A note on robot localization

Simon Parsons
Department of Computer and Information Science
Brooklyn College, City University of New York
Brooklyn, NY 11210
parsons@sci.brooklyn.cuny.edu

January 10, 2005

Abstract

This is a brief note on the localization techniques developed by Wolfram Burgard, Dieter Fox and colleagues, and used by the MetroBots RoboCup Four-legged league team. In particular, we describe the techniques of Markov localization and Monte Carlo localization, summarize the most important publications that describe them, and identify the various flavours of these techniques outlined in those publications.

1 Introduction

The aim of this note is twofold. First it is intended as a summary of my understanding of the probabilistic techniques for localization—establishing the location of a robot from sensor data—that we have used in the MetroBots team. As such it is intended as a source of reference descriptions against which our implementations can be checked, and as record of the variations on the basic technical theme that can be found in the literature. Second, it is intended as a guide, albeit a very specific one, to the literature that describes these techniques. As a guide this note makes no pretentions to be comprehensive—it focuses exclusively on the work of Burgard, Fox and colleagues—and leans heavily towards those aspects of the techniques that are most applicable to the RoboCup task. However, it should still be of use to those interested in localizing in other mobile robot scenarios.

2 Basic theory

The basic schema for these probabilistic localization techniques is as follows. The description is gleaned from [18]). Figure 1 (borrowed from [16], indeed from the chapter written by Sebasian Thrun) represents the general schema for localization as a dynamic Bayesian network—based on the pose X_{t-1} of the robot at time $t-1$, and the action A_{t-1} it carries out at that time, we can predict

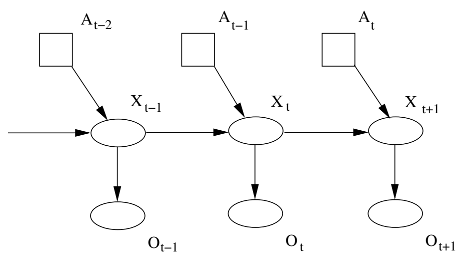

Figure 1: A general schema for robot localization

the pose at time t, X_t, and from the pose we can establish the observations the robot can make O_t. This general model of updating is a form of *Bayes filter*, a *Markov chain* technique.

The key idea is that we calculate a probability distribution over the set of possible poses, and we call this the *belief.* Thus

$$\mathsf{Bel}(x_t) = \Pr(x_t \mid d_{0,\ldots,t})$$

where $d_{0,\ldots,t}$ is all the data from time 0 to t, and includes both observations and actions a_t, is the belief that the robot is in the pose x at time t, x being a possible value of the variable X.

Without loss of generality we assume actions and observations alternate:

$$\mathsf{Bel}(x_t) = \Pr(x_t \mid o_t, a_{t-1}, o_{t-1}, a_{t-2}, \ldots, o_0)$$

and we can use Bayes' rule to write the above as:

$$\mathsf{Bel}(x_t) = \frac{\Pr(o_t \mid x_t, a_{t-1}, \ldots, o_0) \Pr(x_t \mid a_{t-1}, \ldots, o_0)}{\Pr(o_t \mid a_{t-1}, \ldots, o_0)}$$

which reduces to:

$$\mathsf{Bel}(x_t) = \eta \Pr(o_t \mid x_t, a_{t-1}, \ldots, o_0) \Pr(x_t \mid a_{t-1}, \ldots, o_0) \tag{1}$$

since the denominator is constant relative to x_t.

Now, the diagram in Figure 1 summarises the basic principle upon which these techniques are based—we assume that if we know the pose x_t that the robot is currently in, this summarises all information about past poses, x_{t-1} and before, so that the next pose x_{t+1} depends only on x_t and a_t. This is the *Markov* assumption applied to this specific problem.

In this case the Markov assumption says that:

$$\Pr(o_t \mid x_t, a_{t-1}, \ldots, o_0)$$

reduces to:

$$\Pr(o_t \mid x_t)$$

and (1) can be written as:

$$\mathsf{Bel}(x_t) = \eta \,\mathrm{Pr}(o_t \mid x_t)\,\mathrm{Pr}(x_t \mid a_{t-1}, \ldots, o_0)$$

A little more maths gets us a recursive equation for the belief. We integrate over the set of possible poses at time $t-1$ (since we don't know which of these the robot actually adopts, we have to consider all of them and their effects on x_t), to get:

$$\mathsf{Bel}(x_t) = \eta \,\mathrm{Pr}(o_t \mid x_t) \int \mathrm{Pr}(x_t \mid x_{t-1}, a_{t-1}, \ldots, o_0)\,\mathrm{Pr}(x_{t-1} \mid a_{t-1}, \ldots, o_0) dx_{t-1}$$

Then again exploiting the Markov assumption we reduce $\mathrm{Pr}(x_t \mid x_{t-1}, a_{t-1}, \ldots, o_0)$ and get:

$$\mathsf{Bel}(x_t) = \eta \,\mathrm{Pr}(o_t \mid x_t) \int \mathrm{Pr}(x_t \mid x_{t-1}, a_{t-1})\,\mathrm{Pr}(x_{t-1} \mid a_{t-1}, \ldots, o_0) dx_{t-1}$$

and then, by the definition of $\mathsf{Bel}(x_{t-1})$, we get:

$$\mathsf{Bel}(x_t) = \eta \,\mathrm{Pr}(o_t \mid x_t) \int \mathrm{Pr}(x_t \mid x_{t-1}, a_{t-1}) \mathsf{Bel}(x_{t-1}) dx_{t-1}$$

This allows us to calculate the belief recursively based on the *next state density* or *motion model*:

$$\mathrm{Pr}(x_t \mid x_{t-1}, a_{t-1})$$

which tells us how likely a particular pose is given the previous pose and the action carried out, and the *sensor model*:

$$\mathrm{Pr}(o_t \mid x_t)$$

which tells us how likely a particular sensor reading is given a particular pose.

Now, the Bayes filter theory describes how to do the relevant computations given continuous distributions over x, a and o. The challenge in implementing a model based on the theory is finding a way of doing the computations.

For continuous state spaces (such as those we are faced with), the only closed-form solutions that are known restrict the distributions to be Gaussians [17]—such solutions form the basis of *Kalman filters* and are unable to represent situations where, for instance, symmetry means that sensor readings cannot distinguish between two possible robot poses and so the robot position is best represented by a two-peaked distribution. In general, Kalman filter solutions have trouble with the *wake-up robot* problem [7]—when the robot is first booted up it has no idea what its location is—and the *kidnapped robot* problem[4]—when the robot is moved unexpectedly from a position that it knows it is in[1].

To find solutions that go beyond the Kalman filter and can model the situation in which the robot may be in several positions, and which can handle the kidnapped and wake-up robot problems, we need to find different ways to do the Bayes filter computations. Two such solutions are *Markov localization* and *Monte Carlo localization*, and these are presented in the next two sections.

[1]In [7] the authors remark that the wake-up robot problem is a special case of the kidnapped robot problem "in which the robot is told it is carried away". While this is true, a lot is hidden in the fact of telling the robot it has been carried away which makes the wake-up problem

```
if new observation $o_{t+1}$ is obtained.
   for every location $x^i$ in the grid at $t$
      $Bel(x^i_{t+1}) := \Pr(o_{t+1} \mid x^i_{t+1}).Bel(x^i_t)$
   normalize $Bel(x_{t+1})$.

if new odometry information $a_t$ is obtained.
   for every location $x^i_{t+1}$ in the grid at $t$
      $Bel(x^i_{t+1}) = \sum_j \Pr(x^i_{t+1} \mid x^j_t, a_t).Bel(x^j_t)$
```

Figure 2: An algorithm for Markov localization

3 Markov localization

As [10] points out, any approach to localization has to decide how to represent the prior distribution over x_t. Marlov localization, [10] continues, uses a discrete representation—either a topological map, or a grid. The work on Burgard, Fox and colleagues adopts the grid representation.

The basic algorithm, taken from [7] is as in Figure 2, though we have changed the algorithm slightly to tie in with the notation used in [18] (which is the notation used above), and (to ease comparison with the algorithm for Monte Carlo localization in Figure 3) we have considered normalization to be a separate step (whereas Fox *et al.* compute the normalizing factor as they go along which saves another traversal of the grid).

Looking at this algorithm, we can see that there are three main stages.

1. If we receive odometry data, we compute the likelihood of *every* possible new location.

   We do this computation by summing over the whole grid. For every one of the locations in the grid, the x^j in the summation, we compute how likely it is that the last action would take us to the grid location in question, the x^i. The probability of being at grid location x^i is the sum of all contributions from all the x^j. This summation replaces the integral at the end of the previous section.[2]

easier. In the wake-up robot case the robot *knows* that it has no idea where it is (because its clock has just started for example), and this knowledge can be reflected in a uniform distribution over locations which provides a good start point for localizing. In the kidnapped robot case, the robot only *thinks* it is lost (because temporarily its sensor readings don't seem to fit with its notion of where it is), and it has to update a sharply peaked distribution—not a good starting point—which may turn out to be correct (if the confusing readings were caused by an umodelled feature for example).

[2]Another way of thinking of this. For each possible location of the robot—by which we mean we do this for every possible location in turn—we ask the question "if we were here at time t and did a_t, where would we be at $t+1$". Answering this typically gives the robot some probability of being in several different locations, and this is a contribution towards the total probability of being in those locations. The probability of the robot being at any individual location x_{t+1} is the sum of all the contributions that arise from all the x_t.

2. If we receive sensor data, we re-compute the likelihood of the robot being in every possible location given that data. This is done [7] for each possible location x^i by establishing the distance d that the sensor says the robot is from a given obstacle, while the map says that the obstacle is at δ, and then using:

$$\Pr_m(d \mid x^i) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(d-\delta)^2}{2\sigma^2}} \tag{2}$$

   This, of course, is a Guassian distribution with mean at δ, and a standard deviation of σ, and will give higher probabilities to those locations x^i for which d is close to the value δ that is expected.

3. We normalize after updating with sensor data.

The structure of the algorithm makes it clear that we can separate the updating into two parts—the update using odometry data being one, and the combination of update with sensor data and normalization being the other.

Because we represent the entire robot environment, we can solve the problems that occur with Kalman filtering. Maintaining a probability for *every* location gives us a solution to the kidnapped robot and wake-up robot problems—wherever the robot is, there will be a possible location that is close to that position, and the sensor updating stage will give this a suitably large probability. Markov localizatin also trivially provides a solution to the problem of handling two or more likely robot positions, by allowing several possible locations to have (relatively) large probabilities.

The basic method can be extended in a number of ways. Chief among these are the use of various filters that modify the information coming from the proprioceptive and external sensors. In essence these aim to model aspects of the environment to better predict the kind of readings that will be produced.

There are three such filters mentioned by Fox *et al.*, the entropy filter, the distance filter, and the blockage filter[3]. The *entropy* filter makes use of the information theoretic notion of entropy to exclude sensor readings that make the robot's position more confused. We can define the entropy $H(X)$ over the range of possible locations X as being:

$$H(X) = -\sum_x Bel(x) \log Bel(x)$$

The higher the entropy, the more uncertain the robot is of its location (and the less peaked the distribution over X). Given an observation o, we can compute the change in entropy following an update with it (including normalization) as:

$$\Delta H(X \mid o) = H(X \mid o) - H(X)$$

and a negative change indicates that the robot has become more certain about its location. The entropy filter discards any updates that lead to an increase in entropy (though [7] actually says the opposite).

[3][8] describes a *novelty* filter, but this is the same thing that is later called the distance filter (and the latter does indeed seem a better name).

The *distance sensor* is intended to compensate for the fact that with most distance sensors, unmodelled aspects of the environment generate measurements that are shorter than the distance expected from the map[4]. The filter is designed to remove those sensor meaurements which, with probability greater than γ (which is set to 0.99 in the work discussed here).

From (2) we can compute $\Pr_m(d_j \mid x_i)$, the probability that we will measure a distance d_j when the robot is in location x^i. This, in turn, allows us to calculate the probability $\Pr_{short}(d_j \mid x_i)$ that d_i is shorter than expected—this is the same as the probability that the expected measurement is longer than what is measured:

$$\Pr_{short}(d_j \mid x_i) = \sum_{k>j} \Pr_m(d_k \mid x^i)$$

and from this we can calculate what we need, the probability that the measurement is short give the current belief of the robot (which means summing over the whole belief space):

$$\Pr_{short}(d_j) = \sum_{x^i} (d_j \mid x^i) Bel(x^i)$$

and if there is any measurement for which this probability is greater than γ.

The *blocking filter* was presented in Fox's PhD thesis [5], and models the situation in which a proximity sensor is blocked by an unknown obstacle.

The problem with Markov localization is that to get accuracy we need to use a fine grid—[7] suggests a grid of squares where each square is between 10 and 40cm (and angular resolution is between 2 and 5 degrees) on a side, and talks of experiments using 15cm squares and an angular resolution of 2 degrees (the 15cm resolution is also suggested by [3]). For large environments such a grid will require a lot of computation and even for the RoboCup environment used in 2003 (which was 4200mm by 2700mm) we would require approximately 10.5 million grid elements. Furthermore, if we have m grid elements, the odometry update requires that in order to compute the probability of being in any one of these m poses, we look at the probability of getting there from each of the m possible poses we were at in the previous timestep—thus the updating is $O(m^2)$.

There are ways of addressing this computational problem. [7] suggests two ways to do this. First, the sensor model can be pre-computed. Since the location is represented as a set of discrete states, it is possible to compute the relative location of every observable object from every location. From this is then possible to construct a lookup table that gives the probability $\Pr(o \mid x)$ of seeing a given observable at a given distance from every possible location (when the "given distance" is quantised at the same resolution as the grid that the robot is localised against. Accessing this lookup table can then be faster than computing $\Pr(o \mid x)$ from scratch (though this seems less likely for the

[4]In contrast, in the case of legged-league RoboCup, the main unmodelled aspect that affects distance measurements is partial observation of markers—either because they are clipped by the edge of an image, or because they are partially obscured by another robot—which tends to make measurements too long.

RoboCup scenario, at least in the impoverished observation space where only markers count as observations).

The second approach, *selective updating*, works by reducing the number of possible locations that are updated at every iteration of the algorithm. It works by choosing a threshold ϵ (a value that [7] suggests should be set to 1% of the *a priori* probability), and only fully updating those locations which have a higher $\mathsf{Bel}(x_t)$ value than this. Locations with $\mathsf{Bel}(x_t) \leq \epsilon$ cannot be ignored, otherwise we no longer have a probability distribution over the set of possible locations, so we alter the update:

$$\mathsf{Bel}(x^i_{t+1}) := \Pr(o_{t+1} \mid x^i_{t+1}).\mathsf{Bel}(x^i_t)$$

in Figure 2 to be:

$$\mathsf{Bel}(x^i_{t+1}) := \begin{cases} \Pr(o_{t+1} \mid x^i_{t+1}).\mathsf{Bel}(x^i_t) & \text{if } \mathsf{Bel}(x^i_t) > \epsilon \\ \tilde{\Pr}(o_{t+1}).\mathsf{Bel}(x^i_t) & \text{otherwise} \end{cases}$$

where $\tilde{\Pr}(o_{t+1})$ is the prior probability of making the observation o_{t+1}, and can be computed by averaging over all possible locations:

$$\tilde{\Pr}(o_{t+1}) = \sum_x \Pr(o_{t+1} \mid x)\Pr(x)$$

This calculation can be done offline.

Now, as described so far, the advantage of selective updating is that we don't have to establish $\Pr(o_{t+1} \mid x^i_{t+1})$, which is minimal. However, as [7] explain, we can go further, identifying regions of the environment that the robot is unlikely to be in as *passive* if every possible location in that region has $\mathsf{Bel}(x^i_t) \leq \epsilon$, and not updating the probabilities of the individual locations within that region, but instead keeping track of all the normalization factors applied while the region is passive (since these are the only changes carried out to locations with $\mathsf{Bel}(x^i_t) \leq \epsilon$ that vary from update to update). If we also record the maximum probability of any cell in the region, and can tell if and when the region becomes *active* (that is it is no longer passive) when the product of this and the combined normalization factor exceeds ϵ at which point we start to fully update the region again.

While selective updating greatly improves the efficiency of Markov localization, it has been subsumed by Monte Carlo localization.

4 Monte Carlo localization

To avoid the computational overhead associated with Markov localization, we can approximate $\mathsf{Bel}(x_t)$ using a set of samples or *particles* (which gives Monte Carlo localization its other name, particle filtering). The approximation is then:

$$\mathsf{Bel}(x_t) \approx \{x^{(i)}_t, w^{(i)}_t\}\ i = 1, \ldots, m$$

```
$x_{t+1} = \emptyset$

for $j = 1$ to $m$
    pick a random $x_t^{(i)}$ from $x_t$ according to $w_t^{(1)}, \ldots, w_t^{(m)}$
    generate $x_{t+1}^{(j)}$ from $x_t^{(i)}$, $a_t$ and $\Pr(x_{t+1} \mid x_t, a_t)$
    Generate a weight for this new sample $w_{t+1}^{(j)} = \Pr(o_{t+1} \mid x_{t+1})$

normalize $w_{t+1}$ in $x_{t+1}$
return $x_{t+1}$
```

Figure 3: An algorithm for Monte Carlo localization

Each $x^{(i)}$ is a possible pose, and each $w_t^{(i)}$ is the probability of that pose (also called an *importance factor*). Initially we have a set of samples (typically uniform) that give us $\mathsf{Bel}(x_o)$, and we update with the algorithm in Figure 3. This process is guaranteed to converge on a correct set of poses and weights in the sense that after a suitable number of updates, the set of samples are a set that could have been drawn from the correct probability distribution over the set of all possible poses—thus nothing is lost by representing $\mathsf{Bel}(x_t)$ as a set of samples. The rate of convergence is proportional to $1/\sqrt{m}$ [6].

There are four main stages to the algorithm (taken from [18]). The first three are:

1. First, samples are picked from the set of particles representing $\mathsf{Bel}(x_t)$. Just as in Markov localization we start with a guess as to where we are, this time we select one location randomly.

2. Second, we use this sample, the action we know was carried out, and the motion model to predict where the robot will be at $t + 1$ given that it was at the sampled location at t.

   This prediction is itself done by sampling—a location is sampled from $\Pr(x_{t+1} \mid x_t, a_t)$ where x_t is that pose picked by the previous sample, and a_t is the action the robot thinks it carried out at t.

3. Third, we compute a weight for this sample, using the most recent sensor reading to determine how likely this location is, given the reading that was just taken.

These three steps are carried out m times in the basic algorithm, where m is the number of samples representing $\mathsf{Bel}(x_t)$. [3, 6] suggest that values of m between 1000 and 5000 give the best results, and show results that support this. Once this new set of samples have been created, we normalize them so that:

$$\sum_i w_{t+1}^{(i)} = 1$$

and the weights are probabilities. Note that while the updating algorithm appears to be $O(m)$, naive approaches to sampling from the set of particles (which may involve stepping through the whole set of particles for each sample) will raise this to $O(m^2)$.

The convergence property of the particle filtering approach means that we have a solution to the wake-up robot problem—we pick an initial random set of particles, and the process will converge on a set of samples that is an accurate representation of the robot position. The approach can also represent arbitrary distributions so that it has no problems with simultaneously representing multiple hypotheses about the robot pose. The kidnapped robot problem, however, is more problematic.

The difficulty posed by the kidnapped robot problem is not theoretical. Even if the robot is "teleported" a considerable distance, so that none of the samples in its belief vector are close to its new pose, the introduction of new samples from the motion model means that the process will eventually converge. However, in practice, the time it takes for such localization may be unacceptably long, and there are a battery of techniques that can be employed in order to reduce the time until convergence.

Most simply, as described in [6] it is possible to add

> a small number of uniformly distributed, random samples

to the belief vector after each update, and this will also work as a solution to the "sample impoverishment" problem described in [3]. [6] does not suggest how to identify the number of samples, but taking some proportion of the total number of samples seems appropriate.

A more sophisticated approach is *sensor resetting* [12]. As described by [11] (but with notation altered to fit what we are using here), sensor resetting picks the number of samples to add based on how badly the current set of samples seem to represent the location. The number of samples to add, $\tilde{m}$ is determined by:

$$\tilde{m} = m \cdot \max\left(0, 1 - \left(\frac{\tilde{p}}{p_T}\right)\right) \tag{3}$$

$\tilde{p}$ is the average probability of the current set of samples:

$$\tilde{p} = \frac{1}{m} \sum_i \Pr(o_t \mid x_t^i)$$

(recalling that $\Pr(o_t \mid x_t)^i$ is the same as w_i before normalization) and p_T is a constant threshold that is manually determined. Thus when the average probability falls below the threshold some fraction of new samples are added, and this fraction grows as the existing sample set becomes less accurate. [11] give experimental results for p_T of 0.000025 and 0.0000025, showing that the first is robust against sensor noise but fails to handle kidnapping, while the second recovers from kidnapping but is significantly less accurate when sensor noise rises above 40%.

A third approach extends sensor resetting [2]. Rather than comparing $\tilde{p}$ to a fixed threshold in (3), this approach, adopted by the University of Washington's RoboCup Legged-league team the UW Huskies-02, compares two smoothed averages. One is a long term average, and is intended to estimate the noise level in sensors and the environment:

$$\overline{p}_l \leftarrow \overline{p}_l + \eta_l(\tilde{p} - \overline{p}_l)$$

The other is short term and is intended to capture rapid changes due to a sudden failure to predict the robot location (as would arise from kidnapping):

$$\overline{p}_s \leftarrow \overline{p}_s + \eta_s(\tilde{p} - \overline{p}_s)$$

The number of samples to add is then computed by:

$$\tilde{m} = m \cdot \max\left(0, 1 - \nu\left(\frac{\overline{p}_s}{\overline{p}_l}\right)\right)$$

thus adding samples only when the short term average is $1/\mu$ less than the long term average. The parameters η_l and η_s are set according to:

$$0 \leq \eta_l \ll \eta_s \leq 1$$

and the figures given in [11] for η_l and η_s are 0.0001 and 0.1 respectively, while μ is 2.

A fourth approach to picking new samples is that of Mixture Monte Carlo localization (Mixture MCL) as described in [18]. As that paper points out, unlike sensor resetting this is a theoretically justified approach, an in essence [11] weights the new samples according to the existing probability distribution over pose.

All of the above approaches are, as far as I can determine, methods for deciding what proportion of the sample set to replace because the sample set as a whole is out of tune with the observations. Another, related, approach is to make the *size* of the sample set adaptive. In other words, have some threshold η, and when the sum of the unnormalised weights of the sample set exceed η, stop sampling. Thus when the robot is well localised, the sample set is small, and few computational resources need to be spent on localization. When the robot gets lost, the sample set expands, and more computation is used to relocate it. Since this approach is only mentioned in [6], it seems that it is not required once the more sophisticated approaches described above are adopted[5].

Finally, it is worth noting that [11] includes some information on the parameters used by Fox and his students for the University of Washington AIBO

[5]The basic approach handles the kidnapped robot problem by having a large sample set that some of those samples eventually end up being close enough to the real robot location to localise it. Sensor resetting and its kin have small sample sets, but keep renewing samples until some of them end up close enough to the real robot location so one can think of them having the same large set of samples, but cycling through them over time rather than holding them all at once.

team. They use just 30 samples, model noise in the sensors using Gaussian models in which the standard deviation of the angular error is 10 degrees and the standard deviation of the distance measurement is 15%. The paper also provides some useful benchmarks for comparison of performance (though note that all processing was actually done on a 1GHz processor, not that on-board the AIBO—the ERS-210 described in the paper runs at about a quarter of this speed.

5 The literature

This is not a comprehensive study of the literature (for that, see the introduction and related work sections of [18]) but a description of what seem to me to be the most relevant papers of Burgard, Fox and colleagues. It is also not a thematic review, but instead briefly summarises the content of each paper to provide an easy key into this section of the literature, and it should be possible to read this independently of the rest of the paper.

We cover the papers in what I believe to be the approximate chronological order in which the papers were written.

5.1 Markov localization

[1] proposes the use of a *probability grid* (rather than a topological map, as previously used), a grid quantising the possible x and y locations of the robot, and where each cell contains the probability that the robot is in that cell—in the only previous probability grid approach, that of [15], the probability of each cell had denoted the probability that it held an obstacle. The paper also gives the basic updating algorithm, and dicusses both sensor fusion and the use of Gaussian noise in the motion model. Finally, the paper gives an example use of the technique in an office environment.

[10] provides a more detailed mathematical explanation of Markov localization, compares it with Kalman filtering, and gives a detailed description of noise models, in particular noise in the motion of the robot. Results from experiments in both an office environment and a museum (where sensor readings are much more prone to error because of crowds of people, especially those attracted by the robot) are also presented.

[8] concentrates on extensions to the basic Markov localization approach that improve its performance in *dynamic* enironments like the museum where so much of this work was tested. When "obstacles" move, sensor readings need to be handled more carefully, and the paper introduces the *entropy filter* and the *novelty filter* that can be applied to sensor readings to improve the Markov localization model. The entropy filter only incorporates sensor readings that confirm the current estimate of the robot's position. The novelty filter which, like the entropy filter, aims to limit the negative impact of erroeous distance readings caused by unmodelled dynamics (like those of museum visitors). The

novelty filter removes distance readings that with some probability (set to 0.99 in experimental work) are shorter than expected.

[7] is the most comprehensive paper on Markov localization. It describes the *wake-up robot* problem (where the robot has to figure out where it is from scratch) and the *kidnapped robot* problem (where the robot has to cope with being moved unexpectedly, "teleported"). The mathematical description of the updating process is interesting in that it separates the process into two somewhat independent stages—updating with a sensor reading (observation) and updating with motion information. This is the basis of the algorithm in Figure 2 above. The paper goes into further detail on the motion model (giving additional detail to that in [10], but also not including some of the details in [10]).

The paper also explains in great detail how sensor models can be created, and carefully describes the filters introdcued in [8] (though the novelty filter is here renamed the *distance filter*. Computational aspects are discussed, with proposals for the sensor model to be pre-computed, and for *selective updating*—only updating cells with a probability above a certain threshold ϵ (a suggested figure is 1% of the *a priori* probability), allowing the computation to focus on poses that are the most likely. Finally, the paper describes experiments with two different robots in two different museums.

5.2 Monte Carlo localization

[6] then introduces Monte Carlo localization as a means of improving Markov localization both computationally (since you don't need a huge grid) and in terms of accuracy (since positions are computed exactly and there is no quantization error). The paper gives the basic maths, and considers the updating as taking part in two phases, motion and observation. The motion stage generates a new sample set where each sample has a weight of $1/m$, where m is the number of samples (well it is N in the paper, m using the notation we have adopted here from [18]). Updating with observations is here called *resampling*, and the paper describes the process as requiring two stages—the incorporation of sensor data and then normalization—and mentions (though only gives a reference to) an $O(m)$ algorithm for achieving it.

The paper also describes the addition of a set of random samples to the set one updates in order to solve the kidnapped robot problem,and mentions that the "sampling step" can be considered an anytime process that can be interrupted when new data (odomentry or sensory) is obtained (though presumably this only really works if one sorts the set of particles and updates the most likely location first (otherwise if you repeatedly update the least likely location you might get lost when you think you're found). The final algorithmic twist introduced in the paper is the idea of *adaptive sampling*—if you incorporate the motion/sensor update into a single step, and sum the recomputed weights as you go along, you can stop the process when some threshold is exceeded, thus requiring less samples when you are more sure of the location of the robot (again this suggests that the set of particles is ordered by weight). As ever,experimental results are given, including a comparison with Markov localization that shows that the stated advantages of Monte Carlo localization do indeed hold.

[3] covers much of the same ground as [6], providing in addition some deeper theoretical background, and an experimental comparison of the accuracy of position tracking unsing Monte Carlo localization and Markov localization. The upshot of the comparison is that with Monte Carlo localization accuracy of positioning does not increase much once 1000 samples are used (and as few as 100 are feasible with laser sensors), and that with such numbers positioning is as accurate as with Markov localization using a 15 cm grid. Finally, the paper describes the problem of "sample impoverishment", where high probability poses get selected many times in the sampling step and diversity is lost, and gives pointers to some solutions.

[18] gives a nice survey of previous work in localization, and then describes general Bayes filtering techniques (which include Kalman filters and Markov localization), the principles of particle filters, and the basic Monte Carlo localization algorithm. Experimental work is presented, which compare the basic Monte Carlo approach with Markov localization, and also explore the limitations of the technique. In order to deal with the limitations (which cause the average error to increase when the sensor model has too little noise), the dual-MCL and mixture=MCL approaches are introduced, and the mixture approach is shown to localize the robot more accurately than the other approaches (indeed it can outperform standard MCL when using 50 samples as against the 1000 used by the standard approach..

[9] covers a lot of the same ground as [18], but also discusses multi-robot localization (which again reduces error rates).

As the title of the paper suggests, [11] extends the experimental comparison from [10]. Whereas [10] looked at Markov localization and Kalman filtering, [11] is mainly concerned with variations on the Monte Carlo localization theme. In particular the paper compares the use of a Kalman filter, a combination of Kalman filter and grid-based Markov localization, and four types of Monte Carlo localization. These four are all augmentations of the basic algorithm outlined in Figure 3 with methods for solving the kidnapped robot problem.

The basic approach for handling the kidnapped robot, as outlined in [6], is to add in some random samples after each update with sensor information. When the robot is "teleported" this will eventaully provide some samples in the right place and the robot will localise around these, but the process can take time. An improvement, proposed in [12] and described in detail in [11] is to pick more samples when the most recent sensor reading does not agree with where the robot thinks it is—this is established by comparing the average sample weight after updating with some threshold. Two of the approaches used in [11] are forms of this *sensor resetting localization*, with different thresholds. One of the other approaches is the mixture Monte Carlo approach of [18], and the last is *adaptive* Monte Carlo localization, described for the first time in [2], where the likelihoods used by sensor resetting are averaged over a period rather than instantaneously.

The results indicate that some form of additional sampling is desirable for particle filtering (Markov localization of course works fine for the kidnapped robot since it simultaneously computes the probability of all possible locations, and doesn't seem to take much more computational effort intrestingly enough), though it also suggests that with the wrong parameters sensor resetting localization can actually do as badly as Kalman filtering.

Finally, we should note that [17] provides a high-level overview of all the work on particle filters (without giving much detail on the algorithms used) and [14, 13] provides a particle filtering solution to the SLAM problem, where a robot has to build a map while localizing itself within that map.

References

[1] W. Burgard, D. Fox, D. Hennig, and T. Schmidt. Estimating the absolute position of a mobile robot using position probability grids. In *Proceedings of the 14th National Conference on Artificial Intelligence*, 1996.

[2] Z. Crisman, E. Curre, C. Kwok, N. Ratliff, L. Tysbert, and D. Fox. Team description: UW Huskies-02. In G. Kaminka, P. Lima, and R. Rojas, editors, *RoboCup-2002: Robot Soccer World Cup VI*. Springer Verlag, 2002.

[3] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte Carlo localization for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1999.

[4] S. Engelson and D. McDermott. Error correction in mobile robot map learning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1992.

[5] D. Fox. *Markov localization: A probabilistic framework for mobile robot localization and navigation*. PhD thesis, Department of Computer Science, University of Bonn, 1998.

[6] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte Carlo localization: Efficient position estimation for mobile robots. In *Proceedings of the 16th National Conference on Artificial Intelligence*, 1999.

[7] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

[8] D. Fox, W. Burgard, S. Thrun, and A. B. Cremers. Position estimation for mobile robots in dynamic environments. In *Proceedings of the 15th National Conference on Artificial Intelligence*, 1998.

[9] D. Fox, S. Thrun, W. Burgard, and F. Dellaert. Particle filters for mobile robot localization. In *Sequential Monte Carlo Methods in Practice*. Springer Verlag, New York, 2000.

[10] J-S. Gutmann, W. Burgard, D. Fox, and K. Konolige. An experimental comparison of localization methods. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1998.

[11] J-S. Gutmann and D. Fox. An experimental comparison of localization methods continued. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[12] S. Lenser and M. Veloso. Sensor resetting localization for poorly modeled mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2000.

[13] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the 18th National Conference on Artificial Intelligence*, 2002.

[14] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, 2003.

[15] H. P. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 116–121, 1985.

[16] S. Russell and P. Norvig. *Artificial intelligence: A modern approach*. Prentice Hall, 2nd edition, 2002.

[17] S. Thrun. Particle filters in robotics. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence*, pages 511–518, San Francisco, CA, 2002. Morgan Kaufmann.

[18] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, 128(1–2):99–141, 2001.