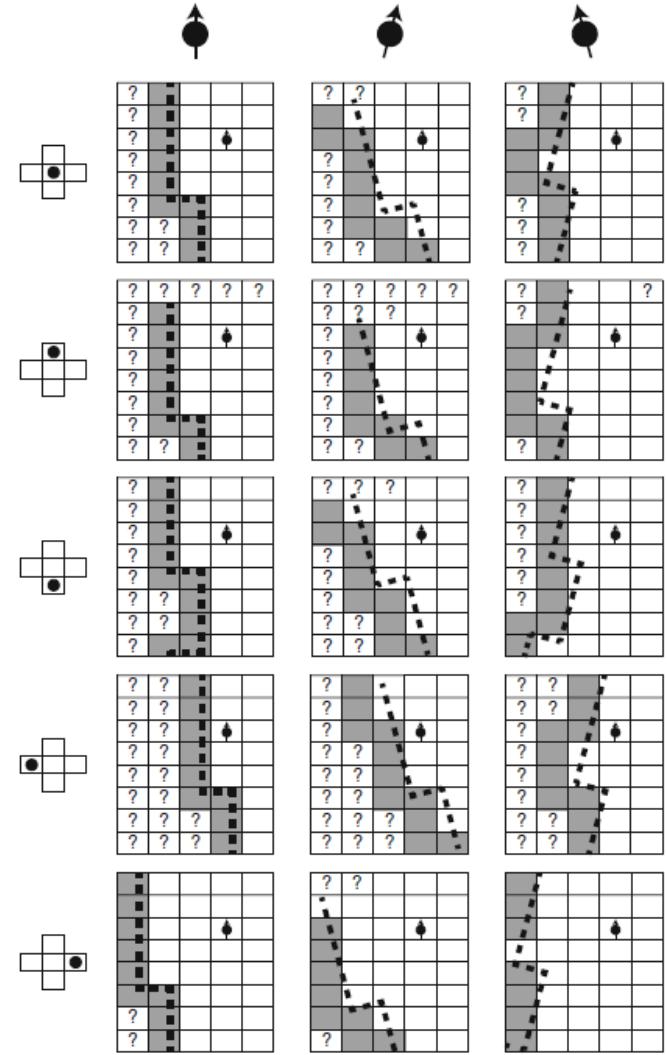
(a)

?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?							?	?
?							?	?
?							?	?
?							?	?
?							?	?
?	?						?	?
?	?						?	?
?	?						?	?



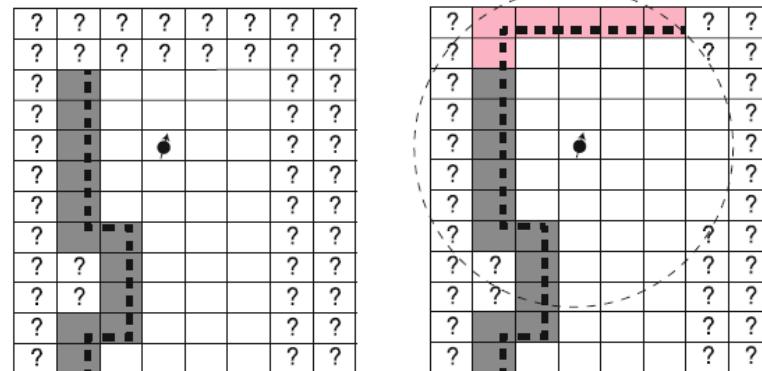
SLAM - Derivation

- Assume odometry gives a reasonable estimation of pose (position and heading) for short motions
- For each relatively small possible error in pose, compute what the perception of the current map would be and compare it with the actual perception computed from sensor data
 - Choose pose that gives best match
 - Set this as actual pose of robot and update map



SLAM - Derivation

- Correct the pose of the robot
 - Use data from perception map to update current map stored in the robot's memory



SLAM Algorithm

Move
Measure

Model
Correct

Update

Algorithm 9.2: SLAM

```
matrix m ← partial map      // Current map
matrix p                      // Perception map
matrix e                      // Expected map
coordinate c ← initial position // Current position
coordinate n                  // New position
coordinate array T            // Set of test positions
coordinate t                  // Test position
coordinate b ← none          // Best position
```

```
1: loop
2:   move a short distance
3:   n ← odometry(c)           // New position based on odometry
4:   p ← analyze sensor data

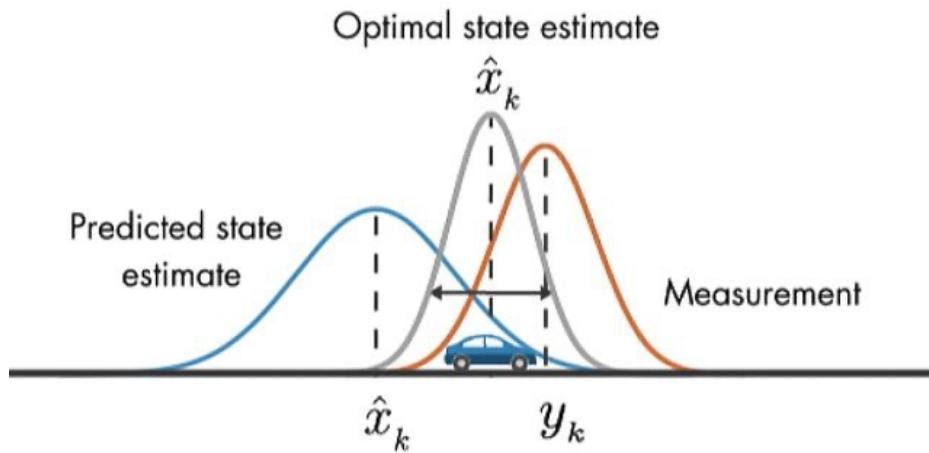
5:   for every t in T           // T is the positions around n
6:     e ← expected(m, t)       // Expected map at test position
```

How do we determine what is the “best position”?

```
9:   n ← b                   // Replace new position by best position
10:  m ← update(m,p,n)        // Update map based on new position
11:  c ← n                   // Current position is new position
```

Kalman filter

- Also known as Linear Quadratic Estimation (LQE)
- assumes **Gaussian noise** in both motion and measurements and uses a **linearized approximation** of the system dynamics
- Uses a series of measurements observed over time
- Follows similar process to particle filter but is not the same!



Kalman filter

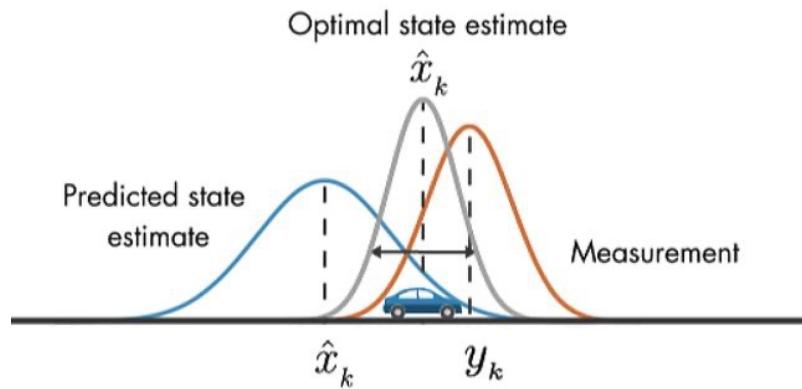
At Each Time Step:

Predict

- Use system model to estimate next state
- Increase uncertainty due to process noise

Update

- Compare prediction with measurement
- Compute optimal correction (Kalman gain)
- Reduce uncertainty



Kalman filter

Model

- State transition: $x_t = Ax_{t-1} + Bu_t + w_t, w_t \sim \mathcal{N}(0, Q)$
- Measurement: $z_t = Hx_t + v_t, v_t \sim \mathcal{N}(0, R)$

Process / motion uncertainty

Measurement / sensor uncertainty

Predict

- State: $\hat{x}_t^- = A\hat{x}_{t-1} + Bu_t$
- Covariance: $\Sigma_t^- = A\Sigma_{t-1}A^T + Q$

Update

- Kalman gain: $K_t = \Sigma_t^- H^T (H\Sigma_t^- H^T + R)^{-1}$
- State update: $\hat{x}_t = \hat{x}_t^- + K_t(z_t - H\hat{x}_t^-)$
- Covariance update: $\Sigma_t = (I - K_t H)\Sigma_t^-$

Extended Kalman filter

Model

- State transition: $x_t = f(x_{t-1}, u_t) + w_t, w_t \sim \mathcal{N}(0, Q)$
- Measurement: $z_t = h(x_t) + v_t, v_t \sim \mathcal{N}(0, R)$

Example (2D differential drive):

$$x_t^r = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, f(x_{t-1}^r, u_t) = \begin{bmatrix} x + v\Delta t \cos \theta \\ y + v\Delta t \sin \theta \\ \theta + \omega\Delta t \end{bmatrix}$$

$$z_t^i = \begin{bmatrix} \sqrt{(x_i - x)^2 + (y_i - y)^2} \\ \text{atan2}(y_i - y, x_i - x) - \theta \end{bmatrix}$$

Predict

- State: $\hat{x}_t^- = f(\hat{x}_{t-1}^-, u_t)$
- Covariance: $= A_t \Sigma_{t-1} A_t^T + Q$

Linearization

$$A_t = \left. \frac{\partial f}{\partial x} \right|_{\hat{x}_{t-1}^-, u_t}, H_t = \left. \frac{\partial h}{\partial x} \right|_{\hat{x}_t^-}$$

Update

- Kalman gain: $K_t = \Sigma_t^- H_t^T (H_t \Sigma_t^- H_t^T + R)^{-1}$
- State update: $\hat{x}_t = \hat{x}_t^- + K_t (z_t - h(\hat{x}_t^-))$
- Covariance update: $\Sigma_t = (I - K_t H_t) \Sigma_t^-$

Extended Kalman Filter (EKF) SLAM

- Localization State
 - 3x1 pose vector
 - 3x3 covariance matrix

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} \quad C_k = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{x\theta} \\ \sigma_{yx} & \sigma_y^2 & \sigma_{y\theta} \\ \sigma_{\theta x} & \sigma_{\theta y} & \sigma_\theta^2 \end{bmatrix}$$

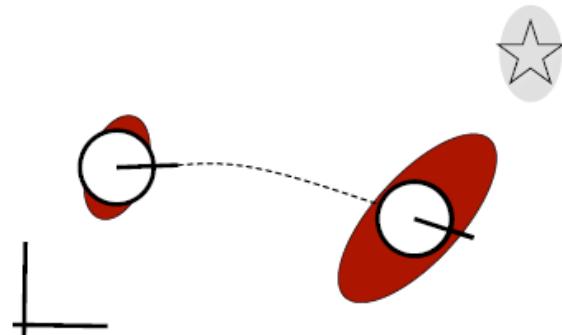
- SLAM
 - Landmarks simply extend the state. Growing both the state vector and covariance matrix.
 - Ex: Map with N landmarks

$$Bel(x_t, m_t) = \left\{ \begin{array}{c|ccccc} \begin{pmatrix} x \\ y \\ \theta \\ l_1 \\ l_2 \\ \vdots \\ l_N \end{pmatrix}, & \begin{pmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{x\theta} & \sigma_{xl_1} & \sigma_{xl_2} & \cdots & \sigma_{xl_N} \\ \sigma_{yx} & \sigma_y^2 & \sigma_{y\theta} & \sigma_{yl_1} & \sigma_{yl_2} & \cdots & \sigma_{yl_N} \\ \sigma_{x\theta} & \sigma_{y\theta} & \sigma_\theta^2 & \sigma_{\theta l_1} & \sigma_{\theta l_2} & \cdots & \sigma_{\theta l_N} \end{pmatrix} \\ \hline \end{array} \right\}$$

EKF SLAM: Building the Map

- Filter cycle:

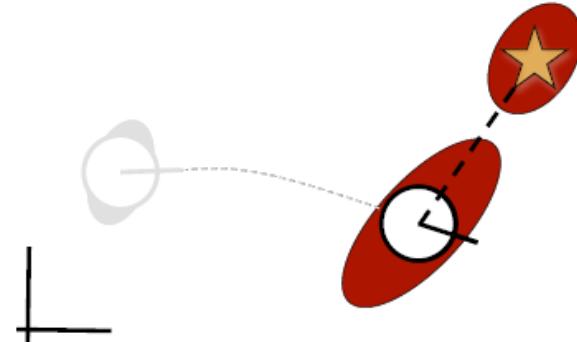
1. **Predict robot's state** (odometry)



$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_R \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \vdots \\ \mathbf{m}_n \end{bmatrix}_k \quad C_k = \begin{bmatrix} C_R & C_{RM_1} & C_{RM_2} & \cdots & C_{RM_n} \\ C_{M_1 R} & C_{M_1} & C_{M_1 M_2} & \cdots & C_{M_1 M_n} \\ C_{M_2 R} & C_{M_2 M_1} & C_{M_2} & \cdots & C_{M_2 M_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{M_n R} & C_{M_n M_1} & C_{M_n M_2} & \cdots & C_{M_n} \end{bmatrix}_k$$

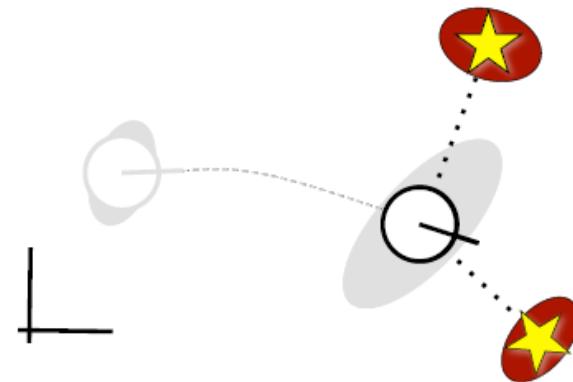
EKF SLAM: Building the Map

- Filter cycle:
 1. **Predict robot's state** (odometry)
 2. **Measurement prediction** (predict what the robot should see based on landmarks that currently exist in map)



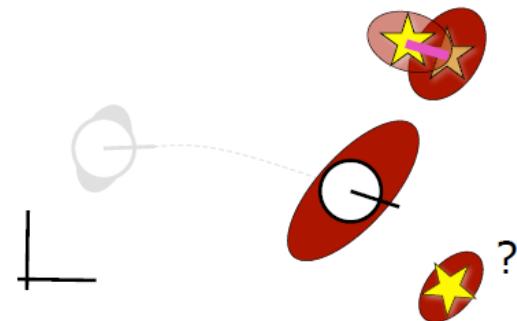
EKF SLAM: Building the Map

- Filter cycle:
 1. **Predict robot's state** (odometry)
 2. **Measurement prediction** (predict what the robot should see based on landmarks that currently exist in map)
 3. **Observation** (measure actual distance and orientation to landmarks)



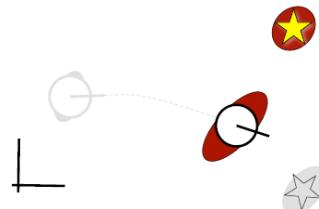
EKF SLAM: Building the Map

- Filter cycle:
 1. **Predict robot's state** (odometry)
 2. **Measurement prediction** (predict what the robot should see based on landmarks that currently exist in map)
 3. **Observation** (measure actual distance and orientation to landmarks)
 4. **Data association** (associates predicted measurements to observations)



EKF SLAM: Building the Map

- Filter cycle:
 1. **Predict robot's state** (odometry)
 2. **Measurement prediction** (predict what the robot should see based on landmarks that currently exist in map)
 3. **Observation** (measure actual distance and orientation to landmarks)
 4. **Data association** (associates predicted measurements to observations)
 5. **Update** (update pose vector and covariance matrix using Kalman Filter)


$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_R \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \vdots \\ \mathbf{m}_n \end{bmatrix}_k \quad C_k = \begin{bmatrix} C_R & C_{RM_1} & C_{RM_2} & \cdots & C_{RM_n} \\ C_{M_1 R} & C_{M_1} & C_{M_1 M_2} & \cdots & C_{M_1 M_n} \\ C_{M_2 R} & C_{M_2 M_1} & C_{M_2} & \cdots & C_{M_2 M_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{M_n R} & C_{M_n M_1} & C_{M_n M_2} & \cdots & C_{M_n} \end{bmatrix}_k$$

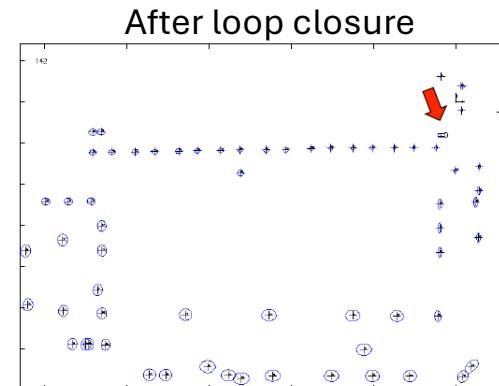
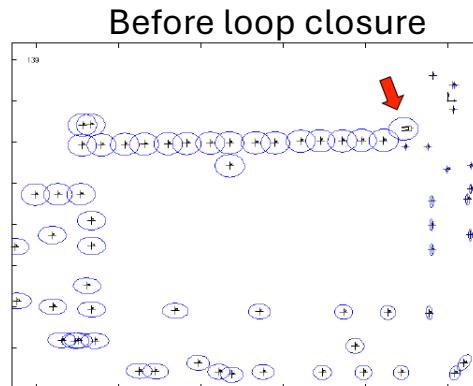
EKF SLAM: Building the Map

- Filter cycle:
 - Predict robot's state** (odometry)
 - Measurement prediction** (predict what the robot should see based on landmarks that currently exist in map)
 - Observation** (measure actual distance and orientation to landmarks)
 - Data association** (associates predicted measurements to observations)
 - Update** (update pose vector and covariance matrix using Kalman Filter)
 - Integration of new landmarks** (extend pose vector and covariance matrix with new landmarks)

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_R \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \vdots \\ \mathbf{m}_n \\ \mathbf{m}_{n+1} \end{bmatrix}_k \quad C_k = \begin{bmatrix} C_R & C_{RM_1} & C_{RM_2} & \cdots & C_{RM_n} & C_{RM_{n+1}} \\ C_{M_1 R} & C_{M_1} & C_{M_1 M_2} & \cdots & C_{M_1 M_n} & C_{M_1 M_{n+1}} \\ C_{M_2 R} & C_{M_2 M_1} & C_{M_2} & \cdots & C_{M_2 M_n} & C_{M_2 M_{n+1}} \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ C_{M_n R} & C_{M_n M_1} & C_{M_n M_2} & \cdots & C_{M_n} & C_{M_n M_{n+1}} \\ C_{M_{n+1} R} & C_{M_{n+1} M_1} & C_{M_{n+1} M_2} & \cdots & C_{M_{n+1} M_n} & C_{M_{n+1}} \end{bmatrix}_k$$

Loop Closure

- Recognizing an already mapped area, typically after a long exploration path (the robot "closes a loop")
- Structurally identical to data association, but
 - high levels of ambiguity
 - possibly useless validation gates
 - environment symmetries
- Uncertainties collapse after a loop closure (whether the closure was correct or not)



Loop closure

- By revisiting already mapped areas, uncertainties in robot and landmark estimates can be reduced
- This can be exploited when exploring an environment for the sake of better (e.g. more accurate) maps
- Exploration: the problem of where to acquire new information

EKF-SLAM Summary

State

Mean: $x_t = [\text{robot pose}, \text{landmarks}]$

Covariance: Σ (captures all correlations)

$$x_t = \begin{bmatrix} x_t^r \\ l_1 \\ l_2 \\ \vdots \\ l_N \end{bmatrix}, \Sigma_t = \text{joint covariance}$$

Algorithm (repeat each time step)

Predict

- Use control/odometry to update robot pose
- Propagate uncertainty through motion model

$$\hat{x}_t^- = \begin{bmatrix} f(\hat{x}_{t-1}^r, u_t) \\ m_{t-1} \end{bmatrix}. \quad \Sigma_t^- = A_t \Sigma_{t-1} A_t^T + Q$$

Associate

- Match each measurement to an existing landmark **Landmark i :** $z_t^i = h(x_t^r, l_i) + v_t, v_t \sim \mathcal{N}(0, R)$
- If no match → initialize new landmark

Update

- Compute measurement residual
- Apply Kalman correction to robot and landmarks
- Reduce uncertainty using sensor information

$$\begin{aligned} K_t &= \Sigma_t^- H_t^{iT} (H_t^i \Sigma_t^- H_t^{iT} + R)^{-1} \\ \hat{x}_t &= \hat{x}_t^- + K_t (z_t^i - h(\hat{x}_t^-, l_i)) \\ \Sigma_t &= (I - K_t H_t^i) \Sigma_t^- \end{aligned}$$

EKF-SLAM: Complexity

- Cost per step:
 - quadratic in n , the number of landmarks: $O(n^2)$
- Total cost to build a map with n landmarks: $O(n^3)$
- Memory consumption: $O(n^2)$

Problem: becomes computationally intractable for large maps!

There exists variants to circumvent these problems

SLAM Techniques

- EKF SLAM
- FastSLAM (particle filter version)
- **Graph-based SLAM (modern standard)**
- Topological SLAM (mainly place recognition)
- Scan Matching / Visual Odometry (only locally consistent maps)
- Approximations for SLAM: Local submaps,
- Sparse extended information filters, Sparse links, Thin junction tree filters, etc.
- ...

MASt3R-SLAM: Real-Time Dense SLAM with 3D Reconstruction Priors

CVPR 2025

Riku Murai* Eric Dexheimer* Andrew J. Davison

Imperial College London



Uncalibrated RGB
8x Playback





That's it for today!

Questions?