Square Root Unscented Particle Filtering for Grid Mapping

Technical Report 2009/246

Simone Zandara and Ann E. Nicholson
Clayton School of Information Technology
Monash University
Clayton, Victoria
Australia.

email: Ann.Nicholson@infotech.monash.edu.au

No Institute Given

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) [13]. Various filtering methods – Particle Filtering, and derived Kalman Filter methods (Extended, Unscented) – have been applied successfully to SLAM. We present a new algorithm that applies the Square Root Unscented Transformation [14], previously only applied to feature based maps [7], to particle filtering for grid mapping. Experimental results show improved computational performance on more complex grid maps compared to a well-known existing grid based particle filtering algorithm, GMapping [2].

1 Introduction

This paper address 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) [13]. Early path tracking methods such as the Kalman Filter (KF) [13] 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) [13] is a widely used and successful method for modeling the uncertainty of a robot's noisy measurements (e.g. encoders, sonar, laser range finders), however it is unstable and imprecise because of Jacobian calculations [1]; the Unscented KF (UKF) [16, 10] avoids these calculations.

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 [13]. Grid maps generate a representation of the surrounding (usually closed) environment through a grid of cells. In contrast, feature based maps describe a (typically open) environment through a set of observed features, usually sensor readings (e.g. range and bearing).

Unscented Particle Filtering [14] for SLAM [9] has been successfully applied to feature based mapping. It mixes Particle Filtering and UKF by updating particles using an unscented transformation, rather then handling the uncertainty through Jacobians operations. Our research draws from this wide spectrum of KF and particle filtering algorithms; in Section 2 we provide a brief introduction (see [17] for more details). We present a new algorithm (Section 4) which we call SRUPF-GM (Square Root Unscented Particle Filtering for Grid Mapping) to apply Unscented Particle Filtering to *grid* based maps. In Section 5 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 [12] is to estimate the joint posterior 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 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. Specifically we have:

$$p(x_t, m|z_t, u_t) = p(m|x_t, z_t) \prod_{i=1}^{n} p(x_t^i|z_t, u_t) \tag{1}$$

This shows we decompose the problem into two factors, updating the robot's state first (i.e. the second factor), then using that to update the map. In particle filtering this second factor calculation is done by maintaining a set of n poses, S, that make up a region of uncertainty (a Monte Carlo method); these poses are called *particles*.

Each particle has its own position and map; its uncertainty 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's proven that to solve SLAM the noise that is inherent in the odometry and sensor readings must be modeled; all SLAM algorithm add a certain amount of noise to do this. The final set of particles becomes the robot's final pose uncertainty ellipse.

The final characteristic key step of Particle Filtering is *r*esampling, which aims to eliminate those particles which are believed to poorly represent the real. Resampling requires a weighting system. Every particle is generally weighted according to the particle principle:

$$w^{[k]} = \frac{p(m|x_t, z_t)}{\pi(m|x_t, z_t)} \tag{2}$$

where k is the index of the particle and π the real proposal distribution. As π is not known, heuristic methods can be applied to assign weights. Resampling is very important for decreasing the uncertainty of the final distribution calculated over the particle poses. Particles which have weight lower then a certain threshold (which may be varied) are deleted and substituted with existing particles with higher weights. A balance must be found between continuing sampling while the approximation is not close enough to the true, while avoiding continuous resampling, which may discard good particles. It has been proved [5] that the so-called *effective factor* for Montecarlo based filters is a good metric to decide whether to resample or not.

$$N_{eff} = \frac{1}{\sum_{k=1}^{M}(w^{[k]})^2} \tag{3}$$

This value represent the variance over particles' weight. The higher the variance, the worse the final approximation. Typically resampling is done if this factor drops below 50% of the total number of particles. Note that the weights are usually normalized before calculating this value. A brief pseudo-code algorithm presented in Algorithm 1, where

- S is the set of particles of variable size,
- *apply_odom*and *apply_measurement* are non-linear functions that describe odometry and measurement with added noise,
- *generate_mean_covariance* is the EKF update where mean and covariance are updated through Jacobian calculations using odometry and measurement,
- *generate_map* generate particle's internal map,
- *sample_new_pose* samples a new random pose from a the gaussian described by the previously generated mean and covariance, and
- *resample* is done if n_eff drops below a given threshold.

2.2 Unscented Transformation

The EKF update in particle filtering introduces unwanted complexity and error. The Unscented Transformation aims to avoid Jacobian calculations and has been proved to better approximate the true values. Instead of linearizing odometry and measurement functions, the Unscented Transform [16] generates a better

Algorithm 1 Update robot's pose using latest odometry and measurement

```
function UpdateStep(z_t, u_t))
for all x_i in S do
  apply_odom(x_i,u_t)
  apply_measurement(x_i,z_t)
  < mean, cov > = generate_mean_covariance(x_i)
  generate_map(x_i,z_t)
  x_i=sample_new_pose(mean,cov)
  update_weight(x_i)
end for
if n_eff < threshold then
  S = resample();
end if
end function
```

approximated Gaussian that represent the real through a set of so-called *Sigma Points*. It has been used to generate feature based maps using laser or visual sensors [10, 7]. $2n + 1$ Sigma points are generated deterministically around the previous mean. They are perturbed by an additive noise (for every step) and by previous covariance.

$$x_0 = \mu_{aug} \tag{4}$$

$$x_i = \mu_{aug} + (\sqrt{\gamma * \Sigma_{aug}})_i \qquad i = 1, ..., n \tag{5}$$

$$x_i = \mu_{aug} - (\sqrt{\gamma * \Sigma_{aug}})_{i-n} \qquad i = n+1, ..., 2n \tag{6}$$

where μ_{t-1} is the previous mean and Σ_{t-1} is the previous covariance, γ is a scaling parameter and depends on implementation and n is usually calculated using the size of the augmented vector. The index around the parenthesis extracts the ith column from the matrix $\sqrt{\gamma * \Sigma}$. The augmented vector is the usual state vector (x, y, θ) with added zeroes. Unscented Transformation typically adds both measurement and odometry noise.

$$\mu_{aug} = \begin{bmatrix} \mu_{t-1} \\ 0 \\ 0 \end{bmatrix}, \Sigma_{aug} \begin{bmatrix} \Sigma_{t-1} & 0 & 0 \\ 0 & Q & 0 \\ 0 & 0 & R \end{bmatrix} \tag{7}$$

where Σ_{t-1} is the previous state covariance, Q and R are odometry and measurement noise. The resulting matrix would be 7x7 in the usual case where P is a 3x3 matrix and both Q and R are 2x2 matrices, having then $n = 7$, although different implementations have different methods.

Sigma points are passed through the odometry function and transformed into vectors of size 3 (x, y, θ). The augmented vector's information is used to incorporate the noise.

$$\overline{x}_i = f(x_i, u_t) \tag{8}$$

the transformed sigma points are then used to generate the new mean and covariance:

$$\mu = \sum_{i=0}^{2n} \omega_g \overline{x}_i \tag{9}$$

$$\Sigma = \sum_{i=0}^{2n} \omega_c (\overline{x}_i - \mu)(\overline{x}_i - \mu)^t \tag{10}$$

where ω_g and ω_c are weights assigned to the Sigma points and their sum for $2n$ points is equal to 1. μ and Σ represent the belief of the state of the robot after incorporating odometry. The covariance matrix tend to increase if no measurement is taken in consideration. The measurement update is similar. For each k feature ν_k in the map of the particle i, the sigma points previously generated are passed to the measurement update function:

$$\overline{\nu}_{l,k} = h(\overline{x_i}, \nu_k, z_t) \tag{11}$$

If we have some measurement information we update the mean and covariance once again:

$$\nu = \sum_{l=0}^{2n} \omega_g \overline{\nu}_{l,k} \tag{12}$$

$$\Psi = \sum_{l=0}^{2n} \omega_c (\overline{\nu}_{l,k} - \nu)(\overline{\nu}_{l,k} - \nu)^t \tag{13}$$

h is used to test the scan result relative to that position. The final update is then:

$$\Sigma^{\overline{x_i},k} = \sum_{i=0}^{2n} \omega_c (\overline{\nu}_{i,k} - \nu)(\overline{x}_i - \mu)^t \tag{14}$$

$$K_i^k = \Sigma^{\overline{x_i},k} * \Psi^t \tag{15}$$

$$\mu = \mu + K_i^k (z_t - \nu) \tag{16}$$

$$\Sigma = \Sigma - K_i^k \Psi (K_i^k)^t \tag{17}$$

where K is the *Kalman Gain*, which is used to decrease uncertainness while odometry update increases it. Other important aspects include feature initialization and data association, but these are not addressed in this paper. For further information on issues with feature based maps and updates, see [13].

Known Problems The Unscented Transformation is proven to be more accurate then 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 [6]. Different solutions has been proposed; here, we look at one such solution, the Square Root Unscented Transformation method [15]. 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.

2.3 Feature based vs. Grid based maps

Historically, the SLAM problem has been applied to two main type of maps: grid and feature based.

1. Grid maps treat the resulting map as a set of cells in a grid. The size of the grid is generally initialized at the beginning of the computation which may induce problems during real scenarios (rather than simulations). It is used specifically for closed environments, as it is unclear how to extract features in a closed place. For grid based maps the problem is to calculate the occupancy grid using measurement (sonar, laser, video). $p(m|z_t, x_t)$. Odometry is not included in the calculation. Typically every cell is then filled with 1 or 0 (occupied or not), although some methods use a normalized occupancy likelihood for every cell. Grid based maps generally suits 2D worlds. The map size varies enormously depending on the precision (usually from 0.1 meter to 1 meter). There are examples for 3D computation but these require a huge amount of memory and computation.
   It has been shown that SLAM depends critically on pose correction from measurements. Grid maps typically use a scan-matching method to correct the pose. Scan-Matching involves checking a new scan over the old map and measuring the likelihood of incorporating the new scan on the existing map. Different scan matching methods have been proposed; they usually return an heuristic value or a pose correction.
2. Feature based maps treat the resulting map as a set of features. Features are considered as obstacles encountered. This kind of map generally suits huge outdoor environments as extracting feature in a closed environment is tricky (e.g. while scanning a wall, everything can be considered as a feature). SLAM uses pose correction using known features versus new features scanned. That means that for every feature recognized, SLAM has to compute a data-association check and correct the pose. After applying odometry, if measurement occurs, all the features are checked in order to calculate the belief. The covariance matrix representing the pose's uncertainty is calculated over the measurement delta between the new scan and the existing features. After updating the pose, a scan is tested from the new pose and for every obstacle returned by the new scan, all the previous scans have to be checked for possible matching ones. If an observed feature is known to

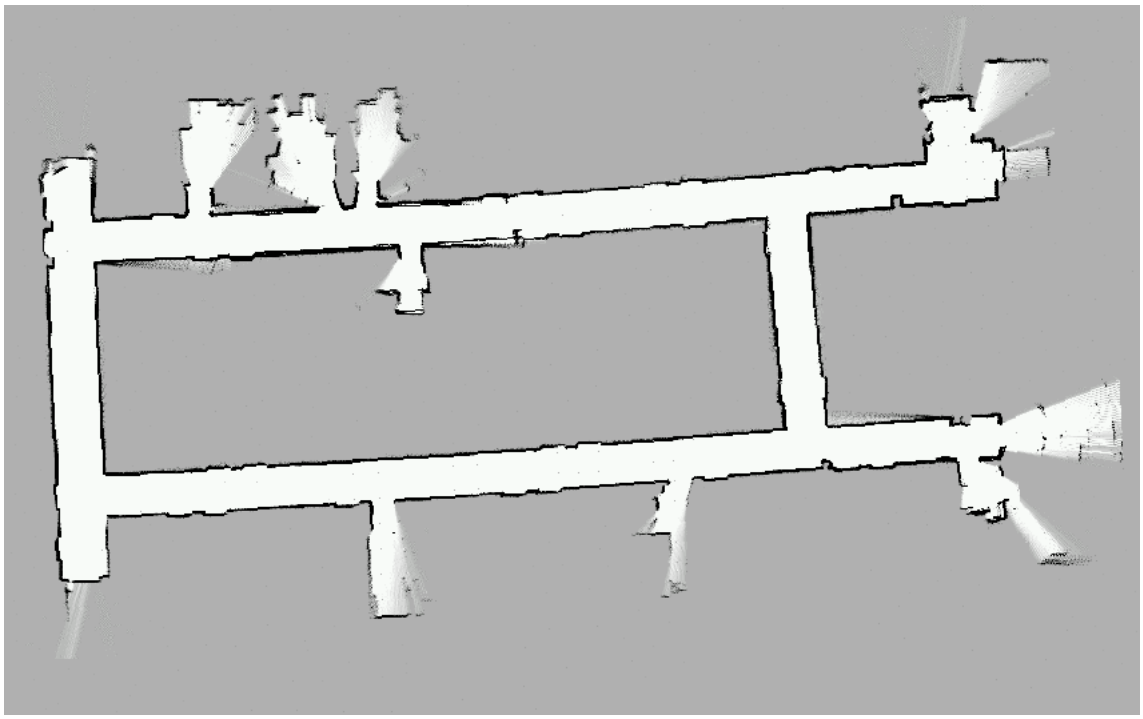

Fig. 1. An example of Grid Based map

already exists on the map, the delta between the scan and the old feature is used to increase/decrease the covariance and the pose.

It is not possible to directly compare the the two methods – grid based and feature based – as different maps are used for different problems.

3 GMapping: a Particle Filter for Grid Based Maps

GMapping is a particle filtering algorithm developed by Grisetti *et al.*[5, 3] to solve SLAM for grid based maps. The algorithm [4] follows the usual particle filtering with two main characteristics.

1. The resulting map is a grid of cells, indicated as occupied or not.
2. The update step is done with a more efficient and fast EKF that avoids calculating Jacobians.

The update odometry step is unchanged. Odometry data with added noise is used to draw particles. To generate the map one would use the laser range finder sensor data; each laser scan returns a set of values representing the point

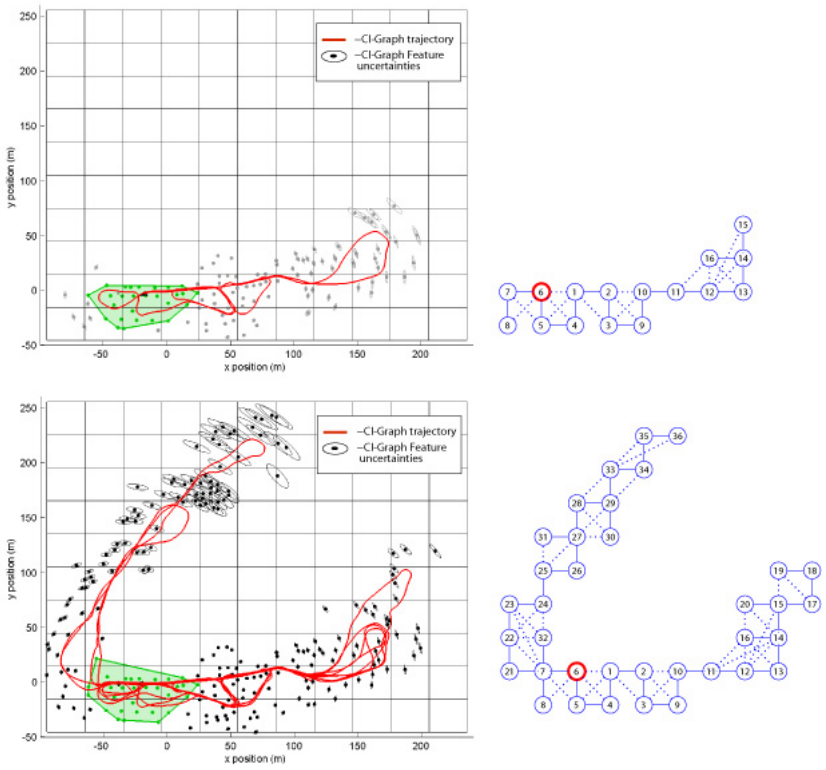

Fig. 2. An example of Feature Based map

at which each laser beam has bounced. This information is noisy and need to be estimated. To generate a map, cells are filled with new laser information (1 = occupied, 0 = unoccupied). The map is generated for every particle in the set.

The next step is to use a scan-matching method to determine the likelihood of the map for that particle including the measurement. Scan-Matching is a process that aims to test a scan from a given position; various Scan-Matching methods exists. GMapping uses a modified Vasco Scan-Matching that compares the new scan with the particle's existing map. A new scan is a set of laser beams that generates a set of occupied points in the space. Those occupied points are compared to a small portion of the existing map surrounding the robot. The more points that match the existing map, the higher the robot's likelihood. This value can be seen as an heuristic weight.

The Gmapping algorithm as implemented in [4] is presented in Algorithm 3; we note that this implementation differs from that presented in [3]. The implemented scan-matching acts as in Algorithm 4. It begins by generating a set of

K samples (with a pre-fixed delta) on the axis. Every sample x_d is passed to the likelihood function that assigns it a weight. This gradient-descent technique is repeated until a pose better then the initial one is discovered out; Figure 3(a) illustrates the GMapping pose correction.

Algorithm 2 Correct particle's pose using measurement information

```
function scanMatch(x_t,readings)
l = -∞
likelihood = likelihood(x_t, readings)
delta = presetDelta
for i = 1 to n_refinements do
  repeat
    delta = delta/2
    for d = 1 to K do
      x_d = deterministic_sample(x_t,delta)
      localL = likelihood(x_d,readings)
      if localL > l then
        l = localL
        bestPose = x_d
      end if
    end for
  until l > likelihood
end for
x_t=bestPose
return l
end function
```

Grisetti *et al.* [3] describe the mean and covariance being then calculated as follows:

$$\mu = \frac{1}{\nu} \sum_{d=0}^{K*n\_refinements} x_d p(z_t|m_{t-1}, x_d) \tag{18}$$

$$\Sigma = \frac{1}{\nu} \sum_{d=0}^{K*n\_refinements} (x_d - \mu)(x_d - \mu)^t p(z_t|m_{t-1}, x_d) \tag{19}$$

with ν being a normalizing factor and $p(z_t|m_{t-1}, x_d)$ being the likelihood of the map including latest measurement and previous map. If no measurements are received, odometry only is used. However, in the actual implementation, the mean and covariance are handled differently. The mean is calculated using the best found pose among the sampled ones using the scan-matching algorithm described above. The covariance is not calculated and the pose is not sampled after scan-matching. This method ensures precision, as it has the advantage of using a tested pose rather then sampling another that could have a worse likelihood and invalidate the map.

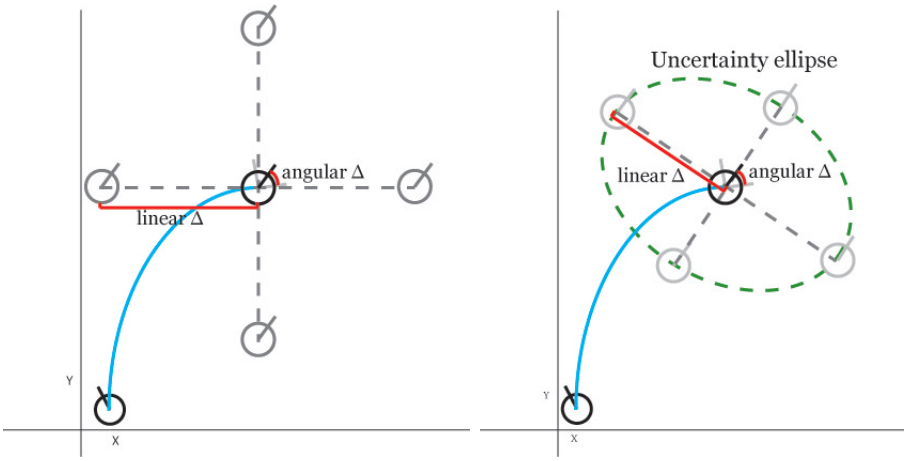

Fig. 3. (a) GMapping pose correction, initial step. Every Δ is divided by two until a better pose is found. (b) SRUPF pose correction delta

To ensure particle spreading, sampling is done during the odometry update over 3 different zero-mean Gaussian with different σ (one for each value: x y and theta). This value is related to the odometry readings and an error factor. Sole odometry without pose-correction would fail map calculation. On the other hand, scan-matching is costly, so it is a good decision to use measurement update only after a certain distance has been traveled. If we consider the odometry as a pair (v,ω) as respectively linear and angular δ we have:

$$\delta_v = v + |v| * sample(\alpha_1) + |\omega| * sample(\alpha_2) \tag{20}$$
$$\delta_\omega = \omega + |v| * sample(\alpha_3) + |\omega| * sample(\alpha_4) \tag{21}$$
$$x_t = x_{t-1} + \delta_v * cos(\theta_t + \delta_\omega/2) \tag{22}$$
$$y_t = x_{t-1} + \delta_v * sin(\theta_t + \delta_\omega/2) \tag{23}$$
$$\theta_t = atan2(sin(\theta_t + \delta_\omega), cos(\theta_t + \delta_\omega)) \tag{24}$$

3.1 Input parameters

The GMapping algorithm has been tested widely on various map types and has been shown to work quite well. The scan-matcher is capable of handling issues related to odometry and measurement noise. By testing various position and guessing the most likelihood position using measurement, the algorithm efficiently models different robots in different situations. The GMapping algorithm uses a number of parameters that can be changed to suit different environments or devices:

– **Added odometry noise**: a 2x2 matrix covariance is used to add noise to the odometry function.

$$\begin{bmatrix} \alpha_1 \ \alpha_2 \\ \alpha_3 \ \alpha_4 \end{bmatrix} \tag{25}$$

The higher these values are, the noisier the odometry is. High noise is sometimes required for high noisy encoders and for complicated and large maps.
– **Window size**: this parameter controls the number of cells around the mean that the scan-matcher checks to determine the likelihood of the new pose over the old map.
– **Number of particles**: this is an important parameter that determines how many particles are used to model the real pose.
– **Minimum Weight**: this parameter controls the scan-match result. When a scan-match is being considered for correcting the pose, the correction must have a map likelihood greater then this threshold to be accepted. Otherwise the scan-match is considered to fail and pure odometry is used to model the real.
– **Scan-matcher iterations N**: this is the main parameter that increases the computation time. Every scan-matching, every particle is tested around its position for K samples over the axis with the usual scan-match algorithm described above. This step is repeated N times. The algorithm often work well with one iteration, but some cases may require more iterations.

The above parameters control the GMapping algorithm behaviour. Still, computation time may be high especially on computers with low memory or slow processors. The scan-matching step is actually the most important feature as it makes use of measurement to correct the pose. Sometimes using high noise is necessary to model big maps, although high noise would make trajectories look definitely unreal and as tested increase computational time. Another issue is the scan-matcher refining process that is done around the axis, basically "guessing" where the particle could be in a very straight way. In this paper, we use the GMapping algorithm as our base algorithm. In the following section, we show how we modified it by substituting the KF update step with an Unscented Transformation and by replacing the pose correction step with a more faster and accurate one.

4 Square Root Unscented Particle Filtering for grid mapping

The Unscented Kalman Filter (UKF) as described in [9, 7] is an algorithm for feature based maps. Here we describe how we adapted the UKF for use with grid based maps. Our base algorithm was the GMapping Particle Filtering algorithm described in the previous section. Our new algorithm involved modifying the GMapping algorithm in four main way:

1. Unstableness of the Unscented Transformation
2. Gmapping approximated pose sampling
3. Unscented Kalman Update without using features
4. Pose-correction

We describe each of these now in turn.

4.1 Unscented Transformation

The numerical unstableness of UKF of the GMapping algorithm is one of its main problems. At certain time in computation, the covariance matrix eventually dropped to non-positiveness and hence the algorithm failed. We solved this problem by implementing a different version of Unscented Particle filtering that propagates the square root of the covariance matrix instead of the covariance matrix itself. Cholesky factorization is then no longer required to calculate the sigma points and this new algorithm does not fail.

4.2 New pose sampling

We reformulated GMapping simple odometry update step (Section 3) with with a more complex Square Root Unscented Transformation. To implement the Square Root Unscented Transformation, several changes to the original Unscented Transformation need to be made:

(1) the state vector (x,y,θ) is augmented with the odometry noise (Q) only and (2) no measurement error (R) is used in the update.

$$\mu_{aug} = \begin{bmatrix} \mu_{t-1} \\ 0 \end{bmatrix}, \ \Sigma_{aug} = \begin{bmatrix} \Sigma_{t-1} & 0 \\ 0 & \sqrt{Q} \end{bmatrix} \tag{26}$$

where $\sqrt{Q}$ is a 2x2 matrix defined as:

$$\begin{bmatrix} |\alpha_1 * v| & 0 \\ 0 & |\alpha_2 * \omega| \end{bmatrix} \tag{27}$$

The regular Unscented Sigma Point creation step is substituted with Square Root unscented step. Note that we approximate $\sqrt{Q}$ with added noise values that are constants. Sigma points are then calculated in the usual way, but with the difference that one does not need to calculate the matrix square root of the augmented covariance matrix:

$$x_0 = \mu_{aug} \tag{28}$$

$$x_i = x_0 + (\gamma\Sigma_{aug})_i \qquad i = 0, ..., n \tag{29}$$

$$x_i = x_0 - (\gamma\Sigma_{aug})_{i-n} \qquad i = n+1, ..., 2n+1 \tag{30}$$

The Sigma points are passed through the odometry function f that incorporates the noise and odometry information to generate vectors $\overline{x}_i$ of size 3 (x, y, θ).

$$\overline{x}_i = f(x_i, u_t) \tag{31}$$

The transformed SPs are then used to generate the new mean and covariance. While the new mean is calculated in the usual way, the covariance is calculated with a QR Decomposition of the weighted deltas between the Sigma points and the new mean:

$$\mu = \sum_{i=0}^{2n} \omega_g \overline{x}_i \tag{32}$$

$$\Sigma = qrdcmp\left[\sqrt{|\omega_c|}(\overline{x}_i - \mu)\right] \qquad i = 1, ...2n+1 \tag{33}$$

$$\Sigma = cholupdate\left(Sigma', \overline{x}_i - \mu, \omega_0\right) \tag{34}$$

where *qrdcmp* means QR Decomposition that returns the transpose of the Cholesky Factorization of the matrix defined as: if S is the result of the QR Decomposition then A = S^t * S = R * R^t = A where R is the result of the Cholesky Factorization on A. Note that QR Decomposition is faster ($O(n^2)$) and more stable then Cholesky factorization($O(n^3)$). *cholupdate* ($O(n^2)$) is the Cholesky update/downdate of the resulting QR Factor; it is used to incorporate the first Sigma point, excluded from the covariance calculation. The weight system, inspired by [7], is as follows:

$$\omega_0 = \frac{1}{n/3} \tag{35}$$

$$\omega_g = \omega_c = \frac{1 - \omega_0}{2n} \tag{36}$$

where n is the augmented vector size: 5. Note that the covariance matrix tends to increase every odometry update. If needed, the γ factor can be changed to reduce or increase the speed at which the uncertainty increases every odometry update step; in our approach we use a gamma factor as suggested in [7]. Finally the new particle's pose is generated sampling around the gaussian generated by:

$$x_{t+1} \sim \mathcal{N}(\mu, \Sigma^t \Sigma) \tag{37}$$

4.3 Measurement Update

The Unscented Transform needs a set of features to compare its Sigma points with and decrease uncertainty; obviously when applying to a grid-based map we do not have such features available. We solved this issue by using the Sigma

points pre-calculated during odometry update. Every Sigma point is passed to the scan-matcher generating a likelihood. The likelihood is used to select the best Sigma point between the existing ones. Our update is then changed to:

$$\nu = h(\overline{x}_i, z_t) \tag{38}$$

$$\Sigma^{\mu,\nu} = \sum_{i=0}^{2n} \sqrt{\omega_c}(\mu - \overline{x}_i)(\nu - \overline{x}_i)^t \tag{39}$$

$$\Sigma = qrdcmp\left[\sqrt{|\omega_c|}(\overline{x}_i - \nu)\right] \qquad i = 1, ...2n+1 \tag{40}$$

$$\Sigma = cholupdate\left(\Sigma, \overline{x}_i - \nu, \omega_0\right) \tag{41}$$

where ν is the best selected Sigma point and $\Sigma^{\nu,\mu}$ is the cross covariance between the newly calculated mean and the previously calculated mean (with odometry). Σ is calculated and the final step is then to calculate the final mean and covariance:

$$K = \Sigma^{\mu,\nu}[\Sigma\Sigma^t]^{-1} \tag{42}$$

$$\mu = \nu \tag{43}$$

$$\Sigma = cholupdate\left(\Sigma, K\Sigma^t, -1\right) \tag{44}$$

where K is the Kalman Gain. We don't want to perturbate the final mean, as the scan matcher returned the best pose so μ is just equal to the best Sigma point. The final Σ is decreased with M successive Cholesky update using all the M columns of the resulting matrix $K\Sigma^t$. No drawing is done after a scan-matching step. The overall algorithm is summarized in Algorithm 5.

4.4 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.

Algorithm 4 describes it in detail.

4.5 Complexity

Square Root Unscented Particle Filter introduces computational complexity compared to the GMapping version used as a base. This is mainly due to the matrix operations. The simple model used by GMapping is substituted with a more

Algorithm 3 SRUPF-GM Update Step Algorithm

Input: sensor z_t and odometry u_t data at time t;
Initialize state (ST) and Sigma point (SP) vectors to $< 0 >$ {Cycle 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 $= [x_{aug} \ x_{aug} + \gamma(cov_{aug})_i \ x_{aug} - \gamma(cov_{aug})_{i-n}]$ for i=0 to 2n
  {Odometry Update Step}
  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}
  if $measurementoccurs$ **then**
    $< x_i, \ cov_i >$ = scanmatch(x_i, z_t)
    Generate_Map(x_i,z_t)
    Update_Weight(x_i)
  end if
end for
if $n_{eff} < threshold$ **then**
  S = resample() {Resampling Step}
end if
x_i = sample_new_pose(x_i, $cov_i' \times cov_i$) {Final Update Step}
end function

accurate mean and covariance calculation. For every particle we introduce $O(m^2)$ computational complexity, with m being the number of Sigma point, given by the QR decomposition and Cholesky Update. The same computation is introduced to the measurement part of the usual QR decomposition and Cholesky Update. We note, however, that an optimized version of those functions can be generated for a small sized matrix. We currently use a fixed small size matrix to calculate mean and covariance; a better algorithm design would decrease the amount of memory and computational time. Nonetheless, as we will see in the following experimental section, testing shows that our approach is faster than the original GMapping method.

Algorithm 4 Correct particle's pose using measurement information

```
function scanMatch(x_t,cov_t,SP,readings)
l = −∞
likelihood = likelihood(x_t, readings)

if NOT (pose_correction ) then

  for all x_d in SP do
    localL = likelihood(x_d,readings)
    if localL > l then
      l = localL
      bestPose = x_d
    end if
  end for


else

  delta = ellipse(cov_t)
  for i = 1 to n_refinements do
    repeat
      delta = delta/2
      for d = 1 to K do
        x_d = deterministic_sample(bestPose,delta)
        localL = likelihood(x_d,readings)
        if localL > l then
          l = localL
          bestPose = x_d
        end if
      end for
    until l > likelihood
  end for

end if

x_t=bestPose
return l
end function
```

5 Experiments

We testing two version of the new SRUPF algorithm – with and without pose correction – against the original GMapping implementation. We now describe our experiments comparing these three algorithms:

i GMapping
ii SRUPF with no pose correction
iii SRUPF with pose correction

5.1 Methodology

Each algorithm was tested on three different grid-map SLAM problems provided by the Radish Repository [8] in CARMEN [11] 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 [8] does not provide a real map neither in a accurate sensor reading format nor in an image format. Hence it was not possible to compare algorithms performance in terms of the quality of the map (for example by generating an error measure). Therefore, we compare the algorithms firstly in 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 three 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 the two important parameters: (1) the number of particles, and (2) the amount of added noise.

5.2 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 5. 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 5.2 shows the difference in computational time (plotted in Figure 4.

Table 1. computational average time with 30 particles for NSH Building

	Avg. Time (sec)	Std. Deviation
(i)	21.7994	0.3635
(ii)	27.2739	0.5679
(iii)	17.3140	0.2340

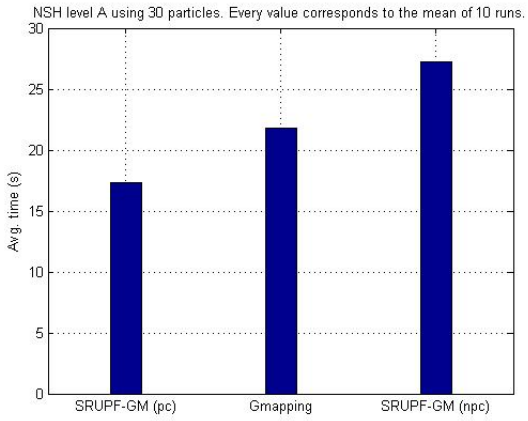

Fig. 4. NSH Building with 30 particles

As one can see, on this simple map, Gmapping is quite fast even using its pose-correction method. SRUPF without pose correction is faster than GMapping, while SRUPF with pose correction is slower, as expected, due to the complexity introduced by the QR and Cholesky computations. For this simple map, SRUPF without pose correction is enough to generate a consistent map with no detectable differences from that generated by GMapping. Simple maps do not require pose correction as they can easy be achieved using correction upon the sigma points. Though in real situation one wouldn't know weather the environment you are going to scan is going to be hard or simple

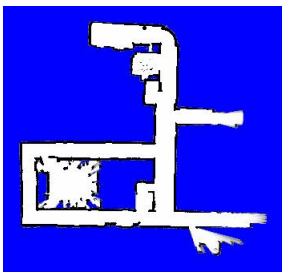

Fig. 5. NSH Building

5.3 Experiment 2

In the second experiment the same algorithms were applied to a medium difficult situation. This dataset comes from a scanning of the Cartesium Building, Univ. of Bremen, depicted in Figure 6.

Fig. 6. Cartesium Building, University of Bremen. Though robot's path is not show in this picture, it generates numerous loop around the building. Loop closure is one of the main issues in SLAM because it requires previous path to be quite accurate.

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 particle number to explore the resultant changes in computational time. In Figure 7 we present the variation of time when increasing the number of particles.

Each version shows a linear increment of time as particles increase. Though, GMapping present the highest inclination. Increasing the number of particles is necessary to increase precision. Some maps require a consistent number of particles to be modeled. Figure 8 shows the variation of time when increasing the odometry noise. We used different minimum noise to model the same map. Note that the added noise is implementation-dependent and is related to how the odometry incorporates this noise into the model function.

First, we can see that in this experiment, across all variations of particles and noise, SRUPF always sproduces a consistent map faster than GMapping, even with pose correction (though of course the no pose correction version is faster). Further, as the number of particles is increased, the computational time of all algorithms looks to be increasing linearly although with different gradients.

As the noise is increased, SRUPF with no pose correction shows no significant difference, that is, it is unaffected as expected. GMapping and SRUPF with pose correction behave differently. GMapping increases its computational time when increasing the added noise, while SRUPF decreases it. The explanation for this relies on the pose correction function. Gmapping use a fixed delta to search the

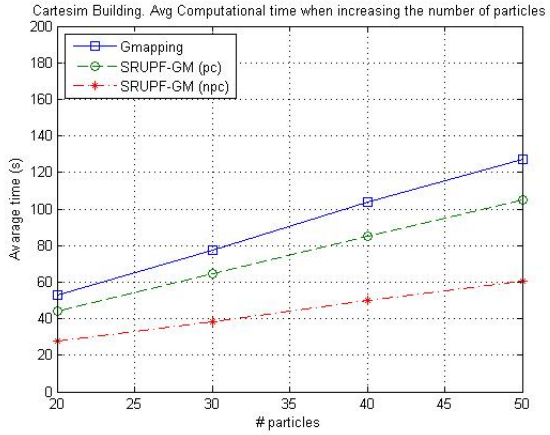

Fig. 7. Cartesium Building: increasing particles number

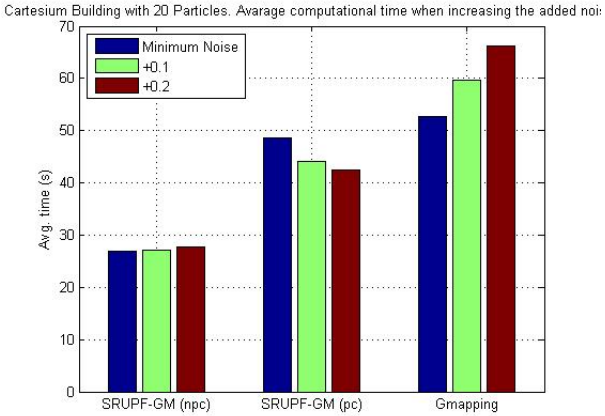

Fig. 8. Cartesium Building: increasing added noise

best pose after a scan. That delta depends on the initial input and does not vary with time. If noise is higher, odometry model is more likely to generate inaccurate particles and pose correction step increase its number of cycles to get a better pose. SRUPF instead generates more accurate particles and its pose correction step delta varies with covariance. When a particle has a high covariance, its uncertainness makes the pose corrector search further away from its mean. If the odometry added noise is low, SRUPF works more to search the correct pose. Low odometry noise also means low particle's uncertainness, so that the pose-corrector searches in vain within a too small surrounding area, most of time

falling back to the original mean. This suggests a differently scaled delta would be a good idea to even the search time and improve the pose corrector.

5.4 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 9).

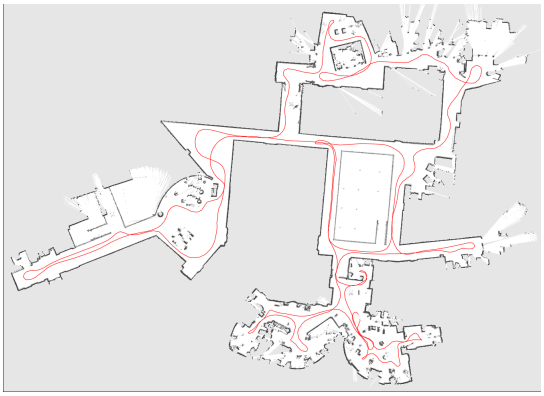

Fig. 9. MIT CSAIL with robot's path

In this particular dataset SRUPF 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 without pose correction unusable unless the map complexity is known a priori (a rare case).

The difference in the computational time (Figures 10 and 11) are even more pronounced than for the simpler maps, although the same linear increment observed previously is again seen when as the the number of particles is increased. And again, SRUPF performs better as the noise increases, while GMapping takes longer.

6 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 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,

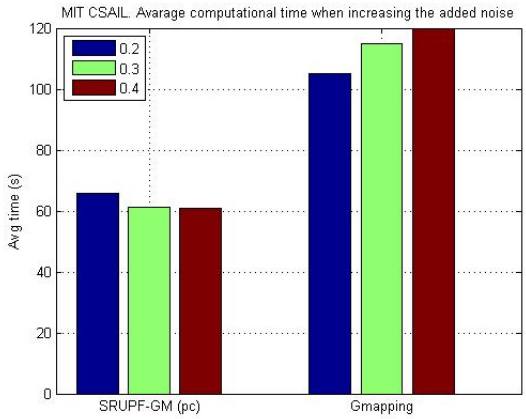

Fig. 10. MIT CSAIL Building: increasing added noise

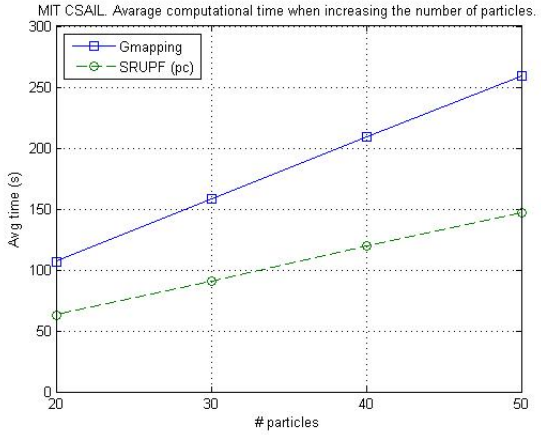

Fig. 11. MIT CSAIL Building: increasing particle number

previously only applied to feature based maps. We adapted this as required for grid-based mapping, increases the precision during the odometry update as well as decreasing the computation time required for pose correction. We have presented results from computational experiments showing that while the full version of SRUPF is slower than GMapping on smaller maps, is performs better on more complex maps, while generating a consistent map that is as accurate (as far as we can determine from visual image inspection).

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. Still much work can be done to decrease the computation time related to SRUPF's pose correction, while further optimisations of the algorithm should may improve its performance also. We envisage these future improvement being based on topological hiererchical methods that should decrease the computation time by focusing the accuracy on smaller submaps.

References

1. Bailey, T., Nieto, J., Guivant, J., Stevens, M., and Nebot, E. Consistency of the EKF-SLAM algorithm. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems* (2006), pp. 3562–3568.
2. Grisetti, G., Stachniss, C., and 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* (2005), pp. 2432–2437.
3. Grisetti, G., Stachniss, C., and Burgard, W. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics 23*, 1 (2007), 34–46.
4. Grisetti, G., Tipaldi, G., Stachniss, C., and Burgard, W. GMapping Algorithm (Online). *http://www.openslam.org/*.
5. Grisetti, G., Tipaldi, G., Stachniss, C., Burgard, W., and Nardi, D. Fast and accurate SLAM with Rao–Blackwellized particle filters. *Robotics and Autonomous Systems 55*, 1 (2007), 30–38.
6. Higham, N. Analysis of the Cholesky decomposition of a semi-definite matrix. *in Reliable Numerical Computation* (1990).
7. Holmes, S., Klein, G., and Murray, D. A Square Root Unscented Kalman Filter for visual monoSLAM. In *Proc. of IEEE Int. Conf. on Robotics and Automation* (2008), pp. 3710–3716.
8. Howard, A., and Roy, N. The Robotics Data Set Repository (radish) [Online]. *http://radish.sourceforge.net/*.
9. Kim, C., Sakthivel, R., and Chung, W. Unscented FastSLAM: A robust algorithm for the simultaneous localization and mapping problem. In *Proc. of IEEE Int. Conf. on Robotics and Automation* (2007), pp. 2439–2445.
10. Martinez-Cantin, R., and Castellanos, J. Unscented SLAM for large-scale outdoor environments. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems* (2005), Citeseer, pp. 328–333.
11. Montemerlo, M., and CARMEN-team. CARMEN: The Carnegie Mellon Robot Navigation Toolkit 2002 [Online]. *http://carmen.sourceforge.net*.
12. Montemerlo, M., Thrun, S., Koller, D., and 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* (2003), vol. 18, pp. 1151–1156.
13. Thrun, S., Burgard, W., and Fox, D. *Probabilistic Robotics*. MIT Press, 2005.
14. Van der Merwe, R., Doucet, A., De Freitas, N., and Wan, E. The unscented particle filter. *Adv. in Neural Information Processing Systems* (2001), 584–590.
15. Van Der Merwe, R., and 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* (2001), vol. 6, pp. 3461–3464.

16. Wan, E., and 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* (2000), pp. 153–158.
17. Zandara, S., and Nicholson, A. Square Root Unscented Particle Filtering for Grid Mapping. Technical report 2009/246, Clayton School of IT, Monash University, 2009.