Why is SLAM Hard: Ambiguity

Where This is Going

• New DP-SLAM 2.0 algorithm
  – Fast maintenance of multiple map hypotheses
  – Linear run time in all relevant parameters
• New model of laser penetration
• Results:
  – Good asymptotics
    (mapping no more expensive than localization!)
  – Very accurate & detailed maps

Outline

• Digression: Tracking
  • Kalman Filter SLAM
  • Full map slam
  • DP-SLAM

Tracking

• Example: Radar
• Hidden state variable(s)
• Dynamic model
• Noisy observations
• Problem: Infer hidden variables

Tracking Algorithm Outline

• Inputs:
  – Initial state estimate
  – Motion model, observation model
• Main loop:
  – Project state estimate forward using motion model
  – Make observations
  – Update state estimate based on observations
**Tracking Example**

- Motion model
- Measurement
- Updated state

**State Representation**

- Assuming:
  - Gaussian initial state
  - Linear dynamics
  - Linear observation model
  - Gaussian noise
- Posterior remains Gaussian
- Closed form solution – Kalman Filter
- See page by Greg Welch and Gary Bishop

**Monte Carlo Approximation (Particle Filter)**

- Motion model
- Measurement
- Resample
- Updated state

**Particle Filter**

- No assumptions about
  - Motion, observation model
  - Form of density
- Simulate → Weight → Resample
- Samples (particles) are fixed in number
- Nota bene: Resampling allocates particles to highest probability areas
- Works well w/concentrated posterior

**Outline**

- Digression: Tracking
  - Kalman Filter SLAM
  - Full map SLAM
  - DP-SLAM

**SLAM as Tracking**

- Hidden state:
  - Robot position
  - Position of distinctive landmarks
- Motion model:
  - Robot control input
  - Landmarks are stationary
- Observations:
  - Measured distances from landmarks
SLAM Pseudocode

- Project robot state distribution forward (robot motion model)
- Observe environment (laser scans)
- Update robot state by P(O|S)
- Update map (add new objects)
- Repeat

Kalman Filter SLAM Properties

- Assumes:
  - Linear motion model
  - Gaussian noise
- Produces
  - Robot position estimates
  - Landmark position estimates
  - Means and full covariance matrix

KF SLAM Example

Video courtesy of Mark Paskin

Problems with KF SLAM

- Reality is not linear Gaussian
- Produces only a map of landmarks
- $n$ landmarks: $O(n^2)$ cost
- Data association problem

Fixes for KF SLAM

- Thin junction tree filters (Paskin)
  - Uses approximate Bayes net inference techniques
  - Fast, adaptive approximation
- FastSLAM (Montemerlo et al.)
  - Sampling robot positions
  - KF for landmark positions
  - Benefits of sampling:
    - Fixes unrealistic linear-Gaussian assumption
    - Landmark positions become independent
    - Linear cost in no. of landmarks seen

Outline

- Digression: Tracking
- Kalman Filter SLAM
- Full map SLAM
- DP-SLAM
Building a Dense Map

Challenges of Full Maps

- Dense concentration of features
  - Makes KF impractical
  - Complicates data association
- Naive approaches fail
  - Ignoring uncertainty = accumulating error in maps
  - Confronting uncertainty = computational problems (solved by DP-SLAM)

Ignoring Uncertainty

- Use a very accurate sensor (laser)
- Maintain PF or KF over robot positions
- Deterministically update map
  - Estimate most likely robot position
  - Insert new observations into map
  - Hope for the best...

Single Map SLAM

Map Patching

- Exploit topology for consistent maps
  - Loop closing [Lu & Milos ’97, Gutmann & Konolige ‘00]
  - Consistency provides accuracy
- Heuristic map correction
- Good maps achieved at intervals
  - Intermediate maps can be poor
  - Removes *Simultaneous* from SLAM

Outline

- Digression: Tracking
- Kalman Filter SLAM
- Full map SLAM
- DP-SLAM
DP-SLAM Goals

- Best of both worlds
  - Soundness/robustness of probabilistic methods
  - Full map detail
- Speed/Efficiency
  - Linear in observation size
  - Linear in number of particles
  - Single pass over sensor data (no map patching required)
- Generality
  - No assumptions about the environment
- Accuracy
  - Full maps without accumulating error

Map Maintenance Challenges

- Want to filter entire, joint pose-map states
- Dense maps are big
- 100’s or 1000’s of particles are needed
- One full map per particle requires
  - $O(MP)$ work (resampling)
  - Gigabytes of memory movement
- Anecdotal reports: Tried, but impractical

Distributed Particle Mapping

- Exploit sampling/resampling steps of PF
  - Common ancestry = Redundant map sections
- History representation: Ancestry Tree
  - Leaves correspond to current particles
- New map Representation
  - Store multiple maps in a single grid

Ancestry Trees
Ancestry Trees

Ancestors with no children can be removed

Ancestry Trees

Ancestors with only one child can be merged
Ancestry Trees

- Maintain a minimal tree (improves complexity)
  - Exactly $P$ leaves
  - Branching factor at least 2
  - Depth no more than $P$
- Explicitly store the ancestry info
  - Node = Ancestor particle w/ unique ID
  - Stores parent link, grid squares updated (list)

Naïve Map Representation

- Map is an occupancy grid
- One full map per particle!

DP-Mapping

- Each ancestry node stores
  - “Vector” of updated grid squares
- Each grid square stores:
  - “Vector” of ancestry nodes that have updated the square
  - Associated observations
- Good News: Minimal redundancy
- Bad News: Sacrifices constant time access to map

SLAM Pseudocode

- Project robot state distribution forward (robot motion model)
- Observe environment (laser scans)
- Update robot state by $P(O|S)$
- Update map (add new objects)
- Repeat

Localization Complexity

- $P$ Particles
- Must compute $P(O|S)$ for each particle
- For each laser cast of the current particle
  - Trace laser cast through grid
  - For each grid square return map occupancy
  - Laser scan probability = $f$ (map occupancy)
  - Trivial for explicit map representation
  - Observation size $A$: $O(AP)$

DP-SLAM Localization

- For DP Maps
  - Must implicitly reconstruct each particle’s map
  - For each ancestor of current particle:
    - Check if node has updated current square
    - Return associated occupancy
  - Naïve Solution:
    - For each $P$ on frontier of ancestry tree: $O(P)$
    - For each square $A$ visited tracing laser casts: $O(A)$
    - For each node $D$ in ancestry tree on path from $P$ to root: $O(P)$
    - Check if node $D$ has updated $A$: $O(P)$
  - $O(AP^3)$ complexity
A Smarter Solution

- Assign particles sequential IDs
- Store observation vectors for each grid square as a balanced tree keyed on IDs

- Working smarter:
  - For each $P$ on frontier of ancestry tree: $O(P)$
  - For each square $A$ visited tracing laser casts: $O(A)$
  - Simultaneously traverse:
    - Ancestry from $P$ to root
    - Balanced tree of observations for $P$
- $O(AP^2)$ complexity

A Linear Solution

- For any iteration of the particle filter:
  - On the first visit to any grid square $A$: $O(A)$
    - Parse all current particles against observation data stored at $A$: $O(P)$
    - Cache result
  - On subsequent visits to $A$: $O(AP)$
    - Return cached result: $O(1)$
- $O(AP) + O(AP) = O(AP)$ complexity

Alternate view will be presented later!

Map Update Complexity

- Updates trickier than they seem at first
- Expensive part: Collapsing

- Q: How to bound collapsing cost?

Amortized Analysis

- $O(AP)$ new observations inserted at leaves
- Total path length in tree bounds total work done collapsing or deleting nodes
- $O(AP)$ amortized cost

Complexity Summary

- Localization: $O(AP)$
  - P particles check A grid squares
  - Lookups are cached
- Map Maintenance: $O(AP)$
- Cost for pure localization with P particles: $O(AP)$

Comparison with Naïve Approach

- Total Time : $O(AP)$
  - Compare to $O(MP)$
  - $M >> A$
  - Linear in observation size
  - Independent of map size
  - Asymptotically, mapping is no more expensive than localization for fixed $P$

A = Area observed
$P$ = Number of particles
$M$ = Map size
Single Map SLAM

DP-SLAM 1.0 Results

Run at real-time speed on 2.4GHz Pentium 4 at 10cm/s

Scale:
1 square = 3cm

Consistency

DP-SLAM 2.0

- Folds in algorithmic improvements
- Improved laser penetration model
- Models behavior of each square to laser
- More accurate
- Can be slower in practice, but recent improvements regain real time speed

Ask for details later

DP-SLAM 1.0 in a Noisy Domain

DP-SLAM 2.0
Caveats and Future Work

- Particle filters have limitations:
  - Still not as robust as Kalman Filter
  - Eventually, unlucky sampling will miss true state (we are working on reducing frequency/severity)
  - Can require LARGE number of particles in presence of high noise or ambiguity

- Extending to 3D (easier and harder)
- Alternate map representations

Conclusions

- Slam is a tracking problem
- Different approaches to tracking permit
  - Different map representations
  - Different uncertainty representations
- Good performance requires
  - Sound probabilistic inference procedures
  - Efficient data structures
  - Good modeling

- Our per particle mapping cost = localization cost
- Moral: *Algorithms and data structures still matter 😊*

Questions?

Linear Solution: Another View

- We build a local “map cache”
- Initialize O(P) local maps of size A
- For each visible grid square
  - Populate local maps from stored observations
- For each “interior” map
  - Push observations down to child maps

Building the Map Cache I

- Visible region O(A)
- Global Map
- Map Cache (one entry per node in ancestry tree) O(AP)
Building the Map Cache II

Why didn’t you use quad trees?

- **Answer 1**: Quad trees solve a different problem
  - Quad trees exploit homogeneity within a single map
  - DP-SLAM exploits homogeneity across maps

- **Answer 2**: Quad trees solve the wrong problem
  - Statistics collected for each grid square make grid squares for any map quite heterogeneous

- Quad trees might be useful as an approximation or to compress unseen regions

Laser Model

- Naïve models condition the probability of a scan on the number of squares penetrated
  
  Equal length scans
  
  One travels through six squares
  
  The other travels through nine squares

- Probability of laser penetration depends on distance traveled through a grid square

Laser Model

- **d** = tendency of environment to stop laser
- Consistency
  
  - Scale of map should not affect probabilities
  
  \[ P(x+y|d) = P(x|d) + (1-P(x|d))P(y|d) \]

Laser Model

- Exponential distribution satisfies desiderata
  
  \[ P(x|d) = 1 - e^{-x/d} \]

- Map updates
  
  - Mean of \( 1 - e^{-x/d} = d \)
  
  - Estimate \( d \approx \) distance observed/stops observed

- No effect on computational complexity

Why don’t you have drift?

- Robot moves slowly

- Most uncertainty is short lived
  
  - Distant scans have most uncertainty
  
  - Dense scanning of close areas resolves ambiguities

- For long-lived uncertainty
  
  - Hierarchical approach
  
  - High level treats low level maps as observations
  
  - High level treats map errors (translations and rotations) as motion model noise