ADDRESSING COMPLEXITY ISSUES IN A REAL-TIME PARTICLE FILTER FOR ROBOT LOCALIZATION

Dario Lodi Rizzini, Francesco Monica, Stefano Caselli
Dipartimento di Ingegneria dell'Informazione, Università di Parma,
Viale G. P. Usberti, 181/A, 43100 Parma, Italy
{rizzini, fmonica, caselli}@ce.unipr.it

Monica Reggiani
Dipartimento di Informatica, Università di Verona,
Ca' Vignal 2, Strada Le Grazie 15, 37134 Verona, Italy
reggiani@metropolis.sci.univr.it

Keywords: Robot Localization, Real-Time, Particle Filters

Abstract: Exploiting a particle filter for robot localization requires expensive filter computations to be performed at the rate of incoming sensor data. These high computational requirements prevent exploitation of advanced localization techniques in many robot navigation settings. The Real-Time Particle Filter (RTPF) provides a tradeoff between sensor management and filter performance by adopting a mixture representation for the set of samples. In this paper, we propose two main improvements in the design of a RTPF for robot localization. First, we describe a novel solution for computing mixture parameters relying on the notion of effective sample size. Second, we illustrate a library for RTPF design based on generic programming and providing both flexibility in the customization of RTPF modules and efficiency in filter computation. In the paper, we also report results comparing the localization performance of the proposed extension and of the original RTPF algorithm.

1 INTRODUCTION

Robot localization is the problem of estimating robot coordinates with respect to an external reference frame. In the common formulation of the localization problem, the robot is given a map of its environment, and to localize itself relative to this map it needs to consult its sensor data. A particularly effective approach to solve this problem is the *probabilistic* one, due to the uncertainty affecting sensor data and movement execution. For this reason, bayesian filtering has become the prevailing paradigm in recent works on localization (Elinas and Little, 2005; Sridharan et al., 2005).

A stochastic estimator provides a result expressed in the form of a *probability density function* (PDF) represented like a continuous function by Gaussian filters (Kalman Filter, EKF) (Leonard and Durrant-Whyte, 1991; Arras et al., 2002) or a discrete decomposition of the state posterior by nonparametric filters. The main nonparametric algorithm is called *Particle Filter* (Fox et al., 1999) and relies on *importance sampling* (Doucet et al., 2001). With importance sampling, the probability density of the robot pose is approximated by a set of samples drawn from a proposal distribution, and an importance weight measures the distance of each sample from the correct estimation.

The nonparametric approach has the advantage of providing a better approximation of the posterior when a parametric model does not exist or changes during iteration, e.g. in initialization or when environment symmetries determine a multi-modal PDF. Even if techniques like *Multi-Hypothesis Tracking* (Arras et al., 2002) attempt to manage multi-modal distributions, particle filters are more efficient and can represent all kinds of PDFs, including uniform distributions. Moreover, particle filters avoid linearizations that can lead to poor performance and divergence of the filter for highly nonlinear problems.

Unfortunately, particle filters suffer from computational complexity due to the large number of discrete samples of the posterior: for each sample a pose update, a correction and a resample step are performed. Since localization can be performed slowly with respect to the usual movement and tasks of the robot, it would be conceivable to perform localization over a large time interval. Therefore, there have been attempts to adapt the number of samples (Fox, 2003). However, during an excessive time interval uncertainty increases and many useful observations are dropped; a proper interval to complete a particle filter iteration should be approximately equal to the rate of incoming data. A trade-off must therefore be reached between time constraints imposed by the need of collecting sensor data incoming with a given rate and the number of samples determining localization performance.

The *Real-Time Particle Filter* (RTPF) (Kwok et al., 2004) is a variant of a standard particle filter addressing this problem. This algorithm relies on the decomposition of the posterior by partitioning the set of samples in smaller subsets computed taking into account the sensor data received in different time intervals. By choosing the proper size for partitions, a particle filter iteration can be performed aligned with the sensor acquisition cycle.

While RTPF represents a remarkable step toward a viable particle filter-based localizer, there are a few issues to be addressed in developing an effective implementation. RTPF convergence is prone to some numerical instability in the computation of important parameters of the algorithm, namely the coefficients of the mixture constituting the posterior. Furthermore, even adopting RTPF as the basic architecture, the design of a flexible and customizable particle filter remains a challenging task. For example, life cycle of samples extends beyond a single iteration and covers an estimation windows in which mixture posterior computation is completed. This extended life cycle of samples impacts over software design. Moreover, RTPF addresses observations management and derived constraints. A good implementation should be adaptable to a variety of sensors.

This paper proposes improvements in both the algorithmic solution and the implementation of RTPF. In section 2, a novel approach in the computation of mixture weights based on the effective number of samples is proposed. This approach simplifies RTPF and tries to avoid spurious numeric convergence of gradient descent methods. In section 3, a localization library implementing a highly configurable particle filter localizer is described. The library takes care of efficient life cycle of samples and control data, which is different in RTPF and standard particle filter, and supports multiple motion and sensor models. This flexibility is achieved by applying *generic programming techniques* and a *policy* pattern. Moreover, differing from other particle filter implementations (e.g., CARMEN (Montemerlo et al., 2003)), the library is independent from specific control frameworks and toolkits. In section 4, simulation and experimental results are reported and compared with the original RTPF performance. These results confirm the effectiveness and viability of the proposed algorithm and its implementation.

2 ADDRESSING ALGORITHMIC COMPLEXITY

In particle filters, updating the particles used to represent the probability density function (potentially a large number) usually requires a time which is a multiple of the cycle of sensor information arrival. For example, range scanners return hundreds of values per scan, at a rate of several scans per second; vision sensors often require advanced algorithms to identify visual landmarks (Se et al., 2002; Sridharan et al., 2005) draining computational resources from the process of localization.

Naive approaches, yet often adopted, include discarding observations arriving during the update of the sample set, aggregating multiple observations into a single one, and halting the generation of new samples upon a new observation arrival (Kwok et al., 2004). These approaches can affect filter convergence, as either they loose valuable sensor information, or they result in inefficient choices in algorithm parameters.

An advanced approach dealing with such situations is the Real-Time Particle Filters (RTPF), proposed in (Kwok et al., 2004), which will be briefly described in the following.

2.1 REAL-TIME PARTICLE FILTER

Assume that the system received k observations within an *estimation window*, i.e. the time required to update the particles. The key idea of the Real-Time Particle Filter is to distribute the samples in sets, each one associated with one of the k observations. The distribution representing the system state within an estimation window will be defined as a *mixture* of the k sample sets as shown in Figure 1. At the end of each

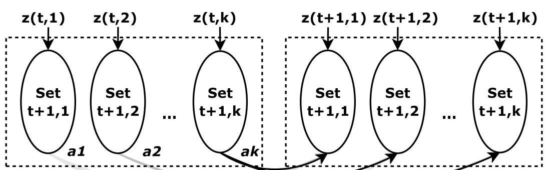

Figure 1: RTPF operation: samples are distributed in sets, associated with the observations. The distribution is a mixture of the sample sets based on weights α_i (shown as a_i in figure).

estimation window, the weights of the mixture belief are determined by RTPF based on the associated observations in order to minimize the approximation error relative to the optimal filter process. The *optimal belief* could be obtained with enough computational resource by computing the whole set of samples for each observation. Formally:

$$Bel_{opt}(x_{t_k}) \propto \int \dots \int \prod_{i=1}^{k} p(z_{t_i}|x_{t_i}) \cdot p(x_{t_i}|x_{t_{i-1}}, u_{t_{i-1}}) \cdot Bel(x_{t_0}) dx_{t_0} \cdots dx_{t_{k-1}} \quad (1)$$

where $Bel(x_{t_0})$ is the belief generated in the previous estimation window, and z_{t_i}, u_{t_i}, x_{t_i} are, respectively, the observation, the control information, and the state for the $i-th$ interval.

Within the RTPF framework, the *belief* for the $i-th$ set can be expressed, similarly, as:

$$Bel_i(x_{t_k}) \propto \int \dots \int p(z_{t_i}|x_{t_i}) \cdot \prod_{j=1}^{k} p(x_{t_j}|x_{t_{j-1}}, u_{t_{j-1}}) \cdot Bel(x_{t_0}) dx_{t_0} \dots dx_{t_{k-1}} \quad (2)$$

containing only observation-free trajectories, since the only feedback is based on the observation z_{t_i}, sensor data available at time t_i.

The weighted sum of the k believes belonging to an estimation windows results in an approximation of the optimal belief:

$$Bel_{mix}(x_{t_k}|\alpha) \propto \sum_{i=1}^{k} \alpha_i Bel_i(x_{t_k}) \quad (3)$$

An open problem is how to define the optimal mixture weights minimizing the difference between the $Bel_{opt}(x_{t_k})$ and $Bel_{mix}(x_{t_k}|\alpha)$. In (Kwok et al., 2004), the authors propose to minimize their Kullback-Leibler distance (KLD). This measure of the difference between probability distributions is largely used in information theory (Cover and Thomas, 1991) and can be expressed as:

$$J(\alpha) = \int Bel_{mix}(x_{t_k}|\alpha) \log \frac{Bel_{mix}(x_{t_k}|\alpha)}{Bel_{opt}(x_{t_k})} dx_{t_k} \quad (4)$$

To optimize the weights of mixture approximation, a gradient descent method is proposed in (Kwok et al., 2004). Since gradient computation is not possible without knowing the optimal belief, which requires the integration of all observations, the gradient is obtained by Monte Carlo approximation: believes Bel_i share the same trajectories over the estimation windows, so we can use the weights to evaluate both Bel_i (each weight corresponds to an observation) and Bel_{opt} (the weight of a trajectory is the product of the weights associated to this trajectory in each partition). Hence, the gradient is given by the following formula:

$$\frac{\partial J}{\partial \alpha_i} \simeq 1 + Bel_i \log \frac{\sum_{i=1}^{k} \alpha_i Bel_i}{Bel_{opt}} \quad (5)$$

where Bel_i is substituted by the sum of the weights of partition set $i-th$ and Bel_{opt} by the sum of the weights of each trajectory.

Unfortunately, (5) suffers from a *bias problem*, which (Kwok et al., 2004) solve by clustering samples and computing separately the contribution of each cluster to the gradient (5). In the next section, an alternative solution is proposed.

2.2 ALTERNATIVE COMPUTATION OF MIXTURE WEIGHTS

This section proposes an alternative criterion to compute the values of the weights for the mixture belief. Instead of trying to reduce the Kullback-Leibler divergence, our approach focuses on evaluating the believes by synthetic values depending on the sample weights of each partition. The concrete approximation given by (Kwok et al., 2004) for the gradient of KL-divergence is a function of weights.

Real-time particle filter prior distribution is the result of two main steps: resampling of samples and propagation of trajectories along the estimation window. The effect of resampling is the concentration of previous estimation window samples in a unique distribution carrying information from each observation. Conversely, the trajectories update given by odometry and observation spreads the particles on partition sets.

Our attempt is to build synthetic values for each element of the resampled distribution and of the partition trajectory; this could be done using weights. Let w_{ij} be the weight of the $i-th$ sample (or trajectory) of the $j-th$ partition set. Then the *weight partition matrix* is given by

$$W = \begin{bmatrix} w_{11} & \dots & w_{1k} \\ \dots & & \dots \\ w_{N_p 1} & \dots & w_{N_p k} \end{bmatrix} \quad (6)$$

The weights on a row of this matrix trace the history of a trajectory on the estimation window; a group of values along a column depicts a partition handling sensor data in a given time. Resampling and trajectory propagation steps can be shaped using matrix W and mixture weights α.

- *Resampling*. The effect of resampling is the concentration of each trajectory in a unique sample whose weight is the weighted mean of the weights of the trajectory. In formula, the vector of trajectory weights is given by $t = W \cdot \alpha$.

- *Propagation*. Projecting a sample along a trajectory is equivalent to the computation of the weight of the sample (i.e., the posterior) for each set given the proper sensor information. Again, matrix W gives an estimation of the weight. Trajectories projection can thus be done with a simple matrix product
$$\hat{\alpha} = W^T \cdot t = W^T\, W \cdot \alpha \tag{7}$$
Vector $\hat{\alpha}$ is a measure of the relative amount of importance of each partition set after resampling and propagation depending on the choice of coefficient α. Hence, $\hat{\alpha}$ is the new coefficient vector for the new mixture of believes.

Some remarks can be made about the matrix $V = W^T\, W$ in (7). First, since we assume $w_{ij} > 0$, V is a symmetric and positive definite matrix. Moreover, each element j on the main diagonal is the inverse of the effective sample size (see (Liu, 1996)) of set j
$$n_{eff_j} = \frac{1}{\sum_{i=1}^{N_p} w_{ij}^2} \tag{8}$$
The effective sample size is a measure of the efficiency of the importance sampling on each of the partition sets. Therefore, the off-diagonal elements of V correspond to a sort of importance covariances among two partition sets. Thus we will refer to this matrix as *weights matrix*.

Hence, a criterion to compute the mixture weights consists of achieving a balance in the mixture forcing $\hat{\alpha}$ in (7) to be equal to α except for scale. The vector is thus obtained by searching an eigenvector of matrix V
$$V\, \alpha = \lambda\, I\, \alpha \tag{9}$$
The eigenvector can be computed using the power method or the inverse power method. This criterion can be interpreted as an effort to balance the effective number of samples keeping the proportion among different partition sets.

3 ADDRESSING SOFTWARE COMPLEXITY

While the choice of the algorithm is a key step, integration of a localization subsystem in a real mobile robot requires a number of practical issues and tradeoffs to be addressed. Real-time execution is the result of different aspects like communication and integration of the localizer with the robot control architecture, careful analysis in object creation/destruction cycles, and tradeoffs between abstraction level management and efficiency.

This section describes a library designed to efficiently support the implementation of particle filter localization algorithms, and specifically of RTPF. The library aims at providing an efficient yet open infrastructure allowing users to take advantage of the provided genericity to integrate their own algorithms. The library has been designed to be easily exploited in different control systems for autonomous mobile robots. In a functional layer, or controller, with the basic computational threads for robot action and perception, the localization task can be simply configured as a computational demanding, low priority thread.

3.1 DESIGN OF THE LIBRARY

Advanced localization algorithms like RTPF address restrictions on real-time execution of localization due to limited computational resources. However, strictly speaking, real-time execution relies on the scheduling and communication capabilities of the robot control system which hosts the localizer. A localization subsystem should therefore be properly integrated in the control architecture. Nonetheless, localizer independence from the underlying low level control layer is a highly desirable property for a localization library. The adaptable aspects of the localization library are the data format and the models of the information provided by the physical devices of a mobile robots.

The functional analysis of the localization problem led to the identification of four main components: the localizer, the dynamic model of the system, the sensor data model, and the map. Each of these components is implemented by a class, which provides a general interface to handle prediction, correction and resampling phases of particle filters. However, there are different ways of modelling details like sensor and motion uncertainty, data formats of system state, control commands and observations, or implementation-specific like map storage and access. In our library, classes `Localizer`, `SystemModel`, `SensorModel` and `LocalizeMap` consist of a general interface which can be adapted.

The strategy pattern (Gamma et al., 1995) is the well-known design solution for decoupling algorithms from system architecture: with this pattern, algorithms can be modified and integrated in the application without modifying the internal code. Thus, using external abstract strategy classes for each changeable aspect of the localization subsystem, adaptation to the robotic platform hosting the localizer can be obtained. While this implementation avoids the hardwiring of user's choices inside the localizer code, it causes inefficiency due to the abstraction levels introduced to support the polymorphic behavior. Furthermore, the strategy pattern cannot handle changes of data types lacking an abstract base class; i.e., observation or control command types given by a generic robot control architecture cannot be used directly. These remarks, together with the observation that choices are immutable at runtime, suggested the use of *static polymorphism*. In the current library implementation, static polymorphism is effectively guaranteed by a *generic programming* approach, and in particular with *policy and policy classes* (Alexandrescu, 2001). This programming technique supports developer's choices at compile time, together with type checking and code optimization, by using templates.

```
template< State ,
          SensorData ,
          Control ,
          SampleCounter ,
          SampleManager ,
          Fusion >
class Localizer : public SampleManager<State>
{
  Fusion<SensorData> fusion_;

public:
  template< ... >      // StateConverter
  Localizer(Pdf &pdf,
            StateConverter<...> &converter);

  ~Localizer();

  template< ... >      // SystemModel Params
  void update(Control &u,
              SystemModel<...> &sys);

  template< ... >      // SensorModel Params
  void update(SensorData &z,
              SensorModel<...> &sen);
};
```

Listing 1: The `Localizer` class.

The main component of the library, the class `Localizer`, exemplifies how policies allow management of different aspects of localization problem. Listing 1 shows a simplified interface of the `Localizer` class, including only two `update()` methods. Note the template parameters of the class: there are both data types (`State`, `SensorData` and `Control`) and policies related to particle filter execution. Methods `update()` allow the execution of prediction and correction phases by using generic sensor and motion models, `SensorModel` and `SystemModel`, which are fully customizable interface classes with their own polices too.

Policies in `Localizer` provide the required flexibility in RTPF implementation. The library supports different versions of particle filters, and template parameters determine the actual algorithm implemented in the system. `SampleCounter` allows choosing the number of samples, that can be fixed or adaptable, e.g. with KL-distance (Fox, 2003). `SampleManager` is the class implementing sample management and creation/destruction of data involved in computation. This class plays an important role, since RTPF determines a complex life cycle for particles, as shown in figure 2. While in standard particle filters samples and control commands survive only during a single iteration (prediction, correction, resampling), RTPF needs the storage of data over the period of an estimation window.

4 RESULTS

In this section, we describe RTPF performance evaluation both in a simulated environment and using experimental data collected by navigating a robot in a known environment. One of the purposes of this evaluation is the comparison of the two RTPF versions differing in their method for computing mixture weights: the original method based on steepest descent and the method described in this paper and relying on the eigenvalues of the weights matrix. Simulations allow comparison of the two methods in a fully-controlled environment, whereas experiments show the actual effectiveness of the proposed technique.

4.1 SIMULATION

Several tests were performed in the simulated environment shown in figure 3, which corresponds to the main ground floor hallway in the Computer Engineering Department of the University of Parma. This environment allows verification of RTPF correctness while coping with several symmetric features, which may cause ambiguities in the choice of correct localization hypotheses. Real experiments with a mobile robot were carried out in the same environment and are described later in the paper: simulations have helped with the setup of experiments and viceversa. Two simulated paths exploited in simulation are also shown in figure 3. These paths, labeled as Path 1 and Path 2, correspond to lengths of approximately 7 m and 5 m.

In simulation, the map is stored as a grid with a given resolution (0.20 m) and is used both to create simulated observations and to compute importance weights in correction steps. Data provided to the localizer consist of a sequence of laser scans and measurements: scanned ranges are obtained by ray trac-

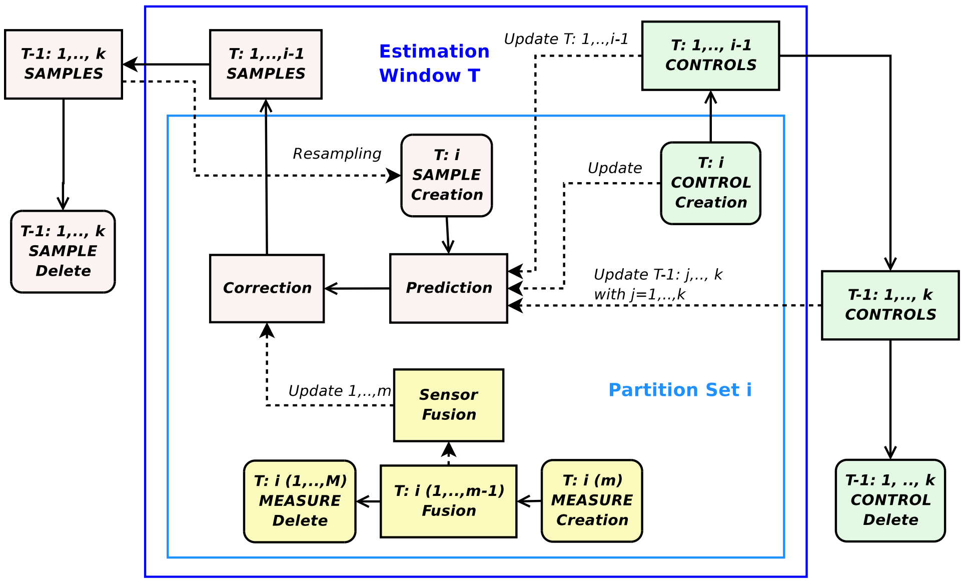

Figure 2: Life cycle for particle, measurement, and control objects within a single step in a real-time particle filter.

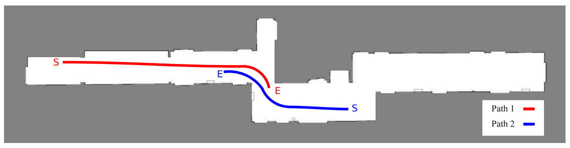

Figure 3: Hallway and simulated paths in the Computer Engineering Department, University of Parma (S=Start, E=End).

ing a beam on the discretized map. The measurement model is also based on ray tracing and follows standard beam models for laser scanners (Thrun et al., 2005). In our tests, we have used only three laser beams measuring distances to left, right and frontal obstacles; such poor sensor data stress the role of algorithm instead of sensor data. A gaussian additive noise is added to both range beams and robot movements representing environment inputs and robot state in simulation. The task of the robot is to achieve localization while moving in the map of figure 3 along different trajectories.

Localization algorithms investigated are RTPFs in the two versions: the original steepest descent-based one (RTPF-Grad) and the proposed one based on the effective number of samples (RTPF-Eig). During the tests the partition set size is 1000 samples.

A summary of simulation results is reported in figure 5. In the figure, curves show the localization error for the two algorithms at each iteration by considering convergence to the nearest hypothesis. For both curves, each value is obtained by averaging the distances of the estimated pose from the real pose over 10 trials where localization converged to the correct hypothesis. For both algorithms there were also a few simulation instances where localization did not converge to the correct hypothesis within the length of the path, although the correct hypothesis was the second best. These unsuccessful cases, mostly occurring on Path 2, were approximately 10% of all simulated localization trials. We did not verify whether the robot would recover its correct pose in the environment with further navigation.

On the average, the two versions of the RTPF-based localizer converged to some few hypotheses after three iterations: the common samples distribution is multi-modal, as shown in figure 4 where there are two local maxima. Hence, cluster search leads to few hypotheses with different weight (an example is shown in figure 4). In our tests a hypothesis close to the correct robot pose always exists, and when this hypothesis prevails there is a sudden change in localization error, as shown in figure 5. Convergence is helped by recognizable features, e.g. the shape of scans, but when the environment is symmetric it can be difficult to reach, especially with limited or noisy sensoriality. Of course, the mean error trend in figure 5 does not correspond to any of the simulated trials; rather, it is the result of averaging trials with quick convergence and trials where the correct hypothesis could only be recovered after many more iterations.

Figure 6 shows the percentage of simulation trials converging to the correct hypothesis (i.e. with localization error less than 1.5 m) at each iteration. Note that for $40-50\%$ of simulation tests, convergence is

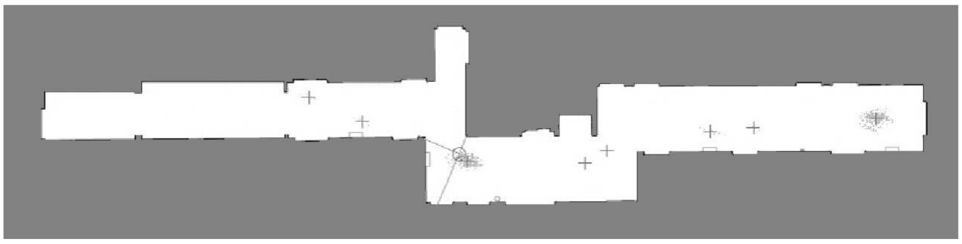

Figure 4: A typical distribution of samples condensed around two prevailing hypotheses (crosses mark hypotheses, the circle is centered in the robot position). When the wrong hypothesis has a higher weight, the localization error is huge.

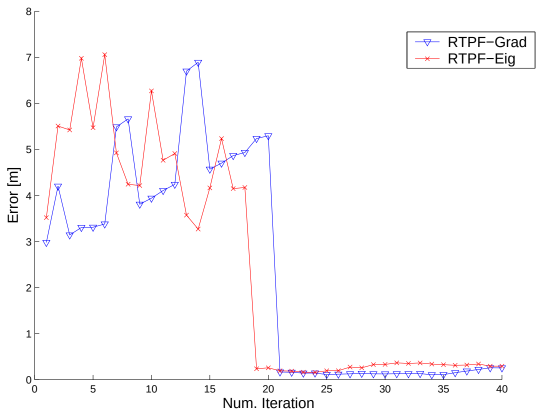

Figure 5: Performance of the two RTPF versions in the simulated environment. The x-axis represents the iterations of the algorithm. The y-axis shows the average error distance of the estimated pose from robot pose.

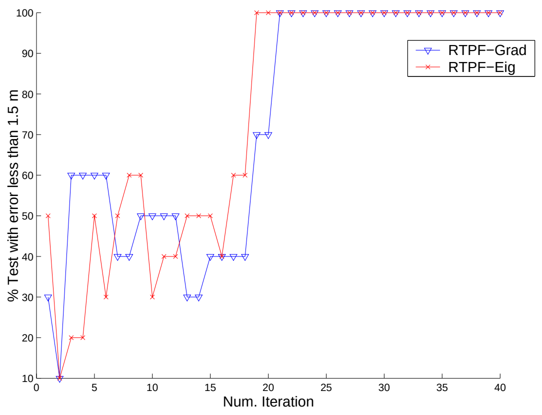

Figure 6: Percentage of simulation trials converged to the correct hypothesis, i.e. with localization error less than 1.5 m, during iterations for Map 1.

reached after few iterations. In other simulations, the correct robot pose is recovered only after about 20 iterations, i.e. after sensing map features that increase the weight of the correct samples.

Empirically, for the examined environment RTPF-Eig seems to exhibit a slightly faster convergence, on the average, to the correct hypothesis, but its average error after convergence appears somehow larger.

4.2 EXPERIMENTS

Real experiments took place in the environment of figure 3 collecting data with a Nomad 200 mobile robot equipped with a Sick LMS 200 laser scanner. The robot moved along Path 1 for about 5 m, from the left end of the hallway in steps of about $15-20$ cm and reading three laser beams from each scan in the same way of the simulation tests. In the real environment localization was always successful, i.e. it always converged to the hypothesis closer to the actual pose in less than 10 iterations (remarkably faster than in simulation). Localization error after convergence was measured below 50 cm, comparable or better than in simulation.

To assess the consistency of the localizer's output on a larger set of experiments, we compared the robot pose computed by the localizer (using the RTPF-Eig algorithm) with the one provided by an independent localization methodology. To this purpose, some visual landmarks were placed in the environment and on the mobile robot, and a vision system exploiting the *ARToolKit* framework (Kato and Billinghurst, 1999) was exploited to triangualate the robot position based on these landmarks. The vision system provided an independent, coarse estimate of the robot pose at any step, and hence allowed to establish convergence of the RTPF-based localizer. The two localization estimates were computed concurrently at each location and stored by the robot.

Figure 7 shows the results of 10 tests of RTPF-Eig over about 20 iterations. These results confirm that RTPF-Eig achieves localization to the correct hypothesis very fast in most experiments. After convergence, the maximum distance between RTPF-based and vision based estimates is about 70 cm due to the compound error of the two systems.

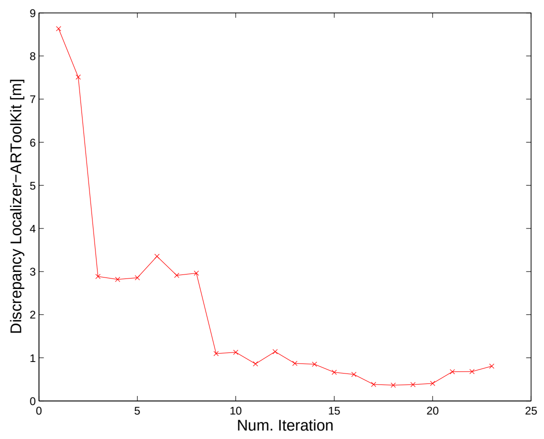

Figure 7: Discrepancy between RTPF-Eig and ARToolKit estimations using real data collected in the hallway of Map 1.

5 CONCLUSION

Localizing a mobile robot remains a difficult task: although effective algorithms like particle filters exist, setting up a concrete localization system must deal with several architectural and implementation problems. One of these problems is the tradeoff between sensor data acquisition rate and computational load. Solutions like RTPF have been proposed to achieve this tradeoff, and are open to a number of improvements. Therefore, localization involves configuring customizable features both in the algorithm and in modules for sensor data, motion models, maps and other algorithmic details. A good generic implementation should provide components that the end user can adapt to his needs.

This paper has presented some improvements of the RTPF algorithm. In the proposed enhancement, the weight mixture of sample sets representing the posterior are computed so as to maximize the effective number of samples.

A novel RTPF implementation based on the enhanced algorithm has been developed. Experiments reported in the paper have shown this implementation to work both in simulated environments and in the real world. Assessing its relative merit with respect to the original RTPF proposal requires further investigation.

ACKNOWLEDGEMENTS

This work was supported by "LARER – Laboratorio di Automazione della Regione Emilia-Romagna." We thank Bruno Ferrarini for his help with the experimental assessment.

REFERENCES

Alexandrescu, A. (2001). *Modern C++ Design: Generic Programming and Design Pattern Applied*. Addison-Wesley.

Arras, K. O., Castellanos, H. F., and Siegwart, R. (2002). Feature-based multi-hypothesis localization and tracking for mobile robots using geometric constraints. *IEEE Int. Conf. on Robotics and Automation*, 2:1371–1377.

Cover, T. M. and Thomas, J. A. (1991). *Elements of Information Theory*. Wiley.

Doucet, A., de Freitas, J., and Gordon, N. (2001). *Sequential Monte Carlo Methods in Practice*. Springer.

Elinas, P. and Little, J. (2005). σMCL: Monte-carlo localization for mobile robots with stereo vision. *Proc. of Robotics: Science and Systems*.

Fox, D. (2003). Adapting the sample size in particle filters through KLD-sampling. *Int. J. of Robotics Research*, 22(12):985–1003.

Fox, D., Burgard, W., and Thrun, S. (1999). Monte Carlo Localization: Efficient position estimation for mobile robots. *Proc. of the National Conference on Artificial Intelligence*.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: elements of reusable object-oriented software*. Addison-Wesley.

Kato, H. and Billinghurst, M. (1999). Marker tracking and hmd calibration for a video-based augmented reality conferencing system. *Proc. of the Int. Workshop on Augmented Reality*.

Kwok, C., Fox, D., and Meilă, M. (2004). Real-time particle filters. *Proc. of the IEEE*, 92(3):469–484.

Leonard, J. J. and Durrant-Whyte, H. F. (1991). Mobile Robot Localization by Traking Geometric Beacons. *IEEE Int. Conf. on Robotics and Automation*.

Liu, J. (1996). Metropolized independent sampling with comparisons to rejection sampling and importance sampling. *Statistics and Computing*, 6(2):113–119.

Montemerlo, M., Roy, N., and Thrun, S. (2003). Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit. *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*.

Se, S., Lowe, D., and Little, J. (2002). Mobile robot localization and mapping with uncertainty using scale-invariant visual landmark. *Int. J. of Robotics Research*, 21(8):735–758.

Sridharan, M., Kuhlmann, G., and Stone, P. (2005). Practical vision-based monte carlo localization on a legged robot. *IEEE Int. Conf. on Robotics and Automation*, pages 3366–3371.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabibilistic Robotics*. MIT Press, Cambridge, MA.