Localization : Part II

November 18, 2008

Maps built using Parker’s DILab robots:

1st Floor Claxton

1st Floor, JICS Building, ORNL
Belief Representation

- a) Continuous map with *single hypothesis*

- b) Continuous map with *multiple hypotheses*

- d) Discretized map with probability distribution

- d) Discretized topological map with probability distribution
Belief Representation: Characteristics

• Continuous
  ➢ Precision bound by sensor data
  ➢ Typically single hypothesis pose estimate
  ➢ Lost when diverging (for single hypothesis)
  ➢ Compact representation and typically reasonable in processing power.

• Discrete
  ➢ Precision bound by resolution of discretization
  ➢ Typically multiple hypothesis pose estimate
  ➢ Never lost (when diverges from one cell, it converges to another cell)
  ➢ Important memory and processing power needed. (not the case for topological maps)
Single-hypothesis Belief – Continuous Line-Map

(a) robot position

(b) $(x, y, \theta)$
Single-hypothesis Belief – Grid and Topological Map
Grid-based Representation – Multi-Hypothesis

- Grid size around 20 cm$^2$.
- Clouds represent possible robot locations
- Darker coloring means higher probability

Courtesy of W. Burgard

Path of the robot

Belief states at positions 2, 3 and 4
Map Representation

1. Map precision vs. application
   - The precision of the map must match the precision with which the robot needs to achieve its goals.

2. Features precision vs. map precision
   - The precision of the map and the type of features represented must match the precision and data types returned by the robot’s sensors.

3. Precision vs. computational complexity
   - The complexity of the map representation has a direct impact on the computational complexity of reasoning about mapping, localization, and navigation

Two primary map choices:
- Continuous Representation
- Decomposition (Discretization)
Representation of the Environment

• Environment Representation
  - Continuous Metric $\rightarrow x, y, \theta$
  - Discrete Metric $\rightarrow$ metric grid
  - Discrete Topological $\rightarrow$ topological grid

• Environment Modeling
  - Raw sensor data, e.g. laser range data, grayscale images
    - large volume of data, low distinctiveness on the level of individual values
    - makes use of all acquired information
  - Low level features, e.g. line other geometric features
    - medium volume of data, average distinctiveness
    - filters out the useful information, still ambiguities
  - High level features, e.g. doors, a car, the Eiffel tower
    - low volume of data, high distinctiveness
    - filters out the useful information, few/no ambiguities, not enough information
Map Representation: Continuous Line-Based

a) Architecture map
b) Representation with set of infinite lines
Map Representation: Decomposition (1)

- Exact cell decomposition
Map Representation: Decomposition (2)

• Fixed cell decomposition
  ➢ Narrow passages can disappear
Map Representation: Decomposition (3)

- Adaptive cell decomposition (i.e., quadtree)
Map Representation: Decomposition (4)

- Fixed cell decomposition – Example with very small cells
Map Representation: Decomposition (5)

- Topological Decomposition
Map Representation: Decomposition (6)

- Topological Decomposition

node

Connectivity (arch)
Map Representation: Decomposition (7)

- Topological Decomposition

- ~ 400 m
- ~ 1 km
- ~ 200 m
- ~ 50 m
- ~ 10 m
State-of-the-Art: Current Challenges in Map Representation

- Real world is dynamic

- Perception is still a major challenge
  - *Error prone*
  - *Extraction of useful information difficult*

- Traversal of open space

- How to build up topology (boundaries of nodes)

- Sensor fusion

- …
Probabilistic, Map-Based Localization (1)

- Consider a mobile robot moving in a known environment.

- As it starts to move, say from a precisely known location, it might keep track of its location using odometry.

- However, after a certain movement the robot will get very uncertain about its position.
  - update using an observation of its environment.

- Odometric information leads to an estimate of the robot’s position, which can then be fused with the sensor observations to get the best possible update of the robot’s actual position.
Probabilistic, Map-Based Localization (2)

- **Action update:**
  
  - *action model Act:*
    
    $$s'_t = Act(o_t, s_{t-1})$$
    
    where $o_t$: Encoder Measurement, $s_{t-1}$: prior belief state
  
  - increases uncertainty

- **Perception update:**
  
  - *perception model See:*
    
    $$s_t = See(i_t, s'_t)$$
    
    where $i_t$: exteroceptive sensor inputs, $s'_t$: updated belief state
  
  - decreases uncertainty
The Five Steps for Map-Based Localization

1. Prediction based on previous estimate and odometry
2. Measurement prediction based on prediction and map
3. Observation with on-board sensors
4. Matching of observation and map
5. Estimation -> position update (posteriori position)
Two general approaches: **Markov** and **Kalman Filter** Localization

- **Markov localization**
  - Maintains *multiple estimates of robot position*
  - Localization can start from *any unknown position*
  - Can recover from ambiguous situations
  - However, to update the probability of all positions within the state space requires a discrete representation of the space (grid); if a fine grid is used (or many estimates are maintained), the computational and memory requirements can be large.

- **Kalman filter localization**
  - *Single estimate of robot position*
  - Requires *known starting position of robot*
  - Tracks the robot and can be very precise and efficient
  - However, if the uncertainty of the robot becomes too large (e.g. due collision with an object) the Kalman filter will fail and the robot becomes “lost”.
Two types of localization problems

• “Global” localization – figure out where the robot is, but we don’t know where the robot started
  
  ➢ Sometimes called the “hijacked robot problem”

• “Position tracking” – figure out where the robot is, given that we know where the robot started

The Markov / Monte Carlo Localization approach of Fox, et al (which is in Player/Stage) can address both problems
Markov Localization

- Markov localization uses an explicit, discrete representation for the probability of all positions in the state space.
  
  ➢ Later, we’ll talk about a more efficient version (called Monte Carlo localization) that randomly samples possible positions, instead of maintaining information about all positions.

- This is usually done by representing the environment by a grid or a topological graph with a finite number of possible states (positions).

- During each update, the probability for each state (element) of the entire space is updated.
Markov Localization

- Key idea: compute a probability distribution over all possible positions in the environment.
  
  - This probability distribution represents the likelihood that the robot is in a particular location.
Markov Localization makes use of Bayes Rule

- P(A): Probability that A is true.
  - e.g. \( p(r_t = l) \): probability that the robot \( r \) is at position \( l \) at time \( t \)

- We wish to compute the probability of each individual robot position given actions and sensor measures.

- P(A|B): Conditional probability of A given that we know B.
  - e.g. \( p(r_t = l | i_t) \): probability that the robot is at position \( l \) given the sensors input \( i_t \)

- Product rule:
  
  \[
  p(A \land B) = p(A | B)p(B) \\
  p(A \land B) = p(B | A)p(A)
  \]

- Bayes rule:
  
  \[
  p(A | B) = \frac{p(B | A)p(A)}{p(B)}
  \]
The “See” update step

• Bayes rule:

\[ p(A|B) = \frac{p(B|A)p(A)}{p(B)} \]

➢ “See” operation: Maps from a belief state and a sensor input to a refined belief state:

\[ p(l|i) = \frac{p(i|l)p(l)}{p(i)} \quad (5.21) \]

➢ \( p(l) \): belief state before perceptual update process

➢ \( p(i|l) \): probability we get measurement \( i \) when being at position \( l \)

   - To obtain this info: consult robot’s map and identify the probability of a certain sensor reading if the robot were at position \( l \)

➢ \( p(i) \): normalization factor so that sum over all \( l \) equals 1.

• We apply this operation to all possible robot positions, \( l \)
The “Act” update step

“Act” operation: Maps from a belief state (i.e., belief in robot being in some prior position) and an action (represented by $o_t$, which is the encoder measurement corresponding to an action) to a new belief state:

$$s'_t = \text{Act}(o_t, s_{t-1})$$

This operation sums over all possible ways in which the robot may have reached position $l$ at time $t$, from any possible prior position at time $t-1$. (Note that there can be more than 1 way to reach a given position, due to uncertainty in encoder measurement.)
The Markov Property

- These two updates (from prior 2 slides):

  ➢ “See”: \[ p(l_i | i_t) = \frac{p(i_l | l)p(l)}{p(i)} \]  
    (5.21)

  ➢ “Act”: \[ p(l_t | o_t) = \int p(l_t | l_{t-1}, o_t) p(l_{t-1}) dl_{t-1} \]  
    (5.22)

constitute the Markov assumption. That is, the current update only depends on the previous state \((l_t)\) and its most recent action \((o_t)\) and perception \((i_t)\).

The Markov assumption may not be true, but it greatly simplifies tracking, reasoning, and planning, so it is a common approximation in robotics/AI.
Markov Localization: Case Study – Grid Map

- Fine _fixed decomposition_ grid \((x, y, \theta)\), 15 cm x 15 cm x 1°

- Action update:
  - **Sum over previous possible positions and motion model:**
    \[
    P(l_t|o_t) = \sum_{l'} P(l_t|l'_{t-1}, o_t) \cdot p(l'_{t-1})
    \]
  - (this is discrete version of eqn. 5.22)

- Perception update:
  - **Given perception i, what is the probability of being in location l:**
    \[
    p(l|i) = \frac{p(i|l)p(l)}{p(i)}
    \]
Perception update details

- The critical challenge is the calculation of $p(i|l)$
  - $p(i|l)$ is computed using a model of the robot’s sensor behavior, its position $l$, and the local environment metric map around $l$.
  - Assumptions:
    - Measurement error can be described by a distribution with a mean at the correct reading
    - Non-zero chance for any measurement
    - Local peak at maximal reading of range sensor (due to absorption/reflection)

\[
p(l|i) = \frac{p(i|l)p(l)}{p(i)}
\]
Markov Localization: General idea for maintaining multiple estimates of robot position

• The 1D case

1. Start
   - No knowledge at start, thus we have a uniform probability distribution.

2. Robot perceives first pillar
   - Seeing only one pillar, the probability of being at pillar 1, 2 or 3 is equal.

3. Robot moves
   - Action model enables estimation of the new probability distribution based on the previous one and the motion.

4. Robot perceives second pillar
   - Based on all prior knowledge, the probability of being at pillar 2 becomes dominant
Markov Localization: Example

- Example 1: Office Building

Courtesy of W. Burgard
Markov Localization: Example #2

- Example 2: Museum
  - Laser scan 1

Courtesy of W. Burgard

© R. Siegwart, I. Nourbakhsh
Markov Localization: Example #2

- Example 2: Museum
  - Laser scan 2
Markov Localization: Example #2

• Example 2: Museum
  ➢ Laser scan 3

Courtesy of W. Burgard

© R. Siegwart, I. Nourbakhsh
Markov Localization: Example #2

- Example 2: Museum
  - Laser scan 13

Courtesy of W. Burgard
Markov Localization: Example #2

- Example 2: Museum
  - Laser scan 21

© R. Siegwart, I. Nourbakhsh
Markov Localization: Using Randomized Sampling to Reduce Complexity  ➔ “Particle filters”, “Monte Carlo algorithms”

- Fine *fixed decomposition* grid results in a huge state space
- Reducing complexity:
  - The main goal is to reduce the number of states that are updated in each step
- Randomized Sampling / Particle Filters
  - Approximated belief state by representing only a ‘representative’ subset of all states (possible locations)
  - E.g., update only 10% of all possible locations
  - The sampling process is typically weighted, e.g., put more samples around the local peaks in the probability density function
  - However, you have to ensure some less likely locations are still tracked, otherwise the robot might get lost

© R. Siegwart, I. Nourbakhsh
Updating beliefs using Monte Carlo Localization (MCL)

• As before, 2 models: Action Model, Perception Model

• Robot Action Model:
  ➢ When robot moves, MCL generates $N$ new samples that approximate robot’s position after motion command.
  ➢ Each sample is generated by randomly drawing from previous sample set, with likelihood determined by $p$ values.
  ➢ For sample drawn with position $l'$, new sample $l$ is generated from $P(l | l', a)$, for action $a$
  ➢ $p$ value of new sample is $1/N$

Sampling-based approximation of position belief for non-sensing robot

(From Fox, et al, AAAI-99)
Updating beliefs using Monte Carlo Localization (MCL), con’t.

- Robot Perception Model:
  - Re-weight sample set according to the likelihood that robot’s current sensors match what would be seen at a given location

- After applying Motion model and Perception model:
  - Resample, according to latest weights
  - Add a few uniformly distributed, random samples
    - Very helpful in case robot completely loses track of its location
  - Go to next iteration
Example Results

Initially, robot doesn’t know where it is
(see particles representing possible robot locations
distributed throughout the environment)

After robot moves some, it gets better estimate
(see particles clustered in a few areas, with a
couple of random particles also distributed around for
robustness)
Adapting the Size of the Sample Set

- Number of samples needed to achieve a desired level of accuracy varies dramatically depending on the situation
  - During global localization: robot is ignorant of where it is → need lots of samples
  - During position tracking: robot’s uncertainty is small → don’t need as many samples

- MCL determines sample size “on the fly”
  - Compare $P(l)$ and $P(l \mid s)$ (i.e., belief before and after sensing) to determine sample size
  - The more divergence, the more samples that are kept
Player/Stage Localization Approach: Monte Carlo Localization

- Based on techniques developed by Fox, Burgard, Dellaert, Thrun (see handout of AAAI’99 article)
More movies

• Dieter Fox movie: MCL using Sonar

Global localization with sonar sensors

• Dieter Fox movie: MCL using Laser

Global localization using a laser range-finder
Summarizing the process: Particle Filtering

<table>
<thead>
<tr>
<th>$S_{t-1}$</th>
<th>$S'_t$</th>
<th>weighted $S'_t$</th>
<th>$S_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Predict" /></td>
<td><img src="image2" alt="ReWeight" /></td>
<td><img src="image3" alt="Resample" /></td>
<td><img src="image4" alt="Predict" /></td>
</tr>
</tbody>
</table>

Slide adapted from Dellaert presentation “19-Particles.ppt”
What sensor to use for localization?

• Can work with:
  ➢ *Sonar*
  ➢ *Laser*
  ➢ *Vision*
  ➢ *Radio signal strength*
The Player/Stage “Localize” Proxy

- Allows you to localize the robot using its sensors (e.g., laser or sonar, or even radio strength of signal)

Public Member Functions:

- `LocalizeProxy` *(PlayerClient* `*aPc, uint aIndex=0)*  
  constructor
- `~LocalizeProxy` ()  
  destructor
- `uint GetMapSizeX () const`  
  Map dimensions (cells).
- `uint GetMapSizeY () const`
- `uint GetMapTileX () const`
- `uint GetMapTileY () const`
- `double GetMapScale () const`  
  Map scale (m/cell).
- `uint GetPendingCount () const`  
  # of pending (unprocessed) sensor readings.
- `uint GetHypothCount () const`  
  Number of possible poses.
- `player_localize_hypoth_t GetHypoth (uint aIndex) const`  
  Array of possible poses.
- `int GetParticles ()`  
  Get the particle set.
- `player_pose_t GetParticlePose (int index) const`  
  Get the Pose of a particle.
- `void SetPose (double pose[3], double cov[3])`  
  Set the current pose hypothesis (m, m, radians).
- `uint GetNumHypotheses () const`  
  Get the number of localization hypotheses
- `uint GetNumParticles () const`  
  Get # of particles (for particle filter-based localization systems).