ME 456: Probabilistic Robotics

Week 5, Lecture 2
SLAM

Reading: Chapters 10, 13

HW 2 due Oct 30, 11:59 PM

Introduction

• In state estimation and Bayes filter lectures, we showed how to find robot’s pose based on a known map
• In mapping lecture, we showed how to find a map of the environment based on known robot poses
• Finding a map when we are localized is easy. Localizing when we have a map is easy. What is hard is doing both localization and finding a map at the same time.
  • This is the problem we most commonly encounter in the real world!
The SLAM Problem

- A robot is exploring an unknown, static environment

- Given:
  - The robot’s controls
  - Observations of nearby features

- Estimate:
  - Map of features
  - Path of the robot

SLAM Applications

- We often can’t localize in these types of environments with things like GPS, so we need SLAM
Representations

- Grid maps or scans

- Landmark Based

Why is SLAM a Hard Problem?

SLAM: robot path and map are both unknown

Robot path error results in errors in map
Why is SLAM a Hard Problem?

- In the real world, the mapping between observations and landmarks is unknown
- Picking wrong data associations can have catastrophic consequences
- Pose error correlates data associations

SLAM: Simultaneous Localization and Mapping

- Full SLAM:
  \[ p(x_{1:t}, m \mid z_{1:t}, u_{1:t}) \]
  Estimates entire path and map!
- Online SLAM:
  \[ p(x_t, m \mid z_{1:t}, u_{1:t}) = \int \ldots \int p(x_{1:t}, m \mid z_{1:t}, u_{1:t}) \, dx_1 \, dx_2 \ldots dx_{t-1} \]
  Estimates most recent pose and map!

Integrations typically done one at a time
Graphical Model of Online SLAM

\[ p(x_t, m \mid z_{t \leq T}, u_{t \leq T}) = \int \ldots \int p(x_{t \leq T}, m \mid z_{t \leq T}, u_{t \leq T}) \, dx_t \ldots dx_{t-1} \]

Graphical Model of Full SLAM

\[ p(x_{1T}, m \mid z_{1T}, u_{1T}) \]
Techniques for Generating Consistent Maps

- Scan matching
- Fast-SLAM
- EKF SLAM
- Probabilistic mapping with a single map and a posterior about poses
  Mapping + Localization
- Graph-SLAM, SEIFs

Fast-SLAM

- Particle-filter based SLAM technique
- Landmark based mapping
- Montemerio, CMU 1995
Fast-SLAM: Assumptions

- Known correspondence between obstacles and sensor readings (we can determine the range and heading to the obstacles)
- The obstacles are fairly far apart (relatively sparse)
- Features do not move (static world)
- Conditional independence of obstacle positions and detections (we are not more or less likely to detect something because of where it is). This allows us to use Rao-Blackwellised particle filters (faster converging particle filters)

Particle Filters

- Discrete approximation of PDF as samples
- Sample from these using importance sampling
- Update the samples based on movement of the robot (pose) and sensing (probability)
- When we move, we update our pose (based on odometry) and add a little noise to it. We modify the probability of being where we think we are based on what we sense
- This method assumes a map in the background is static
Extended Kalman Filter (EKF)

- State estimator
- Bayes filter which represents uncertainty using a Gaussian
- Want to track a parameter $x$. We maintain a Gaussian estimate of $x$.
- We describe $x$ with the mean $\mu$ and covariance $\Sigma$. $\mu$ is an $N$-dimensional vector, and $\Sigma$ is an $N \times N$ matrix
- State transition function needs to be linear
- Observation must be a linear function of the state
- Initial state estimate is a Gaussian distribution
- A set of rules to update the estimates of the state
- These rules provably give you the optimal estimate of the state
- If you have a nonlinear system, we use the extended Kalman filter. Then, we linearize the system at each time step and apply the standard Kalman filter

Landmark Based Mapping

- Suppose the environment consists of a set of isolated landmarks
  - Trees in the forest
  - Rocks in the Martian desert
- Treat a landmark as a point location $(x_k, y_k)$
- SLAM: the robot learns the locations of the landmarks while localizing itself
Structure of the Landmark based SLAM Problem

Landmarks vs. Occupancy Grids

- An occupancy grid makes no assumption about types of features
  - Now we assume point landmarks, but walls and other features are possible
- An occupancy grid (typically) has a fixed resolution
  - Feature models can be arbitrarily precise
- An occupancy grid takes space and time relative to the size of the environment to be mapped
  - A feature-based map takes space and time reflecting the contents of the environment
FastSLAM Example

Fast-SLAM Overview

- Use particles to represent:
  - pose of robot, given history
  - Our map (equivalent to feature locations)

- Particles look like:
  \[
  S_t^{[M]} = \langle s_t^{[M]_1}, \mu_{1,t}^{[M]}, \Sigma_{1,t}^{[M]}, \ldots, s_t^{[M]_N}, \mu_{N,t}^{[M]}, \Sigma_{N,t}^{[M]} \rangle
  \]

- \( t \) is the time step, \( M \) is the particle, \( s_t^{[M]} \) is the path to time \( t \) in the map corresponding to particle \( M \)

- The tuple \( (\mu_{t}^{[M]}, \Sigma_{t}^{[M]}) \) is feature \( i \)'s EKF estimate at time \( t \), with respect to map \( M \)

- The path to get to time \( t \) is given by
  \[
  s_t^{[M]} = \{(x_1, y_1, \theta_1), \ldots, (x_t, y_t, \theta_t)\}
Fast-SLAM Overview

The mean estimate is given by:

\[ \mu_{1,t}^{[M]} = (r, \theta) \]

And \( \Sigma_{1,t}^{[M]} \) is a 2x2 covariance matrix

So, our particle \( S_i^{[M]} \) contains \( 6N + 3t \) numbers

\( S_i^{[M]} \) is what goes in to the particle filter

Particles in Fast-SLAM

<table>
<thead>
<tr>
<th>Particle ( k )</th>
<th>robot path</th>
<th>feature 1</th>
<th>feature 2</th>
<th>\ldots</th>
<th>feature ( N )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k = 1 )</td>
<td>( x_1^{[1]} = (x_1, y_1, \theta)^T )</td>
<td>( \mu_1^{[1]}, \Sigma_1^{[1]} )</td>
<td>( \mu_2^{[1]}, \Sigma_2^{[1]} )</td>
<td>\ldots</td>
<td>( \mu_N^{[1]}, \Sigma_N^{[1]} )</td>
</tr>
<tr>
<td>( k = 2 )</td>
<td>( x_1^{[2]} = (x_2, y_2, \theta)^T )</td>
<td>( \mu_1^{[2]}, \Sigma_1^{[2]} )</td>
<td>( \mu_2^{[2]}, \Sigma_2^{[2]} )</td>
<td>\ldots</td>
<td>( \mu_N^{[2]}, \Sigma_N^{[2]} )</td>
</tr>
<tr>
<td>\ldots</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( k = M )</td>
<td>( x_1^{[M]} = (x_M, y_M, \theta)^T )</td>
<td>( \mu_1^{[M]}, \Sigma_1^{[M]} )</td>
<td>( \mu_2^{[M]}, \Sigma_2^{[M]} )</td>
<td>\ldots</td>
<td>( \mu_N^{[M]}, \Sigma_N^{[M]} )</td>
</tr>
</tbody>
</table>
Fast-SLAM

- Do the following $M$ times:
  - **Retrieval.** Retrieve a pose $x_{t-1}^{[k]}$ from the particle set $Y_{t-1}$.
  - **Prediction.** Sample a new pose $x_{t}^{[k]} \sim p(x_{t} \mid x_{t-1}^{[k]}, u_{t})$.
  - **Measurement update.** For each observed feature $z_{t}^{j}$ identify the correspondence $j$ for the measurement $z_{t}^{j}$, and incorporate the measurement $z_{t}^{j}$ into the corresponding EKF, by updating the mean $\mu_{t}^{[k]}$ and covariance $\Sigma_{t}^{[k]}$.
  - **Importance weight.** Calculate the importance weight $w^{[k]}$ for the new particle.
- **Resampling.** Sample, with replacement, $M$ particles, where each particle is sampled with a probability proportional to $w^{[k]}$.

**Fast-SLAM: Basic Algorithm**

1) Sample a particle $S_{t-1}^{[M]}$ from $S_{t-1}$
   - with importance sampling
2) Sample a new pose
   - $S_{t}^{[M]} = P(s_{t}^{[M]} \mid S_{t-1}^{[M]}, u_{t})$ (this is probabilistic forward model of system)
3) For each feature in $S_{t}^{[M]}
   - Calculate bearing and range with sensors
   - Update EKF estimates of position (w.r.t. $M$)
4) Calculate importance weight $w^{[M]}$ of $S_{t}^{[M]}
5) Resample, according to $w^{[M]}$, to get the set $S_{t}$
Notes on Fast-SLAM

• Step 3 (update position estimates): 2 cases
  - Landmark is seen: update EKF as usual
  - Landmark is not seen: use old estimate
  - Do adjustment for new pose after EKF updated

• Step 4 (assign importance weights)
  - If we see what we expect, the particle’s importance weight is improved, and we are more likely to sample that particle

A Practical Note

• SLAM becomes much more accurate when we close a loop.

• If we can identify that we have already been somewhere, we can use this knowledge to offset odometry errors.

• Much of the research in SLAM involves optimizing loop closure correction algorithms.
Another Practical Note

• In theory, FastSLAM should scale well:
  – $O(KM \log N)$
  – $N$ is the number of landmarks in the map
  – $K$ is the number of landmarks observed
  – $M$ is the number of particles
• But, in practice...

Why Doesn’t FastSLAM Scale?

• At each time step
  – $K$ features added to search tree
  – $MK$ landmarks are added to the FastSLAM tree

• Memory fragmentation:
  – In time, nearby features are separated in memory, so CPU cache miss rate goes up
  – For large maps, page fault rate will also increase

• So the problem is the memory hierarchy, due to failure of locality
Another Practical Note

• We have focused on SLAM in environments with point obstacles.

• What about other types of features such as walls?

• For point obstacles, we only need (x,y) positions.

• For other features, we need something else...

Feature Extraction: Hough Transform

• Feature extraction algorithm
• Finding parameterized shapes in image data
• Good for lines and circles, but exponentially expensive in the number of parameters
  – We are relatively constrained in the things we can detect
• Basic idea: we have two points a and b
  – An infinite number of lines go through a
  – An infinite number of lines go through b
  – Only ONE line goes through both a and b
  – The Hough transform finds this line
• If we have many points that are close to collinear, then the Hough transform will find this line
  – Points on a wall aren’t collinear b/c of sensor noise, extrusions such as outlets or fire alarms, etc.
SLAM with Many Features

• Basic FastSLAM keeps track of (x,y) positions of point obstacles.
• If we know we have point obstacles in addition to other features such as walls, we keep estimates of parameterized features rather than only points.
• The SLAM algorithm works the same as before, we simply change how we are representing the environment.
• We need to parameterize shapes because this is a memory-efficient way of storing a large number of features.

ROS Already Has SLAM...

• Why do we care about how it works?
ROS Already Has SLAM...

• Why do we care about how it works?

• Many different types of SLAM:
  – Full SLAM
  – Online SLAM
  – FastSLAM
  – Scan Matching
  – Grid Based Rao-Blackwellized Particle Filter SLAM
  – Graph Mapping
  – At least 10 different “flavors” of SLAM presented every year at robotics conferences

Differences Between SLAM Implementations

• Different algorithms have different assumptions: VERY important. You want to use the algorithm suited to your environment
• Tradeoff between computational complexity and performance
• Different SLAM algorithms are often tweaked to work well in a very specific type of environment
ROS Already Has SLAM...

- You need to have a basic understanding of the different implementations of SLAM to understand which one you should use
- For the problems analyzed in this class, FastSLAM is sufficient
- But, for larger environments or 3D environments, FastSLAM is far too slow to be implemented in practice

SLAM vs. HIMM

Recall HIMM

- Cut world into a grid
  - Values in grid are between 0 and 15
  - Represent probabilities between 0 and 1

- Move the robot
- Get sensor reading, s
- Update the grid:
  - All cells in zone 1, subtract 1
  - All cells in zone 2, add 3
  - Truncate at 0 and 15
- Repeat
HIMM - ManyDots.world

Simulated world

Map created with HIMM

SLAM - ManyDots.World

Simulated world

Map created with SLAM
Why is SLAM So Much Better?

- Localization errors
  - HIMM created with odometry localization, where error propagates over time. As localization errors grow, estimates of obstacle locations become inaccurate
  - SLAM constantly fixing localization errors
Why is SLAM So Much Better?

- Green and Red: Fast-SLAM and full SLAM localization estimate
- Blue: Odometry localization estimate