J Intell Robot Syst (2016) 82:513–536
DOI 10.1007/s10846-015-0245-8

Differential Evolution Markov Chain Filter for Global Localization

Luis Moreno · Fernando Martín · María Luisa Muñoz · Santiago Garrido

Received: 21 May 2015 / Accepted: 28 May 2015 / Published online: 20 June 2015

Abstract A key challenge for an autonomous mobile robot is to estimate its location according to the available information. A particular aspect of this task is the global localization problem. In our previous work, we developed an algorithm based on the Differential Evolution method that solves this problem in 2D and 3D environments. The robot's pose is represented by a set of possible location estimates weighted by a fitness function. The Markov Chain Monte Carlo algorithms have been successfully applied to multiple fields such as econometrics or computing science. It has been demonstrated that they can be combined with the Differential Evolution method to solve efficiently many optimization problems. In this work, we have combined both approaches to develop a global localization filter. The algorithm performance has been tested in simulated and real maps. The population requirements have been reduced when compared to the previous version.

Keywords Differential evolution · Markov chain Monte Carlo · Optimization method · Global localization · Mobile robots

L. Moreno · F. Martín (✉) · S. Garrido
Carlos III University, Madrid, Spain
e-mail: fmmonar@ing.uc3m.es

M. L. Muñoz
Universidad Politécnica, Madrid, Spain

1 Introduction

One of the most important skills of an autonomous mobile robot is the capability of estimating its own location according to the available information. Two different systems can be defined according to the information source: positioning systems and self-localization systems. The positioning systems receive signals from external sources (for example, the well-known GPS). The self-positioning systems rely on sensors implemented onboard the robot. This work is included in the second option because our robot works in indoor environments and it is equipped with a laser range finder.

Depending on the initial knowledge, it is possible to distinguish between two different problems: re-localization or tracking and Global Localization (GL). In re-localization, the initial robot's pose (position and orientation) is known, or at least we have information that simplifies the localization problem by reducing the area to be explored. The localization module tries to estimate the current pose as accurately as possible while the robot is moving around the environment. The objective of this problem is to correct the estimate based on proprioceptive sensors (for example, odometry provided by the wheel encoders) by using local information obtained by perceptive sensors [1] (ultrasounds, laser scanners, vision, etc.). In the GL problem (also called kidnapping problem), the initial pose is unknown or highly uncertain and the search of the robot's true location is not limited to a local area (it

is assumed that the global map is known). The robot's pose has to be estimated using local information provided by perceptive sensors and the odometry is not available yet. GL is, from a theoretical point of view, more difficult to solve than re-localization.

In our previous work, we developed an algorithm based on the Differential Evolution (DE) method [2] that solves the GL problem in 2D [3] and 3D [4] environments. These methods are evolutionary optimization techniques that rely on the representation of the robot's pose (position and orientation) by a set of possible location estimates (population) weighted by a fitness function. This function represents the difference between the real observations and the estimated observations from each possible candidate. The observation vector is composed of the measurements obtained by a laser scanner. The state is recursively estimated using a set of results selected according to the weight associated to each possible solution. The set of solutions evolves in time to integrate the sensor information and the robot motion information. A new version that uses the Kullback-Leibler divergence to deal with different types of occlusions was recently published [5].

The basic idea of the Monte Carlo (MC) principle is to represent a probability density function by a set of samples. This is an old concept that was first formulated by Stan Ulam in 1946. He was playing solitaire when he was sick and he thought that, instead of trying to calculate the probabilities of being success, it was easier to play the game a significative number of times (from a probabilistic point of view), and sum the number of cases in which he won. He and Nick Metropolis investigated about how to apply this principle in computing and published the first document about MC simulation in 1949 [6].

Many papers about MC simulation appeared in the physics literature after that. One of the most interesting versions was developed by Metropolis [7] and later generalized by Hastings [8]. They created a large class of sampling algorithms that are called Markov Chain Monte Carlo (MCMC)[9]. These methods are used to generate a number of samples by successive jumps that depend on a transition probability. The set of samples explore the state space following a Markov chain mechanism. The sequence of samples is drawn to imitate a target distribution.

The MCMC algorithms have been successfully applied to multiple fields such as econometrics or computing science [10]. In [11], Ter Braak has demonstrated how to combine the MCMC sampling technique with the DE algorithm. His method has been called DE-MC. This new version is able to obtain the solution in multiple optimization problems. In this work, we have applied these concepts to develop a new version of the GL module based on the DE-MC method. Several reasons encouraged us to implement this method. First, the good behavior shown in [11]. Ter Braak reports advantages with respect to the classic MCMC regarding simplicity, speed of calculation and convergence. Second, the possibility of improving our localization module. Third, to combine the statistical robustness of the MCMC technique and the exploration properties of the evolutionary filter.

The new algorithm has been tested in simulated and real maps. The experimental results show that this new method is an appropriate approach to solve the GL problem. We have not found any drawback when compared to the previous versions of the filter. The most important contribution that may be inferred from the experiments is that there is a significant improvement in the population requirements. The optimum number of particles will be measured by a parameter that we will define as the success rate. The population requirements are much lower than the demands of our previous version of the DE-based GL filter.

The results presented here are focused on solving the problem with a single laser scan, but our technique is not limited to this assumption. The method also works with motion and multiple laser scans can be used to filter the localization results (explained at the end of Section 4). We present the results after a single perception cycle (the robot does not know its location, known map) because the evolutionary search of the DE-MC algorithm uses a single observation vector. Working with a single laser scan is a strong assumption and the cases where the GL problem can be solved with this information are limited by the environment. However, the new information can be integrated in order to solve the ambiguities when necessary. The performance of the evolutionary method with multiple scans from different locations and robot motion has been discussed in our previous work [4, 5]. The population set keeps the hypotheses and the new information eliminates the ambiguities, converging to the true pose after receiving more information from the new locations.

The rest of this paper is organized as follows. First, Section 2 contains a brief review of related work. The main concepts about the algorithms used here (MC, MCMC, and DE) are reviewed in Section 3. In Section 4, the new version of the GL localization module is presented. The experimental results are detailed in Section 5 and, finally, the most important conclusions are summarized in Section 6.

2 Related Work

Different families of algorithms can be used to solve the GL problem. In particular, we will distinguish between Bayesian-based, optimization-based, and hybrid methods.

The Bayesian-based filters work in two steps. Firstly, the available information (motion and perception) is integrated into the *a posteriori* density function. Secondly, the robot's pose is estimated according to a specific criterion such as the maximum density point or the average value. The maximum *a posteriori* (MAP) is an estimator that is frequently used in localization to update the robot's pose in this step. The key of these methods is the generation of accurate models for the density function to represent the most feasible zones. All the distribution is concentrated in a small area after convergence. The particle filters are a well-known class of methods that are used to solve the integral expressions of the Bayesian filter.

There is a wide group of researchers that work on this group of techniques. Some examples are grid-based probabilistic filters [12, 13] and MC localization methods [14–16]. Fox et al. [17] have developed a Markov localization module that is applied to dynamic environments. The MC version presented by Thrun et al. [14] solves the GL and the re-localization problems. Different variations were proposed after that [18, 19].

Some interesting approaches take into account the observations likelihood function to improve the refinement of the hypotheses. The objective is to reduce the particle requirements of the localization filters. Biswas et al. [20] have introduced the Corrective Gradient Refinement method. Their technique diminishes the population requirements of the particle filter by using gradients of the observation model. In [21], the authors include the movement of the robot and the most recent observation in the proposal distribution. The current observation is also included in the proposal distribution in [22]. Zhang et al. [23] have developed the Self-Adaptive MC localization method. They have defined the concept of similar energy region to distribute more efficiently the samples.

The optimization-based methods rely on a fitness function that is minimized in each motion-perception cycle. The information is integrated to generate a fitness function that can be implemented in different ways. This cost function is the key of this group of techniques because the estimate will be the element with the best fitness value. The most common choice is the quadratic cost function. However, different options have also been considered in our previous work. The Kullback-Leibler divergence is an appropriate metric to deal with different types of occlusions [5]. The Manhattan distance (L1-norm) is a more suitable approach in environments with dynamic obstacles [24]. Donoso et al. [25] have used the Hausdorff distance. Fox et al. [26] have considered the entropy of future belief distributions. Arras et al. [27] have implemented a feature-based method that relies on the Mahalanobis distance.

Two different approaches can be followed to solve the optimization problem. The first option is to use the derivative of the cost function to obtain the solution. The main advantage of this approach is the computational speed, but it is not possible to deal with multi-hypotheses problems. The Kalman filters can be included here. They are more frequently used in tracking problems (re-localization) because they only need to manage one hypothesis. The second group executes a stochastic search to find the best solution. Multiple classes of algorithms can be included in this approach: DE, Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), etc. An exhaustive review can be found in [28, 29]. Vahdat et al. [30] have published a comparison between two evolutionary methods (DE and PSO) and MC. Lisowski [31, 32] has implemented the DE-based MC. A GL filter based on the Harmony Search algorithm [33] has been developed by Mirkhania et al. [34]. Ronghua et al. [19] have proposed a genetic algorithm optimizer in MC. They exploit the idea of coevolution to avoid premature convergence. The DE algorithm has been applied in our previous work [3, 4]. Kwok et al. [35] have utilized three evolutionary techniques (GA, PSO, and ACO) to figure out the

solution of the well-known Simultaneous Localization and Mapping problem.

In the hybrid methods (multi-hypotheses Kalman filters) [27, 36–39], the set of solutions is formed by normal probability distributions. Nevertheless, the creation or elimination of solutions is not purely Bayesian. They keep a set of multi-hypotheses with an associated Gaussian probability where each distribution is guided by a Kalman filter. Most of them rely on a decision tree search mechanism based on geometric constraints together with probabilistic attributes to manage the global data association problem. In [40], the authors use a set of Gaussians to model the likelihood function of the robot's location given the information provided by a laser range finder. In [41], the authors combine Kalman and particle filters.

In this paper, the MCMC sampling approach is combined with the DE evolutionary algorithm to develop a GL module. The literature about this sampling method is explored in the next section.

3 Fundaments of the Method

In this section, the methods that will be used to develop the GL module are explained. For a more detailed explanation, the reader can consult different references that will be cited in this section.

First of all, it is necessary to describe the problem that will be solved here. The GL problem consists of estimating the robot's pose in a known map considering the available information which, in this case, is a 2D laser reading obtained by a laser scanner. A population-based algorithm in which each population member is a possible solution (robot's pose in a known map) is applied to solve the cited problem.

Since the map is a priori known, it is possible to obtain a simulated observation vector from each estimate. This vector can be compared to the true observation from the real location to determine how good the estimate is. This comparison is done by implementing a fitness function. The cost value obtained by this function can be considered as a probability, and the main objective of the GL problem will be to find the population member with a highest probability, which is basically an optimization problem where the objective is to determine the MAP estimate:

$$\hat{\mathbf{x}}_{MAP} = \arg \max_{\mathbf{x}_i : i=1.....N_P} p(\mathbf{x}_i|\mathbf{z}), \tag{1}$$

where there are N_P population members $\mathbf{x}_i$, and the observation vector from the true location is $\mathbf{z}$.

The GL module proposed here to solve the optimization problem combines concepts of MCMC and DE. The theory under these algorithms is explained below.

3.1 Monte Carlo Sampling

Particle filters have been applied with remarkable success to many different problems. In robotics, they have been widely implemented to solve the localization problem. The most common particle filters are based in the MC principle, which consists of drawing a "i.d.d. set of samples $\{\mathbf{x}_i\}_{i=1}^{N_P}$ from a target density $p(\mathbf{x})$ defined on a high-dimensional space $\mathcal{X}$"[1] (e.g. the space of possible locations in this case). This population set can be used to approximate empirically the probability distribution:

$$p_{N_P}(\mathbf{x}) = \frac{1}{N_P} \sum_{i=1}^{N_P} \delta_{\mathbf{x}_i}(\mathbf{x}), \tag{2}$$

where $\delta_{\mathbf{x}_i}(\mathbf{x})$ represents the Dirac delta mass associated to the candidate $\mathbf{x}_i$. In this case, all population members have the same probability mass $1/N_P$. Generally, $p_{N_P}(\mathbf{x}) \to p(\mathbf{x})$ when $N_P \to \infty$.

As can be observed, the N_P samples can be utilized to obtain the solution of the MAP problem. However, in this basic version, the method will success only if $p(\mathbf{x})$ has a standard form (Gaussian). Andrieu et al. [10] have stated that for more complicated distributions it is necessary to implement more sophisticated techniques such as Rejection Sampling (RS), Importance Sampling (IS) and Sampling Importance Resampling (SIR). These options are described in the next paragraphs. The probability distribution of the problem addressed here depends on the sensor measurements and the geometry of the environment.

The first alternative to the basic MC sampling method is called RS. In this technique, the distribution $p(\mathbf{x})$ is sampled from another distribution $q(\mathbf{x})$ that is easier to sample and satisfies that $p(\mathbf{x}) \leqslant Mq(\mathbf{x})$ with $M < \infty$. For example, the known distribution $q(\mathbf{x})$ could be a uniform distribution over the space of possible solutions and the objective distribution $p(\mathbf{x})$

[1] A useful explanation about how to apply this method to machine learning is given in [10].

could be based on the cost value of each possible solution.

An accept/reject procedure is implemented in two steps. In the first step, two samples are generated: $\mathbf{x}_i \sim q(\mathbf{x})$ and $u \sim \mathcal{U}_{(0,1)}$. In the second step, $\mathbf{x}_i$ is accepted if $u < p(\mathbf{x}_i)/Mq(\mathbf{x}_i)$. According to this procedure, the set of accepted samples follows the distribution $p(\mathbf{x})$ [42].

The RS variation presents some limitations. It is not always possible to establish an adequate constant M to restrict $p(\mathbf{x})/q(\mathbf{x})$ over the whole space. If M is too large, the probability of acceptance is too small, which makes this method inefficient in high-dimensional spaces. In particular, this is an important drawback in the localization problem. In planar maps, there are three dimensions (position and orientation) and the definition of an adequate value for M is not an easy task.

Most MC filters are based on the IS strategy [43, 44]. The idea of the IS technique is to define "an arbitrary importance proposal distribution $q(\mathbf{x})$ such that its support set includes the support set of $p(\mathbf{x})$" [10]. It is possible to use $q(\mathbf{x})$ to generate N_P i.i.d. samples $\{\mathbf{x}_i\}_{i=1}^{N_P}$ and measure the importance $w(\mathbf{x}_i)$ of each population member, which is

$$w(\mathbf{x}_i) = \frac{p(\mathbf{x}_i)}{q(\mathbf{x}_i)}. \tag{3}$$

The probability density $p(x)$ is now approximated by

$$\hat{p}_{N_P}(\mathbf{x}) = \sum_{i=1}^{N} w(\mathbf{x}_i)\delta_{\mathbf{x}_i}(\mathbf{x}). \tag{4}$$

There are different difficulties when approximating a probability distribution by this method. First, if a sample falls in an area with low probability its weight is drastically reduced, which implies a great increase in the weight of those samples closer to the maximum values. This factor can cause a fast degeneration of the method. The most classical way of dealing with this problem is the SIR strategy [45]. Those particles with higher weights are replaced by a set of particles of equal weights around the original position of the higher weights particles, and those particles with lower weights are removed from the set. After the resampling step all particles have the same weight.

Second, areas without particles are not evaluated. The classical solution to this problem is to increase the number of particles, which can result in big sets of particles and prohibitive computational costs.

Third, the method, in each iteration, includes the new sensor information obtained to modify the probabilistic weight of each particle in the set. This implies that a new iteration is executed whenever there is new information.

On the one hand, these characteristics make the MC method based on IS or SIR to be very effective in re-localization problems where the area to be sampled is small and, and on the other hand, it can be very inefficient in the GL problem because it requires a very high number of samples to ensure a proper initial density. Besides, it requires a high number of observation-motion cycles until convergence. As advantages, it should be noted that this method is very robust statistically and it supports high levels of noise.

As it is stated in [10], even with SIR or other variations that are not commented here, there are problems where it is almost impossible to obtain proposal distributions that are easy to sample from and good approximations of the state space at the same time. For this reason, it is necessary to introduce other sampling algorithms based on Markov chains. In the topic studied in this paper, the initial information is the map of the environment and the laser scan from the robot's location. If the state space is formed by all the possible poses that can be the solution to the problem, the probability cannot be approximated by a known distribution (except in very simple maps). This probability is mainly influenced by the sensor measurements and the geometry of the environment.

Besides, if MC-based sampling methods are directly applied to the GL problem, the population requirements will be huge to cover the whole space in the initial stages. The jumping step of the DE-MC technique will help the localization process in this aspect (see Section 4).

3.2 Markov Chain Monte Carlo - Metropolis-Hastings Method

An interesting variation of the RS idea has produced the MCMC algorithms [9]. MCMC is a sampling method where the space is explored using a Markov chain mechanism. Each Markov chain is formed after generating N_P samples that explore the space by successive jumps. These jumps are based on a state transition probability in such a way that the generated

sequence of states imitates samples drawn from the target distribution $p(\mathbf{x})$. This idea was proposed by Metropolis [7] and later refined by Hastings [8].

The Metropolis-Hastings algorithm (MH) is the most famous MCMC method. It relies on an accept/reject approach. In the basic version of the MH algorithm, a random value $u \sim \mathcal{U}_{(0,1)}$ and a trial sample $\mathbf{x}_{i*} \sim q(\mathbf{x}_{i*}|\mathbf{x}_i)$ are generated. The trial sample is accepted according to an acceptance probability:

$$u < \mathcal{A}(\mathbf{x}_i, \mathbf{x}_{i*}) = min\left\{1, \frac{p(\mathbf{x}_{i*})q(\mathbf{x}_i|\mathbf{x}_{i*})}{p(\mathbf{x}_i)q(\mathbf{x}_{i*}|\mathbf{x}_i)}\right\}. \quad (5)$$

If the trial sample is accepted the next element of the chain is $\mathbf{x}_{i+1} = \mathbf{x}_{i*}$; otherwise, the candidate is rejected and $\mathbf{x}_{i+1} = \mathbf{x}_i$.

To illustrate this method, the practical application where this method will be applied is briefly described. Instead of having a Markov chain represented by a set of particles, the Markov chain will be defined in a slightly different way. Each sample will be a possible solution of the GL problem (robot's pose). For each population member, a trial sample will be generated and the probability value is a variable dependent on the cost function that will be calculated for both samples $(\pi(\mathbf{x}_i), \pi(\mathbf{x}_{i*}))$. The notation of the acceptance probability will be

$$u < \mathcal{A}(\mathbf{x}_i, \mathbf{x}_{i*}) = min\left\{1, \frac{\pi(\mathbf{x}_{i*})}{\pi(\mathbf{x}_i)}\right\}. \quad (6)$$

It can be noticed that the Markov chain will evolve to the poses that maximize the cost function.

The MH algorithm is conceptually simple, but it requires a careful choice of the proposal distribution $q(\mathbf{x}_{i*}|\mathbf{x}_i)$. To guarantee that the algorithm converges, it is necessary to ensure that there are no cycles (aperiodicity) and each state having positive probability can be reached in a finite number of steps (irreducibility). The efficiency of the algorithm depends on the proposal distribution and suffers from two major problems:

- The local-trap problem in systems whose landscape has multiple basins. The samples can be trapped in a local minimum.
- The difficulty to sample from distributions with difficult or even intractable integrals.

At the beginning, the MCMC methods were purely sequential, with a single Markov chain. More recently, many variants in which different Markov chains are run in parallel have been devised. These methods, which are called population-based MCMC, reduce the local-trap problem of the original version. Each Markov chain can follow different distributions. It is possible to exchange information between chains, learning from past samples and improving the convergence speed. Different algorithms can be included in this category: adaptive direction sampling [46], conjugate gradient Monte Carlo [47], parallel tempering [48, 49], evolutionary Monte Carlo [50], equi-energy sampler [51], etc.

The idea of combining evolutionary algorithms with population-based MCMC approaches have been explored by different researchers: Ter Braak [11], Linage and Wong [50], Liang [52], Laskey and Myers [53], etc. Ter Braak has combined MCMC and DE to solve many different optimization problems. He has concluded that the simplicity, speed of calculation and convergence are improved when compared to the MCMC method [11]. In this work, the MH version of the population-based MCMC algorithm will be combined with the DE evolutionary technique according to the method proposed by Ter Braak to design a GL module.

3.3 Differential Evolution Algorithm for GL

The DE algorithm [2] can be applied to multiple optimization problems. The solution adopted in our previous work to solve the GL problem is detailed in this section. It has been widely explained in our previous papers [3, 4], reason why only a brief reminder is given in this section. In order to do that, the reader can see Algorithm 1.

There is a set of elements that corresponds to possible solutions and the fitness function represents the error between real and estimated data. Three coordinates that define a state space with three Degrees of Freedom (DOF) in a 2D map must be estimated to determine the robot's location.

The exploration starts with a group of N_P candidates which are introduced in the localization module and evolve with the time to the best solution. Each candidate $\mathbf{x}_i^k$ is a possible solution to the GL problem (the robot's pose, with 3 DOF, at iteration k). The initial population will be chosen randomly to cover the whole map. The other input parameters are the laser scan

Algorithm 1 DE-based GL

```
1: function DE_GL(real_dist, pop, known_map, conf_parameters)
2:     for i = 1 : N_P do
3:         estimated_dist(i) ← dist_est(pop_i, known_map)
4:         cost(i) ← fitness(estimated_dist(i), real_dist)
5:     end for
6:     while (CONVERGENCE CONDITIONS) do
7:         for i = 1 : N_P do
8:             MUTATION
9:             CROSSOVER
10:            SELECTION
11:            estimated_dist(i) ← dist_est(pop_i, known_map)
12:            cost(i) ← fitness(estimated_dist(i), real_dist)
13:                ▷ cost function value calculation for next generation
14:        end for
15:        [error, ind_best] ← min(cost)
16:        bestmem ← pop(ind_best)
17:        conv_conditions_checking(...)
18:    end while
19: end function        ▷ return bestmem, error and population
```

from the true location ($real\_dist$), the known map, and the configuration parameters of the DE method.

The population size is a crucial factor in any population-based optimization algorithm. An initialization mechanism to estimate this parameter has been developed in our previous work [54].

For each population member, its associated fitness function is calculated (line 2 to 5 of Algorithm 1). The true observation vector from the real pose is compared to the estimated data from the candidate solution.

The main loop starts in line 6. If one of the convergence conditions is satisfied, the localization process ends successfully.

Another loop that contains the evolutionary search starts in line 7. It consists of the generation of a new population for the next generation. In a single iteration the algorithm is executed to obtain the next candidates, evolving to the correct pose.

The current population member is perturbed to generate a mutated vector $\mathbf{x}_{i*}^k$ according to the following expression:

$$\mathbf{x}_{i*}^k = \mathbf{x}_{r_0}^k + F\left(\mathbf{x}_{r_1}^k - \mathbf{x}_{r_2}^k\right), \tag{7}$$

where $\mathbf{x}_{r_0}^k$, $\mathbf{x}_{r_1}^k$, and $\mathbf{x}_{r_2}^k$ are parameter vectors chosen randomly from the population at iteration k and are different from the running index. The scale factor $F \in (0, 1)$ is a real and constant coefficient that controls the amplification of the differential variations $\left(\mathbf{x}_{r_1}^k - \mathbf{x}_{r_2}^k\right)$. It controls the population evolution rate. It is usually defined in the interval $[0.4, 0.9]$, with an empirical upper limit equal to 1 [2]. Zaharie [55] has restricted the lower limit of F. This parameter will be fixed to 0.7 in the experiments according to the optimum values found in our previous work [24].

There is an initial population member $\mathbf{x}_i^k$ and the perturbation is done with three random variables and the constant factor F, generating the new parameter vector $\mathbf{x}_{i*}^k$.

In order to increase the diversity of the new generation, the crossover is introduced. Denoted by $\mathbf{s}_i^k = \left(s_{i,1}^k, s_{i,2}^k, \ldots, s_{i,D}^k\right)^T$, the new parameter vector is

$$s_{i,j}^k = \begin{cases} x_{i*,j}^k & \text{if } p_{i,j}^k < \delta \\ x_{i,j}^k & \text{otherwise,} \end{cases} \tag{8}$$

where $p_{i,j}^k$ is a randomly chosen value from the interval $[0, 1]$ for each parameter j of the population member i at step k, and δ is the crossover probability and constitutes the crossover control variable. $x_{i*,j}^k$ and $x_{i,j}^k$ are each one of the parameters of the mutated and the current population vectors, respectively. D is usually defined as the number of chromosomes, which is three in this case. The random values $p_{i,j}^k$ are made anew for each trial vector i.

The new population candidate $\mathbf{s}_i^k$ is compared to $\mathbf{x}_i^k$ to choose the member of the next generation $i + 1$. If the vector $\mathbf{s}_i^k$ yields a better value for the fitness function than $\mathbf{x}_i^k$, then it is replaced by $\mathbf{s}_i^{k+1}$; otherwise, the old value $\mathbf{x}_i^k$ is retained for the new generation. The general ideas of the previous mechanism (mutation, crossover, and selection) are well known and can be found in literature [56].

Finally, the algorithm returns the best population member according to the fitness function, which is the solution of the GL problem.

However, this is the basic version of the DE-based GL module. This algorithm will be used here in a different way. The new version that combines MCMC and DE will be introduced in the next section. The idea is to exploit the utility of the RS concept of the MH approach combined with the evolutionary-based method.

4 Differential Evolution Markov Chain GL Filter

A GL filter based both on MCMC and DE has been proposed in this paper. The main idea is to implement the same concepts that applied Ter Braak [11] to convert the N_P particles of the DE algorithm into N_P Markov chains, exploiting the statistical robustness of the MCMC sampling method and the exploration properties of the evolutionary algorithm. He proposed a simple modification to combine the MCMC approach with the DE optimizer. This algorithm has been applied here to solve the GL problem. It is detailed in pseudocode in Algorithm 2.

Each population member evolves as a Markov chain where new potential samples are generated in each iteration. In the first iteration, the whole population is generated to cover uniformly the free map (lines 2-4 of Algorithm 2).

The DE-MC method uses the mutation step of the DE algorithm to generate the new potential samples. The new candidates are accepted or rejected according to a selection mechanism. This combined scheme takes advantage of the exploratory efficiency of the DE method to run the exploration jumps and the statistical efficiency of the MC RS strategy via the usual Metropolis ratio, which defines the probability with which a new proposal is accepted.

The use of the DE approach in the jumping step of the MCMC sampling algorithm solves an important difficulty in MCMC in real parameter spaces, which is the selection of a suitable scale and orientation for the jumping distribution. This problem is only solved in orientation but not in scale when using other adaptive direction sampling methods.

In the default option of the DE method (Section 3.3), the new proposals are generated from three random vectors according to Eq. 7. Once a new

Algorithm 2 DE-MC GL module

1: **function** $DE\_MC\_GL(real\_dist,\ known\_map,\ conf\_parameters)$
2: **for** $i = 1 : N_P$ **do** ▷ Initialization of N_P Markov chains
3: $\mathbf{x}_i^0 = uniform(free\_map)$
4: **end for**
5: $j = 1$
6: **while** (CONVERGENCE CONDITIONS) **do**
7: **for** $i = 1 : N_P$ **do**
8: $\mathbf{x}_{i*}^j = \mathbf{x}_i^j + F\left(\mathbf{x}_{r_1}^j - \mathbf{x}_{r_2}^j\right) + \mathbf{e}$ ▷ Mutation
9: $r_{log} = fitness\left(\mathbf{x}_{i*}^j\right) - fitness\left(\mathbf{x}_i^j\right)$
10: $u \sim \mathcal{U}_{(0,1)}$
11: **if** $r_{log} < \log u$ **then** ▷ Selection, next sample of each chain
12: $\mathbf{x}_i^{j+1} = \mathbf{x}_{i*}^j$
13: **else**
14: $\mathbf{x}_i^{j+1} = \mathbf{x}_i^j$
15: **end if**
16: **end for**
17: $j \leftarrow j + 1$ ▷ Next iteration index
18: $optimum\_location = \mathbf{x}_i^j : \min\{\pi(\mathbf{x}^j)\}$
19: $conv\_conditions\_checking(...)$
20: **end while**
21: **end function** ▷ Return solution

set of proposals are obtained, a crossover mechanism is used to mix the new candidates with the old ones. The crossed and mutated vector ($\mathbf{s}_i^k$) is retained for the next generation if its fitness value is better than the fitness value of the current population member $\pi\left(\mathbf{x}_i^k\right)$. In other words, the proposal is accepted if $r = \pi\left(\mathbf{s}_i^k\right)/\pi\left(\mathbf{x}_i^k\right) > 1$, and the whole population evolves to the best candidates optimizing the fitness function. Although Ter Braak proposes a variant with crossover that could be useful in some cases, this option has not been included in our method to keep the procedure closer to the basic concept of a population-based MCMC algorithm.

To make a proper conversion from the DE mechanism to a population-based MCMC algorithm for drawing samples from the target distribution, different researchers have concluded that the generation of new samples and their acceptance condition must be chosen to satisfy the "balance condition" [42, 57, 58]. This basically means that if a sample $\mathbf{x}_i^j$ is drawn from

the target distribution, then the next sample $\mathbf{x}_i^{j+1}$ must be drawn from the same target distribution, possibly dependent on $\mathbf{x}_i^j$. This condition cannot be met when the strategy shown in Eq. 7 is followed.

There are multiple options that can be chosen to generate new samples. According to the work of Ter Braak, a more promising option has been chosen to generate the new samples:

$$\mathbf{x}_{i*}^j = \mathbf{x}_i^j + F\left(\mathbf{x}_{r_1}^j - \mathbf{x}_{r_2}^j\right) + \mathbf{e}, \qquad \mathbf{e} \sim \mathcal{N}(0, b)^d, \quad (9)$$

where $\mathbf{e}$ is a symmetric normal distribution in a d-dimensional space that is added to guarantee that the whole parameter space is covered. b is small when compared to the variance of the target. Note that we use j instead of k to distinguish between the iterations in the new method (Markov chains) and the iterations in the old version. This strategy is adopted to generate the N_P new candidates (line 8).

Figure 1 shows the mutation options described in Eqs. 7 and 9. A simple simulated indoor map with sampled particles at random places is drawn. Each subfigure represents one of the proposed techniques.

The key idea of the procedure proposed by Ter Braak is to include a probabilistic acceptance rule in the evolutionary method. Instead of using a fixed ratio, the proposal generated in Eq. 9 is accepted with probability $\min(1, r)$, where $r = \pi\left(\mathbf{x}_{i*}^j\right)/\pi\left(\mathbf{x}_i^j\right)$.

This acceptance mechanism has to be defined in a different way taking into account the properties of the fitness function. According to the available information, it is not possible to measure direct probabilities in this type of method. Therefore, an important aspect to explain is the cost function implemented in the optimization filter. This is a crucial factor because the evolutionary search of the Markov chains is guided by this parameter. It has been widely studied in our previous work [4, 5]. A common choice for this function when it is assumed that the sensor errors are Gaussian-distributed is the sum of the squared errors.

If the observation vector from the true location is $\mathbf{z} = (z_1, \ldots, z_{N_S})^T$ (N_S being the number of laser beams or observations), the estimated observations from the population member $\mathbf{x}_i$ in the known map are $\mathbf{z}_i = (z_{1,i}, \ldots, z_{N_S,i})^T$, and the observation error variance is σ_e^2, the fitness function to be minimized could be given by the following expression:

$$fitness(\mathbf{x}_i) = \sum_{k=0}^{N_S} \frac{(z_k - z_{k,i})^2}{2\sigma_e^2} + \frac{1}{2}(\mathbf{x}_i - \hat{\mathbf{x}})P^{-1}(\mathbf{x}_i - \hat{\mathbf{x}})^T, \quad (10)$$

where $\hat{\mathbf{x}}$ represents the best estimate among all members of the population. The second term of the previous expression is useful when the robot is moving [4]. Since the GL problem in a single perception cycle (no motion) is considered here, the cost function can be simplified:

$$fitness(\mathbf{x}_i) = \sum_{k=0}^{N_S} \frac{(z_k - z_{k,i})^2}{2\sigma_e^2}. \quad (11)$$

This is the final expression of the fitness function implemented in the GL filter. It measures the difference between the simulated laser scan estimated from a population member in the known map and the observations from from the true location. It can be observed that, if the sensor noise is modeled as a Gaussian in each laser beam, the cost function in the optimum case will be a sum of Gaussians with zero mean, thus the fitness value has to be minimized (the objective is to maximize the probability in the DE-MCMC original notation).

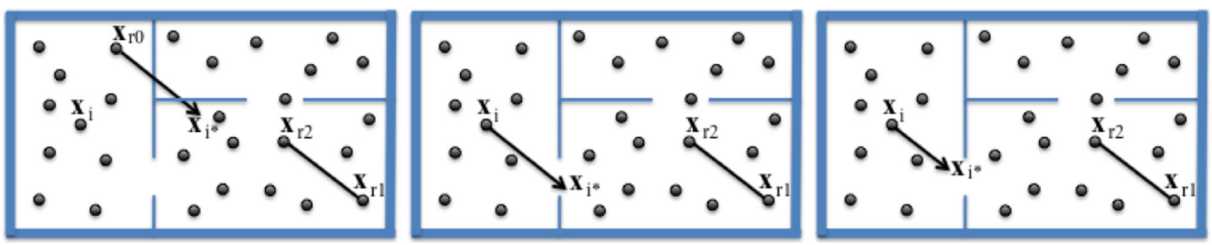

Fig. 1 Different mutation strategies in DE. *Left*: Mutation strategy of Eq. 7. *Middle*: Mutation strategy of Eq. 9 ($F = 1$). *Right*: Mutation strategy of Eq. 9 ($F = 0.7$). Orientation of the particles and noise $\mathbf{e}$ of the mutated values not drawn for simplicity

The acceptance mechanism has been modified here by taking logarithms (lines 9 − 15). After making experiments with different ratios (the simplest one was using the inverse of the fitness value in the original expression), an empirical method based on the difference between the fitness value of the proposal and the current member is proposed:

$$r_{log} = fitness\left(\mathbf{x}_{i*}^{j}\right) - fitness\left(\mathbf{x}_{i}^{j}\right). \tag{12}$$

The logarithm of the random number $u \sim \mathcal{U}_{(0,1)}$ is used to define an acceptance criterion. The new member is accepted if $r_{log} < \log u$; otherwise, it is rejected. $\log u$ is in the interval $(-\infty, 0)$ and r_{log} is negative when the fitness value is improved. The new candidate is accepted with a probability that depends on the random number u if the fitness value is improved. This idea has been taken from an example shown in [11], where logarithms are used to modify the acceptance probability. The probability of acceptance depending on the improvement of the fitness value is tabulated in Table 1. As can be noticed, if there is a significant improvement, the new candidate is almost always accepted.

Analyzing the selection mechanism, several differences are found when compared with the original method proposed by Ter Braak. His method always accepts the proposal if the probability is improved. In this filter, the new candidate is accepted with a probability that is increased with the difference between fitness values. In other words, the new candidate is only accepted if there is a significant improvement and small improvements that can be caused by the noise are filtered, which reduces the optimization in the noise band. This interesting property was also wanted in our previous work [4]. Ter Braak can accept a new candidate even if it holds a lower probability. In our method, it is not possible to accept the proposal if the fitness function is not improved. Though this mechanism has been empirically fixed, this capabilities are more adequate for a GL filter according to our experience in this problem, fact that will be shown in the experimental results section. Different options could be defined to meet different requirements.

This procedure of generating new candidates and accepting them according to a probability ratio is repeated for the whole population in each iteration. The N_P population members or Markov chains evolve to the locations with the best fitness values. This loop is repeated until the convergence conditions are satisfied (line 19), returning the best member of the population (line 18), which is also the solution of the GL problem. As can be noticed, one of the advantages of this filter is that it is even simpler to implement than the basic version of the DE algorithm.

The practical situation in the GL problem is that it is almost impossible to determine the target distribution. The distribution of the cost function from the possible locations (position and orientation) in the whole map does not follow any known distribution at all. However, it will be demonstrated that the solution adopted in this paper works in an efficient way.

The best estimate is saved as the robot location after convergence. If the robot moves to another location (although motion is not considered here, the algorithm is not limited to this assumption), the population set is moved according to the robot motion model $\mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$, where t represents the time instant when the robot receives information from its sensors (odometry information $\mathbf{u}^t$ and laser readings $\mathbf{z}^t$). The problem is now converted into the tracking one. The algorithm is executed when the robot is situated in the new location, but instead of having a random population at the beginning, the initial population is formed by the results of the last execution of the algorithm. In other words, the population that is obtained after the execution of the localization filter is moved according to the odometry, and this population will be the initial population set for the new location.

Table 1 Probability of acceptance depending on the fitness value improvement

Fitness improvement (r_{log})	0.10	0.35	0.69	1.20	1.60	2.39	6.90
p(acceptance) (%)	10	30	50	70	80	90	99.99

Fig. 2 GL in an architectural plan (DLR). All units in cells. Laser readings in purple. Robot's location in blue. Points of study marked with dots and orientation with arrows

5 Experiments

Different experiments with simulated and real maps obtained from the OpenSLAM repository[2] have been carried out to check de performance of the GL filter.

It is interesting to remark different aspects that will be measured:

- GL ability without motion (perceptual differentiation capacity, static). The robot is standing still at a given location and the objective is to estimate its position in a single perception cycle, i. e., using a single laser scan from the true location.
- Ability to handle multiple hypotheses. In indoor environments such as office buildings it is very common to find similar places that are very difficult to differentiate because the perceptive information from both of them is almost the same. The ability to keep a sufficient number of hypotheses until it is possible to eliminate the ambiguity between them is an interesting property.

The algorithm performance has been studied in an architectural plan (Section 5.1) and in a real map (Section 5.2). After that, the results are analyzed in Section 5.3, including a detailed comparison with respect to the previous version of the DE-based GL filter. The most important conclusions are deduced in this section.

[2] www.openslam.org

DE configuration parameters: $F = 0.7$, $\delta = 0.5$.

5.1 Architectural Plan

Figure 2 shows the map provided by the DLR[3] that will be used in this section. A cell size equal to 10 cm will be assumed. It is a medium-large size map (about $650 \times 800 = 520000$ cells $= 5200$ m^2) with a high degree of repeatability, which means that it contains many similar places (for example, offices with almost the same appearance).

Three points of interest (locations in the environment, denoted by $\mathbf{p}_{ai}$) have been chosen:

1. $\mathbf{p}_{a1} = (300, 50, 90)$. The robot is located in an office, and there are many offices with similar dimensions.
2. $\mathbf{p}_{a2} = (150, 105, 0)$. This place is a corridor.
3. $\mathbf{p}_{a3} = (550, 400, 0)$. This place is a hall.

According to the map of Fig. 2, the first coordinate corresponds to the horizontal axis, the second one is the vertical axis and the third one represents the orientation (zero being pointing right, horizontal direction, increasing clockwise). These locations have been chosen because they represent characteristic places of the environment (hall, corridor, and office). More results from random places are given at the end of this section.

[3] Thanks to Christoph Hertzberg for making available this data set.

The sensor noise has been modeled as a Gaussian distribution over the laser distance where the standard deviation is the parameter that must be fixed to specify the noise. This noise is added to the distances observed from the true location. It has to be noticed that the noise will be typically worse than the noise of the commercial devices.

The "success rate" (S_r in the tables) is an interesting variable that has been defined to measure the algorithm robustness. For a specific location, two different results can be reached when estimating the robot's pose: the estimate matches the real pose (success) or the estimate and the true pose do not coincide (failure). S_r can be measured if the algorithm is run multiple times for the same location. The success rate is a statistical variable that is equal to the number of runs in which the estimate matches the real pose (it means that the correct pose is estimated in one perception cycle) divided by the total number of trials. The result of this division has to be multiplied by 100 because S_r is given in %. Since the true pose is known, it is possible to define a distance threshold to measure the success rate, which is 50 cm in these experiments. For example, $S_r = 100$ means that if the algorithm is run 50 times the correct pose is obtained in all cases.

The influence of the population size is measured in Table 2 for the first location ($\mathbf{p}_{a1}$). The position errors (e_x, e_y) are the distances between the estimated values and the real positions of the robot, in cartesian coordinates. The orientation error (e_θ) is the difference between the estimated orientation and the real one.

N_P has to be large enough to be successful in most cases, but higher values have a negative effect on the computational cost. The success rate reaches the 100 % when the population size is 180. Therefore, this size will be adequate in this case.

The estimation of an adequate population size will depend on several factors. The first one is the size of the area perceived by the mobile robot. The required size will be lower for larger areas. This conclusion is completely logical because, in general, the basin of attraction of the local minimum will be larger for larger areas. Therefore, it will be easier to converge to the optimum value. The map size is also a key factor because the number of elements has to be increased for larger maps. There are other factors such as the number of symmetries, the sensors information, and the occlusions that also have an influence on this aspect. An interesting study about the population requirements is included in our previous work [54].

The algorithm performance depending on the noise level has been measured for the same point of interest. The results are presented in Table 3. It can be observed that the accuracy decreases when the sensor noise increases. Although the error is increased, the algorithm converges in almost all cases to the correct pose up to 7.5 % of noise. When the sensor noise is 10 %, the algorithm does not converge to a single pose, requiring additional motion and a new laser scan to solve the GL problem.

The same tests have been conducted for the other points of interest. The results for $\mathbf{p}_{a2}$ are detailed in

Table 2 Error and success rate depending on the population size

N_P	S_r	e_x(cells)	e_y(cells)	e_θ(degrees)
40	54	0.0181 ± 0.0191	0.0677 ± 0.0842	0.0655 ± 0.0609
60	70	0.0218 ± 0.0306	0.0530 ± 0.0398	0.0495 ± 0.0633
80	80	0.0249 ± 0.0326	0.0554 ± 0.0632	0.0647 ± 0.0765
100	92	0.0206 ± 0.0339	0.0586 ± 0.0729	0.0710 ± 0.0812
120	96	0.0169 ± 0.0320	0.0490 ± 0.0679	0.0640 ± 0.0593
140	98	0.0235 ± 0.0361	0.0474 ± 0.0613	0.0723 ± 0.0777
160	98	0.0147 ± 0.0268	0.0660 ± 0.0976	0.0598 ± 0.0612
180	100	0.0241 ± 0.0343	0.0433 ± 0.0729	0.0743 ± 0.0722
200	100	0.0213 ± 0.0346	0.0597 ± 0.0864	0.0650 ± 0.0704
220	100	0.0170 ± 0.0301	0.0700 ± 0.1093	0.0680 ± 0.0715

Simulated map. True location: (300, 50, 90). Sensor noise: standard deviation of 1%. S_r in %. Errors in mean $\pm$ standard deviation

Table 3 Error and success rate depending on the sensor noise level

Sensor noise	S_r	e_x(cells)	e_y(cells)	e_θ(degrees)
0.5	100	0.0074 ± 0.0128	0.0235 ± 0.0295	0.0332 ± 0.0333
1.0	100	0.0213 ± 0.0346	0.0597 ± 0.0864	0.0650 ± 0.0704
2.5	98	0.0925 ± 0.1038	0.1837 ± 0.1876	0.1992 ± 0.2070
5.0	98	0.2355 ± 0.2553	0.5007 ± 0.3931	0.3604 ± 0.3759
7.5	96	0.2797 ± 0.2824	0.8544 ± 0.7115	0.6011 ± 0.5831
10.0	74	0.5315 ± 0.3790	1.1011 ± 0.7711	1.0305 ± 0.7043

Simulated map. True location: (300, 50, 90). Sensor noise in standard deviation. S_r in %. Errors in mean $\pm$ standard deviation. $N_P = 200$

Tables 4 and 5. In this case, the success rate is optimum when the population size is 50. The algorithm converges in almost all cases to the correct pose up to 2.5 % of sensor noise. The deterioration of the results is significant with a 10 % of noise.

Tables 6 and 7 show the results for $\mathbf{p}_{a3}$. An optimum population size is 60 or more in this case. The success rate is less influenced by the noise when the robot is located at this place. The success rate is still 100 % even with a 7.5 % of sensor noise.

In order to obtain a more illustrative comparison between points of interest, the influence of the sensor noise and the population size on the position error can be observed in Fig. 3. In the left part of the figure, the position error is drawn depending on the sensor noise. In the right part of the figure, the same error is plotted against the population size. The reader has to notice that this figure shows the same results presented in the previous tables (regarding the position errors) in a graphic format. It can be easily deduced that the position error grows linearly with the sensor noise. Regarding the population size, the position error does not depend on this variable. The error stays in the same range when the population size is changed.

Because the sensor noise is proportional to the distances measured by the laser scanner, larger errors are expected in larger areas where the closest obstacles are far away from the robot. The worst errors are obtained when the robot is in $\mathbf{p}_{a3}$. It can be seen that this point corresponds to the largest area. In smaller zones, such as $\mathbf{p}_{a1}$ and $\mathbf{p}_{a2}$, the localization error is smaller.

For a fixed noise (Tables 2, 4 and 6), the errors are in the interval [2.57, 27.13] mm in position and [0.03, 0.11] degrees in orientation. These errors are slightly lower than those obtained in our previous work [5, 24], and they are low enough to conclude that the GL problem is efficiently solved.

More results from different random places are given in Table 8. For simplicity, only the errors with 1 % noise and optimum population size (lowest N_P with maximum S_r) are given. As can be observed, the method performance is similar for these new places. The errors are within the same interval and the success rates present optimum values.

Table 4 Error and success rate depending on the population size

N_P	S_r	e_x(cells)	e_y(cells)	e_θ(degrees)
20	56	0.0712 ± 0.1093	0.0126 ± 0.0126	0.0340 ± 0.0372
30	80	0.0626 ± 0.0901	0.0151 ± 0.0099	0.0414 ± 0.0319
40	94	0.0624 ± 0.0917	0.0115 ± 0.0131	0.0399 ± 0.0390
50	100	0.0601 ± 0.0512	0.0128 ± 0.0122	0.0338 ± 0.0311
60	100	0.0547 ± 0.0605	0.0140 ± 0.0155	0.0424 ± 0.0422
70	100	0.0403 ± 0.0364	0.0099 ± 0.0069	0.0290 ± 0.0281
80	100	0.0400 ± 0.0358	0.0099 ± 0.0079	0.0327 ± 0.0320

Simulated map. True location: (150, 105, 0). Sensor noise: standard deviation of 1 %. S_r in %. Errors in mean $\pm$ standard deviation

Table 5 Error and success rate depending on the sensor noise level

Sensor noise	S_r	e_x(cells)	e_y(cells)	e_θ(degrees)
0.5	100	0.0233 ± 0.0216	0.0067 ± 0.0037	0.0130 ± 0.0132
1.0	100	0.0400 ± 0.0358	0.0099 ± 0.0079	0.0327 ± 0.0320
2.5	98	0.1886 ± 0.2262	0.0312 ± 0.0259	0.1084 ± 0.0851
5.0	80	0.4573 ± 0.3482	0.0780 ± 0.0798	0.1850 ± 0.1595
7.5	80	0.6351 ± 0.3687	0.0922 ± 0.0853	0.1920 ± 0.2016
10.0	68	0.8161 ± 0.4292	0.1581 ± 0.1912	0.3522 ± 0.2479

Simulated map. True location: (150, 105, 0). Sensor noise in standard deviation. S_r in %. Errors in mean $\pm$ standard deviation. $N_P = 80$

Table 6 Error and success rate depending on the population size

N_P	S_r	e_x(cells)	e_y(cells)	e_θ(degrees)
20	80	0.2312 ± 0.1835	0.0800 ± 0.0519	0.1012 ± 0.0791
30	94	0.2586 ± 0.2533	0.0819 ± 0.0611	0.1200 ± 0.0815
40	98	0.2244 ± 0.2462	0.0731 ± 0.0496	0.1013 ± 0.0681
50	98	0.2400 ± 0.2632	0.0870 ± 0.0541	0.0993 ± 0.0666
60	100	0.2229 ± 0.2242	0.0743 ± 0.0519	0.0959 ± 0.0600
70	100	0.2551 ± 0.2547	0.0906 ± 0.0554	0.0896 ± 0.0715
80	100	0.2600 ± 0.2342	0.0993 ± 0.0683	0.1149 ± 0.1028
90	100	0.2244 ± 0.1514	0.0798 ± 0.0698	0.1163 ± 0.0859
100	100	0.2523 ± 0.2444	0.0855 ± 0.0718	0.0988 ± 0.0827

Simulated map. True location: (550, 400, 0). Sensor noise: standard deviation of 1%. S_r in %. Errors in mean $\pm$ standard deviation

Table 7 Error and success rate depending on the sensor noise level

Sensor noise	S_r	e_x(cells)	e_y(cells)	e_θ(degrees)
0.5	100	0.0543 ± 0.0628	0.0439 ± 0.0387	0.0504 ± 0.0463
1.0	100	0.2244 ± 0.1514	0.0798 ± 0.0698	0.1163 ± 0.0859
2.5	100	0.5695 ± 0.3788	0.1521 ± 0.1216	0.2391 ± 0.1756
5.0	100	0.8449 ± 0.7479	0.3469 ± 0.2947	0.4425 ± 0.3888
7.5	100	1.4733 ± 0.9826	0.7037 ± 0.4973	0.7951 ± 0.6600
10.0	90	1.9609 ± 1.4879	0.8846 ± 0.6723	1.0414 ± 0.8635

Simulated map. True location: (550, 400, 0). Sensor noise in standard deviation. S_r in %. Errors in mean $\pm$ standard deviation. $N_P = 90$

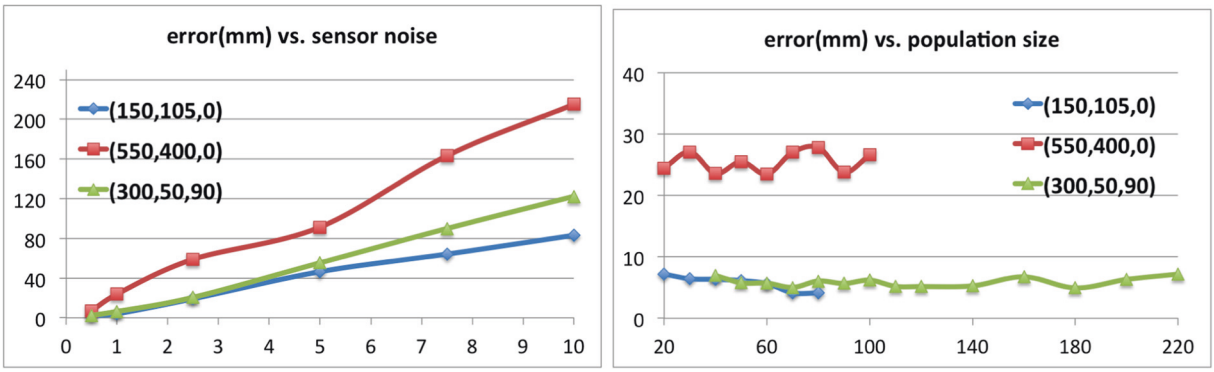

Fig. 3 Left: position error vs. sensor noise. Right: position error vs. population size. Simulated map. Sensor noise in standard deviation (%). Robot's poses between brackets

5.2 Learned Map

Similar experiments have been conducted in the real map of Fig. 4 to test the algorithm performance in this type of environments[4]. The area covered by this map is 29×29 m^2, thus it has been assumed that the cell size is 5 cm. This is a medium-size map with highly cluttered areas (Fig. 5). In general, this map is very similar to the architectural plan in corridors and halls, but it is completely different in offices because there are many obstacles.

In this case, the points of interest will be:

1. $\mathbf{p}_{r1} = (100, 450, 45)$. This place is a hallway.
2. $\mathbf{p}_{r2} = (470, 425, 37)$. The robot is located in an office. It is a cluttered area with many obstacles.
3. $\mathbf{p}_{r3} = (550, 400, 0)$. This place is an office and there are many offices with similar dimensions and furniture. Less cluttered than $\mathbf{p}_{r2}$.

The influence of the population size is measured in Table 9 for the first location. This place corresponds to a hallway. Since the area that can be measured by the sensors is large and there are no similar places, this place will be one of the easiest in this map. The success rate reaches the 100 % when the population size is 160. Therefore, this size will be adequate in this case.

The effect of the noise level when the robot is situated in $\mathbf{p}_{r1}$ has been measured in Table 10. As in the previous cases in the simulated map, the accuracy decreases when the sensor noise increases. The algorithm converges in all cases to the correct pose up to 2.5 % of noise. The success rate shows high values even with a 10 % of noise.

Tables 11 and 12 show the results for $\mathbf{p}_{r2}$. The robot is inside a cluttered room in this case (Fig. 5). The basin of attraction of the local minimum is smaller, which means that small changes in position or orientation cause very different values in the cost function. The success rate reaches the maximum value (96 %) when the population size is 600. The GL method does not succeed in some cases because it is a very challenging location. However, the success rate presents a promising value, as will be demonstrated in the next section when comparing to the previous version of the filter. The algorithm converges in almost all cases to the correct pose when the sensor noise is 1 %. The deterioration of the results is significant with a 5 % of noise.

Tables 13 and 14 display the results when the robot is located at $\mathbf{p}_{r3}$. This place is an office and there are many offices with similar dimensions and furniture, but it is less cluttered than $\mathbf{p}_{r1}$. The size that is required is 480 in this case. The success rate is less influenced by the noise when compared to $\mathbf{p}_{r2}$. The success rate is optimum even with a 5 % of sensor noise. As in the previous location, the localization process fails in some cases.

The comparison between all locations according to the sensor noise and the population size can be seen in Fig. 6. Like in the architectural plans, the position error grows linearly with the sensor noise and it does not depend on the population size.

[4]Thanks to Dieter Fox for making available this map.

Table 8 Error and success rate for different random places

Location	N_P	S_r	e_x(cells)	e_y(cells)	e_θ(degrees)
(802,123,23)	100	100	0.0516 ± 0.0622	0.0354 ± 0.0393	0.0434 ± 0.0502
(252,516,167)	120	98	0.1018 ± 0.0929	0.0333 ± 0.0470	0.0522 ± 0.0478
(688,470,271)	140	98	0.0359 ± 0.0232	0.0163 ± 0.0101	0.0458 ± 0.0374
(561,150,100)	80	100	0.0411 ± 0.0336	0.0687 ± 0.0492	0.1358 ± 0.1171
(551,498,43)	80	98	0.0773 ± 0.0661	0.0573 ± 0.0680	0.1022 ± 0.0698
(300,172,255)	100	100	0.0241 ± 0.0304	0.1511 ± 0.1063	0.0562 ± 0.0852

Simulated map. Sensor noise: standard deviation of 1%. S_r in %. N_P is the lowest that optains the maximum S_r. Errors in mean $\pm$ standard deviation

The worst errors are obtained for $\mathbf{p}_{r1}$ (larger sensing area). For a fixed noise (Tables 9, 11 and 13), the errors are in the interval $[0.27, 3.17]$ mm in position and $[0.01, 0.05]$ degrees in orientation. These errors are slightly lower than the errors in the simulated map. The main reason is that the map resolution is higher in this section.

In general, the population size should be larger and the success rate is more influenced by the sensor noise in learned maps.

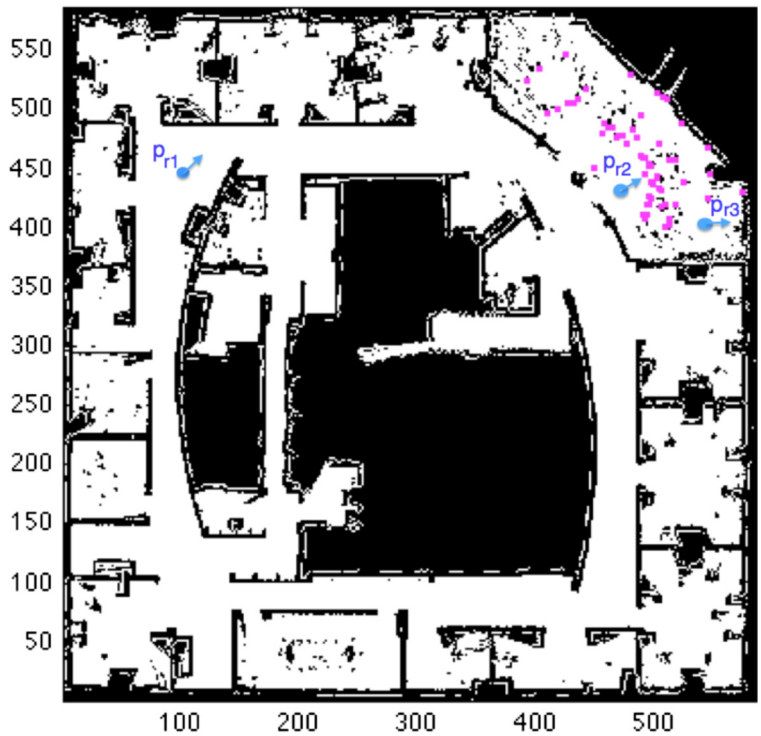

Fig. 4 GL in an learned map (Intel lab). All units in cells. Laser readings in purple. Robot's location in blue. Points of study marked with dots and orientation with arrows

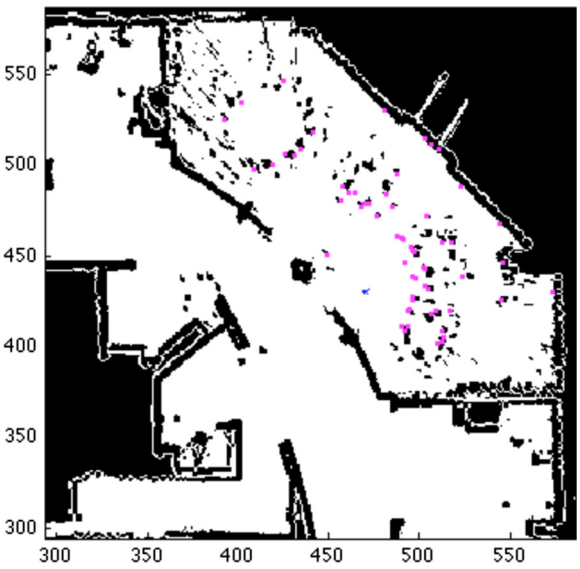

Fig. 5 Map details in a cluttered area. Zoom of the region of interest in the learned map of Fig. 4. All units in cells. Laser readings in purple. Robot's location in blue

More results from different random places are given in Table 15. For simplicity, only the errors with 1 % noise and optimum population size (lowest N_P with maximum S_r) are given. The method performance is similar for these new places.

5.3 Results analysis - Comparison with Previous Version

The real benefits of the new method can be appreciated if it is compared to the previous version of the GL filter. To do that, we will use the results obtained in the previous sections.

For the simulated map, the localization errors and the success rates computed for the points of interest with a 1 % noise and the optimum number of particles (Tables 3, 5, and 7) are rewritten in the top rows of Table 16 (DE-MC in the table). e_d is the position error and $iter$ represents the average number of iterations that are needed to converge. The same measurements are calculated using the traditional version of the DE-based GL algorithm detailed in Section 3.3 (DE in the table). For each location, the number of particles is increased until the maximum success rate is reached.

If the traditional version is run with the same population size (first row for each location, N_P equal to the population size used for the new version), the success rate is much lower in all cases. It means that the old version is less robust and the population size must be increased to obtain an optimum performance. Therefore, the population size required by the new technique is much lower than the population size that is needed by the previous version. In particular, for $\mathbf{p}_{a1}$, $\mathbf{p}_{a2}$, and $\mathbf{p}_{a3}$, the population size has to be multiplied by

Table 9 Error and success rate depending on the population size

N_P	S_r	e_x(cells)	e_y(cells)	e_θ(degrees)
40	72	0.0337 ± 0.0313	0.0389 ± 0.0409	0.0461 ± 0.0632
60	76	0.0312 ± 0.0347	0.0423 ± 0.0335	0.0426 ± 0.0445
80	88	0.0483 ± 0.0413	0.0411 ± 0.0391	0.0522 ± 0.0589
100	94	0.0496 ± 0.0554	0.0312 ± 0.0285	0.0409 ± 0.0422
120	94	0.0346 ± 0.0417	0.0357 ± 0.0262	0.0545 ± 0.0505
140	98	0.0329 ± 0.0411	0.0413 ± 0.0499	0.0506 ± 0.0659
160	100	0.0394 ± 0.0429	0.0324 ± 0.0328	0.0382 ± 0.0347
180	100	0.0423 ± 0.0544	0.0260 ± 0.0220	0.0306 ± 0.0271

Real map. True location: (100, 450, 45). Sensor noise: standard deviation of 1 %. S_r in percentage. Errors in mean $\pm$ standard deviation

Table 10 Error and success rate depending on the sensor noise level

Sensor noise	S_r	e_x(cells)	e_y(cells)	e_θ(degrees)
0.5	100	0.0173 ± 0.0132	0.0210 ± 0.0193	0.0270 ± 0.0332
1.0	100	0.0423 ± 0.0544	0.0260 ± 0.0220	0.0306 ± 0.0271
2.5	100	0.0922 ± 0.1195	0.0615 ± 0.0599	0.1012 ± 0.1001
5.0	92	0.2060 ± 0.2317	0.2182 ± 0.1857	0.2216 ± 0.1939
7.5	92	0.3094 ± 0.2248	0.2127 ± 0.2100	0.2801 ± 0.2320
10.0	90	0.4866 ± 0.3660	0.3752 ± 0.3556	0.3504 ± 0.2630

Real map. True location: (100, 450, 45). Sensor noise in standard deviation. S_r in %. Errors in mean $\pm$ standard deviation. $N_P = 180$

Table 11 Error and success rate depending on the population size

N_P	S_r	e_x(cells)	e_y(cells)	e_θ(degrees)
100	32	0.0084 ± 0.0056	0.0095 ± 0.0065	0.0152 ± 0.0116
200	62	0.0157 ± 0.0233	0.0140 ± 0.0193	0.0231 ± 0.0275
300	66	0.0085 ± 0.0061	0.0079 ± 0.0065	0.0140 ± 0.0102
400	82	0.0214 ± 0.0771	0.0145 ± 0.0147	0.0407 ± 0.0615
500	94	0.0266 ± 0.0206	0.0121 ± 0.0076	0.0039 ± 0.0027
600	96	0.0100 ± 0.0082	0.0053 ± 0.0046	0.0084 ± 0.0068

Real map. True location: (470, 425, 37). Sensor noise: standard deviation of 1%. S_r in percentage. Errors in mean $\pm$ standard deviation

Table 12 Error and success rate depending on the sensor noise level

Sensor noise	S_r	e_x(cells)	e_y(cells)	e_θ(degrees)
0.5	100	0.0122 ± 0.0099	0.0059 ± 0.0049	0.0101 ± 0.0063
1.0	96	0.0100 ± 0.0082	0.0053 ± 0.0046	0.0084 ± 0.0068
2.5	76	0.0110 ± 0.0068	0.0121 ± 0.0069	0.0200 ± 0.0137
5.0	62	0.0166 ± 0.0136	0.0156 ± 0.0118	0.0236 ± 0.0148
7.5	56	0.0170 ± 0.0148	0.0303 ± 0.0320	0.0275 ± 0.0160
10.0	44	0.0528 ± 0.1022	0.0495 ± 0.0629	0.0505 ± 0.0745

Real map. True location: (470, 425, 37). Sensor noise in standard deviation. S_r in %. Errors in mean $\pm$ standard deviation. $N_P = 600$

Table 13 Error and success rate depending on the population size

N_P	S_r	e_x(cells)	e_y(cells)	e_θ(degrees)
80	36	0.0118 ± 0.0058	0.0048 ± 0.0028	0.0106 ± 0.0053
160	72	0.0110 ± 0.0065	0.0042 ± 0.0033	0.0078 ± 0.0064
240	78	0.0127 ± 0.0061	0.0047 ± 0.0058	0.0096 ± 0.0097
320	84	0.0141 ± 0.0112	0.0052 ± 0.0058	0.0093 ± 0.0078
400	92	0.0106 ± 0.0042	0.0047 ± 0.0033	0.0093 ± 0.0067
480	96	0.0095 ± 0.0061	0.0036 ± 0.0025	0.0074 ± 0.0060
560	96	0.0130 ± 0.0067	0.0056 ± 0.0064	0.0103 ± 0.0099

Real map. True location: (550, 400, 0). Sensor noise: standard deviation of 1%. S_r in percentage. Errors in mean $\pm$ standard deviation

Table 14 Error and success rate depending on the sensor noise level

Sensor noise	S_r	e_x(cells)	e_y(cells)	e_θ(degrees)
0.5	96	0.0107 ± 0.0037	0.0042 ± 0.0029	0.0076 ± 0.0053
1.0	96	0.0095 ± 0.0061	0.0036 ± 0.0025	0.0074 ± 0.0060
2.5	96	0.0222 ± 0.0163	0.0175 ± 0.0169	0.0241 ± 0.0217
5.0	82	0.0384 ± 0.0237	0.0259 ± 0.0199	0.0623 ± 0.0715
7.5	80	0.0676 ± 0.0773	0.0438 ± 0.0796	0.1068 ± 0.1323
10.0	80	0.0604 ± 0.0455	0.0638 ± 0.1231	0.1334 ± 0.1512

Real map. True location: (550, 400, 0). Sensor noise in standard deviation. S_r in %. Errors in mean $\pm$ standard deviation. $N_P = 480$

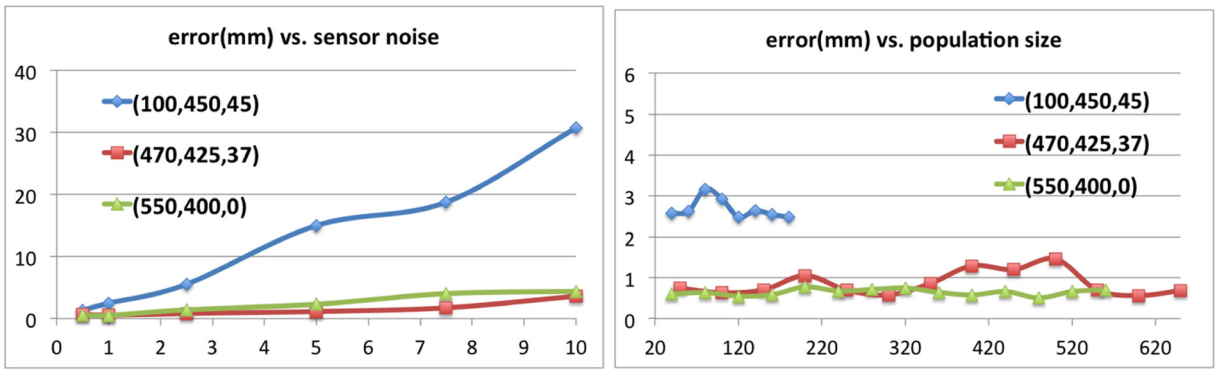

Fig. 6 Left: position error vs. sensor noise. Right: position error vs. population size. Learned map. Sensor noise in standard deviation (%). Robot's poses between brackets

Table 15 Error and success rate for different random places

Location	N_P	S_r	e_x(cells)	e_y(cells)	e_θ(degrees)
(320,460,10)	60	100	0.0464 ± 0.0362	0.0324 ± 0.0353	0.0390 ± 0.0291
(531,48,85)	120	98	0.0309 ± 0.0222	0.0349 ± 0.0355	0.0415 ± 0.0258
(440,322,21)	60	100	0.0502 ± 0.0460	0.1201 ± 0.1251	0.0598 ± 0.0551
(75,60,3)	240	96	0.0284 ± 0.0188	0.0118 ± 0.0117	0.0547 ± 0.0582
(210,401,167)	140	100	0.0169 ± 0.0196	0.0200 ± 0.0232	0.0417 ± 0.0403
(351,477,99)	80	98	0.0266 ± 0.0265	0.0330 ± 0.0245	0.0230 ± 0.0243

Real map. Sensor noise: standard deviation of 1%. S_r in %. N_P is the lowest that optains the maximum S_r. Errors in mean $\pm$ standard deviation

Table 16 Comparison with traditional DE-based GL filter

DE-MC					
Location	N_P	S_r	e_d(mm)	e_θ(degrees)	$iter$
(300,50,90)	200	100	6.3386 ± 9.3071	0.0650 ± 0.0704	2091
(150,105,0)	80	100	4.1207 ± 3.6661	0.0327 ± 0.0632	1002
(550,400,0)	90	100	23.8167 ± 16.6715	0.1163 ± 0.0859	600
DE					
Location	N_P	S_r	e_d(mm)	e_θ(degrees)	$iter$
(300,50,90)	200	64	6.5947 ± 4.7508	0.0980 ± 0.0625	304
(300,50,90)	400	82	7.5427 ± 9.7576	0.1093 ± 0.0933	343
(300,50,90)	600	98	8.3072 ± 9.1914	0.1109 ± 0.0781	260
(300,50,90)	800	98	6.5076 ± 7.9818	0.0873 ± 0.0785	300
(150,105,0)	80	44	19.8139 ± 17.6466	0.0916 ± 0.0634	242
(150,105,0)	160	76	17.1486 ± 11.9873	0.0520 ± 0.0520	231
(150,105,0)	240	78	17.8856 ± 19.9534	0.0610 ± 0.0480	221
(150,105,0)	320	94	14.9048 ± 18.6853	0.0540 ± 0.0447	234
(150,105,0)	400	98	15.0640 ± 14.1963	0.0508 ± 0.0409	244
(150,105,0)	480	100	12.9933 ± 12.5968	0.0494 ± 0.0397	251
(550,400,0)	90	82	32.6667 ± 28.7437	0.0846 ± 0.0754	110
(550,400,0)	135	88	26.5698 ± 30.1158	0.1007 ± 0.0824	135
(550,400,0)	180	100	17.0854 ± 21.7391	0.0972 ± 0.0854	182

Simulated map. Sensor noise: standard deviation of 1 %. S_r in percentage. Errors in mean $\pm$ standard deviation

4, 6, and 2, respectively, to obtain the optimum performance regarding the success rate. Moreover, it is not possible to reach $S_r = 100\%$ when the robot is situated in $\mathbf{p}_{a1}$.

The errors of the new technique are in the interval [4.12, 23.82] mm in position and [0.03, 0.11] degrees in orientation, which are slightly lower than the errors of the classic version. It is possible to make a comparison between these errors and those obtained by other authors in 2D maps. The position error reported by Donoso et al. [25] is in the interval [8 cm, 15 cm]. Se et al. [59] have published an average position error equal to 7 cm and a rotation error of 1°. To the best of our knowledge, there are no research groups that have obtained localization errors significantly lower than our errors in planar maps. However, these comparisons have to be considered only as indicative figures. It is not easy to make a fair comparison between methods that rely on different concepts.

The results for the real map are introduced in Table 17. Similar conclusions can be drawn.

The population when using the traditional technique must be increased to obtain the optimum success rate. In particular, for $\mathbf{p}_{r1}$, $\mathbf{p}_{r2}$, and $\mathbf{p}_{r3}$, the population size has to be multiplied by 6, 4, and 2.5, respectively. Furthermore, when the robot is in $\mathbf{p}_{r2}$, the maximum success rate that can be obtained with the old version is equal to 50 % (much lower than the value obtained with the new technique). These facts reinforce the conclusion that the new method is more robust and reliable.

The errors of the new version are in the interval [0.51, 2.48] mm in position and [0.01, 0.03] degrees in orientation, which in most cases are lower than the errors of the traditional method.

The time complexity of both GL methods is

$$O(DE-MC) = O(DE) = iter \times N_P \times n, \quad (13)$$

where n represents the number of measurements of the laser scan. The computational times can be compared by analyzing the population size and the number of iterations to converge ($N_P \times iter$), because n is

Table 17 Comparison with traditional DE-based GL filter

DE-MC					
Location	N_P	S_r	e_d(mm)	e_θ(degrees)	$iter$
(100,450,45)	180	100	2.4826 ± 2.9340	0.0306 ± 0.0271	1202
(470,425,37)	600	96	0.5659 ± 0.4701	0.0084 ± 0.0068	4304
(550,400,0)	480	96	0.5080 ± 0.3296	0.0074 ± 0.0060	1308
DE					
Location	N_P	S_r	e_d(mm)	e_θ(degrees)	$iter$
(100,450,45)	180	54	1.1102 ± 0.3543	0.0187 ± 0.0227	245
(100,450,45)	360	76	3.1219 ± 2.9649	0.0446 ± 0.0411	291
(100,450,45)	540	76	2.6114 ± 2.5429	0.0486 ± 0.0495	300
(100,450,45)	720	80	3.7547 ± 4.1548	0.0651 ± 0.0705	322
(100,450,45)	900	94	3.1741 ± 3.8807	0.0534 ± 0.0699	306
(100,450,45)	1080	96	2.1037 ± 2.7321	0.0297 ± 0.0297	374
(470,425,37)	600	22	1.1102 ± 0.3543	0.0187 ± 0.0227	730
(470,425,37)	1200	24	1.2580 ± 2.0617	0.0183 ± 0.0192	752
(470,425,37)	1800	50	0.7738 ± 0.5423	0.0101 ± 0.0067	900
(470,425,37)	2400	50	0.6574 ± 0.4294	0.0135 ± 0.0082	948
(550,400,0)	480	52	2.0495 ± 1.2039	0.0930 ± 0.0220	452
(550,400,0)	720	72	1.6789 ± 0.8801	0.0262 ± 0.0181	422
(550,400,0)	960	78	1.6409 ± 0.9159	0.0193 ± 0.0143	358
(550,400,0)	1200	96	1.2801 ± 0.8180	0.0163 ± 0.0146	401

Real map. Sensor noise: standard deviation of 1%. S_r in percentage. Errors in mean $\pm$ standard deviation

the same for both methods. Observing the tables, the reduction of the convergence speed is compensated by the decrease of the population size.

6 Conclusions

A new method that combines the DE evolutionary algorithm and the population-based MCMC technique has been proposed in this paper to solve the GL problem. This new version maintains the advantages of both approaches. It inherits the statistical robustness of the MCMC technique and it keeps the exploration properties of the evolutionary filter. Besides, the DE-based mutation stage solves the scale and orientation problem of the jumps of the MCMC algorithms.

The experimental results lead us to conclude that this new method is an appropriate approach to solve the GL problem in both simulated and real maps, and it does not present any drawback when compared to the previous version of the DE filter. In the experiments carried out, we have measured the localization error and the success rate depending on the sensor noise and the population size.

An important conclusion that can be deduced from the experiments is that there is a significant improvement in the performance depending on the number of particles, which is strongly related to the parameter that we have defined as the success rate. The success rate has been computed when the robot is located at different places and it is increased in all experiments when compared to the traditional version if the same population size is used, making this new version a more suitable approach in challenging environments where it is difficult to obtain the true location.

The population requirements are much lower than the demands of our previous version of the DE-based GL filter. Although the population size of the MC sampling algorithm is huge when compared to the

DE method, this hybrid version needs less particles to obtain satisfactory results. In general, the population size should be larger and the success rate is more influenced by the sensor noise in learned maps.

The accuracy is slightly improved in position and orientation. If the sensor noise is increased, there is a low degradation of the estimation results because the error grows linearly with the sensor noise. The noise level that still allows a maximum success rate is higher than the noise of commercial devices.

The GL module developed here will be a useful tool when a map of the environment provided by perceptive sensors like laser range finders is available. With a single laser scan (which is a strong assumption), the mobile robot can obtain its own location. Besides, additional laser scans could be used to update the estimate. The accuracy shown in the experimental results is good enough to use the current method in manipulation tasks. Although the method presents a good performance regarding the sensor noise, high uncertainties in the sensors or the map will decrease the algorithm performance.

Since the sum of the squared errors is not the most adequate metric for environments with occlusions or dynamic obstacles because big errors strongly penalize the cost value, the current method will not be the most appropriate for these cases. In our previous work [5], a GL technique that relies on different cost function assumptions has obtained a great performance in this type of environments. An interesting work to carry out is to implement this different fitness function in the current method.

A detailed study about the convergence properties is a challenging task that must be accomplished in the future work because the computational cost is highly dependent on this factor.

The evolutionary nature of this filter causes many interesting features: it can deal with nonlinear state space dynamics and noise distributions; it does not require any assumptions on the shape of the posterior density; the computational resources are focused on the most relevant zones.

Acknowledgments The research leading to these results has received funding from the RoboCity2030-III-CM project (Robótica aplicada a la mejora de la calidad de vida de los ciudadanos. fase III; S2013/MIT-2748), funded by Programas de Actividades I+D en la Comunidad de Madrid and cofunded by Structural Funds of the EU.

References

1. Leonard, J.J., Durrant-Whyte, H.: Mobile robot localization by tracking geometric beacons. IEEE Trans. Robot. Autom. **7**, 376–382 (1991)
2. Storn, R., Price, K.: Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. **11**, 341–359 (1997)
3. Moreno, L., Garrido, S., Muñoz, M.L.: Evolutionary filter for robust mobile robot localization. Robot. Auton. Syst. **54**(7), 590–600 (2006)
4. Martín, F., Moreno, L., Garrido, S., Blanco, D.: High-accuracy global localization filter for three-dimensional environments. Robotica **30**, 363–378 (2011)
5. Martín, F., Moreno, L., Blanco, D., Muñoz, M.L.: Kullback–Leibler divergence-based global localization for mobile robots. Robot. Auton. Syst. **62**, 120–130 (2014)
6. Metropolis, N., Ulam, S.: The Monte Carlo method. J. Am. Stat. Assoc. **44**(247), 335–341 (1949)
7. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equations of state calculations by fast computing machines. J. Chem. Phys. **21**, 1087–1091 (1953)
8. Hastings, W.K.: Monte Carlo sampling methods using Markov chains and their applications. Biometrika **57**, 97–109 (1970)
9. Gilks, W.R., Richardson, S., Spiegelhalter, D.J.: Markov chain monte carlo in practice. Chapman & Hall, London (1996)
10. Andrieu, C., de Freitas, N., Doucet, A., Jordan, M.: An introduction to MCMC for machine learning. Mach. Learn. **50**, 5–43 (2003)
11. Braak, C.J.F.T.: A Markov chain Monte Carlo version of the genetic algorithm differential evolution: easy Bayesian computing for real parameter spaces. Stat. Comput. **16**, 239–249 (2006)
12. Burgard, W., Fox, D., Henning, D., Schmidt, T.: Estimating the absolute position of a mobile robot using position probability grids. In: Proceedings of the fourteenth national conference on artificial intelligence (AAAI'96) (1996)
13. Fox, D., Hightower, J., Liao, L., Schulz, D., Borriello, G.: Bayesian filters for location estimation. Pervasive Computing **2**, 24–33 (2003)
14. Thrun, S., Burgard, W., Fox, D.: A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping. In: Proceedings of the IEEE international conference on robotics and automation (ICRA'00) (2000)
15. Gamallo, C., Regueiro, C.V., Quintía, P., Mucientes, M.: Omnivision-based KLD-Monte Carlo Localization. Robot. Auton. Syst. **58**, 295–305 (2010)
16. Zhang, L., Zapata, R., Lepinay, P.: Self-adaptive Monte-Carlo localization for mobile robots using range sensors. In: Proceedings of the lEEW/RSJ international conference on intelligent robots and system (IROS'09) (2009)
17. Fox, D., Burgard, W., Thrun, S.: Markov localization for mobile robots in dynamic environments. J. Artif. Intell. Res. **11**(11), 391–427 (1999)
18. Thrun, S., Fox, D., Burgard, W., Dellaert, F.: Robust Monte Carlo localization for mobile robots. Artif. Intell. **128**, 99–141 (2001)

19. Ronghua, L., Bingrong, H.: Coevolution based adaptive Monte Carlo localization (CEAMCL). Int. J. Adv. Robot. Syst. **1**(3), 183–190 (2004)
20. Biswas, J., Coltin, B., Veloso, M.: Corrective gradient refinement for mobile robot localization. In: Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS'11) (2011)
21. Grisetti, G., Grzonka, S., Stachniss, C., Pfaff, P., Burgard, W.: Efficient Estimation of Accurate Maximum Likelihood Maps in 3D. In: Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS'07) (2007)
22. Montemerlo, M., Thrun, S.: FastSLAM 2.0, FastSLAM: A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics (2007)
23. Zhang, L., Zapata, R., Lépinay, P.: Self-adaptive Monte Carlo localization for mobile robots using range finders. Robotica **30**, 229–244 (2011)
24. Moreno, L., Blanco, D., Muñoz, M.L., Garrido, S.: L1–L2-norm comparison in global localization of mobile robots. Robot. Auton. Syst. **59**, 597–610 (2011)
25. Donoso-Aguirre, F., Bustos-Salas, J.P., Torres-Torriti, M., Guesalaga, A.: Mobile robot localization using the Hausdorff distance. Robotica **26**, 129–141 (2008)
26. Fox, D., Burgard, W.: Active Markov localization for mobile robots. Robot. Auton. Syst. **25**, 195–207 (1998)
27. Arras, K.O., Castellanos, J.A., Siegwart, R.: Feature-based multi-hypothesis localization and tracking for mobile robots using geeometric constraints. In: Proceedings of the IEEE international conference on robotics and automation (ICRA'02), (Washington DC, USA), pp. 1371–1377 (2002)
28. Back, T., Fogel, D.B., Michalewicz, Z.: Evolutionary computation I: basic algorithms and operators. IOP Publishing Ltd (2000)
29. Back, T., Fogel, D.B., Michalewicz, Z.: Evolutionary computation II: advanced algorithms and operators. IOP Publishing Ltd (2000)
30. Vahdat, A.R., Ashrafoddin, N.N., Ghidary, S.S.: Mobile robot global localization using differential evolution and particle swarm optimization. In: Proceedings of the congress on evolutionary computation (CEC'07) (2007)
31. Lisowski, M.: Differential evoution approach to the localization problem for mobile robots. Master's thesis, Technical University of Denmark (2009)
32. Lisowski, M., Fan, Z., Ravn, O.: Differential evolution to enhance localization of mobile robots. In: Proceedings of the 2011 IEEE international conference on fuzzy systems, (Taipei, Taiwan), pp. 241–247 (June 2011)
33. Geem, Z., Kim, J., Loganathan, G.: A new heuristic optimization algorithm: harmony search. Simulation **76**(2), 60–78 (2001)
34. Mirkhania, M., Forsatib, R., Shahric, M., Moayedikiad, A.: A novel efficient algorithm for mobile robot localization, Robotics and Autonomous Systems (2013)
35. Kwok, N.M., Liu, D.K., Dissanayake, G.: Evolutionary computing based mobile robot localization. Artif. Intell. **19**, 857–868 (2006)
36. Cox, I.J., Leonard, J.J.: Modeling a dynamic environment using a Bayesian multi hypothesis approach. Artif. Intell. **66**, 311–44 (1994)
37. Austin, D.J., Jensfelt, P.: Using multiple Gaussian hypotheses to represent probability distributions for mobile robot localization. In: Proceedings of the IEEE international conference on robotics and automation (ICRA'00), (San Francisco, USA), pp. 1036–1041 (2000)
38. Jensfelt, P., Kristensen, S.: Active global localization for a mobile robot using multiple hypothesis tracking. IEEE Trans. Robot. Autom. **17**(5), 748–760 (2001)
39. He, T., Hirose, S.: A global localization approach based on Line-segment relation matching technique. Robot. Auton. Syst. **60**, 95–112 (2012)
40. Pfaff, P., Plagemann, C., Burgard, W.: Gaussian mixture models for probabilistic localization. In: Proceedings of IEEE international conference on robotics and automation (ICRA'08), (Pasadena, CA, USA) (2008)
41. Jochmann, G., Kerner, S., Tasse, S., Urbann, O.: Efficient multi-hypotheses unscented kalman filtering for robust localization, RoboCup 2011: Robot Soccer World Cup XV (pp. 222-233) (2012)
42. Robert, C.P., Casella, G.: Monte Carlo statistical methods, 2nd edn. Springer, New York (2004)
43. Geweke, J.: Bayesian inference in econometric models using Monte Carlo intergration. Econometrica **24**, 1317–1399 (1989)
44. Rubinstein, R.Y.: Simulation and the Monte Carlo method. John Wiley & Sons (1981)
45. Rubin, D.B.: Using the SIR algorithm to simulate posterior distributions. Bayesian Statistics **3**(1), 395–402 (1988)
46. Gilks, W.R., Roberts, G.O., George, E.I.: Adaptive direction sampling. The Statistician **43**, 179–189 (1994)
47. Liu, J.S., Liang, F., Wong, W.H.: The use of multipletry method and local optimization in metropolis sampling. J. Am. Stat. Assoc. **94**, 121–134 (2000)
48. Geyer, C.J.: Markov chain Monte Carlo maximum likelihood, in computing science and statistics. In: Keramigas, E.M. (ed.) Proceedings of the 23rd symposium on the interface, pp. 153–163 (1991)
49. Hukushima, K., Nemoto, K.: Exchange Monte Carlo method and application to spin glass simulations. J. Phys. Soc. Jpn. **65**(6), 1604–1608 (1996)
50. Liang, F.M., Wong, W.H.: Real-parameter evolutionary Monte Carlo with applications to Bayesian mixture models. J. Am. Stat. Assoc. **96**, 653–666 (2001)
51. Kou, S.C., Zhou, Q., Wong, W.H.: Equienergy sampler with applications to statistical inference and statistical mechanics. Ann. Stat. **32**, 1581–1619 (2006)
52. Liang, F.M.: Dynamically weighted importance sampling in Monte Carlo computation. J. Am. Stat. Assoc. **97**, 807–821 (2002)
53. Laskey, K.B., Myers, J.W.: Population Markov Chain Monte Carlo. Mach. Learn. **50**, 175–196 (2003)
54. Martín, F., Moreno, L., Muñoz, M.L., Blanco, D.: Initial population size estimation for a Differential-Evolution-based global localization filter. Int. J. Robot. Autom. **29**(3), 245–258 (2014)
55. Zaharie, D.: Critical values for the control parameters of differential evolution algorithms (2002)

56. Goldberg, D.E.: Genetic algorithm in search, optimization and machine learning. Addison Wesley Publishing Company (1989)
57. Waagepetersen, R., Sorensen, D.: A tutorial on reversible jump MCMC with a view toward applications in QTL-mapping. Int. Stat. Rev. **69**, 49–61 (2001)
58. Gelman, A., Carlin, J.B., Stern, H.S., D B, R.: Bayesian data analysis, 2nd edn. Chapman & Hall, London (2004)
59. Se, S., Lowe, D.G., Little, J.J.: Vision-based global localization and mapping for mobile robots. IEEE Trans. Robot. **21**, 3 (2005)