CMPUT 499 Probabilistic Robotics
MCMC SLAM - Implementation Report

Roshan Shariff

September 24, 2010

This implementation of the MCMC SLAM algorithm assumes that the edge labels form a group. It stores only the edge labels and recomputes vertex labels when necessary. The spanning tree of the graph is assumed to contain all the state change edges, and one observation edge for each feature. In this implementation, the first observation of a feature is used, but the algorithm's design is not contingent on this.

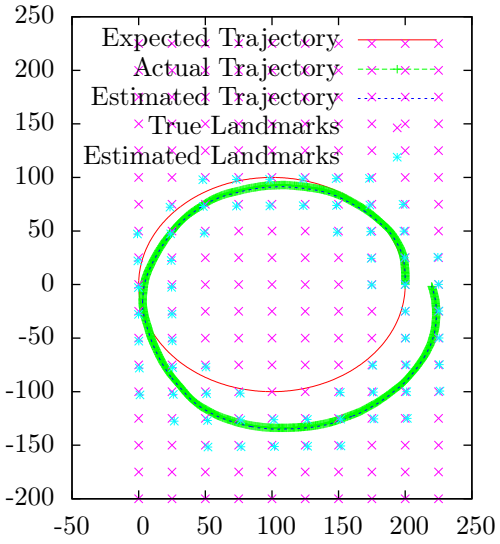
To efficiently retrieve the label of a state vertex while storing only the edge labels in a way that allows the edge labels to be modified easily, a *binary indexed tree* is used.

Binary Indexed Tree

A binary indexed tree stores a sequence of elements (which must form a group) and admits the following operations:

- Retrieve an element of the sequence, in amortised $O(1)$ time.
- Append an element to the end of the sequence, in amortised $O(1)$ time.
- Modify an element of the sequence, in $O(\lg n)$ time.
- Compute the product of any contiguous subsequence, in $O(\lg n)$ time where n is the length of the subsequence.
- If the group is linearly ordered and the sequence contains only positive elements, then find the largest initial subsequence with product less than some given bound, in $O(\lg n)$ time using a modified binary search.

The binary tree structure is stored in a contiguous block of memory, which improves memory locality and therefore has better cache behaviour. An implicit tree structure is defined over the elements, with the root of the tree being the element with index 0. The parent of the element with index i is given by zeroing the least significant set bit in the binary representation of i. Thus the depth of any element is the number of set bits in the binary representation of its index, which is bounded by $\lg(i)$.



Every index stores the partial product of the elements from its parent to itself. For example, index $6_{10} = 110_2$ has parent $4_{10} = 100_2$, and so the 6th index stores the product of the 5th and 6th elements. To retrieve the partial product up to a given element, the tree is traced back to the root, accumulating products in the correct order. Any particular element can be retrieved by computing the partial product up to that element and the previous element.

The binary indexed tree is used to store the estimates of state changes in the SLAM problem, while allowing the estimates to be modified efficiently. It is also used to sample from the edges, using the modified binary search operation on a binary indexed tree containing the edge weights.

More information on binary indexed trees (but for storing scalar values instead of group elements) is available in [1].

Edge Weighting Function

The original MCMC SLAM paper [2] proposes the edge weighting function

$$p_T(e \mid l) \propto \frac{1}{J_e(l(e))}.$$

However, the use of this function results in failure to converge when the three-parameter odometry model for differential drive robots is used. State edges are almost never resampled; only observation edges are resampled. Much better results were obtained by using the weighting function

$$p_T(e \mid l) \propto \frac{\kappa}{\sqrt[\kappa]{J_e(l(e))}}$$

where $\kappa = 2$ for observation edges and $\kappa = 3$ for state change edges. This reflects the fact that for 2D planar robots, each observation represents two independent parameters (for the position of the landmark) whereas each state change represents three independent parameters (two dimensions of translation and one of rotation).

Pose Representation

The pose of the 2D planar robot is represented as a pair of complex numbers, which allows only multiplication and addition operations to be used to compute compositions of pose changes and invert pose changes. It is also expected that complex arithmetic operations are implemented efficiently on superscalar processor architectures.

Running Time

MCMC iterations on the graph take running time proportional to the number of edges whose weights must be recomputed for each edge relabelling. Relabelings of the state change edges additionally take time linear in the total number of observed landmarks.

Experimental Results

Some experiments were run to verify the correctness of the implementation and measure its performance characteristics. The experimental set up was identical to the "large" SLAM problem described in the MCMC SLAM paper, with 612 landmarks arranged in an equidistant grid with spacing 25 m, and the robot driving in a circle of radius 200 m. The maximum observation range is 30 m. The robot completes two revolutions in 3518 steps.

The experiments were performed on an Apple MacBook Pro notebook computer with an Intel Core 2 Duo processor running at 2.2 GHz. Running two experiments simultaneously achieved full processor utilisation on both cores with no noticeable slowdown, indicating that the simulation is processor-bound and memory bandwidth is not an issue. The following results were averaged over 10 runs of the simulation:

- MCMC iterations per step: 100
- Average observations per step: 4.47
- Cumulative state error: 46.87 m
- Landmark estimation error: 36.37 m
- Running time: 532.16 s

These results are comparable to those of the original Matlab implementation running on a cluster, with that implementation averaging a running time of 27178.39 s.

References

[1] P.M. Fenwick. A new data structure for cumulative frequency tables. *Software: Practice and Experience*, 24(3):327–336, 1994.

[2] P. Torma, A. Gyorgy, and C. Szepesvari. A Markov-Chain Monte Carlo Approach to Simultaneous Localization and Mapping. 2009.