Agenda

- Lecture: Robot Navigation → MAPPING!
- TheConstructSim:
  - Try out the ROS/Gazebo/Rviz simulators for Turtlebot3
  - More information on assignments will be posted to Piazza
    (ignore the schedule online)
- Upcoming:
  - Lecture next week: Ethics of Robotics and Automation
- References:
  - This lecture is partially based on "Introduction to AI Robotics", chapter 11, Robin Murphy, 2000. For SLAM, see online theory tutorial paper "SLAM: Part 1 The Essential Algorithms" by Durrant-Whyte et al., 2006 and online practical tutorial paper "SLAM for Dummies" S. Ringgaard, and M. Blas (2005)

Today: Robots Navigating the World

Scenarios
- Hospital Helper (e.g. Diligent, Tugs)
- Office security or mail-delivery (e.g. Cobal, Savioke)
- Tour Guide robot in a museum (Minerva)
- Autonomous Car with GPS and Nav system

Biological analogies: Humans, bees and ants, migrating birds, herds

Second Part of CS189: High-level reasoning
From finite state machines to complex representation and memory

- Path Planning: How to I get to my Goal?
- Localization: Where am I?
- Mapping: Where have I been?
- Exploration: Where haven’t I been?

Last Lecture: Today!
You are roaming around in an unknown space, what can you learn about it?

Two parts of the problem:
- **Mapping**: As you roam around the world, how do you build a memory of the shape of the space you have moved through?
- **Exploration**: Given that you don’t know the shape or size of the environment, how do make sure you covered all of it?

Both have many uses:
- Returning back to home/charger after some task.
- Cleaning a new room efficiently OR Systematic search for survivors
- Mapping a collapsed mine or building.

Mapping and Exploration are also “collections of algorithms”
- E.g. Many representations of a “map”, random walks are exploration
- We will focus on “Occupancy Grid” algorithms

Today’s topics
- Mapping and Exploration Algorithms
- Occupancy Grids and Sensor Models
- A First-cut Simple Mapping Algorithm
- Three Improvements
  - Exploration strategies
  - Frontier based exploration (guaranteed coverage)
  - Managing sensor uncertainty
  - Probabilistic algorithms for Occupancy Grid Mapping (Bayes Rule)
  - Managing motion uncertainty and sensor uncertainty together
  - Simultaneous Localization and Mapping (SLAM)
- Maybe? Pset 4: Your Autonomous OG Mapper!*

What is an Occupancy Grid?
- A way of representing a map as a gridded world where each cell is either “occupied” or “empty” or “unknown”.

* uses material from all 3 navigation lectures
Step 1: Constructing a Sensor Model
- A sensor measures raw values in an environment.
- You have to map that into a Grid Cell Value.
- Robots can have very different sensors and configurations.
- Examples:
  - Think about LIDAR/Depth Camera
  - Vs. a 360 degree vision/ranging system

Example: Depth Sensor Model
- $R = \text{maximum range}$, $B = \text{maximum angle}$
- Let say the sensor at point $p$ returns $\text{distance} = r$
- Region 1 ($\text{dist} < r$, grid cell probably empty)
- Region 2 ($\text{dist} = r$, grid cell probably obstacle)
- Region 3 ($\text{dist} > r$, grid cell unknown/obscured)
Example: Depth Sensor Model
R = maximum range, B = maximum angle
Let say the sensor at point p returns distance = \( r \)

Region 1 (dist < r, grid cell probably empty)
Region 2 (dist = r, grid cell probably obstacle)
Region 3 (dist > r, grid cell unknown/obscured)

Constructing a Sensor Model

Simplest Sensor Model
Where I stand is Empty (white)

A Better Model
Set Region 1 cells as Empty (white)
Set Region 2 cells as Occupied (black)
Pick a max range/angle where data is reliable
Rest is still Unknown (gray)

A Simple OG Mapping Algorithm

1. Initialize a Grid
   - Set all locations as “unknown”, pick a start location and orientation
2. Update the Grid
   - Mark your current grid position as “empty”
   - Using your better sensor model, mark all visible grid locations as “empty” or “occupied”
3. Pick a Next Move
   - Look at neighboring grid positions in your map
   - Pick a neighboring grid location that is empty (randomly)
   - Move to it and update your current position in the Grid
4. Loop forever
   - Keep moving and updating the grid (unless you are “done”)
A Simple Mapping Algorithm

1. Initialize Grid
2. Update the Grid
   - Mark your current position as “empty”
   - Mark sensed nearby grid locations as “empty” or “occupied”
3. Pick a Next Move
   - Look at neighboring grid positions
   - Choose a random empty direction
   - Move and update your position in the Grid
4. Loop forever

Exploration

- Basic Concept in Math: Random Walks in bounded 2D
  - With Probability=1 you will eventually visit every spot
- Basic Concept in CS: Systematic Graph Coverage
  - You are given a “graph” with V nodes
  - Write an algorithm that visits all of the nodes
    - Breath-First Search and Depth-First Search; Time Complexity: O(V+E)
- Basic Concept in Robotics: Traversing a GRID Graph is different
  - DFS works, but will still make a robot retrace steps
  - Better choice: Frontier Based Exploration
Exploration in Grid Worlds

- **Frontier Based Exploration**
  - A common technique for building maps
  - **Key Idea:**
    - Identify the “frontiers” between known and unknown
    - Frontier cell = a unknown cell with at least one empty cell nbr
  - Pick a frontier cell (e.g. the closest)
  - Plan a path to go explore it.

- **Done Condition:***
  - No more frontier nodes left => your map is Complete!
  - If finite world, then any algorithm that systematically explores frontier nodes is guaranteed to cover the whole world.

A Less Simple Mapping Algorithm

1. **Initialize Grid**
2. **Update the Grid**
   - Mark your current position as “empty”
   - Mark sensed nearby grid locations as “empty” or “occupied”
3. **Pick a Next Move**
   - Identify frontier cells
   - Pick one (e.g. maybe the closest)
   - Plan a path* to the nbr empty cell.
   - Go to that location using this path (and keep track of your position as you move)
4. **Loop** until no frontier nodes are left

* We covered path planning two lectures ago

The smart sensor model and smart exploration strategy make for much faster mapping!
Turtlebots can do this!

Today’s topics

- Mapping and Exploration Algorithms
- Occupancy Grids and Sensor Models
- A First-cut Simple Mapping Algorithm

- Three Improvements
  - Exploration strategies
  - Managing sensor uncertainty
    - Probabilistic algorithms for Occupancy Grid Mapping (Bayes Rule)
    - Managing motion uncertainty and sensor uncertainty together
  - Simultaneous Localization and Mapping (SLAM)

Questions?
A Less Simple Mapping Algorithm

1. Initialize Grid
   - Mark your current position as "empty"
   - Mark sensed nearby grid locations as "empty" or "occupied"

2. Update the Grid
   - Identify frontier cells
   - Pick one (e.g., the closest)
   - Plan a path to the empty cell
   - Go to that location using this path (and keep track of your position as you move)

3. Pick a Next Move
   - Identify frontier cells
   - Pick one (e.g., maybe the closest)
   - Plan a path to the empty cell
   - Go to that location using this path (and keep track of your position as you move)

4. Loop until no frontier nodes are left

Improvement 2: Sensors aren’t perfect
Take advantage of the fact that you are often retracing steps
And taking measurements multiple times of the same location

Part 1: A Probabilistic Sensor Model

Example: Depth Sensor Model
R = maximum range, B = maximum angle
Let say the sensor at point p returns distance = “r”
Region 1 (dist < r, grid cell probably empty)
Region 2 (dist = r, grid cell probably obstacle)
Region 3 (dist > r, grid cell unknown/obscured)

A More Complex Sensor Model: Probabilistic
For a cell at distance r and angle a
P("correctness") = [(R - r/R) + (B - a/B)]/2
I.e. Uncertainty in my assessment grows with distance and angle from the centerline

Part 2: A Bayesian OG Map

For every grid location (i,j), store current probability value
P(Occupied) = Probability this grid location is Occupied
P(Empty) = 1 - P(Occupied)

Bayesian Map
For every grid location (i,j), store current probability value
P(Occupied|s) = Probability this grid location is Occupied ("n" timestep)

Bayesian Map Update Rule
P(Occupied|s) = P(s|Occupied) * P(Occupied) / P(s)

Bayesian Rule
P(A|s) = P(s|A) * P(A) / P(s|A) + P(s|1-A) * P(1-A)

For every grid location (i,j), store current probability value
P(Occupied|s) = Probability this grid location is Occupied ("n" timestep)
Bayesian OG Mapping

- In the beginning of time,
  - \( P(\text{Occupied}) = P(\text{Empty}) = 0.5 \)
- Let's say I observe grid(5,6) for the first time, and let's say my sensor reading \( s=\text{obstacle} \) (but it's far away, i.e., less sure)
  - New Reading: \( P(s|\text{Occupied}) = 0.62, P(s|\text{Empty}) = 0.38 \)
  - Old Map Estimate \( P(\text{Occupied}) = P(\text{Empty}) = 0.5 \)
  - \( P(\text{Occupied}|s = \text{obs}) = (0.62 \times 0.5) / (0.62 \times 0.5 + 0.38 \times 0.5) = 0.62 \)
    Which is what you'd expect because we have no better knowledge
- Later if we observe location grid (5,6) again, we have prior knowledge
  - We now think \( P(\text{Occupied}) = 0.62 \), \( P(\text{Empty}) = 0.38 \)
  - New sensor reading \( P(s=\text{obstacle}|\text{Occupied}) = 0.80 \) (we are closer & surer)
  - \( P(\text{Occupied}|s = \text{obs}) = (0.8 \times 0.62) / (0.8 \times 0.62 + 0.2 \times 0.38) = 0.87 \)
    (my new confidence is higher, that this grid cell is occupied)

Bayesian Map Update Rule:

\[
P(\text{Occupied}|s) = \frac{P(s|\text{Occupied}) \times P(\text{Occupied})}{P(s|\text{Occupied}) \times P(\text{Occupied}) + P(s|\text{Empty}) \times P(\text{Empty})}
\]

Improvement 2: Probabilistic Mapping

- Overarching idea
  - Store probabilities of occupancy rather than 3 values.
  - Caveat: We treat each grid cell as independent even though its not.
- But how do you move in this probabilistic map?
  - You periodically must turn probability into Occupied/Empty!
  - Use some threshold to decide, e.g., \( P(\text{occupied}) > 0.8 \) and \( P(\text{empty}) < 0.2 \), rest is "unknown".
  - Then do frontier exploration and path planning as before on your deterministic map.

A Probabilistic OG Mapping Algorithm

1. Initialize Grid to 0.5
2. Update the Grid
   - Mark your current position as high probability "empty"
   - Use your sensor model and Bayes rule to update grid
3. Pick a Next Move
   - Threshold your map into empty, occupied
   - Identify frontier nodes, and pick one
   - Plan a path to the clear node nearest
   - Go to that location and update position
4. Loop until no frontier nodes are left

Improvement 3: Motion isn’t perfect either!
Maybe you are not where you think you are!
And you are just messing up your grid over time due to drift

Recall: Probabilistic Localization...

- Probabilistic Localization
  - \( P(x_t | Z_0, U_0, \text{map}) \)
  - Where am I? Given that I took the noisy actions \( U \) and noisy observations \( Z \) of things in my perfect map.
- Kalman Filter (observed known landmarks)
- Particle Filter (match with known map)

Improvement 3:
Motion isn’t perfect either!
Maybe you are not where you think you are!
And you are just messing up your grid over time due to drift
Probabilistic Localization and Mapping

- **Probabilistic Localization**
  - \( P(x_t | Z_0 - t, U_0 - t, \text{map}) \)
  - Where am I? Given that I took the noisy actions \( U \) and noisy observations \( Z \) of things in my perfect map.

- **Probabilistic Mapping**
  - \( P(\text{map} | Z_0 - t, U_0 - t) \)
  - What is my map like? Given that I made noisy observations \( Z \) as I walked along my perfect path dictated by \( U \).

1 lecture ago:
- Bayesian Filters
- Particle Filters

Today:
- Bayesian Occupancy Grids

Probabilistic Localization and Mapping

- **Probabilistic SLAM ("Simultaneous")**
  - \( P(x_t, \text{map} | Z_0 - t, U_0 - t) \)
  - Where am I and what is my map?
  - Given noisy actions \( U \) and made noisy observations \( Z \)
  - Distribution of a huge space! (all possible positions and maps)

Many Methods
- EKF-SLAM (Kalman Filter) and Fast-SLAM (Particle Filters/OG)

In original EKF,
- State == robot position, represented as a Gaussian \( \{x, \sigma\} \)

In EKF-SLAM,
- State \( = \) [robot and all landmark positions as Gaussians]
- Position \( X_t = \{x_t, m_1, m_2, m_3, \ldots \} \) (number of landmarks grows!)
- Co-variance \( \sigma = (n+1) \times (n+1) \) matrix (uncertainty is correlated?)
- Supply a motion model and observation model as before (Gaussian)

Interesting factors
- Number of landmarks \( n \) grows with time (i.e. you build a map).
- But good news: Landmark correlations can help you converge faster and better.
Extended Kalman Filter SLAM

- Let's say EKF-SLAM State at time $t$ is
  - Position $X = \{x, m_1, m_2, m_3, m_4\}$ (robot + landmarks-so-far)
  - Covariance $\sigma = 5 \times 5$ matrix (uncertainty and correlations)
- Basic Procedure: Four Steps (Repeat)
  1. Motion Step: Update $P(x_t, \text{map} | Z_{0-t} U_{0-t})$ based on action $U_t$
  2. Observation Step: Update $P(x_t, \text{map} | Z_{t} U_{0-t})$ based on $Z_t$
  3. Combine into Single Estimate
     - Data Association: Determine which landmarks are re-observed* (let's say $m_2, m_3$)
     - Your motion state estimate $= x_t, m_2', m_3'$ (where you expect to see these landmarks)
     - Kalman Gain: Compute relative confidence and combine estimates
     - Then update the whole map ($m_1$-$m_4$), thanks to co-variance matrix
  4. Add Landmarks: Add New landmarks to the State (say $m_5$)
- Important – implementing Data Association and landmark choice!

More About SLAM

- Data Association and Loop Closure
  - We don’t really have perfect landmarks
  - Instead we have lasercan “features” (e.g. major corners)
  - Tradeoff: Uniqueness and frequency
  - Local matching is easier than long term matching
  - Can do loop closure with human assistance.
- Practical Implementations
  - These algorithms are theoretically well-grounded
  - But practical implementation still requires significant work (e.g. constructing sensor/motion models, choosing landmarks.)
- References (online)
  - SLAM for Dummies, Riisgaard et al 2005 (practice)
  - Gmapping in ROS! (PRR chapter 9 = offline map making)

Conclude: Robots Navigating the World

Second Part of CS189: High-level reasoning
From finite state machines to complex representation and memory

PathPlanning
- Visual Homing
- Forward Kinematics (direct methods)
- Bug Algorithms (obstacles)
- A* Algorithm (maps)

Localization
- Dead-Reckoning (using internal motion)
- Landmarks (using external sensing)
- Kalman Filter
- Particle Filters (combine uncertain motion and sensing)

Mapping/Exploration
- Occupancy Grid Mapping
- Sensor models
- Frontier Exploration (faster mapping)
- Bayesian Mapping
- SLAM (uncertainty in maps and location)

Conclude: Robots Navigating the World

Second Part of CS189: High-level reasoning
From finite state machines to complex representation and memory

PathPlanning
- Visual Homing
- Forward Kinematics (direct methods)
- Bug Algorithms (obstacles)
- A* Algorithm (maps)

Localization
- Dead-Reckoning (using internal motion)
- Landmarks (using external sensing)
- Kalman Filter (ROS pkg)
- Particle Filters (combine uncertain motion and sensing)

Mapping/Exploration
- Occupancy Grid Mapping
- Sensor models
- Frontier Exploration (faster mapping)
- Bayesian Mapping
- SLAM (ROS pkg: Gmapping) (uncertainty in maps and location)