Statistical Techniques in Robotics (16-831, F11) Lecture#20 (November 21, 2011)

Gaussian Processes, SLAM, Fast SLAM and Rao-Blackwellization

Lecturer: Drew Bagnell *Scribes: Junier Oliva* [1]

1 Comments on Gaussian Processes

1.1 GP vs. MRF

- GP is like a MRF (with Gaussian RVs) with a continuous number of RVs; that is, it is extendable to an infinite number of RVs).

1.2 Precision Matrix

- GP cannot be parameterized using precision matrix P because submatrices of P do no correspond to the precision matrix of that subset of elements.

  I.e. if P is precision matrix for $\{a, b\}$ then the precision matrix for $\{a, b, c, d, \dots\}$ is not

$$\begin{bmatrix} \begin{bmatrix} \mathrm{P} \end{bmatrix} & \dots \\ \vdots & \ddots \end{bmatrix}$$

2 About SLAM

Simultaneous localization and mapping (SLAM) is one of the canonical problems in robotics. A robot set down in an unknown environment with noisy odometry and sensing needs to figure out where it is in the world and learn details about the world at the same time. It is a sort of chicken-egg problem where the mapping problem is relatively easy given exact knowledge of the robot's location, and localizing the robot is relatively easy given a good map of the environment. The challenge is found in performing these tasks simultaneously.

2.1 Variations on the SLAM problem

SLAM can be approached in two ways:

- Batch: Given all data, $y_{1:T}$, up front, solve for robot path $x_{1:T}$ and map M all at once (we won't cover this)
  - This approach can be thought of as a smoothing operation on our data.

[1] Some content adapted from previous scribes: Justin Haines, Neal Seegmiller, and Bryan Wagenknecht

- Online/filtering: compute $\{x_t, m_t\}$ at each timestep as data y_t is received (we focus on this)
  - This approach can be thought of as a filtering problem.
  - Note: this approch will produce a best guess for the only x_t, m_t not the entire trajectory
  - If, alternatively, you are interested in the robot trajectory, calculate $\{x_{1:t}, m_t\}$ at each timestep

The mapping part of SLAM can also be approached in two ways:

- Filling an occupancy grid $M = \{m_1, m_2, ...\}$ (we'll come back to this)
- Tracking landmarks $l_i = (l_{xi}, l_{yi})$ (focus on this for now)

3 Basic Approaches to SLAM

3.1 Naïve approach: PF over joint

The first potential solution is to build a Particle Filter to sample from the joint distribution

$$p(r_x, r_y, l_{x1}, l_{y1}, l_{x2}, l_{y2}, ...)$$

This becomes bad very quickly due to the high dimensionality of the state space when landmark position is included (this example has dimension $2 + 2 * l$.)

- PF doesn't scale well with dimensionality - particle sparsity means a particle that is good at describing one landmark is probably bad at describing others; thus, it will need too many particles
- Observation distributions/densities get very sharp due to multiplying distributions

3.2 X-KF over joint

The next potential solution (often called the "classical solution") might be to use your favorite Kalman Filter (X-KF here could mean EKF, UKF, etc.) to track all variables in one joint distribution

$$p(r_x, r_y, l_{x1}, l_{y1}, l_{x2}, l_{y2}, ...)$$

This results in a big covariance matrix with the marginal gaussian distribution of each landmark contained in blocks along the covariance diagonal. Off-diagonal terms are correlations from multiple landmarks appearing in the robot's view at once.

The motion update would look like:

$$\begin{bmatrix} r_x \\ r_y \\ l_{x1} \\ l_{y1} \\ \vdots \end{bmatrix}_{t+1} = \begin{bmatrix} \begin{bmatrix} A_r \end{bmatrix}_{2x2} & 0 \\ 0 & \begin{bmatrix} I \end{bmatrix} \end{bmatrix} \begin{bmatrix} r_x \\ r_y \\ l_{x1} \\ l_{y1} \\ \vdots \end{bmatrix}_t + \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \\ 0 \\ \end{bmatrix}$$

where A_r is the Jacobian of the motion model of the robot and ϵ_x and ϵ_y are gaussian noise. The identity block represents the assumption that landmarks do not move over time.

The observation model would look like:

$$\begin{bmatrix} range_i \\ bearing_i \end{bmatrix} = \begin{bmatrix} \sqrt{(r_x - l_{xi})^2 + (r_y - l_{yi})^2} \\ atan2((r_y - l_{yi}), (r_x - l_{xi})) \end{bmatrix} + \begin{bmatrix} \epsilon_{range} \\ \epsilon_{bearing} \end{bmatrix}$$

KEY ASSUMPTION: The robot must be able to differentiate landmarks

3.2.1 Potential problems with this approach

- The solution could be multimodal, especially with bad odometry
- Data association: If the robot cannot distinguish between landmarks, problems can arise with "loop closure" where the robot returns to a location it has already mapped. In this case, the robot needs to match the landmarks with those previously observed.

  Maximum Likelihood data association with hard thresholds is often used for this
- Scales as $O(l^2)$ where l is number of landmarks

  Although this is better than the complexity of the particle filter SLAM
- What to use for initial landmark location, μ_{x0}? Any guess could be arbitrarily wrong and take a long time to correct

3.2.2 Some hacks that could be helpful

- Use submaps (segment the large world map into smaller maps) and work with smaller chunks of the covariance matrix.
- When a landmark is seen for the first time, initialize its μ_x to the observed location, or wait until seen a couple of times and triangularize location
- Throw out confusing landmarks

3.3 Alternative: Information Filter for SLAM

- Information filter implementation is $O(l^3)$ in the number of landmarks
- However, the sparse nature of the lxl matrix inversion allows the algorithm run in closer to $O(l \log(l))$

4 SLAM as a Bayes Net

If we draw the Bayes Network representation of SLAM we get something like Figure 1. Here the data associations are presumed to be known and fixed. The main intuition here is that landmarks are correlated only by observations through the path of the robot.

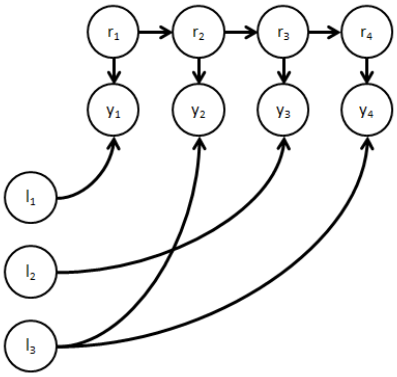

Figure 1: Bayes Network representation of SLAM. Robot states r_i form a Markov chain and data associations (which landmarks l_i are in view at each observation y_i) are presumed known and fixed. Observe that the landmark locations are conditionally independent given the robot states.

4.1 Fast SLAM factorization

Investigation of the Bayes Net shows us that we can make the landmark locations independent by conditioning on the robot locations. This allows us to factorize the distribution as

$$p(y_{1:t}|z_{1:t}, u_{1:t}) = p(x_{1:t}|z_{1:t}, u_{1:t}) \cdot \prod_i p(l_i|x_{1:t}, z_{1:t})$$

Thus we can track the landmark locations independently by conditioning on the assumed robot position.

Fast SLAM is an algorithm that runs a particle filter in robot location space using augmented particles. Each particle carries a history of its past locations $r_1 ... r_t$ and an individual Kalman filter (consisting of μ_{l_i} and Σ_{l_i}) for each landmark l_i it sees. This is visualized in Figure 2.

5 Smarter Approach: Fast SLAM

5.1 Fast SLAM 1.0

The first version of Fast SLAM runs a particle filter with augmented particles, then applies importance weighting to the particles using the latest estimates of any landmarks within view. The landmarks are tracked independently in Kalman filters within each particle.

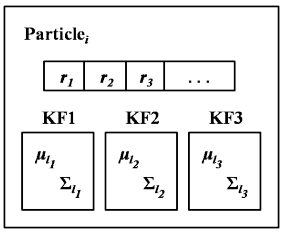

Figure 2: Fast SLAM particles include a history of their positions and individual Kalman filters for each landmark.

5.1.1 Fast SLAM 1.0 Algorithm

1. Motion update: for all $particle_i$, move r_i according to motion model (leave KF's for landmarks alone)
$$r_t \sim p(r_t|r_{t-1}, u_t)$$
2. Observation update: weight $particle_i$ by $p(y|associations\, l, r_i) \sim N(\mu_y, \Sigma_{yy})$
$$p(y|associations\, l, r_i) = \frac{1}{\sqrt{(2\pi)^2\,|\Sigma_{yy}|}} \exp\left[-\frac{1}{2}(y-\mu_y)^T\Sigma_{yy}(y-\mu_y)\right]$$
3. Run KF update on each particle's estimate of landmark l
4. Resample particles

5.1.2 Timestep Complexity: $O(l)$

- Data association is $O(l)$ (use KD tree)
- Memory copy is $O(l)$

5.1.3 Number of parameters:

- $\#particles \cdot [(2+3) \cdot \#landmarks + 2]$; (2+3) for μ, Σ, last 2 for pose.

5.1.4 Occupancy Grids

- In theory each particle should carry its own map grid (instead of KF's)
- Practically this is too memory intensive, use data structure tricks: DP-SLAM

  Use a single map grid with data for all relevant particles in each cell

5.2 Fast SLAM 2.0

The problem with Fast SLAM is paricle deprivation, especially for loop closure. The key is to have a smarter motion model (sampling distribution). The Fast SLAM 2.0 motion model is conditioned on the observation z_t in addition to the control, u_t.

5.2.1 Fast SLAM 2.0 Algorithm

1. Motion update: Sample robot position from Kalman filter $\tilde{p}(x_{t+1}|y_{t+1}, x_t, z_t)$
2. Observation update: Weight $particle_i$ by

$$w_i \propto \frac{p(y_{t+1}|associations\, l, r_i)}{\tilde{p}(x_{t+1}|y_{t+1}, x_t)}$$

6 Wrap-up Topics

6.1 Data association

If we want to be lazy with data association (or we can't label the landmarks a priori) then we can track associations within the particle. Wrong assignments are corrected in retrospect for free through particle resampling.

- For each landmark l_i, sample associations from $p(y|l_i, r)$

6.2 Rao-Blackwell theorem

The Rao-Blackwell theorem says running an algorithm that samples from a distribution on one variable $p(x_1)$ then determines another variable x_2 analytically conditioned on x_1 will do no worse than an algorithm that samples from the joint distribution $p(x_1, x_2)$.

$$var\left[sampler(x_1, x_2)\right] \geq var\left[sampler(x_1) + does\, x_2|x_1\, analytically\right]$$

Fast SLAM is an example of Rao-Blackwellization since by using KF's inside particles, we don't have to sample landmark info from the whole joint distribution. In general Rao-Blackwellized filters make for the most efficient SLAM algorithms.

7 Occupancy Grid SLAM

7.1 DP SLAM

Murphy initially proposed Fast SLAM using occupancy grid maps[2]. In the simplest implementation, a complete map is stored for each particle; however, this is too memory intensive (and computa-

[2]Kevin Murphy, "Bayesian Map Learning in Dynamic Environments," NIPS 1999. (Neural Info. Proc. Systems)

tionally expensive to copy the maps). Researchers at Duke University proposed DP SLAM which uses dynamic programming to reduce the memory/computational requirements[3].

The basic idea: If two particles share a common ancestor, both their maps will agree with the observations of that common ancestor. A single occupancy grid map is stored with an observation tree at each cell (storing the observations of all particles that affect that cell).

Efficient maintenance of this data structure is non-trivial.

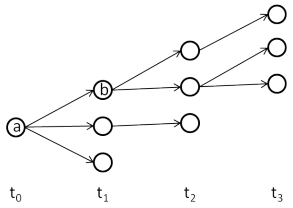

Figure 3: Example of particle ancestry for DP SLAM. Note that all particles at t_3 are descended from the root a as well as b at t_1. All particles at t_3 will share the observations of b and its predecessors, so these sections of the map need only be stored once.

7.1.1 Loss of diversity

Ideally, in order to close loops diversity must be maintained on the order of the largest loop that will ever be traversed. However, DP SLAM typically only maintains diversity on the order of 2-3 meters (i.e. all current particles are descended from a single particle 2-3 meters back). Despite this, DP SLAM still works due to high localization accuracy.

7.2 3D Fast SLAM

Nathaniel Fairfield's research on DEPTHX[4] is an example of efficient occupancy grid SLAM in 3D. An octree data structure is used to maintain hundreds of maps required by the particle filter. The octree structure exploits spatial locality and temporal shared ancestry between particles to reduce the processing and storage requirements.

[3] `http://www.cs.duke.edu/~parr/dpslam/`

[4] Nathaniel Fairfield, George Kantor, and David Wettergreen, "Real-Time SLAM with Octree Evidence Grids for Exploration in Underwater Tunnels," Journal of Field Robotics 2007.

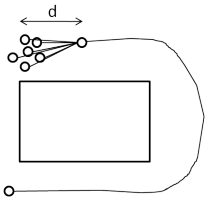

Figure 4: Illustration of loss of diversity in DP SLAM. In this example, correcting the map after closing the loop around the rectangular obstacle would require diversity of paths all the way back to the root; however, the map can only be corrected over length d because the maps of all current particles share the observations of the common ancestor.

8 Stochastic Gradient Descent approach to SLAM

Olsen proposed as fast gradient descent approach to recovering a time sequence of robot poses[5]. The steps are:

1. Marginalize out the landmarks
2. Use stochastic gradient descent to optimize pose graph
3. (if desired) Perform mapping using the optimized pose data

8.1 Marginalize out the landmarks

Observations of landmark location relative to robot pose are converted to constraints between poses from which the same landmark was observed. This results in fewer variables (no landmark variables) but more constraints. The resulting pose graph will have two types of constraints:

1. temporal (odometry) constraints between poses at adjacent timesteps
2. landmark constraints between poses from which the same landmark was observed

8.2 Critical trick: represent the state variables as pose deltas

At each iteration a single constraint is chosen at random for the update.

[5]Edwin Olson, John Leonard, and Seth Teller, "Fast Iterative Alignment of Pose Graphs with Poor Initial Estimates," ICRA 2006

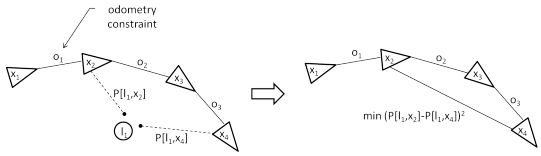

Figure 5: Illustration of marginalizing out a landmark. The observations of landmark l_1 at poses x_2 and x_4 (left) are converted to a single pose constraint (right).

Instead of representing the state variables as robot poses, represent them as the difference in pose between timesteps.

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \end{bmatrix} \rightarrow \begin{bmatrix} x_0 \\ \Delta x_1 \\ \Delta x_2 \\ \vdots \end{bmatrix}$$

Robot pose is calculated by summing the pose deltas, so a change in any one pose delta affects the robot pose at all subsequent timesteps. Alternatively, this change in representation can be thought of as a change in the distance metric used by gradient descent.

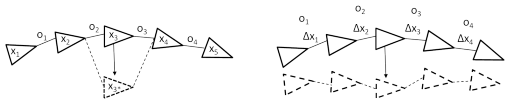

Figure 6: (left, original representation) Consider x_3 undergoes a large update due to a landmark constraint. Likely many iterations will pass before o_2 or o_3 are selected, finally updating the neighboring poses. (right, pose delta representation) Because a change in pose delta affects all subsequent poses, sequences of poses will update together.

8.3 Other notes

- it's magic! tends to jump over local minima
- be clever about the learning rate (increase for loop closure, etc.)
- this is a batch algorithm, but it can be implemented online using a sliding window/lag filter