Hierarchical Monte-Carlo Localisation Balances Precision and Speed

V. Estivill-Castro B. McKenzie
Institute of Integrated and Intelligent Systems
Griffith University
Brisbane 4111 QLD Australia

Abstract

Localisation is a fundamental problem for mobile robots. In dynamic environments (robotic soccer) it is imperative that the process be very efficient. Techniques like Monte-Carlo Localisation or Markov Models have been shown to be effective in dealing with partial recognition of landmarks, errors in odometry and the kidnap problem. But they are particularly CPU intensive. However, many times decision-making does not need high accuracy, and thus, we have developed a hierarchical version that allows us to balance real-time efficiency of computation with precision in localisation.

1 Introduction

Localisation is a fundamental problem of mobile autonomous robots [Gutmann and Fox, 2002]. It is hard to imagine how to make appropriate decisions while moving around, with no information of our whereabouts. In the past, problems where knowledge of location is important have been solved only at great expense. Usually by embedding equipment into the environment that would make the solution simple: "complex position-guidance equipment ... built into the factories and buildings"[Leonids and R., 1995]. There are many technologies available for robot localisation, for example GPS, active/passive beacons, odometry (dead reckoning) and sonar. While the improvements in accuracy can be obtained with more costly hardware, there is always a time delay between sensor input and determination of location. Moreover, with more sensors, finding efficient algorithms is more important. From the Mars rovers to humble automated vacuum-cleaners, it is becoming impractical to rely on complete supporting infrastructure. As robots leave the labs and constrained environments, they will require CPU cycles for other tasks, and fast localisation will continue to be critical. An important challenge in small-scale robotics (like robotic soccer in the Aibo league [Veloso *et al.*, 1998]) is finding the robot's position when only limited sensor information is available. Vision is almost the only input to localisation and, as opposed to the middle size or small size leagues, the 4-legged league only has partial views of the environment.

The inadequacy of triangulation in situations with noisy sensory data [Betke and Gurvits, 1994] has led to more robust solutions. *Bayes Probabilistic Localisation* and *Kalman Filters* (KM) utilise Bayes law to introduce the concept of "confidence" or "probability" to an estimated (current) position. Thus, the current location has a probability attached to it. As sensory information comes in, not only is the location adjusted, but some measure of the sensors reliability is used to update the probability that the location is correct. To do this, the filter uses Gaussian representations (or other probabilistic models) for motor and perception noise variation. The model of the robot's actions to move from its current position to a desired position is a predictor that is usually called the motion model, while the correction of the prediction using sensor data is the perceptual model.

When data comes from a source that is known to be highly accurate, the location's probability rises. When highly uncertain data comes in the probability may fall [Dahm and Ziegler, 2003; Thrun, 1998; Thrun *et al.*, 2000]. While these techniques went a long way to solving the noisy sensor problem, they rely heavily on having some prior (and accurate) location information. They also require information on sensor performance, which often varies according to the landmark and its position.

More recently, probability theory has been used in a different way to develop a technique know as *Markov Models* (MM) [Fox *et al.*, 1999a]. The subject space is divided into discrete parts; usually by drawing a grid onto the internal map. Now, instead of monitoring a single possible location, this method monitors the probability of each cell in the grid being the correct one. When information about distance to a landmark comes in, the probability of each cell is adjusted slightly. In this way no prior knowledge of location is needed but can be built.

If odometry is provided, then the entire grid of probabilities is moved appropriately [Betke and Gurvits, 1997; Thrun *et al.*, 2000]. The advantage of this approach is that while critics say that it does not work in this or that situation, it can easily be adjusted and strengthened so that it does. Unfortunately it can require extremely large amounts of processing power to use, particularly in situations calling for high accuracy.

By monitoring many potential locations instead of one, not only can ambiguous situations (such as when landmarks are identical) be dealt with, but with less cost than with Markov models. This is the philosophy of *Monte Carlo Localisation* (MCL) [Fox *et al.*, 1999a; 1999b; Thrun *et al.*, 2001] (it could be considered a multiple stochastic ensemble of Kalman Filters). Variations on MLC include dynamically adjusting the size of the sample population, removing less likely samples and adding new random ones, and generating samples directly from sensor information [Thrun *et al.*, 2000]. This technique combines the strengths of the other major approaches and avoids the main weaknesses. While the literature has recorded combinations and variations of the three basic ideas (KF, ML, MCL), some are specific to the type of sensor, for example sonar or upward camera, and others to the variation of localisation problem involved [Thrun *et al.*, 2000]. Currently the focus is variations of the MCL and Markov model techniques[Röfer and Jüngel, 2004; Thrun *et al.*, 2000].

In robotic soccer the level of accuracy in localisation is required with different degrees at different stages of the game. For example, a robot in control of the ball may identify it is in direct alignment of the opponents open goal. This is usually sufficient information to decide on an action (namely, take a shot on goal). Although it may have significantly inaccurate information on localisation (distance to goals from vision is extremely inaccurate; colour-coded goal recognition is simple, but recognising goal edges is very difficult). On the other hand, a high level of location accuracy is required by the goalie to guard its goal. Localisation in the RoboCup 4-legged league has attracted so much attention that in 2003 and 2004 a localisation challenge was part of the technical challenges. Reducing landmarks and colour coding remains a focus for approaching an playing environment closer to human soccer. In this direction, the most relevant work to our paper regarding localisation to SONY Aibo soccer is the achievements using only the soccer-field lines [Röfer and Jüngel, 2004], and its seems apparent that this is possible only though MCL [Röfer and Jüngel, 2004].

2 Vanilla MCL

Self-localization is the problem by which the software on board the robot determines at time t the probability function $p(\vec{x}_t)$ of being currently at a pose described by the vector $\vec{x}_t$. The inputs to the localization algorithms are observations $\vec{o}_t$ and executed actions $\vec{a}_t$. It usually has three subvariants: *position tracking* involves regularly monitoring position and updating previous estimates as the robot moves around. *Global positioning* consist of finding out where a robot is with only a map of the environment as initial information. Finally *the kidnap problem* consists of recuperating from being transported uninformed to another position after achieving very high accuracy in recognizing a position. A generic framework for KM, MM and MLC [Gutmann and Fox, 2002; Thrun *et al.*, 2001] indicates that the robot will iteratively use a motion model to make a prediction for $\vec{x}'_{t+1}$ using $p(\vec{x}_t)$ and $\vec{a}_t$. Then, it will correct the prediction using the data $\vec{o}_t$ from its sensors to correct $\vec{x}'_{t+1}$ and produce the new $p(\vec{x}_{t+1})$.

Kalman filters are a popular choice for many tracking systems and mobile robotics localization. However, the approach suffers from the representation of $p(\vec{x}_t)$ as a Gaussian distribution that tracks only one hypothesis for $\vec{x}_t$. It also cannot solve well the global localization problem nor the kidnap problem. Although many variants have been proposed to remedy such drawbacks, it seems rather unsuitable for environments like robotic soccer, where Gaussian densities are not suitable models and global position or kidnapping situations are frequent. Alternatively, Markov Localization uses a piece-wise linear representation for $p(\vec{x}_t)$ and thus offers far more flexibility to model the effect of actions and the noise in sensor input. However, its computational requirements grow fast with the dimension of the space (the dimension of $\vec{x}$) and the resolution of the grid partitioning the space. As a result, accuracy is in direct trade-of with computational effort and seems impractical for inexpensive mobile robots like the SONY Aibo.

Monte-Carlo Localization and its variants have been shown to be superior to the Extended Kalman Filter [Gutmann and Fox, 2002] and to vanilla Markov Models [Gutmann and Fox, 2002]. They seem to be the most effective methods for the RoboCup 4-legged league [Röfer and Jüngel, 2004]. They also are known as *particle filters* and have parallels with non-parametric statistics in that they are universal density approximations. They do not need to impose a parametric model on the distribution of $p(\vec{x}_t)$. Some of their advantages are that they can accommodate arbitrary sensors, motion models and noise properties. They use resampling and naturally focus their computational effort in those areas believed to be more relevant, and controlling the number of particles allows regulation of computational effort. Moreover, they are significantly simpler to implement, as we will illustrate.

The MCL approach still has some disadvantages. The

1. First, a *resampling step* draws another m hypothesis $\vec{x}_t^{i,r}$ by selecting from S with replacement using the discrete distribution defined through the weights w_i.
2. Next, a *sampling step* uses a stochastic representation of the motion model. The motion model used has the form $p(\vec{x}_t|\vec{x}_{t-1}, \vec{a}_{t-1})$ in the theoretical derivations but is implemented as a subroutine `Motion_Model` that receives a pose $\vec{x}$ and an action $\vec{a}$ and returns a new pose $\vec{x}'$ that is the prediction of the movement. The code ensures that $\vec{x}'$ is distributed as $p(\vec{x}'|\vec{x}, \vec{a})$ (For the 4-legged league, we use the *max distance model* [Gutmann and Fox, 2002] that gives equal probability to positions up to a range equal to the time step times the maximum speed of the robot). Thus, for $i = 1, \ldots, m$ we have $\vec{x}_{t+1}^i \leftarrow$ `Motion_Model`$(\vec{x}_t^{i,r}, \vec{a}_t)$.
3. The process continues with an *importance sampling step*, where the sensor information is used to correct the prediction $\vec{x}_{t+1}^i$. Namely, $\nu_{t+1}^i \leftarrow p(\vec{o}_t|\vec{x}_{t+1}^i)$.
4. The last step of the process is to normalize the weights by $w_{t+1}^i \leftarrow \nu_{t+1}^i / \sum_{j=1}^m \nu_{t+1}^j$.

Figure 1: The 4 steps in the *iteration loop*.

stochastic nature of the algorithms implies that the number of particles can not be small. Vanilla MCL does not handle the kidnap problem well, and thus it needs to model the possibility of a kidnap by randomly introducing new particles to the space. It also seems slow to converge if the sensors are too accurate and until recently little theoretical foundation existed for some of its fixes [Thrun *et al.*, 2001]. A derivation from the theory of Bayes filters [Thrun *et al.*, 2001] shows that the vanilla MCL (under some general assumptions[1]) estimates the belief of the pose recursively. Moreover, this belief is modeled by a distribution $p(\vec{x}_t)$ (called the posterior) as is represented in non-parametric form by a sequence S of m hypothesis $S = \langle \vec{x}_t^i \rangle_{i=1,\ldots,m}$. For each hypothesis $\vec{x}_t^i$ there is a weight $w_i^t \in \Re$.

Initially all weights are equal to $1/m$ and the $\vec{x}_0^i$ are randomly drawn from the space (for $i = 1, \ldots, m$). At time $t > 0$, the algorithm repeats the *iteration loop* in Figure 1.

2.1 Illustration

We illustrate vanilla MCL with an example. Suppose we have a robot that moves on a linear array of 11 tiles. Suppose that the tiles are colored with gray-scaled tones such that the x-th tile has value x (for $x = 0, \ldots, 10$). The robot can decide to perform one of three actions `L`, `S`, `R` (that correspond to moving to the left tile, staying in the same or shifting to the right tile, respectively[2]). The goal of the robot is to stay centered (say in tile 5), although occasionally it is shifted to either end by an external agent. When it decides to carry out one of the actions, it may fail, due to conditions of the physical environment. Let us assume that we model this by a discrete probability distribution $p(x_{t+1}|x_t, a_t)$, that with probability $1/2$, it succeeds in performing the action and the possible failures share the remaining $1/2$ equally (thus, $p(4|5, \texttt{L}) = 0.5$, $p(5|5, \texttt{L}) = 0.25$, $p(6|5, \texttt{L} = 0.25)$, $p(4|5, \texttt{S}) = 0.25$, $p(5|5, \texttt{S}) = 0.5$, $p(6|5, \texttt{S}) = 0.25$, $p(0|0, \texttt{S}) = 0.5$, and $p(1|0, \texttt{S}) = 0.5$). To complete the method we need to decide upon $p(GrayValue|x)$. Let us assume that we model the noise in the light sensor by a discrete probability distribution where the chance of misreading the gray-scale value is inversely proportional to the distance from the true location. Let $r_{g,x} = 1/(1 + |g - x|)$ and let $p(GrayValue|x) = r_{GrayValue,x} / \sum_{g=0}^{10} r_{g,x}$. Note that the motion model and the sensor model would always be an approximation to reality obtained though experimentation and calibration.

Let us assume that the robot starts at position 3 with a population $S_0 = \langle 2, 2, 8, 2, 1, 10 \rangle$ of 6 particles and all weights set to $1/6$. The robot believes that it is in cell 2 because all weights are equal; so it instructs its motion control to go right. Let's assume that the robot sensor reads gray level 3. The localization will resample S_0 using some random indexes $\langle 3, 2, 3, 3, 1, 5 \rangle$; thus, we have $\langle 8, 2, 8, 8, 2, 1 \rangle$ as the population of particles to which the `Motion_Model` is to be applied. Suppose that the effect is as follows `Motion_Model`(8,R)=7, `Motion_Model`(2,R)=3, `Motion_Model`(8,R)=9, `Motion_Model`(8,R)=9, `Motion_Model`(2,R)=2, and `Motion_Model`(1,R)=2. That is, the self-localization uses a model that generates two moves that fail (the first and the fifth). Using the sensor reading of we obtain $p(3|7)$=0.06, $p(3|3)$=0.27, $p(3|9)$=0.05, $p(3|9)$=0.05, $p(3|2)$=0.14 and $p(3|2)$=0.14. Normalizing this values results in the new belief $S_1 = \langle 7, 3, 9, 9, 2, 2 \rangle$ and weights $\vec{w}_1 = \langle 0.08, 0.38, 0.07, 0.07, 0.20, 0.20 \rangle$. Note that although half of the hypothesis are to the right of cell five, the correction by the sensor model makes location 3 the most likely. Figure 2 illustrates this iteration.

3 Hierarchical MCL

We present two methods that organise the particles in a hierarchical data structure. Because we assume the pose

[1] For example, the Markov assumption says that the probability of observing $\vec{o}_t$ from pose $\vec{x}_t$ does not depend on all previous observations and actions; namely, nobody is moving landmarks.

[2] On the 0-th tile the robot cannot move left and on the 10-th tile it can not move right.

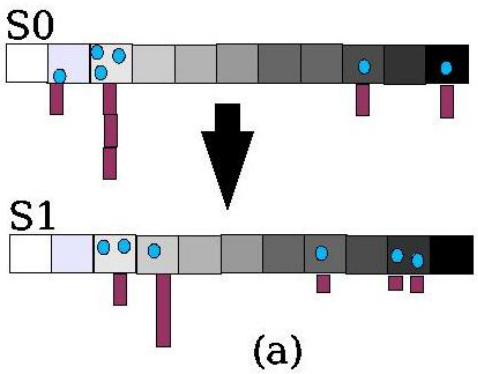

Figure 2: One iteration loop of the vanilla MCL has 4 steps that transform the beleif representaiton S_0 into S_1.

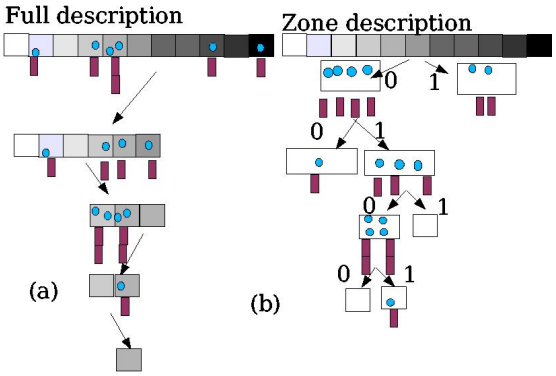

Figure 3: The two belief representations for a hierarchical MCL. (a) Particles that represent poses. (b) Particles with values in $\{0, 1\}$ that represent which half of the dimensional axis the location is believed to be.

$\vec{x}$ is multidimensional we will use k-dimensional trees or kd-trees [Bentley, 1975a]. We introduce the idea with the one-dimensional example from the previous section. Note that if more robustness to the stochastic nature of the MCL method was required, the number m of hypothesis would have to be much larger than 6. Also, in realistic environments, the hypothesis space would not be so simple as one out of 11 cells. Thus, also more precision in the localisation would require a large number of particles. Clearly, the iteration loop of MCL is proportional to $O(m)$, the number m of particles.

To aid the intuition, consider a scenario where self-localisation was required only to determine if the robot is on the left side of the array of cells as opposed to the right side, then we essentially have a very small space of hypothesis. The robot in the previous illustration needs to remain close to the centre, so such loose self-localisation is sufficient for its control. Moreover, much fewer particles (and processing time) are required to make this decision. If further resolution/accuracy is required for the location of the robot, then the method proceeds under the condition that it has determined which side of the array of cells it is in. Thus, there is no need to process particles/hypothesis of higher resolution on the other side. Applying the idea recursively, if the robot has already determined it is in the first half, then it will apply vanilla MCL to determine if it is in the first or second quarter of the space. Again, once it has determined a quarter, then it partitions the hypothesis space into two and uses a local vanilla MCL to determine which half to proceed.

Let m_0 be the number of particles at depth 0 (the top level of the hierarchical structure). In our first version (referred here as *full description*) the method uses particles whose pose descriptor is the vector $\vec{x}$. That means that at the top level we have a vanilla MCL with particles with full pose resolution. At depth d, we have m_d particles whose pose descriptor is again a full vector $\vec{x}$. The *zone description* version has discrete universe $\{0, 1\}$ of size two for all levels. At depth 0, a pose with value 0 means the first half of the partition in the kd-tree while a value of 1 means the second half of the partition.

Figure 3(a) illustrates the data structure for the vanilla MCL illustration step described in the previous section. For the *full description* we refer to Figure 3 (a). The data structure at level 0 holds particles with 1-dimensional poses covering the entire domain of the first coordinate. It also has weight for each particle and poses have full resolution. The particles at depth 0 represent a distribution $p(\vec{x})$ where the agent knows it is in the first half of the domain. It follows a link to a new set of particles that present information about $p(\vec{x}|\texttt{In\_first\_half})$. Using the particles at depth 0 that are in the first half and the particles at depth 1 in the left child, the localisation determines that the robot is in the second quarter of the strip of tiles. Now if more precision is required for a decision/behaviour, the self-localisation can explore the particles at depth 2. These particles (and implicitly the particles in parent nodes) correspond to a representation of $p(\vec{x})|\texttt{In\_second\_quarter}$) . The self-localisation determines a belief that it is among the 4-th and 5-th cells. Finally, by going one level down, tile 5 is determined as the current pose (the belief represented by the data structure).

Note that the data structure is a tree, but we have chosen to illustrate only the path that provides a belief of location where accuracy increases with depth. Second, note that for the *full description* version, poses (particles) and weights are of the same type as for the vanilla MCL. Therefore, the same motion model and sensor model could directly apply to them when computing

with particles for the vanilla MCL. This makes a large part of the code re-usable; however, we will need to adjust sensor models for very precise sensors to be useful in higher levels of the structure. We will say more about this later. More importantly, the iteration loop in the localisation algorithm performs the 4-steps of resampling, sampling, importance sampling and normalisation only for the particles in a path on the tree from the root to the leaf (and not over all particles).

We argue that because work happens on a path from the root to the leaf, the overhead of the hierarchical data structure is negligible (and in fact less than in other hierarchical data structures like Octrees where the complexity has remained proportional to the total number of nodes in the tree [Burgard *et al.*, 1998]). We can have two schemes to allocate the number of particles to nodes of the tree. First, we allocate a number m_0 of particles to each node. Then the complexity of the iteration loop is $m_0(depth+1)$ where $depth$ is the depth of the leaf (recall the root is depth 0). If vanilla MCL is using m particles, $m_0 = m/(depth+1)$ results in a hierarchical MCL that has the same time complexity as vanilla MCL but can reply to global localisation queries in $1/(depth+1)$ of the time. Alternatively, we can allocate the number m of particles that vanilla MCL would use uniformly across levels of the tree; that is, $m_i = 2m_{i+1}$ we have $m = m_0 + 2m_1 + 4m_2 + 8m_3 + \ldots 2^{depth}m_{depth}$ but the complexity of the algorithm is only $\sum_{d=0}^{depth} m_d = m_0(1 - 1/2^{depth})$. Thus, a value $m_0 = m/(1 - 1/2^{depth})$ makes hierarchical MCL have the same CPU requirements as vanilla MCL. Another way to think about this is, under this second allocation scheme, let $m_o = m$, use the number of particles at depth 0 that one would use for vanilla MCL and half them for each child (the overhead will be negligible).

This also means that the algorithm's CPU time requirements for hierarchical MCL depends less on the degree of dispersion, concentration or clustering of the particles; that is, of the certainty of the location. If one is prepared to trade accuracy for CPU time, then we would observe a larger ratio of improvement in CPU time as the tree is deeper. It means that the difference in the speed of our algorithm with vanilla MCL is more noticeable as we handle domains with requirements for both loose and high accuracy. A bit of reflection confirms this, since our methods reduces to vanilla MCL when the chosen depth is 0. That is the hierarchical method will be faster for queries of location within a zone, but of equivalent CPU for the high accuracy queries. With the many particles as vanilla MCL, we would expect very solid robustness to stochastic noise.

It is important to mention that variations and additions to MCL methods are easily adapted to our methods. This is because our methods can be regarded as stacked vanilla MCLs. Thus, in particular, techniques like randomly adding particles to allow resilience to the kidnap problem are applicable as well as dual MCL.

Now we are in a position to describe a k-dimensional hierarchical structure. That is, we assume the dimension of a pose $\vec{x}$ is k. However, we will present illustrations with $k = 2$. Note that a pose for a robot in a soccer field may be bi-dimensional (x, y) or perhaps tri-dimensional (x, y, ϕ) (if we consider the orientation ϕ). Some environments, like submersible robots may require 3 Cartesian values for location (x, y, z) and another three angular values (pitch, yaw and roll) to indicate orientation (in that case, the dimension of the pose would be 6).

Our k-dimensional hierarchical structure is based on the notion of a binary kd-tree. Binary kd-trees are tree structures first introduced by Bentley [Bentley, 1975a] to solve the problem of associative retrieval of multidimensional keys. Each node in the tree has an associated discriminant $\delta \in \{1, 2, \ldots, k\}$. All the nodes in the same level have the same discriminant, and discriminant values are assigned cyclically in this way: the root node has discriminant 1, its two subtrees have discriminant 2 and so on up to the $k - th$ level in which the discriminant is k. In level $k + 1$ the discriminant is 1 again and the cycle described before is repeated. Each node of the hierarchical data structure is a vanilla MCL that uses the particles at that level to decide between two possibilities for the pose. Namely, for a node ν_n at depth d, we test if the pose $(d \bmod k) + 1$ coordinate is less that a value v_{ν_n} or its is greater than this value. For example, consider the illustration in Figure 4 for a pose vector (x, y). At the root, the discriminant is a value x_0 to compare against the fist component of the pose vector and determines that the belief is either that $x \leq x_0$ or that $x > x_0$. The figure illustrates the case when the vanilla MCL at this level determines that the robot believes the pose is such that $x \leq x_0$. At depth one, and on the left branch of the kd-tree, the discriminant is a value y_1 for the second coordinate y of the pose. continuing in this fashion, the path of the kd-tree illustrates the determination of a position. Note that the illustration only shows a path in the kd-tree and not the entire tree. The partition of the domain by the discriminant values is illustrated by solid lines (while the grid is shown with dashed lines).

For our algorithm, each node of the kd-tree has associated with it a vanilla MCL. In full representation, as in the 1-dimensional case, there are m_d particles for a node ν_d at depth d. These particles are set $S_t^{\nu_d}$ that are managed using the iteration loop with its four steps. Since the kd-tree is binary, the allocation of particles to nodes can follow any of the two schemes we already discussed for the 1-dimensional case. Again, the computational overhead is now even more negligible, since the dimension k is at least a linear factor in the complexity

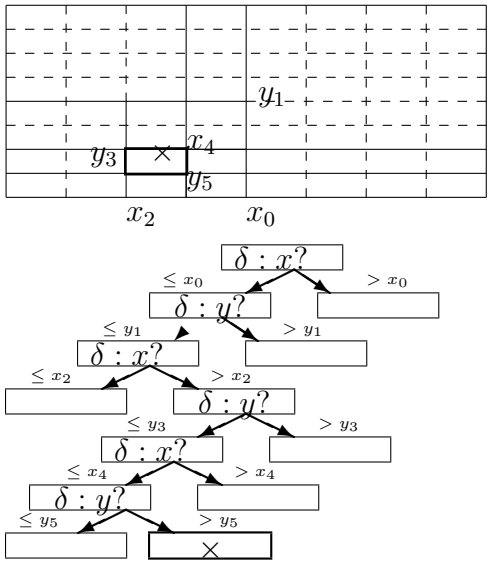

Figure 4: A *kd*-tree represents a grid of the domain of posses. A cell is identified by a path that alternates discriminating among the coordinates of the pose vector.

of vanilla MCL. That is, with higher dimensions, our algorithms become more affordable with respect to vanilla MCL. A zone representation for the multi-dimensional case uses particles with values in $\{0, 1\}$ as in the 1-dimensional case.

4 Implementation and experiment

There are four important points regarding the successful implementation of our methods. In fact, our methods are not as effective if these points are not incorporated.

1. The first point is that particles assigned to a node in the data structure do not migrate to siblings/neighbours in the tree, even if the actual pose of the robot is very far from the grid-cell holding those particles. To illustrate this consider again Figure 4. The grid-cell marked $\times$ represents a pose $\vec{x} = (x, y)$ where $x_2 < x \leq x_4$ and $y_5 < y \leq y_3$. Let us suppose that the robot is actually in the top-right corner of the domain with values for x much larger than x_0 and values for y much larger than y_1. Moreover, the robots is over there for a long while. Vanilla MCL (or Markov Localisation) eventually would move most of its particles (respectively probability density) to that top-left corner. For vanilla MCL, the number of particles representing a pose in the grid-cell marked $\times$ (or a pose nearby this grid-cell) would decrease rapidly. Only because of the random placement of particles in the domain (the mechanism to protect against the kidnap problem), would particles nearby the grid-cell marked $\times$ would appear.

   This is not the case in our methods, a node at depth d remains with m_d particles associated with it, and therefore attached to values close to its grid-cell. This is true for all levels. For example, the left child of the root in Figure 4, which covers the region $x \leq x_0$ and $y \leq y_1$ also would keep m_1 particles even if the robot believes and has being for a while in the top right corner of the domain. Note that this is possible because at depth 0 there would be enough particles (and corresponding weights) to determine that the pose has an x value greater than x_0. This allows our methods to have rapid coverage in all areas of the domain.

2. Second, we allow particles associated with a node to represent a pose outside the grid-cell (or region) of the domain associated with such node. There is a fundamental reason for this. Suppose we are in the *sampling step* of a pose in the grid-cell marked $\times$ and this pose is (x_4, y_3) (that is, on the border of the cell). Moreover, lets assume that such particle has the highest weight (so the robot believes it is actually at $\vec{x} = (x_4, y_3)$). And lets assume the robot wants to go to the centre of the domain, so it has performed an action a to move to the top-right. So $(x_4 + \epsilon, y_3 + \epsilon) \leftarrow$ `Motion_Model`$((x_4, y_3), a)$. Alto ugh the new particle $(x_4 + \epsilon, y_3 + \epsilon)$ is outside the grid-cell, it is kept associated with the vanilla MCL at the node marked $\times$ in the *kd*-tree. What we achieve with this is a complete representation from the point of view of the belief at that node. That is, the sum of the weights of particles with pose values outside the region corresponding to the node is the representation (by the local vanilla MCL at that node) that the robot **is not** in that region.

   Note that our representation of a robot being somewhere else besides the region of coverage is similar to the approach used for speeding Markov Localisation with *selective update* [Fox *et al.*, 1999a]. However, also note that the pose values outside a node's region must be close to the region's legal values. We call this technique "collecting the pose values". Namely, we bound ϵ to a small value. In fact all particles in a node ν with values $\vec{x}_\epsilon$ outside the region of ν are given pose values so that with high probability `Motion_Model`$(\vec{x}_\epsilon)$ returns a value inside ν's region. This is important because, when the robot's belief moves back to being close to the grid-cell in question (because the robot is coming back and maybe it even gets back into the region of the node that abandoned much earlier), the particles of that node ν must rapidly represent belief of a pose in the region corresponding to ν. Note that while the robot

believes to be outside the region for ν, it will not update particles in the vanilla MCL at ν (that is where the CPU time savings come from in our method). However, when updating particles at ν, it is because it believes is is at ν, at least at upper levels in the tree. It would be contradictory if at ν the belief indicated a pose far from the region for ν.

3. If a sensor has very high noise or/and low precision, it is incorporated into the sensor model of only the vanilla MCL at shallow nodes in the tree (shallow depth). Such a sensor in deeper nodes is not informative, it would at best just confirm that we are at the node (and would not provide further resolution within the region for that node).

   A sensor of high accuracy is modelled closely for deep nodes in the tree. However, it is converted to what we call a *virtual low resolution sensor* for its use in high nodes of the tree (shallow depth). We shall explain this further so the rationale behind our approach is justified.

   We describe why high precision sensors are a problem for the vanilla MCLs at shallow nodes. Consider again our one-dimensional example, but now we have 128 cells (and not just 11 as before). The assumption that each of these cells has a gray-scale illumination that identifies the cells remains, also the lowest value is at the leftmost cell, and values linearly increase to the rightmost cell. To illustrate our point we suggest here a sensor that is very precise. Say, it returns the gray-value plus or minus the gray-scale value of one cell (that is the sensor gives information of which out of 3 cells the robot is). What happens at the top level when we have full description particles and essentially we are deciding if the robot is on the fist or second half of the array of cells? Consider a particle in the second half with value 80, while the robot is actually at 40 and has just read the value 41. Then, during the importance sampling, $p(o_t|\vec{x}_{t+1})$ would be zero most of the time. In our example, the probability of reading 41 given that one is at 80 is 0 for such a high precision sensor. The problem is that this happens for almost all particles, even those that are in the first half. As a result, the vanilla MCL may artificially (as a result of stochastic noise) assign a slight difference but just few more particles to the wrong half. From there, the vanilla MCL would not be to review its belief of being in the wrong half because, on a high precision sensor, $p(o_t|\vec{x}_{t+1})$ is equally poorly informative for all $\vec{x}_{t+1}$ in the wrong half.

   What we need is a sensor model for $p(o_t|\vec{x}_{t+1})$ that presents the high precision sensor as a low precision sensor where

   (a) $p(o_t|\vec{x}_{t+1})$ is different from 0 for all particles in the domain,

   (b) the modes (points where the distribution is maximum) are the same as for the high precision sensor.

   (c) the distribution for the low precision sensor is (locally) strictly monotonic.

   As a consequence of these properties, the vanilla MCLs will converge their belief to the poses of the reading sensors. Thus, in our example, $p(41|80)$ would not be zero, but a very small value. Moreover, while $p(41|41)$ would be the largest value, $p(41|x_l) > p(41|x_2)$ for all $41 < x_1 < x_2$ (for instance $p(41|41) < p(41|60) < p(41|80)$.

   As a result, when at the root of our hierarchical MCL, the particles there would indeed be using the sensor readings for revising the belief. It is intuitively clear, that even if a robot was at 80 and read 40 from a very high precision sensor, it should revise its belief and at least believe it is in the first half of the 127 cells (rather than trust its odometry too much). Moreover, the availability of a set of particles will allow particles close to the value 40 to re-test the hypothesis of being even close to 40 in the next iteration loop.

4. Recall that the number of particles associated with a node is constant. Although 90% of those particles are processed using the vanilla MCL iteration loop, we always replace 10% of the particles with random particles in the entire domain. This is implemented by skipping the resampling step and the sampling step for a randomly selected 10% of the particles. New particles are feed directly to the third step *importance sampling* of the iteration loop. Feeding randomness into ML or MCL is a standard technique because the possibility of having an incorrect belief must be incorporated regularly into the method (and thus, it works as protection to the kidnap problem). In our methods, it becomes more important since particles at a node will not be updated for long periods of time if the robot is not operating in the region represented by that node. More importantly, we must rapidly revise the belief at nodes high up.

5. Our methods work the *importance step* under what we call a *conditioned approach*. We now explain what this means and justify its use. Consider again Figure 4 and a situation in where the robot believes to be in the grid-cell marked $\times$ with a pose $\vec{x} = (x_4, y_3)$. That is, if queried for the pose, the localisation model would reply with an region corresponding to the path in Figure 4; the depth of the node used would depend on the requested accuracy.

For just what side of x_0 the root node would be used, for example. With this information, a robot aiming for the middle of the domain would issue a motion a for $(x_4 + \epsilon, y_3 + \epsilon)$, and read sensor information s_t indicating another grid-cell. Then, the iteration loop would be performed on the vanilla MCL at the root. Once the iteration loop is completed, a child is selected to continue the iteration loop one level deeper. In our example, the left child of the root would be selected and an iteration loop with the new sensor value performed there. Once again, the iteration loop is completed at the depth 1 before using it to select the child to proceed. This would be again a left node. But once the iteration loop is completed on this child, the belief in this child would indicate a value for the x-coordinate of the pose greater than y_3, so that the child selected at this depth would be the right node. That is, although the path in Figure 4 is the path that represents the belief at a certain time step, after a motion and a sensor reading are executed, it may not be that the iteration loop is executed in the nodes of this path. The updating of the belief is on another path that is obtained by updating the belief at a node and following the child using the updated belief at that node.

This strategy is sound, since it provides an updating rule of the belief always using the most recent version (the most up-to-date belief) as suggested by all localisation recursive methods based on a Bayesian formalism (including Kalman and Markov Localisation). If fact, if for whatever reason the updating of the belief was halted halfway and did not complete its revision of nodes from the root to a leaf, the data structure still represents a localisation belief where the most influential nodes (the ones at the root) are as accurate as possible.

4.1 Experiment

We have implemented our methods and embedded them is a simulation testing system. Embedding in a simulation for testing self-localisation algorithms is common practice [Burgard *et al.*, 1998; Fox *et al.*, 1999a; 1999b; Fox, 2003; Thrun *et al.*, 2001] because we can measure exactly the physical (virtual) position of the robot, and several error conditions can be tested for sensors or for errors in motion. We have confirmed empirically the claims discussed earlier. Figure 5 shows the global localisation error and the tracking performance of our method against some previous versions of vanilla MCL. The robot is initially placed in one extreme of the domain with a belief of random poses and weights all equal to $1/m$ where m is the number for particles. Global localisation is evaluated by the speed by which the robot reduces the error to less than 5 (which is the error in the

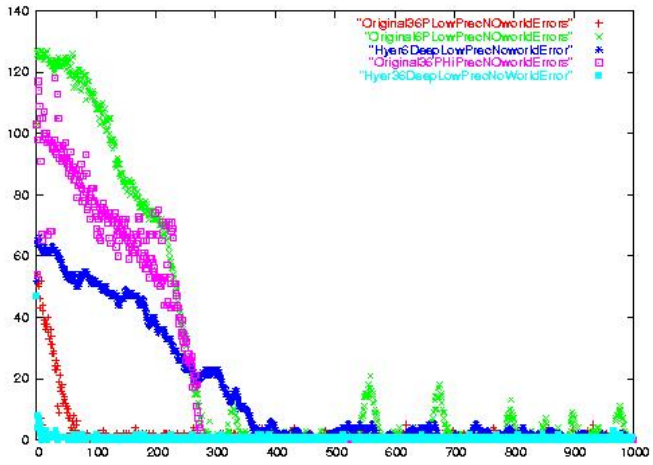

Figure 5: Results that illustrate that our method has the fastest discovery of localisation and also the most stable tracking (minimising error between true position and believed position).

sensor). Tracking is evaluated by the robot maintaining the error this low. The worst performance is by the vanilla MCL with a ratio of 6 particles to 128 grid-cells and a low quality sensor. The plot indicates this version with green $\times$ symbols. This version not only takes about 300 iterations to localise but also has high peaks during tracking. By incrementing the particles to 36 particles and a high precision sensor, the error in global localisation improves just a bit (this is in pink with $\square$) . The tracking is much better but hardly compensates for the weak performance in global localisation. Interestingly, if this version of vanilla MCL is provided with a low precision sensor, we obtain the red + plot. Global localisation is much effective, requiring only 80 iteration loops, and the penalty in tracking in negligible. This illustrates that our technique of a *virtual low resolution sensor* is effective for vanilla MCL (and therefore for the root and other nodes of our hierarchical MCL).

Our method with the first allocation of particles scheme are represented in dark blue and symbol $\star$. That is all nodes have 6 particles, even the root has 6 particles for 128 grid cells. This is the method that runs the fastest (together with green $\times$). It shows an error in global localisation that is better than the two vanilla MCLs. Although it takes 380 steps of the iteration loop, it always has the correct answer for which half of the domain (that is, at the root there is no error). Its tracking performance is solid and definitely acceptable.

Our method (36 particles for each 128 grid-cells at the root and then half the number of particles the parent per child) is shown in light blue and $\blacksquare$. The method not only is efficient (requiring competitive CPU as the

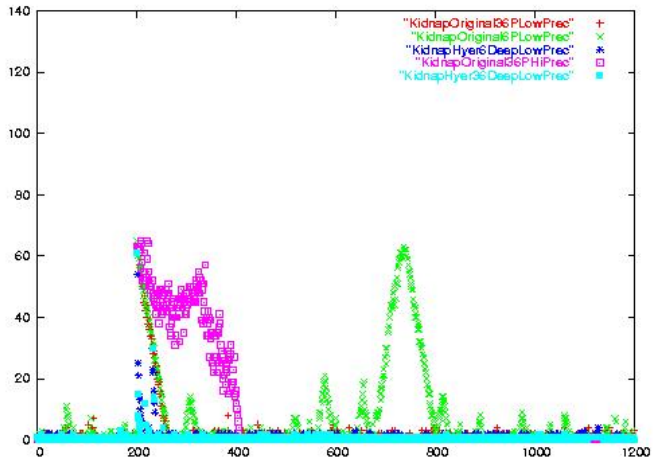

Figure 6: Results that illustrate that our method has the fastest recovery after a kidnapping and also the most stable tracking afterwards.

vanilla MCL with pink □), but its reduction of global localisation is remarkable. It offers the smallest global localisation error, recovering from a total neutral belief in less than five iteration loop steps. The tracking is also remarkable as can be appreciated in the plot in Figure 5.

We have also considered the kidnap problem. Figure 6 displays the result of a kidnap after 200 iteration loop cycles for the methods discussed previously. We can see that the vanilla MCL with a ratio of 6 particles for every 128 grid-cells is very unstable. Other vanilla MCLs do recover from the robot being translated 63 grid cells away after converging to a location; however, our methods show the fastest recovery.

5 Discussion

We now contrast of methods with the literature.

- The approach called *kd*-trees in [Gutmann and Fox, 2002] when referring to [Thrun *et al.*, 2001] as an efficiency improvement is because in [Thrun *et al.*, 2001] the data structure is called *kernel density trees*. The data structure is used to efficiently represent a piece-wise linear distribution for the Dual MCL. This is not a mechanism to structure the particles efficiently. It provides efficient manipulation of pice-wise distributions. Although a reference to Bentley [Bentley, 1980] appears in [Thrun *et al.*, 2001], this is because Bentley suggests a possibility of representing piece-wise linear probability densities and operations on densities using multi-dimensional trees. The *kd*-tree by Bentley was originally used for multidimensional associate searching [Bentley, 1975a; 1975b; 1979; Gaede and Günther, 1998; Sproull, 1991].
- Because Markov Localisation imposes a grid that grows very rapidly with the dimension of the pose vector, or with the desired accuracy, techniques were developed for improving performance of implementations. *Pre-computation of the sensor model* stores it as a look up table [Fox *et al.*, 1999a] and *selective update* labels some cells in the grid as passive, because their probability is very small, and skips their updating. However, these techniques demand high memory requirements still, are laborious to implement (as they demand monitoring of passive/active cells) and are brittle to the parameters/thresholds chosen. They also lack theoretical intuition and are effective when the robot has a clear picture of where it is (not in the kidnap problem). It should be noted that the management of passive and active cells is not trivial. The motion model increases the active cells (adds cells to the list of active cells as part of the prediction step that takes account of the uncertainty of actions) while the perception model will delete some active cells and place them among the passive cells. The approach by [Burgard *et al.*, 1998] suggests using an Octrees to represent poses (x, y, ϕ) consisting of two coordinates x and y (for a robot's location in 2D) and an orientation *phi*. This allows to refine the resolution if the localisation software dynamically adjust the branches of the tree at each localisation step. The overhead of dynamically updating the tree is not trivial. The Octree representation reduces significantly the complexity of Markov Localisation form the large number of cells in the grid (an exponential number in the height of the tree), to just the number of nodes in the tree (which of course is not complete). The normalisation and updating complexity becomes proportional to the number of leaf nodes in the Octree. Our methods improve performance because we are concerned with only the particles in a path from the root to a leaf in our *kd*-tree.
- The first paper that seems to deal with the issue of managing the particles for efficiency is [Fox, 2003]. However, the problems is rather different, since the issue is how to keep and adequate number of particles (and random particle introduction) in a vanilla MCL. The allocations schemes proposed here solve this problem. The particle number management is handled as they are distributed along the nodes of our *kd*-tree.

6 Conclusions

We have implemented a hierarchical approach to Monte Carlo Localisation. It allows the incorporation of sensors of different accuracy or information content (the sensor could be very accurate but there could be many

poses where that reading could be obtained). It allows to rapidly secure information about the pose and increase the accuracy for response time.

The methods become very effective in comparison with standard MCL approaches (we have referred to them as vanilla MCL). In should be noted that MCL approaches are now regarded as superior to Markov Localisation of Kalman filters in many scenarios. We introduced five techniques that complement our methods and that provide robustness to the kidnap problem or to the stochastic noise implicit in MCL. These provide particle management, mechanisms to represent complementary information, generalisation of high resolution sensors to low resolution sensors, rapid belief revision, and hierarchical conditioning of belief update. As a result, the methods become very effective and more competitive than previous approaches.

References

[Bentley, 1975a] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[Bentley, 1975b] J.L. Bentley. A survey of techniques for fixed radius near neighbor searching. Technical Report Report STAN-CS-78-513, Dept. Comput. Sci., Stanford Univ., Stanford, CA, 1975.

[Bentley, 1979] J.L. Bentley. Decomposable searching problems. *Information Processing Letters*, 8:244–251, 1979.

[Bentley, 1980] J.L. Bentley. Multidimensional divide and conquer. *Communications of the ACM*, 23(4):214–229, 1980.

[Betke and Gurvits, 1994] M. Betke and L. Gurvits. Mobile robot localization using landmarks. *IEEE Int. Conf. Robotics and Automation*, volume 2, pages 135–142, San Diego, CA, May, 9-13 1994.

[Betke and Gurvits, 1997] M. Betke and L. Gurvits. Mobile robot localization using landmarks. *IEEE Transactions on Robotics and Automation*, 13(2):251–263, April 1997.

[Burgard *et al.*, 1998] W. Burgard, A. Derr, D. Fox, and A. Cremers. Integrating global position estimation and position tracking for mobile robots: the dynamic markov localization approach. *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS '98)*, pages 730–735, Victoria, BC, Canada, October 1998. IEEE.

[Dahm and Ziegler, 2003] I. Dahm and J. Ziegler. Adaptive methods to improve self-localization in robot soccer. G. A. Kaminka, et al eds, *RoboCup 2002: Robot Soccer World Cup VI*, pages 393–408. Springer Verlag Lecture Notes in Artificial Intelligence, 2003.

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

[Fox *et al.*, 1999b] D. Fox, W. Burgard, and S. Thrun. Monte Carlo localization: Efficient position estimation for mobile robots. *Sixteenth National Conf. on Artificial Intelligence*, pages 343–349, Orlando, CA, July 18-22 1999. AAAI, AAAI Press.

[Fox, 2003] D. Fox. Adapting the sample size in particle filters through KLD-sampling. *Int. J. of Robotics Research (IJRR)*, 22(12):985–1040, December 2003.

[Gaede and Günther, 1998] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, June 1998.

[Gutmann and Fox, 2002] J.-S. Gutmann and D. Fox. An experimental comparison of localization methods continued. *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2002.

[Leonids and R., 1995] J. Leonids and Motwani R. The robot localization problem. *Algorithmic Foundations of Robotics*, pages 269–282, 1995.

[Röfer and Jüngel, 2004] T. Röfer and M. Jüngel. Fast and robust edge-based localization in the SONY four-legged robot league. *7th Int. Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences)*, Padova, Italy, July 2004. Springer Verlag Lecture Notes in Artificial Intelligence.

[Sproull, 1991] R.F. Sproull. Refinements to nearest neighbour searching in k-dimensional trees. *Algorithmica*, 6:579–589, 1991.

[Thrun *et al.*, 2000] S. Thrun, D. Fox, and W. Burgard. Monte Carlo localization with minsture proposal distribution. *National Conf. on Artificial Intelligence*, pages 859–865, Austin, USA, 2000. AAAI, MIT Press.

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

[Thrun, 1998] S. Thrun. Bayesian landmark learning for mobile robot localization. *Machine Learning*, 33(1):41–76, 1998.

[Veloso *et al.*, 1998] M. Veloso, W. Uther, M. Fujita, M. Asada, and H. Kitano. Playing soccer with legged robots. *Proc. IROS-98, Intelligent Robots and Systems Conf.*, Victoria, Canada, October 1998.