The effect of using negative information on the accuracy of FastSLAM

(Bachelor thesis)

Marko Doornbos (m.doornbos.2@student.rug.nl)

July 31, 2010

Abstract

For years, the standard algorithm for Simultaneous Localization And Mapping (SLAM) has been the Extended Kalman Filter (EKF). A more recent and more accurate solution is the FastSLAM-algorithm, which uses a Rao-Blackwellized particle filter. However, unreliable landmarks still pose a major problem to this algorithm. Two approaches to deal with these landmarks have been tested and compared to the performance of the original FastSLAM-algorithm. The results are promising, but not entirely applicable to real-world situations, for which further research will be needed.

1 Introduction

One of the most important parts of autonomous systems research is Simultaneous Localization And Mapping, generally referred to as SLAM. A SLAM-algorithm will simultaneously create a map of an autonomous system's surroundings based on sensor and motor data and determine the system's location on that map. This is more difficult than it might seem at first glance, as both sensor and motor data is generally very noisy, with which the algorithm should be able to cope. Things become even more difficult when parts of the environment look very similar to each other. For a more expansive explanation of the SLAM-problem, see (Montemerlo and Thrun, 2003).

Despite these difficulties, having reliable SLAM-algorithms that can function in real-time will be very important for the widespread use of robotics. After all, many of these machines will need to be able to return to certain points in the world they are in. For example, many of them will have to return to a station set up somewhere to recharge batteries, and it simply is not feasible to provide a map for each location a system may find itself in or to have someone create the map for the system. Another example is that one would like a vacuum cleaner robot to reach every part of one's house regularly. After all, such a robot would be of little use if it simply did not go to certain parts of the house. To ensure that it goes to every location it can reach, and that it goes there at least once within a certain amount of time, real-time SLAM will be necessary.

1.1 SLAM algorithms

For a long time, the primary algorithm used for SLAM was the extended Kalman filter (Welch and Bishop, 2001). This algorithm estimates the current state as a Gaussian based on the previous state and sensor and motor data, with the shape of the Gaussian based on the uncertainty of the previous state and the uncertainty of the sensor and motor data. The main problems with this algorithm are that it will become increasingly uncertain as running time increases and that an error in estimation or the modelling of the environment can cause the algorithm to construct a faulty map from which it will be unable to recover. The primary reason it was used so much anyway is that it can often work quite decently, and it simply was the best algorithm available.

An alternative for the extended Kalman Filter is fastSLAM (Montemerlo and Thrun, 2003). This algorithm is a Rao-Blackwellized particle filter. It maintains a set number of robot location estimates, each of which maintains a number of Kalman filters.

These estimate landmark locations for that robot position estimate. Implementation of a tree-based datastructure allows this algorithm to run in O(M log K), where M is the number of particles and K is the number of landmarks. For the full algorithm, see (Thrun, Montemerlo, Koller, Wegbreit, Nieto, and Nebot, 2004). This algorithm does have a number of weaknesses, which include the use of just motor data for pose estimation (solved in a later version, fastSLAM2.0 (Thrun et al., 2004)) and a tendency to generate false landmarks based on sensor noise, which then complicate data association and increase computing time.

1.2 Improvements to FastSLAM

One way to improve the fastSLAM-algorithm is to keep track of the certainty of the existence of observed landmarks. After all, things like people moving around or sensor noise may be registered as landmarks, which can then influence the localization. One can keep count of the number of times a landmark has been detected in comparison to the number of times it should have been detected. This will provide a nice estimate of the certainty of the existence of that landmark. Should this estimate fall below a set threshold, the landmark will then be discarded, preventing a loss of accuracy and speeding up the processing. These gains are caused by having a shorter list of possible matches in data-association and preventing associating an observation with a non-existent landmark. An example implementation of this is discussed in (Montemerlo and Thrun, 2003).

However, it is not certain how much implementing this will affect the accuracy of the algorithm's localization. This is important to know. Therefore, an implementation of this negative information usage with counters was added to a FastSLAM simulation. It was then compared with an implementation of fastSLAM that does not discard landmarks. Both implementations are in Matlab and make use of known data association, with each landmark having a chance not to be detected. Another improvement using negative information has also been tested (Kuipers, 2010), and the results of that implementation will also be compared to my results. The question to be answered here is:

Does implementation of landmark removal based on existence certainty in fastSLAM affect its location estimation accuracy?

2 Methods

2.1 The fastSLAM algorithm

FastSLAM is a SLAM-algorithm developed by (Thrun et al., 2004). It uses a rao-blackwellized particle filter to estimate a number of possible robot poses based on odometric and sensor data for the current time step. The basic running of the algorithm is explained below.

A more detailed explanation of the fastSLAM1.0-algorithm with some improvements can be found in (Thrun et al., 2004).

2.1.1 Initialization

At initialization, a pre-set number of N particles is generated. These share the same position and orientation, i.e. the robot's starting location. Each of these particles will maintain its pose (position and orientation), a list of EKFs for the observed landmarks, and a measure of their own certainty (starting off as $1/N$). The algorithm from this point onwards is looped in each time step.

2.1.2 Loop

1 First, data from the agent's actuators is used to estimate the agent's movement. This estimated movement is then applied to the particle, including updates to the landmark positions (these are moved to where they would be if the agent's movement estimate is accurate). This applied movement is also modified by simulated motor noise.

2 Then, observations by the agent's sensors are associated to either known landmarks or determined to be new landmarks not previously encountered. The Kalman filters for existing landmarks are then updated based on the new data. Then, new Kalman filters are created for any newly observed landmarks. Then, the particle's internal likelihood is calculated based on how well the current observations match the estimated landmark positions in this particle.

The previous two steps of the algorithm are executed for each particle.

3 After this, a new set of particles is sampled from the current set. This sampling is weighted by the internal likelihood of the particles. This leaves the algorithm with a set of particles which is probably distributed across all possible agent positions where the more likely positions are populated more densely. After this, the algorithm can continue with the next loop.

2.2 The experiment

In order to measure the effect of using negative information in a FastSLAM-implementation a simulation in Matlab was used. [1] This simulation uses known data association and has the option of using either fastSLAM 1.0 or fastSLAM 2.0. For this experiment fastSLAM 1.0 was used, as it is is easier to work with. The simulation uses basic maps consisting of a number of landmarks and a number of way points. The agent starts at the coordinates (0,0) and will travel across the way points on the map in an order that was determined when the map was created. The number of times the agent follows the set of waypoints can be chosen by the user. The software is quite customisable and incorporates a simple map-editor. The used settings are described later in this text.

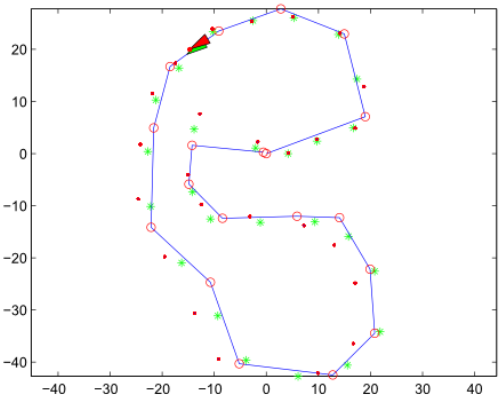

Figure 1: A capture of the simulation during runtime. Distances are in meters. Red circles are waypoints, connected by blue lines to show the order in which they are followed. Green stars are landmarks. The red dots next to the landmarks are the estimations of landmark positions by the algorithm. The green triangle (mostly covered by the red triangle) represents the position of the agent, while the red triangle shows the position estimated by the algorithm.

2.3 Additions to the simulation

2.3.1 Missing landmarks

An important part of the experiment is that landmarks are not always observed. Therefore, the simulation had to be modified to allow for landmarks not to be observed. Three ways to do this were created. However, only one of these methods was actually used in the experiment.

This method is used for missing a given percentage of landmarks after the first loop across the waypoints. Which landmarks will be missed is determined at initialization and does not change during the run.

A missrate between 0 and 1 is added to the variables used by the simulation. At initialization, an array with a length equal to the number of landmarks is initialized. Each of these elements cor-

[1] This simulation can be found at `http://www-personal.acfr.usyd.edu.au/tbailey/software/slam_simulations.htm`

responds to a specific landmark. Each element of this array is given a random value between 0 and 1. Then, all elements in the array with a value higher than the missrate are set to 1, while the rest are set to 0. This determines which landmarks will be missed after the agent has started its second circuit of the waypoints. This method does not cause missed landmarks in the first circuit, as that would prevent those landmarks from ever being observed.

2.3.2 Using negative information

The goal of this experiment was to determine the effect of negative information usage. A version of the method suggested by Montemerlo and Thrun (2003) was used. This implementation checks to see which landmarks could have been observed given the particle's internal list of landmarks. It uses the same code used to determine the actual observations and creates a list of these (P). Landmarks that have actually been observed (O) are then removed from that list. This leaves a list of landmarks that should have been observed according to the map modelled by the particle, but were not. ($P - O = M$)

A pair of counters is maintained for each landmark in a particle. One keeps track of the number of times the landmark has been observed (Co). The other keeps track of how often the landmark was not observed while it should have been (Cm). These counters are updated each time step. This is done by increasing the first counter for all landmarks in O by 1 ($Co(O) = Co(O) + 1$) and increasing the second counter of all landmarks in M by 1. ($Cm(M) = Cm(M) + 1$) After the counters have been updated, any landmarks where the ratio between the two counters falls below a set threshold (set to 1.0 in the experiment) are deleted from the particle. This is not done if the landmark to be deleted is the only landmark in the particle because the simulation cannot handle that situation.

The removed landmarks will no longer be part of the observations for that particle. This way, unreliable or disappeared landmarks are kept out of the model.

2.3.3 An alternative method

Another method tested at the same time was implemented by (Kuipers, 2010). This method determines which landmarks should have been observed by the particle currently being processed in the same manner as the previous method. The certainty of that landmark's position within the particle currently being processed is then increased. Increasing this uncertainty is done by increasing the values in that landmark's covariance matrix which represent that uncertainty. Any landmark for which the size of these two values is above a set threshold is then deleted. This deletion works just like in the previous version.

2.4 Performance measurement

Deviation in position has been chosen as the primary performance measure because it is arguably the most important part of a SLAM-algorithm. After all, the primary purpose of these algorithms is to determine robot position both accurately and efficiently. The position of the particle with the highest internal probability is used to estimate the location of the agent. Each time step, the euclidean distance ($Sqrt((\Delta X)^2+(\Delta Y)^2)$) between this most probable particle and the ground truth of the simulation is measured and put into an output file. Only the measurements from the final circuit of the waypoints were used to determine performance. As the maps used in the experiment were designed to have a lap take 3500 time steps, the last 3500 entries in this output file could be used for this.

2.5 Experimental setup

In order to test the implementations, six different maps with routes of similar lengths were created for the simulation. Three further versions of each of these maps were made, with approximately 25, 50 and 75 landmarks in each version. The routes can be seen in figure 2.

The implementations were tested with three different missrates: 0.1, 0.25 and 0.5. To ensure the same landmarks would be missed in both versions of the algorithm, the seed for the pseudorandom number generator in Matlab was set to the same value for both. This ensures Matlab produces the exact same sequence of pseudorandom numbers in both runs.

The simulation was run for each combination of route, number of landmarks, missrate and version (a version that does miss landmarks but makes no

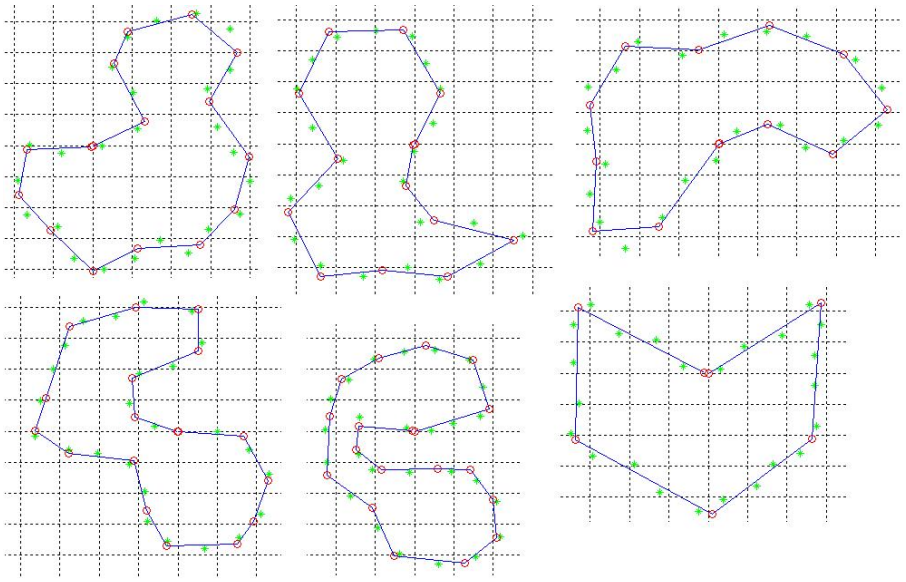

Figure 2: The maps used for the experiment. Both the height and width of a space in the grid is 10 meters in-simulation.

use of negative information and the version implementing the negative information usage through counters). A third version of the simulation that makes use of negative information in a different way as implemented by [Kuipers (2010)] was also tested under the same circumstances. Each of these runs used the same random seed between versions. This seed determines the sequence of pseudorandom numbers generated by Matlab. Therefore, by using the same seed in different runs of the same map, the same observations and motor noise are provided to the algorithm. The additions to the algorithm do not add or prevent calls to the functions that provide these pseudorandom numbers. Therefore, any difference in performance between runs under the same circumstances is entirely caused by the additions to the algorithm.

2.5.1 Simulation variables

The simulation's default settings were used with the following exceptions:

1. MAX_RANGE: 5.0 The agent's vision range was decreased considerably.

2. *removalThreshold*: 1.0 This variable was added in the counter-based version.

3 Results

After all test runs were made, the data from these was gathered. There were 81 runs with the unmodified algorithm, 81 with the version implementing counters as described in this article and another 81 with the algorithm implemented by (Kuipers, 2010). For each run, the last 3500 data points were used, as explained earlier. The values for these points were averaged per run. Figure 3 contains boxplots of these averages. Statistical analysis with paired t-tests showed that my implementation of negative information usage has a significantly higher error than the unmodified algorithm, unlike the version modified by (Kuipers, 2010). The values for the T-tests are shown in figure 1.

4 Discussion

4.1 Extrapolation from experimental results

As stated in the previous section, using the modified fastSLAM-algorithm results in a significantly higher error than using the unmodified algorithm, at least for the tested circumstances. However, the circumstances of the experiment, especially the use

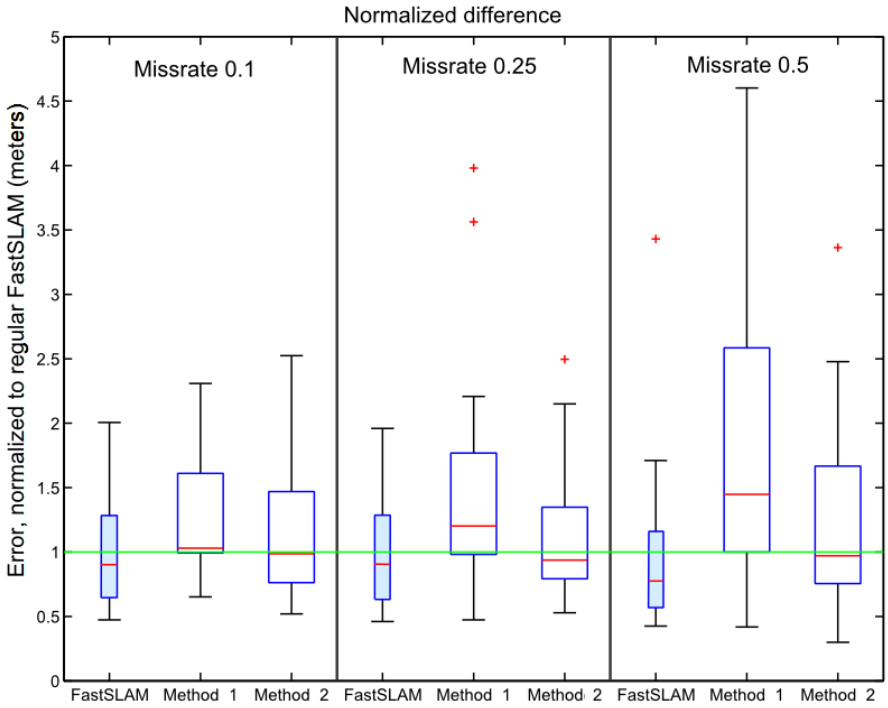

Figure 3: The results of the experiment. The data points have been averaged per run. Method 1 is the counter implementation described here, while method 2 is the modification implemented by (Kuipers, 2010). Normalization has been done separately for the three missrates.

Missrate	Method 1	Method 2
0.1	0.030	0.329
0.25	0.029	0.581
0.5	0.045	0.852
Combined	0.002	0.422

Table 1: The results fo the paired T-tests between the unmodified algorithm and the two versions with additions. A value smaller than 0.05 means the difference with the unmodified algorithm is statistically significant. Method 1 is the version my myself, while method 2 is the version by (Kuipers, 2010).

of known data association, severely limits the impact of these results to circumstances very close to those of the experiment.

Removing unreliable landmarks becomes a lot more important when unknown data association becomes involved. There, having spurious landmarks will greatly affect the time in which data association can be completed. They can also result in more potential incorrect associations, which reduces the accuracy of the algorithm.

4.2 Results for the alternate method

The loss of accuracy was considerably less in the version modified by (Kuipers, 2010). In fact, the loss of accuracy was not statistically significant for this method. In addition to this, the addition does not use additional memory compared to the unmodified version, unlike the first addition.

4.3 Conclusions

Based on the experimental data, we can determine that both additions are likely to decrease the accuracy of the FastSLAM-algorithm. However, the counter-based method will probably perform worse than the method modifying the covariance matrix. However, there is the possiblity that this effect is less profound (or might even disappear) in situations with unclear data association. In order to

prove or disprove this hypothesis, further research would be necessary.

Another interesting point of study would be the the computing time requirements for the additions in comparison to the unmodified version. However, testing that would probably require using a different simulation or building one from scratch, as the implementations used in this experiments were not computationally optimal. This was due to limitations in the original simulation, which was not made with deleting landmarks in mind. This kind of research could very well be combined with the research using unknown data association, probably as a large project.

References

J. Kuipers. Negative landmark information influence in fastslam, rijksuniversiteit groningen (department of artificial intelligence), 2010.

M. Montemerlo and S. Thrun. Simultaneous localization and mapping with unknown data association using fastslam. *IEEE Robotics and Automation Magazine*, 2:1985–1991 vol.2, 2003.

S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto, and E. Nebot. Fastslam: An efficient solution to the simultaneous localization and mapping problem with unknown data association. *Journal of Machine Learning Research*, 2004.

G. Welch and G. Bishop. An introduction to the kalman filter. *In Practice*, 95:1–16, 2001.