Square Root Unscented Particle Filtering for Grid Mapping

Simone Zandara and Ann Nicholson

Monash University, Clayton 3800, VIC

Abstract. In robotics, a key problem is for a robot to explore its environment and use the information gathered by its sensors to jointly produce a map of its environment, together with an estimate of its position: so-called SLAM (Simultaneous Localization and Mapping) [12]. Various filtering methods – Particle Filtering, and derived Kalman Filter methods (Extended, Unscented) – have been applied successfully to SLAM. We present a new algorithm that adapts the Square Root Unscented Transformation [13], previously only applied to feature based maps [5], to grid mapping. We also present a new method for the so-called pose-correction step in the algorithm. Experimental results show improved computational performance on more complex grid maps compared to an existing grid based particle filtering algorithm.

1 Introduction

This paper addresses the classical robotics problem of a robot needing to explore its environment and use the information gathered by its sensors to jointly produce a map of its environment together with an estimate of its position: so-called SLAM (Simultaneous Localization and Mapping) [12]. SLAM is an inherently sequential problem, which suggested the use of Bayesian Filters. Early path tracking methods such as the Kalman Filter (KF) [12] are based on the idea that, given knowledge about the position and heading of a moving object, observed data can be used to track that object; the problem becomes more difficult when the sensors are mounted on the moving object itself. The Extended KF (EKF) [12] is a successful method for modeling the uncertainty of a robot's noisy measurements (e.g. encoders, range finders), however it is unstable and imprecise because of linearization[1]; the Unscented KF (UKF) [8,15] avoids such approximation.

Particle filtering is a popular sequential estimation technique based on the generation of multiple samples from the distribution that is believed to approximate the true distribution. Studies have shown that particle filtering can better approximate a robot's real position than KF techniques, but the method is computationally intense because every particle is updated through a lightweight KF derived technique. Particle filtering has been used successfully to solve SLAM for both grid and feature based maps [12]. Grid maps generate a representation of the surrounding (usually closed) environment through a grid of cells, using raw sensor data for precisation. In contrast, feature based maps describe a (typically

A. Nicholson and X. Li (Eds.): AI 2009, LNAI 5866, pp. 121–130, 2009.

open) environment through a set of observed features, usually sensor readings (e.g. range and bearing).

Unscented Particle Filtering [13] for SLAM [7] has been successfully applied to feature based mapping. It mixes Particle Filtering and UKF by updating particles using an unscented transformation, rather than updating the uncertainty through Taylor linearisation of the update functions. Our research draws from this wide spectrum of KF and particle filtering algorithms; in Section 2 we provide a brief introduction (see [16] for more details). We present a new algorithm (Section 3) which we call SRUPF-GM (Square Root Unscented Particle Filtering for Grid Mapping) to adapt Unscented Particle Filtering to *grid* based maps. We also present a new method for the so-called pose-correction step in the algorithm. In Section 4 we present experiments comparing its performance to the well-known GMapping algorithm [2], on three grid environments. Our results show that while SRUPF-GM is slower on simpler maps, it is faster on more complex maps, and its performance does not degrade as quickly as GMapping as the number of particles increases.

2 Background

2.1 Particle Filtering for SLAM Problem

The main idea behind Particle Filtering applied to SLAM [11] is to estimate sequentially the joint posterior $p(x_t, m|x_{t-1}, z_t, u_t)$ for the robot's state x (which is usually its position X, Y and bearing θ), and the map of the environment m. This is done using its previous state (x at time $t-1$), odometry information (u at time t), that is, the robot's own measurements of its movements from its wheels, and the measurements from sensors (z at time t), e.g. lasers, sonars, etc.

$p(x_t, m|x_{t-1}, z_t, u_t)$ has no closed solution. In Particle Filtering, its estimation can be decomposed by maintaining a set of n poses, S, that make up a region of uncertainty (a Monte Carlo method); these poses are called *particles* [10]. Each particle has its own position, map and uncertainty; the latter is represented by a Gaussian defined by its position μ (mean) and covariance Σ. The generated distribution is called the *proposal*. The proposal is meant to represent the movement of the robot and is usually derived from u and z. It is proven that to solve SLAM the noise that is inherent in the odometry and sensor readings must be modeled; all SLAM algorithms add a certain amount of noise to do this. The final set of particles becomes the robot's final pose uncertainty ellipse. Another key characteristic of Particle Filtering is *resampling*, which aims to eliminate those particles which are believed to poorly represent the true value. This resampling is done using a weighting mechanism, where each particle has an associated weight. A high-level description of the particle filtering algorithm is given in Algorithm 1.

2.2 Unscented Transformation

The odometry update in particle filtering can be implemented in different ways; thus far the most well-known technique is based on an EKF update, which unfortunately introduces unwanted complexity and error [1]. The Unscented

Algorithm 1. Particle filtering algorithm

```
while Robot received data from sensors do
  for all x_i in S do
    Apply_odometry_to_update_robots_position(x_i,u_t)
    Apply_sensor_measurement_to_correct_pose(x_i,z_t)
    Generate_Map(x_i,z_t)
    x_i updated by sampling new pose
    Update_Weight(x_i)
  end for
  S = resample();
end while
```

Transformation (UT) [15] aims to avoid Jacobian calculations and has been proved to better approximate the true values. Instead of linearizing odometry and measurement functions, the UT generates a better approximated Gaussian that represents the true distribution through a set of so-called *Sigma points*. It has been used to generate feature based maps using laser or visual sensors [8,5]. Sigma points are generated deterministically using the previous mean and added noise.

Known Problems. The Unscented Transformation is proven to be more accurate than EKF, but its calculation is difficult. During the selection of the Sigma points one needs to calculate the square root of the augmented covariance matrix; this is usually done through a Cholesky Factorization. However, this method needs the matrix to be positive-defined, otherwise the method dramatically fails. Unfortunately, after several updates, the matrix may become non-positive [4]. Different solutions have been proposed; here, we look at one such solution, the Square Root Unscented Transformation method [14]. In this method, the square root of the covariance matrix is propagated during the updating sequence; this requires a number of other changes in the updating algorithm. Complexity is also reduced from $O(N^3)$ to $O(N^2)$ for N Sigma points.

3 The New Algorithm: SRUPF-GM

The Unscented Particle Filter (UPF) as described in [7,5] is an algorithm for feature based maps. Here we combine elements of the UPF with aspects of the GMapping Particle Filtering algorithm [2] to give an improved algorithm, SRUPF-GM (see Algorithm 2), for SLAM with grid based maps.

3.1 Updating Robot's State Using Odometry Information

All SLAM filters add noise when updating the state using the robot's odometry information. While investigating the most recent GMapping implementation [3], we found differences from the method reported in [2]. For example, rather than updating using a KF method, they approximate it, generating the noise around four independent pre-defined zero-mean Gaussians. While fast, this does not

Algorithm 2. SRUPF-GM Update Step Algorithm

Input: previous state set $S =< x_{[t-1,0]}, ..., x_{[t-1,n]} >$, sensor z_t and odometry u_t data at time t;
{Cycle through all particles}
for all x_i in S **do**
  {Sigma Point Generation}
  $x_{aug} = [x_i \ 0 \ 0]$
  $cov_{aug} = [cov_i$ Cholesky(Q)$]$
  SP $= [x_{aug} \quad x_{aug} + \gamma(cov_{aug})_i \quad x_{aug} - \gamma(cov_{aug})_{i-n}]$ for i=0 to 2n
  {Odometry Update Step, function f}
  for all x_j in SP **do**
    $< v, \delta > = u_t$
    $V = v + x_j[3]$
    $G = \delta + x_j[4]$
    $x_j = \begin{pmatrix} X_{x_j} + V \times cos(G + \theta_{x_j}) \\ Y_{x_j} + V \times sin(G + \theta_{x_j}) \\ \theta_{x_j} + G \end{pmatrix}$
    add(ST,x_j)
  end for
  $x_i = \sum_{i=0}^{2n} \omega_c x_j$ for all x_j in ST
  $cov_i = QRDecomposition(< x_j - x_i >)$ for j = 1 to 2n
  $cov_i = CholeskyUpdate(cov_i, x_0 - x_i, w_0)$
  {Measurement Update Step, Map Generation and Weight Update}
  if $measurementoccurs$ **then**
    $< x_i, cov_i >$ = scanmatch(x_i, z_t)
    Generate_Map(x_i,z_t)
    Update_Weight(x_i)
  end if
  x_i = sample_new_pose(x_i, $cov_i' \times cov_i$) {Final Update Step}
end for
if variance of particle weight is above threshold **then**
  S = resample() {Resampling Step}
end if

provide the mean and covariance information we need. Instead, we replace the GMapping odometry update step with a more accurate Square Root Unscented Transformation update step. We augment the mean and the covariance, which are then used to compute the Sigma points (SPs):

$$\mu_{aug} = \begin{bmatrix} \mu_{t-1} \\ 0 \end{bmatrix}, \ \Sigma_{aug} = \begin{bmatrix} \Sigma_{t-1} & 0 \\ 0 & \sqrt{Q} \end{bmatrix} \tag{1}$$

where Q is the added noise matrix and is constant for every update step. SPs sp_i are then calculated using the augmented covariance matrix:

$$sp_0 = \mu_{aug} \tag{2}$$

$$sp_i = sp_0 + (\gamma\Sigma_{aug})_i \qquad i = 1, ..., n \tag{3}$$

$$sp_i = sp_0 - (\gamma\Sigma_{aug})_{i-n} \qquad i = n+1, ..., 2n \tag{4}$$

γ controls how fast uncertainty increases. The SPs are passed through the odometry function f that incorporates the noise and odometry information to generate vectors $\overline{sp}_i$ of size 3 (X, Y, θ).

$$\overline{sp}_i = f(sp_i, u_t) \qquad i = 0, ..., 2n \tag{5}$$

The new mean is calculated from the set of SPs. The covariance is calculated with a QR Decomposition of the weighted deltas between the SPs and the new mean.

$$\mu = \sum_{i=0}^{2n} \omega_g \overline{sp}_i \tag{6}$$

$$\Sigma = QRDecomposition\left[\sqrt{|\omega_c|}(\overline{sp}_i - \mu)\right] \qquad i = 1, ...2n \tag{7}$$

$$\Sigma = CholeskyUpdate\left(\Sigma, \overline{sp}_0 - \mu, \omega_0\right) \tag{8}$$

where ω_g and ω_c are weights on the SPs;[1] our weight system follows [5]. Finally, we add randomness; the particle's new pose is sampled around the Gaussian generated by:

$$x_i \sim \mathcal{N}(\mu, \Sigma^t \Sigma) \tag{9}$$

3.2 Measurement Update

Gmapping's measurement update step uses laser information both to correct the robot's pose and to generate the map. While we keep the map generation unchanged, we changed the measurement update step. Our method uses the usual UT measurement update step but with a different pose correction method.

Uncertainty Decrement. SPs are passed to the measurement update function h, which acts as a lightweight pose corrector and returns the best SP ν.

$$\nu = h(\overline{sp}_i, z_t) \tag{10}$$

$$\Sigma^{\mu,\nu} = \sum_{i=0}^{2n} \sqrt{\omega_c}(\overline{sp}_i - \mu)(\overline{sp}_i - \nu)^t \tag{11}$$

$$\Sigma_\nu = QRDecomposition\left[\sqrt{|\omega_c|}(\overline{sp}_i - \nu)\right] \qquad i = 1, ...2n \tag{12}$$

$$\Sigma_\nu = CholeskyUpdate\left(\Sigma_\nu, \overline{sp}_0 - \nu, \omega_0\right) \tag{13}$$

$\Sigma^{\mu,\nu}$ is the cross covariance between the newly calculated mean and the previously calculated mean (with odometry). The last step is then to calculate the final mean and covariance:

$$K = \Sigma^{\mu,\nu}[\Sigma_\nu \Sigma_\nu^t]^{-1} \tag{14}$$

$$\mu = \nu \tag{15}$$

$$\Sigma = CholeskyUpdate\left(\Sigma, K\Sigma_\nu^t, -1\right) \tag{16}$$

[1] Not to be confused with the particle weights.

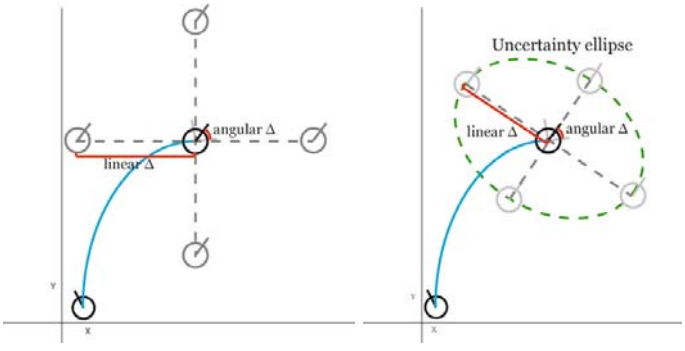

Fig. 1. (a) GMapping pose correction, initial step. Δ is divided by two until a better pose is found. (b) SRUPF-GM pose correction delta uses uncertainty ellipse.

where K is the Kalman Gain. We do not want to perturbate the final mean, as the scan matcher returns the best pose, so μ is just the best Sigma point. The final Σ is decreased with M successive Cholesky updates using all the M columns of the resulting matrix $K\Sigma^t$. No sampling is done after the scan-matching step.

Pose Correction. One reason GMapping works quite well is due to its accurate pose-correction step. The pose corrector generates a number of deterministic samples around a particle's mean and tries to find the best pose within a given δ. Pose correction checks, for every measurement update, if the particle is generating a consistent map. It uses the scan-matching method that incorporates a new scan into particle's existing map to generate a weight. This value is also used to weight the particle. Our square root unscented particle approach works quite well with no accurate pose correction (function h) on simple maps, however more challenging maps do require a pose-correction step. Thus SPs are no longer used to search for the best pose, instead their mean is taken as the starting point.

Our pose correction follows GMapping's with the difference that the δ on which the deterministic samples are generated depends on the covariance cov_t generated during the odometry update. This makes the computation faster than the original GMapping pose-correction, because it searches for the best pose inside the covariance ellipsis, whereas Gmapping version searches over a user pre-defined δ that would eventually include a huge number of improbable poses. Note that (10) in this case is omitted. Figure 1 illustrates the intuition behind our pose-correction approach compared to GMapping pose-correction.

4 Experiments

We tested two versions of the new SRUPF-GM algorithm – with and without pose correction – against the original GMapping implementation. Each algorithm was tested on three different grid-map SLAM problems provided by the Radish

Repository [6] in CARMEN [9] format. Each problem consists of a dataset generated by the sensors of a robot driven around by a human controller, inside a building. The dataset consists of odometry and laser scan readings.

The Radish Repository does not provide a real map neither in an image format nor in an accurate sensor form. Hence it was not possible to compare the algorithms in terms of the quality of the map (for example by generating an error measure). Therefore, we compare the algorithms firstly by whether they achieve the main goal of generating a consistent map. The consistency test assessed whether the overall shape of the map follows the real map, by visually inspecting an image of the results. Secondly, we compared the computation time of the algorithms. Each result reported for the following experiments is the mean of the computational time calculated for 10 runs. To explore the relative computational performance of the algorithms, we also varied two important parameters: (1) the number of particles, and (2) the amount of added noise.

4.1 Experiment 1

The first test was done on a very simple map, generally squared with one single loop and no rooms, as shown in Figure 2(a). For this experiment, we used 30 particles, and the noise parameters were linear 0.1, angular 0.2. (The noise parameters were chosen through preliminary investigations which showed, coincidentally, that these were the best values for all three algorithms.) Table 1 shows the difference in computational time. As one can see, on this simple map, Gmapping is quite fast even using its pose-correction method. SRUPF-GM without pose correction is faster than GMapping, while SRUPF-GM with pose correction is slower, as expected, due to the complexity introduced by the QR and Cholesky computations.

(a) (b) (c)

Fig. 2. (a) NSH Building, (b) Cartesium Building, (c) MIT CSAIL

Table 1. Expt 1: Computation time (30 particles, 10 runs)

Algorithm	Average Computation Time (sec)	Std. Deviation
SRUPF-GM without pose correction	17.3140	0.2340
GMapping	21.7994	0.3635
SRUPF-GM with pose correction	27.2739	0.5679

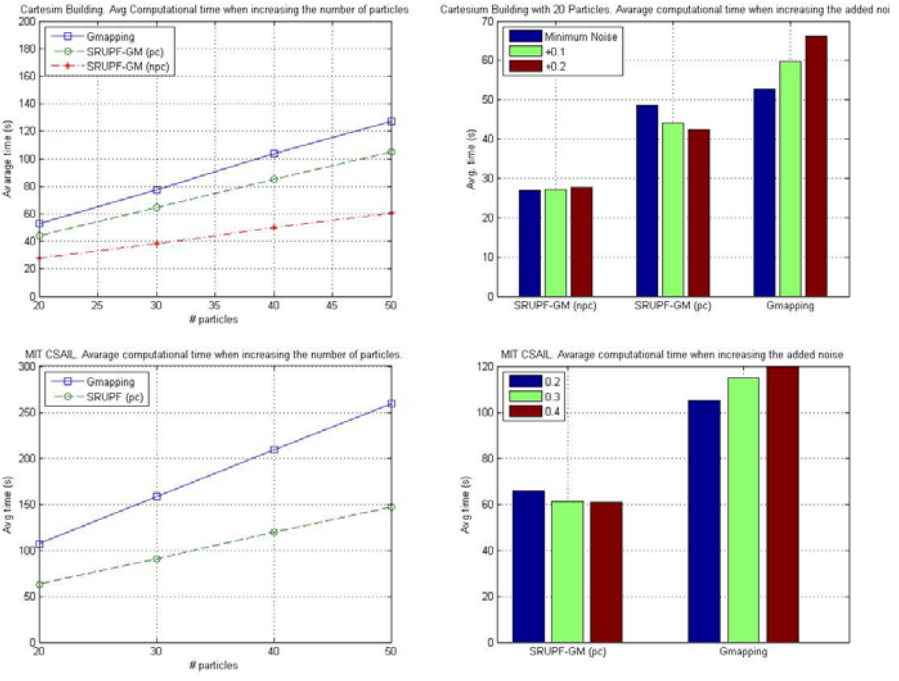

Fig. 3. Expt 2 (above) and Expt 3 (below) comparing SRUPF-GM with and without pose correction (pc/npc) to GMapping, increasing (a) no. of particles (b) added noise

4.2 Experiment 2

In the second experiment the same algorithms were applied to a dataset of medium difficulty, shown in Fig. 2(b). This map increases the difficulty due to the robot's path, which generates a number of loops; closing a loop in the right way is not a simple task. We minimized the error and the number of particles for each algorithm for which it was always successful (generating a consistent accurate map). In this experiment, we varied the noise parameters and the number of particles to explore the resultant changes in computational time.

Figure 3(a)(above) shows that the computation time for all the algorithms increases linearly in the number of particles, however GMapping's results have the largest gradient. Increasing the number of particles is necessary to increase

precision. Figure 3 (b)(above) shows the variation of time when increasing the odometry noise. Note that we had to use different minimum added noise for each algorithm; the added noise is algorithm-specific as it depends on how the odometry incorporates this noise into the model function. As the noise is increased, SRUPF-GM with no pose correction shows no significant difference, the computation time for GMapping increases, while the time for SRUPF-GM with pose correction decreases. The explanation for this is found in the pose correction function: by searching inside the uncertainty ellipse SRUPF-GM avoids checking improbable pose. On the other hand, if the ellipse is too small SRUPF-GM may search in vain, hence SRUPF-GM always requires added noise that is not too low. For all the algorithms, increasing the added noise across these values did not decrease accuracy. Of course if the added noise is too high, all the algorithms may no longer find a solution; this 'too high' noise value depends on both the algorithm and the map.

4.3 Experiment 3

In this last experiment, a relatively complicated map was used, still relatively small in area but with an irregular shape and numerous loops in the path. This dataset was taken in the MIT CSAIL building (see Figure 2(c)). On this dataset SRUPF-GM with no pose correction always failed to generate a consistent map (regardless of the number of particles and the amount of noise). Hence it is clear that pose correction *is* actually needed to generate more difficult maps. This fact of course makes SRUPF-GM without pose correction unusable unless the map complexity is known a priori (a rare case). Results are shown in Fig. 3(below). The difference in the computational time is even more pronounced than for the simpler maps, with the computation time again increasing linearly as the number of particles is increased. And again, SRUPF-GM performs better as the noise increases, while GMapping takes longer.

5 Conclusions and Future Works

In this paper, we have presented an improved particle filtering algorithm for solving SLAM on grid based maps. We used as our starting point the GMapping particle algorithm which has been shown (as we confirmed), to generate very accurate maps even for large scale environments. To improve this algorithm, we took aspects from the square root Unscented particle filtering algorithms, previously only applied to feature based maps. We adapted this as required for grid based mapping, increasing the precision during the odometry update as well as decreasing the computation time required for pose correction. One obvious future step is to obtain suitable test datasets that give the real map in a form that allows accurate error measurements to be computed, which will allow us to compare the quality of the resultant maps more accurately. We expect to be able to improve the computation time of SRUPF-GM's pose correction and further optimize the overall algorithm. We envisage these improvements being based on topological hierarchical methods that should decrease the computation time by focusing the accuracy on smaller submaps.

References

1. Bailey, T., Nieto, J., Guivant, J., Stevens, M., Nebot, E.: Consistency of the EKF-SLAM algorithm. In: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pp. 3562–3568 (2006)
2. Grisetti, G., Stachniss, C., Burgard, W.: Improving grid-based slam with Rao-Blackwellized particle filters by adaptive proposals and selective resampling. In: Proc. of the IEEE Int. Conf. on Robotics and Automation, pp. 2432–2437 (2005)
3. Grisetti, G., Tipaldi, G., Stachniss, C., Burgard, W.: GMapping Algorithm, `http://www.openslam.org/`
4. Higham, N.: Analysis of the Cholesky decomposition of a semi-definite matrix. Reliable Numerical Computation (1990)
5. Holmes, S., Klein, G., Murray, D.: A Square Root Unscented Kalman Filter for visual monoSLAM. In: Proc. of IEEE Int. Conf. on Robotics and Automation, pp. 3710–3716 (2008)
6. Howard, A., Roy, N.: The Robotics Data Set Repository (radish), `http://radish.sourceforge.net/`
7. Kim, C., Sakthivel, R., Chung, W.: Unscented FastSLAM: A robust algorithm for the simultaneous localization and mapping problem. In: Proc. of IEEE Int. Conf. on Robotics and Automation, pp. 2439–2445 (2007)
8. Martinez-Cantin, R., Castellanos, J.: Unscented SLAM for large-scale outdoor environments. In: Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, Citeseer, pp. 328–333 (2005)
9. Montemerlo, M., CARMEN-team: CARMEN: The Carnegie Mellon Robot Navigation Toolkit 2002, `http://carmen.sourceforge.net`
10. Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B.: FastSLAM: A factored solution to the simultaneous localization and mapping problem. In: Proc. of the National Conf. on Artificial Intelligence, pp. 593–598. AAAI Press/MIT Press, Menlo Park, Cambridge (1999/2002)
11. Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B.: FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In: Proc. of Int. Joint Conf. on Artificial Intelligence, vol. 18, pp. 1151–1156 (2003)
12. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics. MIT Press, Cambridge (2005)
13. Van der Merwe, R., Doucet, A., De Freitas, N., Wan, E.: The unscented particle filter. In: Adv. in Neural Information Processing Systems, pp. 584–590 (2001)
14. Van Der Merwe, R., Wan, E.: The square-root unscented Kalman filter for state and parameter-estimation. In: Proc. of IEEE Int. Conf. on Acoustics Speech and Signal Processing, vol. 6, pp. 3461–3464 (2001)
15. Wan, E., Van Der Merwe, R.: The unscented Kalman filter for nonlinear estimation. In: Proc. of the IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium, pp. 153–158 (2000)
16. Zandara, S., Nicholson, A.: Square Root Unscented Particle Filtering for Grid Mapping. Technical report 2009/246, Clayton School of IT, Monash University (2009)