Giving a Compass to a Robot

Probabilistic Techniques for Simultaneous Localisation and Map Building (SLAM) in Mobile Robotics

Roland Willdor van Loh Wenzel
Department of Computer Science
University of British Columbia
2366 Main Mall
Vancouver, B.C. V6T 1Z4, Canada
rwenzel@cs.ubc.ca

Abstract

An important feature of an autonomous mobile robotic system is its ability to accurately localize itself while simultaneously constructing a map of its environment. This problem is complicated because of its chicken-and-egg nature: in order to determine its location the robot needs to know the map, and in order to build an accurate map the robot must know where it is. In addition, a robust system must account for the noise in odometry and sensor readings. This project explores the probabilistic methods of solving the SLAM problem using Rao-Blackwellisation.

1 The odometric model or "why noise?"

Robots can use a variety of inputs in order to find their way through the world: tactile and force sensors (bumpers), speed of sound sensors (sonar), and speed of light sensors (laser, vision and stereo vision). A cheap and commonly used (because easy to realise) method is the integration of odometry data using wheel encoders. For short trips, they provide accurate information at a low computational cost.

As wheel encoders only look at the wheels, and not on the real world, errors accumulate on longer trips, and it can be prone to sudden events, e.g. wheel slippage. Because we cannot be sure of what the sensors tell us (this is true for all sensor types named above, however, in different degrees), we need to include other information that gets "external" data from the world and reviews the internal representation of the world based on the new data (which will be discussed with either laser range scanners or vision in 2.1 and 2.2, respectively). That is the reason why we need to include noise in our perception model, as we will do in 3.3.

2 Localisation

One very important operation in a robot's life is the process of determining its position in the world. This usually involves matching sensor readings with some model of the world (using a model of the sensor, including the sensor noise).

In order to do localisation, we have to determine the probability distribution over the space of possible robot poses. We use the following model, which follows the Markov assumption (z_t are the locations, y_t the observations at time t):

$$\begin{tikzcd}
z_0 \arrow[r] \arrow[d] & z_1 \arrow[r] \arrow[d] & \dots \arrow[r] \arrow[d] & z_{t-1} \arrow[r] \arrow[d] & z_t \arrow[r] \arrow[d] & z_{t+1} \arrow[d] \\
y_0 & y_1 & \dots & y_{t-1} & y_t & y_{t+1}
\end{tikzcd}$$

Figure 1: Markov model for localisation

This means that $P(y_t \mid z_t, y_{0:t-1}) = P(y_t \mid z_t)$, i.e. future data is independent of past data, if we know the state of the system.

In order to find the right place where our robot is at a certain time, we basically have to determine which part of the map gave rise to the sensor reading of that time. In order to perform this map matching process, the "data association", we use particle filtering. We spread particles (which are position proposals) over the map and compare the expected sensor readings with the real one.

This approach causes an interesting ambiguity: Whenever the robot sees a certain configuration of landmarks (e.g. a wall with a door), he could (ignoring the past) be on any place of the map where he could see this same configuration. The robot is not sure in front of which door he is standing in reality. Only by taking his past belief of his position into account (as a prior), he can narrow his belief, which is represented by multiple Gaussians. Thus, the Gaussians tend to get a lower variance on the robot's walk. However, mainly the noise of his scanners and the slippage of his wheels cause these Gaussians to get wider on the other hand, i.e. they get a higher variance. Because those two effects are thwarting each other up to a certain degree, the robot can never be absolutely sure about his position – albeit he gets a very good impression of where he is. See figures 2 and 3.

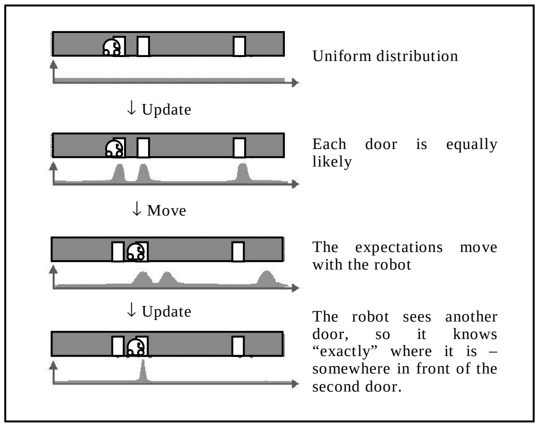

Uniform distribution
↓ Update
Each door is equally likely
↓ Move
The expectations move with the robot
↓ Update
The robot sees another door, so it knows "exactly" where it is – somewhere in front of the second door.

Figure 2: one-dimensional example [7]

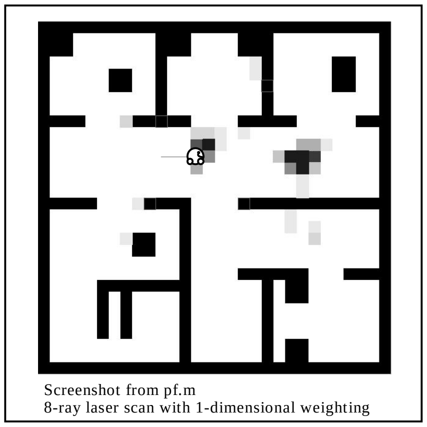
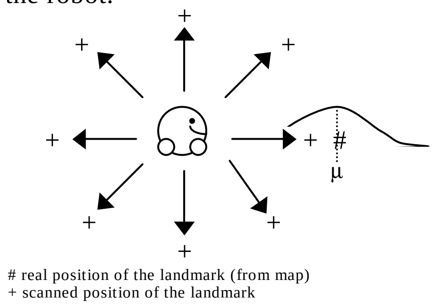


Figure 3: two-dimensional example

2.1 Laser range scan

Laser scanners have the big advantage that they measure distances by the speed of light, and the required information can be extracted much easier and faster than in the case of vision. Since it is very expensive to install fast moving 360-degree laser scanner units, one can use some fixed rays pointing in different directions, giving a planar and fragmentary view of the world. Supposed that the robot isn't able to rotate (and it can only move in x/y-directions), we know the bearings of the beams exactly. To contrast the expected and the perceived scanner readings, which represent distances to the objects in the world, we therefore only compare these distances in the direction of each beam, leading to a 1-dimensional Gaussian model. Whenever the scanner readings could be anywhere around the robot, however, we should use a 2-dimensional Gaussian model to weight the particles instead. At this point, it is very critical to have good algorithms to match the structures in the map and the real world, especially if there may be errors in the measured orientation of the robot.



# real position of the landmark (from map)
+ scanned position of the landmark
μ

Figure 4: one-dimensional model for laser range scanner with 8 beams

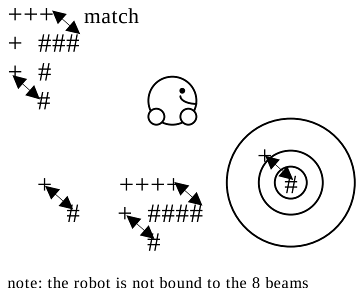


Figure 5: two-dimensional model for vision and laser range scanner

2.2 Vision

Unfortunately, vision is computationally very expensive and can have problems with accuracy and reliability, mainly because of the limited resolution of the camera image. Nevertheless, vision is very attractive because it is a non-invasive, passive way to gather information about the world. Because of the high number of unique features that can be found in real life, e.g. using the SIFT feature approach [1], the localisation process comes up with quite good results very early. In case of a 360-degree view, the vision algorithm comes up with about 40 features per step instead of 8 features for the laser scan version in our simulated environment.

Because of its inherent nature, vision requires a 2-dimensional weight function as discussed in the previous section 2.1.

3 Tackling SLAM using Rao-Blackwellisation

Simultaneous Localisation and Mapping (SLAM) is the problem of concurrently estimating the robot’s position and the position of landmarks or features in the environment. In other words, SLAM addresses the problem of building a map of an environment from a sequence of sensor measurements obtained from a moving robot that had no idea how the world looked like before he started moving. Therefore, SLAM is just an “extension” of the localisation problem, but it is trickier than the original task.

The problem arises from the uncertainty of sensor measurements and motion tracking. Consider the apparent “sub problems” of mapping and localisation:

- Given reliable motion tracking (odometry), it would be easy to build maps despite of sensor uncertainty.
- Given a map, it would be easy to correct errors of motion tracking despite of sensor uncertainty.

SLAM is so complex because of the large number of cross-correlations that appear, i.e. because the above sub problems cannot be separated. That’s why the literature often refers to it as a chicken-and-egg situation.

3.1 standard approach for small worlds: Kalman filtering

We can use Kalman filtering to solve our SLAM problem: Each landmark's position is represented by its x- and y-coordinates, but is disguised by measurement errors that result primarily from measurement noise and a wrong belief about the current position of the robot. To simplify the problem, we assume that the true position of each landmark is affine to the measured position and includes some noise. We then can use the Kalman filter to estimate the true position for each step, as it looks for a "true" state (μ, Σ) that has the smallest uncertainty (the minimal variance) [6]. The gathered information can be used to construct a map of the world that represents the current belief state.

Since all co-variances between the n landmarks have to be represented in the covariance matrix Σ, we need n^2 entries or $O(n^2)$ space when using the standard approach using a Kalman filter. On top of that, since we need to update all of these entries at each time step, we also run into a time complexity of $O(n^2)$. To tackle larger problems in SLAM, we therefore need other approaches or at least modifications of the above approach. One common way to do that is to delay some of the incorporation work in the model that has to be done, or to apply the Kalman filter to each landmark separately instead of to the whole map. This approximation will be chosen for the Rao-Blackwellisation method, where we use an exact model, but approximate the inference process by only "reflecting" about each landmark individually.

3.2 Rao-Blackwellisation: an approach to reduce complexity

If we assume that the robot's position is given, the position of one landmark is independent of all other landmark's positions[1]. Hence, we use Rao-Blackwellisation to "guess" the robot's path using a particle filter, and – given the position of the robot in the respective time step – we use a Kalman filter for each landmark to estimate it's "true" position [2]. Since we have to deal with completely different "views" for each particle, including a different data association and even a different number of identified landmarks, the integration of all these information in a single map is computational very expensive. On the other hand we need a certain number of particles to be sure to "hit" the true current position of the robot with one of the particles in order for the filter not to diverge [6]. Therefore, we have to live with a trade-off between a high computational load and a diverging filter. By using a simple data association algorithm and enough particles, the Rao-Blackwellised Particle Filter (RBPF) gives good results in our implementation.

3.3 The probabilistic model

In the case at hand, we assume a model with the conditional probability distributions from figure 6, based on the jump Markov chain in figure 7. The distribution (1) is the motion model or state transition model. After executing a movement command at a known position, the distribution of the following position is also known.[2] (2) serves as our measurement model, while (3) is our observation model.

[1] In other words: when you know all locations from which you saw a specific landmark, and you know the relative positions of those view points, then you don't need more information to update your belief about the landmark's position.

[2] This is relatively easy on a flat surface with wheels, but seems to be hard outdoors, with legs.

(1) $z_t \sim P(z_t \mid z_{t-1})$

(2) $x_t \sim P(x_t \mid x_{t-1}, z_t, z_{t-1}) = N(x_{t-1} - (z_t - z_{t-1}), BB')$

(3) $y_t \sim P(y_t \mid z_t, x_t) = N(x_t C(z_t), DD')$

Figure 6: Conditional probability distributions

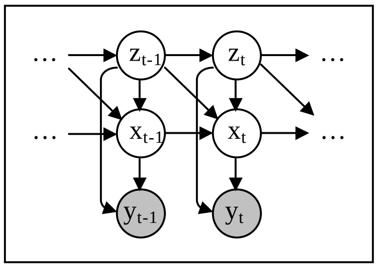


Figure 7: Jump Markov chain

The unknown discrete state z_t represents the positions of the robot, the unknown Gaussian state x_t the distances to all the detected features (and therefore the map) and y_t the observations at time t. B and D are noises that have to be added to account for inaccuracies in odometry and scanner readings, respectively, that occur during the map update and the observation. The problem of novelty detection (the robot might encounter previously unseen landmarks) is solved using a selector matrix C, which only updates those locations in the map that have been detected in the current step. In addition to that, we assume that the environment has F uniquely distinguishable landmarks, where F is fixed beforehand. A practical realisation of the landmarks could be red stickers depicting numbers “1” to “F”. A research assistant could then train a visual classifier well enough to make the assumptions hold. Given perfect distinguishability, the uncertainty of sensor measurements equals the uncertainty of landmark position estimates (relative to the robot).

The approach that has been implemented for this project uses two types of modelling tools: particle filters that handle the path posteriors, and Kalman filters, which handle the landmark location posteriors.

Particle filtering, as outlined above, is a Monte Carlo technique for sampling from the distribution of particle positions. Each particle $z_t^{(i)}$ represents a guess of the location z_t. Locations $z_t^{(i)}$ come from simulated transitions (e.g. the motion model). At time t, we have a population of particles $Z_t = \{z_t^{(i)}\}_{i=1}^N$, where N is the number of particles. The population Z_t is updated when t increases. First, we simulate the transition model to move each location $z_t^{(i)}$ like the robot would do. Second, we generate Z_{t+1} by sampling from the locations generated in the first step. We evaluate and normalise the importance weights $w_t^{(i)} \propto P(y_t \mid y_{1:t-1}, \hat{z}_t^{(i)})$ in order to penalise locations that are inconsistent with sensor measurements. This is followed by a resampling step, such that the samples with low importance weights are discarded. As the simulation continues, unlikely locations “die off”. The remaining ones are continuations of the likely.

After sampling $z_t^{(i)}$ in the above proceeding, we use a Kalman filter to propagate the mean $\mu_t^{(i)}$ and the covariance $\Sigma_t^{(i)}$ of the map x_t. The Kalman filter acts the same way as outlined in 3.1. For a detailed view on the used formulas, please see [3], [4] or the code of the accompanying implementation.

4 Implementation

We used a particle filter to do Monte Carlo Localisation (pf.m) and RBPF to do SLAM (rbpf.m) in a synthetic, two-dimensional, grid-based world model. All implementations are done in Matlab Version 6.1. See the readme.txt for explanations of the data format. Due to some limitations of the artificial world, the following remarks have to be made:

(1) The world is discrete. This especially causes noise to vanish under certain circumstances in the rounding process.

(2) The two dimensional version of the weighting algorithm (which is used for laser and vision) currently uses a very simple data association method (it simply chooses the closest object). This is not correct, as features may be matched multiple times. One way to fix this is to simply drop ambiguous points (compare to the algorithm to match stereo pictures in [1]).

(3) Right now, the implemented version of the vision algorithm samples the orientation of the robot randomly from a uniform distribution, without including any prior information. This has only an effect if the vision angle is smaller than 360 degrees, because the robot will see all non-occluded features otherwise. In a later stage, it would be nice to include the rotation angles in the position proposition function, so that the rotation can be determined as well with the particle filtering process. However, a small vision angle only makes sense because it better applies to the real world – it will reduce the processing time, but will lead to less features and thus to a worse localisation behaviour.

5 Conclusions

Rao-Blackwellisation is a fast and easy to understand way of solving the SLAM problem. There are some implementations that try to optimise the existing shortcomings in order to speed the algorithm up once more. The FastSLAM algorithm bases on the independence of the landmarks and a smart representation of the particles that prevents superfluous copying of the landmark estimates. Thus, FastSLAM gains an astounding fast O(N log F) time filter complexity (where N is the number of particles) [5]. When delaying the incorporation of observations in the majority of the map, which is another approach that is chosen in the Thin Junction Tree Filtering algorithm, the time complexity of the filtering operation can be even reduced to constant-time [6].

Acknowledgements

I want to thank my team members, Iryna Skrypnyk and James Cook, for their contribution to the implementation, Nando de Freitas for his support during this project, and my mouse for providing me more than 5 kilometres of accurate cursor motion that were needed to implement this project.

References

[1] Se, St., Lowe, D. & Little, J. (2001) Vision-based Mobile Robot Localization And Mapping using Scale-Invariant Features. In *Proceedings of IEEE International Conference on Robotics and Automation, Seoul, Korea*, pp. 2051-58.

[2] Doucet, A., de Freitas, N., Murphy K. & Russell, St. (2000) Rao-blackwellised particle filtering for dynamic Bayesian networks. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pp. 176-183.

[3] Skrypnyk, I.B. (2002) Simultaneous Localization and Mapping Using Rao-Blackwellised Particle Filtering. Online at *http://www.cs.ubc.ca/~andrones/530A-projects.html*.

[4] De Freitas, N. (2002) Rao-Blackwellised Particle Filtering for Fault Diagnosis. In *IEEE Aerospace*.

[5] Montemerlo, M., Thrun, S., Koller, D. & Wegbreit, B. (2002) Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pp. 593-598. AAAI Press.

[6] Paskin, M.A. (2002) Thin Junction Tree Filters for Simultaneous Localization and Mapping. Report No. *UCB/CSD-02-1198*.

[7] Thrun, S. (2000) AAAI-2000 Tutorial on Probabilistic Robotics. Online at *http://www.cs.cmu.edu/~thrun/tutorial/tutorial.ppt*, slide 27.