Corrective Gradient Refinement for Mobile Robot Localization

Joydeep Biswas
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
joydeepb@ri.cmu.edu

Brian Coltin
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
bcoltin@andrew.cmu.edu

Manuela Veloso
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
mmv@cs.cmu.edu

***Abstract*— Particle filters for mobile robot localization must balance computational requirements and accuracy of localization. Increasing the number of particles in a particle filter improves accuracy, but also increases the computational requirements. Hence, we investigate a different paradigm to better utilize particles than to increase their numbers. To this end, we introduce the Corrective Gradient Refinement (CGR) algorithm that uses the state space gradients of the observation model to improve accuracy while maintaining low computational requirements. We develop an observation model for mobile robot localization using point cloud sensors (LIDAR and depth cameras) with vector maps. This observation model is then used to analytically compute the state space gradients necessary for CGR. We show experimentally that the resulting complete localization algorithm is more accurate than the Sampling/Importance Resampling Monte Carlo Localization algorithm, while requiring fewer particles.**

I. Introduction

Particle filters have proven to be successful at state estimation and tracking for nonparametric stochastic processes. However, for every particle filter implementation, the mantra to reach higher accuracy has traditionally been "add more particles." This approach does work, but the required number of particles scales exponentially with the number of dimensions. Therefore, significant work has gone into developing algorithms that work with fewer particles but *do more* with the same particles, such that the desired accuracy is still met.

In this paper we introduce one such novel algorithm, Corrective Gradient Refinement (CGR), which reduces the required number of particles for a particle filter by using the *gradients* of the observation model. This results in more efficient sampling of the robot's localization belief by sampling less along directions of low uncertainty given an observation, while densely sampling along directions of high uncertainty. For example, if the LIDAR scans of the robot detect parallel walls, there will be fewer samples along the direction perpendicular to the walls (the direction of low uncertainty), and more samples parallel to the walls (the direction of high uncertainty).

We apply Corrective Gradient Refinement to the task of indoor mobile robot localization with point cloud sensors. By point cloud sensors, we refer to sensors that generate 2D or 3D points corresponding to the surfaces of detected obstacles. We use both a 2D LIDAR scanner and a 3D Microsoft Kinect sensor to generate point clouds. To compare our algorithm with existing approaches, we first review the concept of particle filtering in general, and then discuss existing algorithmic improvements and extensions. Next, we describe the Corrective Gradient Refinement algorithm in detail and experimentally demonstrate its effectiveness.

II. Background And Related Work

Bayesian filters are used to track the probability distribution of a d-dimensional state variable $x \in \mathbb{R}^d$ over time given the history of observations $y_{1:t}$ and control inputs $u_{1:t-1}$. The resulting probability distribution of x is called the *belief*, $Bel(x_t) = p(x_t|y_{1:t}, u_{1:t-1})$. The belief is recursively updated using the equation:

$$Bel(x_t) = \eta p(y_t|x_t) \int p(x_t|x_{t-1}, u_{t-1}) Bel(x_{t-1}) dx_{t-1} \tag{1}$$

The true belief (the probability distribution given all the observations) is often called the posterior distribution. To simplify the terminology, in the rest of the paper, we will refer to the true belief as the "posterior," and the estimated belief simply as the "belief."

Particle filters, and in particular, Monte Carlo Localization (MCL)[1] represent the belief by a set of weighted samples, or "particles," $Bel(x_t) = \{x_t^i, w_t^i\}_{i=1:m}$, where x_t^i is the state of the particle i at time t and w_t^i its associated weight. Recursive updates of the particle filter can be implemented via a number of methods, notably among which is the sampling/importance resampling method (SIR) [2]

In the SIR algorithm for MCL, m samples x_{t-}^i are drawn with replacement from the prior belief $Bel(x_{t-1})$ proportional to their weights w_{t-1}^i. These samples x_{t-}^i are used as priors to sample from the motion model of the robot $p(x_t|x_{t-1}, u_{t-1})$ to generate a new set of samples x_t^i that approximates $p(x_t|x_{t-1}, u_{t-1})Bel(x_{t-1})$. Importance weights for each sample x_t^i are then computed as

$$w_t^i = \frac{p(y_t|x_t^i)}{\sum_i p(y_t|x_t^i)}. \tag{2}$$

Although SIR is largely successful, several issues may still create problems:

1) Observations could be "highly peaked", *i.e.* $p(y_t|x_t^i)$ could have very large values for a very small subspace $\epsilon \subseteq \mathbb{R}^d$ and be negligible everywhere else. In this case, the probability of samples x^i overlapping with the subspace ϵ is diminishingly small.

2) If the posterior is no longer being approximated well by the samples x^i, recovery is only possible by chance, if the motion model happens to generate new samples that better overlap with the posterior.
3) The required number of samples necessary to overcome the aforementioned limitations scales exponentially in the dimension of the state space, d.

These problems are well known, and a number of algorithmic improvements address them successfully with several assumptions.

KLD-Sampling [3] adapts the number of samples based on the Kullback-Liebler Distance between the belief and the posterior, thus reducing the computational requirements when the belief accurately models the posterior, and increasing the number of particles when the posterior distribution is likely to spread out. However, this approach does not refine the samples based on observations, but only varies the number of samples.

Sensor resetting [4] takes into account mismatch between observations and the belief by drawing samples from the observation model, $p(y_t|x_t)$ to incorporate into the belief. However, this assumes that $p(y_t|x_t)$ can be sampled efficiently over the map, which may not be possible (*e.g.* for observation models of LIDAR scans on a map).

Monte Carlo Markov Chain (MCMC) [5], [6] based approaches, and in particular, the Hybrid Monte Carlo (HMC) filter [7], [8] have been shown to be successful at maintaining the belief with small sample sets, even in large dimensional state spaces. In the HMC filter, the posterior $p(x_t|y_{1:t}, u_{1:t-1})$ is modeled as a hypothetical potential energy term $E = -\log(p(x_t|y_{1:t}, u_{1:t-1}))$, and for each sample x_t^i, a Markov Chain (MC) is evolved using Hamiltonian dynamics over the Hamiltonian $H = E + K$ where K is the potential energy of the sample, $K = \frac{1}{2}\dot{x}^T\dot{x}$. The disadvantages of this method are that it is slow (the MCMCs requires a minimum number of steps of evolution before they reach equilibrium), and it assumes that the state space gradient of the full posterior, $\frac{\delta}{\delta x}p(x_t|y_{1:t}, u_{1:t-1})$ can be computed, which is often impossible or computationally infeasible.

The Auxiliary Particle Filter (APF) [9] improves the proposal distribution by "simulating" the evolution of the samples x_t^i using the observation y_t to generate a new sampling x_{t-}^i, which is then used to generate samples x_t^i using the motion model. Since the simulation step is performed prior to applying the motion model, the simulation model must account for the motion model as well.

In FastSLAM 2.0 [10], the proposal distribution is refined by linearizing the observation model with respect to pose and landmark locations, and then performing a single EKF-like update step. The GMapping [11] algorithm refines its proposal distribution by sampling the observation likelihood around every particle, and fitting a Gaussian to the sampled points. The proposal distribution is then updated using this Gaussian estimate.

The Corrective Gradient Refinement (CGR) algorithm, which we introduce, addresses the aforementioned problems in a computationally efficient manner. It refines samples locally to better cover neighboring regions with high observation likelihood, even if the prior samples had poor coverage of such regions. Since the refinement takes into account the gradient estimates of the observation model, it performs well even with highly peaked observations. This permits CGR to track the location of the robot with fewer particles than MCL-SIR, although for the task of global localization, the number of particles would still have to be scaled up as in the case of MCL-SIR to cover the entire map. The approach of KLD-Sampling, if used along with CGR, is likely to provide the same dynamic reduction of particles as in the case of MCL-SIR once the localization belief collapses to a unimodal distribution. Unlike the HMC and APF algorithms, CGR does not require computation of the transition function of the full posterior (such as the simulation model in the APF) or the gradients of the motion model (such as the evolution of Hamiltonian dynamics in HMC). We formulate the gradient estimates of the observation model in a manner that allows analytical computation of the gradient using a vector map, unlike the approaches of FastSLAM 2.0 and GMapping. Similar approaches to improving the proposal distribution using gradients have been applied to vision-based tracking (*e.g.*[12]). We show experimentally that CGR has higher accuracy and lower variance across trials than MCL-SIR. Furthermore, for the same level of desired accuracy, CGR requires far fewer particles than MCL-SIR for localization tracking.

III. Corrective Gradient Refinement

The CGR algorithm iteratively updates the past belief $Bel(x_{t-1})$ using observation y_t and control input u_{t-1} to estimate the latest belief $Bel(x_t)$ as follows:

1) Samples of the belief $Bel(x_{t-1})$ are evolved through the motion model, $p(x_t|x_{t-1}, u_{t-1})$ to generate a first stage proposal distribution q^0.
2) Samples of q^0 are "refined" in r iterations (which produce intermediate distributions $q^i, i \in [1, r-i]$) using the gradients $\frac{\delta}{\delta x}p(y_t|x)$ of the observation model $p(y_t|x)$.
3) Samples of the last generation proposal distribution q^r and the first stage proposal distribution q^0 are sampled using an acceptance test to generate the final proposal distribution q.
4) Samples x_t^i of the final proposal distribution q are weighted by corresponding importance weights w_t^i, and resampled with replacement to generate $Bel(x_t)$.

We explain the four steps of the CGR algorithm in detail.

A. The Predict Step

Let the samples of the past belief, $Bel(x_{t-1})$ be given by x_{t-1}^i. These samples are then evolved using the motion model of the robot to generate a new set of samples $q^0 = \left\{x_{q^0}^i\right\}_{i=1:m}$ as $x_{q^0}^i \sim p(x_t|x_{t-1}^i, u_{t-1})$. This sample set q^0 is called the first stage proposal distribution, and takes time complexity $\mathcal{O}(m)$ to compute.

B. The Refine Step

The Refine step is central to the CGR algorithm. It corrects sample estimates that contradict the observations (*e.g.* when the LIDAR scan shows the robot to be in the center of the corridor, but the sample is closer to the left wall). This results in the CGR algorithm sampling less along directions that have low uncertainty in the observation model while maintaining samples along directions of high uncertainty.

For the CGR algorithm, estimates of the first order differentials (the gradients) of the observation model, $\frac{\hat{\delta}}{\delta x} p(y_t|x)$ must to be computable. Given these gradients, the Refine step performs gradient descent for r iterations, generating at iteration i the i-th stage proposal distribution. Algorithm 1 outlines the Refine step.

Algorithm 1 The Refine step of CGR

```
1: Let $q^0 = \left\{ x^j_{q^0} \right\}_{j=1:m}$
2: for $i = 1$ to $r$ do
3:     $q^i \leftarrow \{\}$
4:     for $j = 1$ to $m$ do
5:         $x^j_{q^i} \leftarrow x^j_{q^{i-1}} + \eta \left[ \frac{\hat{\delta}}{\delta x} p(y_t|x) \right]_{x = x^j_{q^{i-1}}}$
6:         $q^i \leftarrow q^i \cup x^j_{q^i}$
7:     end for
8: end for
```

The computation of $\frac{\hat{\delta}}{\delta x} p(y_t|x)$ for localization using point cloud sensors is described in detail in Section IV-B. Performing multiple iterations of gradient descent allows the estimates of the gradients of the observation model to be refined between iterations, which results in higher accuracy. The computation time required for the Refine step scales as $\mathcal{O}(rm)$. In general, smaller values of the step size η, paired with larger values of r result in higher accuracy.

The Refine step performs rm total iterations, each of which requires the computation of the gradient of the observation model. Therefore, the time complexity of the refine step is $\mathcal{O}(rmf(\ldots))$ where f is a function that depends on the observation model.

After the Refine step, samples from the r-th stage distribution are compared to the samples from the first stage proposal distribution by an acceptance test to generate the final proposal distribution q, as we now present

C. The Acceptance Test Step

To generate the final distribution q, Samples $x^i_{q^r}$ from the r-th stage distribution are probabilistically chosen over the corresponding samples $x^i_{q^0}$ of the first stage distribution proportional to the value of the acceptance ratio r^i, as:

$$r^i = \min \left\{ 1, \frac{p(y_t|x^i_{q^r})}{p(y_t|x^i_{q^0})} \right\} \tag{3}$$

This Acceptance Test allows the algorithm to probabilistically choose samples that better match the observation model $p(y_t|x)$. If the Refine step does not produce samples with higher observation likelihood than the samples in the first stage distribution, the final proposal distribution q will have a mixture of samples from q^0 and q^r. Furthermore, if the Refine step results in most samples having higher observation likelihood than the samples in the first stage distribution, the final proposal distribution q will consist almost entirely of samples from q^r.

Samples from the final proposal distribution q thus generated are weighted by importance weights, and resampled to compute the latest belief $Bel(x_t)$ in the Update step. The acceptance test step has time complexity $\mathcal{O}(m)$ since it reuses the cached values of the observation likelihood computed on the first and last iterations of the Refine step.

D. The Update Step

The importance weights for the CGR algorithm are different from those of the MCL-SIR algorithm, since in the CGR algorithm, the proposal distribution q is not the same as the samples of the motion model. To derive the expression for the importance weights of the CGR algorithm, we first factor out the belief update (Eq. 1) as:

$$Bel(x_t) \propto p(y_t|x_t) p(x_t|u_{t-1}, Bel(x_{t-1})) \tag{4}$$

$$p(x_t|u_{t-1}, Bel(x_{t-1})) = \int p(x_t|x_{t-1}, u_{t-1}) Bel(x_{t-1}) dx_{t-1} \tag{5}$$

The proposal distribution from which the belief $Bel(x_t)$ is computed, is q. Hence, the (non-normalized) importance weights w^i_t for samples x^i_t in q are given by:

$$w^i_t = \frac{p(y_t|x^i_t) p(x^i_t|u_{t-1}, Bel(x_{t-1}))}{q(x^i_t)} \tag{6}$$

In this expression, $p(y_t|x^i_t)$ is computed using the observation model (see Section IV-B), and the terms $p(x^i_t|u_{t-1}, Bel(x_{t-1}))$ and $q(x^i_t)$ are computed by kernel density estimates at the locations x^i_t using the samples from q^0 and q for support of the kernel density respectively. Since the importance weight accounts for the motion model (from the term $p(x^i_t|u_{t-1}, Bel(x_{t-1})))$ as well as the refined proposal distribution $(q(x^i_t))$, the CGR algorithm avoids "peak following" behavior which contradict the motion model.

The samples in q are resampled in proportion to their importance weights using low variance resampling [13] to obtain the samples of the latest belief, $Bel(x_t)$. The update step has time complexity $\mathcal{O}(m^2)$. It is quadratic in the number of particles since it requires computation of kernel density functions.

Thus, given the motion model $p(x_t|x_{t-1}, u_{t-1})$, the observation model $p(y_t|x)$, the gradients of the observation model $\frac{\delta}{\delta x} p(y_t|x)$ and the past belief $Bel(x_{t-1})$, the CGR algorithm computes the latest belief $Bel(x_t)$. Adding the runtime of all the steps together, the CGR algorithm is performed in $\mathcal{O}\left(rmf(\ldots) + m^2\right)$ time. The formulation of the observation model and its gradients is the subject of the next Section.

IV. Vector Localization Observation Models

We are interested in localizing an indoor mobile robot using the CGR algorithm applied to point cloud sensors and a vector map.

A. Vector Map Representation

The map M used by our localization algorithm is a set of s line segments l_i corresponding to all the walls in the environment: $M = \{l_i\}_{i=1:s}$. Such a representation may be acquired by mapping (*e.g.* [14]) or (as in our case) taken from the blueprints of the building. The reason we use a vector map for localization is three-fold:

1) A vector map allows higher precision than an occupancy grid map, in a more compact representation.
2) Ray casts can be performed analytically on a vector map, and thus are much faster than ray casts on an occupancy grid map.
3) The gradients of the observation model (which are required for CGR) can be computed analytically on a vector map.

Using the vector map, we next develop an observation model for 2D point cloud sensors.

B. 2D Point Cloud Observation Model

The 2D point cloud observation model is used for sensors like a planar LIDAR scanner, which generate point clouds in 2D space.

Let the 2D point cloud generated by the sensor (*e.g.* 2D LIDAR scanner) be denoted by the set of 2D points $P = \{p_i\}_{i=1:n}$. Without loss of generality, we assume that the origin of the sensor coincides with the origin of the robot. Let the observation be $y = P$, and the pose of the robot x be given by $x = \{x_l, x_\theta\}$ where x_l is the 2D location of the robot, and x_θ its orientation angle. The observation likelihood $p(y|x)$ is then computed as follows:

1) For every point p_i in P, line l_i ($l_i \in M$) is found by ray casting such that the ray in the direction of $p_i - x_l$ and originating from x_l intersects l_i before any other line.
2) The perpendicular distance d_i of p_i from the (extended) line l_i is computed.
3) The total (non-normalized) observation likelihood $p(y|x)$ is then given by:

$$p(y|x) = \prod_{i=1}^{n} \exp\left[-\frac{d_i^2}{2f\sigma^2}\right] \tag{7}$$

Here, σ is the standard deviation of the distance measurements of a single ray, and f (where $f > 1$) is a discounting factor to discount for the correlation between rays. The gradient of the observation model is therefore given by:

$$\frac{\delta}{\delta x} p(y|x) = -\frac{p(y|x)}{f\sigma^2} \sum_{i=1}^{n} \left[d_i \frac{\delta d_i}{\delta x}\right] \tag{8}$$

The term $\frac{\delta d_i}{\delta x}$ in this equation has two terms, corresponding to the translation component $\frac{\delta d_i}{\delta x_l}$ and the rotational component $\frac{\delta d_i}{\delta x_\theta}$. These terms are computed by rigid body translation and rotation of the point cloud P respectively. Due to errors in the pose of the robot, the point to line correspondences may be incorrect for some points, which would lead to errors in the gradient estimate. Hence, some estimated outlier rejection is necessary before evaluating Eq. 8 for all the points. Outlier rejection is performed by two checks:

1) If the distance d_i is greater than a threshold d_{max}, the corresponding point p_i is rejected.
2) If the angle between the line joining successive points p_i, p_{i+1} and either of the corresponding lines l_i, l_{i+1} is greater than a threshold θ_{max}, then the points p_i, p_{i+1} are rejected.

Performing the ray cast operation from the estimated pose takes time $\mathcal{O}(s+n)$, and the computation of the observation likelihood and gradients is $\mathcal{O}(n)$. So the total complexity for the observation model is $f(s,n) = \mathcal{O}(s+n)$. Hence the complexity of vector-based 2D point cloud localization with CGR is $\mathcal{O}\left(rm(s+n) + m^2\right)$. In comparison, the time complexity of MCL-SIR using a vector map is $\mathcal{O}\left(m + m \cdot (s+n)\right)$. Thus, for the same number of particles, CGR is generally slower than MCL-SIR. However, since CGR-localization provides greater accuracy than MCL-SIR, it can be run with far fewer particles, as we will demonstrate experimentally.

C. 3D Point Cloud Observation Model

The 3D point cloud observation model is used for sensors like depth cameras that generate point clouds in 3D space. Let the 3D point cloud be denoted by the set of 3D points $P = \{p_i\}_{i=1:n}$. The observation model does not use all the 3D points since this computation would be slow, and there are many detected points that would not correspond to features on the map. Hence, the points in P are sampled by a RANSAC [15] based algorithm that only accepts points grouped in local neighborhoods corresponding to planes. Thus, points that do not belong to vertical planes (walls on the map) are rejected. Figure 1 shows a sample scene with the raw unfiltered point cloud, and after filtering. The non-planar objects (two humans in this scene) are completely absent in the filtered scene.

The filtered points and their corresponding plane normal estimates are then projected onto 2D to yield the set of points $P' = \{p'_i, r'_i\}_{i=1:n'}$ where p'_i are the projected 2D points, and

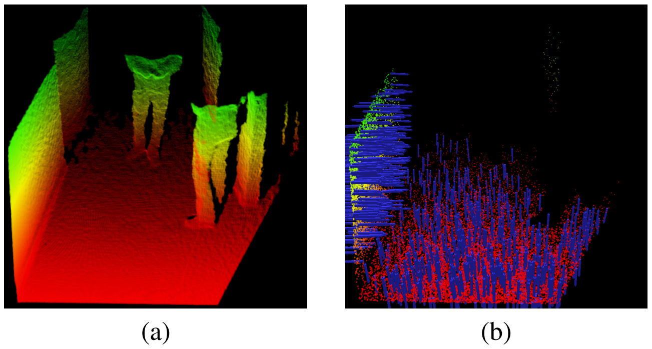
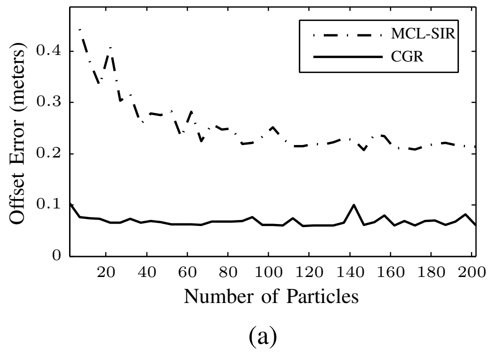


Fig. 1. 3D Point clouds captured by the Kinect sensor: (a) Unfiltered and (b) Plane filtered, with plane normals estimates shown as blue arrows.



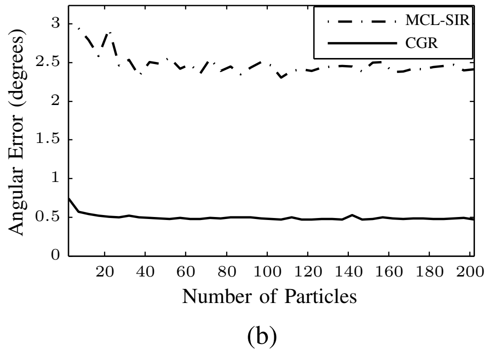
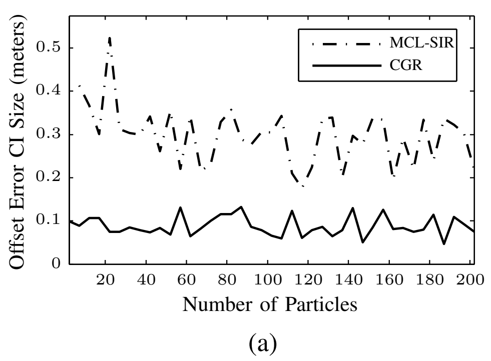


Fig. 2. The offset (a) and angular (b) errors from the true robot locations for the MCL-SIR and CGR algorithms



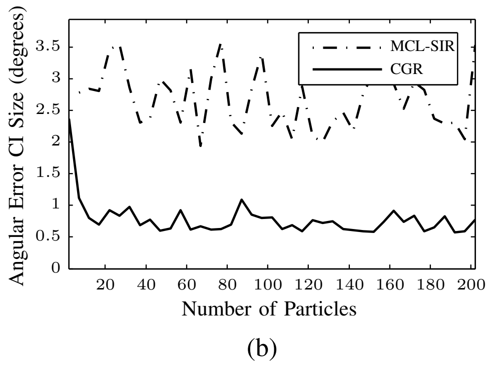


Fig. 3. The size of the 70% confidence intervals for the offset (a) and angular (b) errors for the MCL-SIR and CGR algorithms

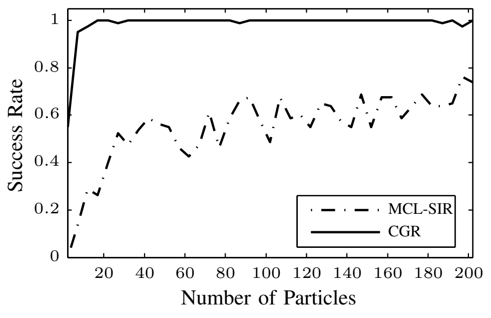


Fig. 4. Success rates of the CGR and MCL-SIR algorithms

r'_i the corresponding 2D normals. The projected points in P' are then be used to compute the observation model using the 2D observation model (Section IV-B). The only difference is that the outlier rejection for points p'_i in P' is estimated using the normals r'_i and the point to line correspondences l_i, instead of using pairs of points.

V. Experimental Results

Our experiments were performed on our custom built omnidirectional indoor mobile robot, which is equipped with a Hokuyo URG-04LX planar LIDAR scanner (which produces 2D point clouds) and a Microsoft Kinect sensor (which produces 3D point clouds).

We ran a series of experiments to compare the performance of the CGR algorithm with the MCL-SIR algorithm and to test the effectiveness of the CGR algorithm alone over extended periods of time. The MCL-SIR algorithm was implemented using the same map representation and observation models as those used for the CGR algorithm. The only difference is that for the MCL-SIR algorithm, there were no refine step, and the usual MCL-SIR update equations were used.

A. Comparison of CGR and MCL-SIR

In the first series of experiments, we logged sensor data while driving the robot around the map, traversing a path 374m long. The true robot location was manually annotated by aligning the sensor scans to the map. This data was then processed offline with varying number of particles to compare success rates, accuracy and run times for CGR as well as MCL-SIR. At the begining of every trial, the particles were randomly initialized with errors of up to $\pm 4\text{m}$ and $\pm 40°$. The number of particles m was varied from 2 to 302, with 80 trials each for CGR and MCL-SIR for each value of m. The values of f (the observation discount factor) and r (the number of iterations in the Refine step) were 2000 and 3 respectively for all trials.

1) Accuracy: Figure 2 shows the mean localization errors for the two algorithms for different numbers of particles. Both graphs show that localization using CGR is consistently more accurate than when using MCL-SIR. In particular, localizing using CGR with 20 particles has smaller mean errors than localizing using MCL with even 200 particles.

Another advantage of the CGR algorithm over MCL-SIR is that the CGR algorithm has lower variance across trials. Figure 3 shows the 70% confidence interval sizes for the two algorithms, for different numbers of particles. The variance across trials is consistently smaller for the CGR algorithm.

2) Success Rates: A trial was counted as being "successful" if the estimated end location was within $1m$ of the true robot location and if the error at every time step was less than $1m$. Figure 4 shows the success rates of the CGR and MCL-SIR algorithms for varying numbers of particles.

3) Computational Tradeoffs: Figure 5 shows the computational tradeoff for running CGR and MCL-SIR as a scatter plot of the localization offset error and the average run time of the complete belief update. Different data points represent different numbers of particles. This figure indicates that for any desired maximum computational runtime, CGR provides lower mean localization errors than MCL-SIR.

B. Long Run Trials

We have been using the CGR localization algorithm on our robot over deployment, and have been logging the localization data over all these runs. The localization algorithm used data from both the LIDAR scanner as well as the Kinect sensor, and ran in real time on the robot with 20 particles, updating at 30Hz. In total, over a span of 19 days, the robot had an autonomous run time of 52 hours, and traversed a total distance of over 13km, traversing a variety of environments including corridors, open areas and areas with clutter and pedestrian traffic. Figure 6 shows the combined trace of the robot's location, as reported by CGR localization.

VI. Conclusion And Future Work

We introduced the Corrective Gradient Refinement algorithm to refine the samples of the belief in particle filters. We developed an observation model using point cloud sensors for the task of indoor mobile robot localization. By experimental evaluation, we showed that the CGR localization algorithm using this observation model outperforms Sampling/Importance Resampling Monte Carlo Localization in terms of accuracy and computational requirements.

In the future, we are interested in applying CGR to higher dimensional state estimation, where we believe its advantages over standard particle filters will be even more pronounced. In addition, we intend to extend the work of 3D point cloud localization to fuse data from multiple depth cameras and perform complete 6 degree of freedom localization.

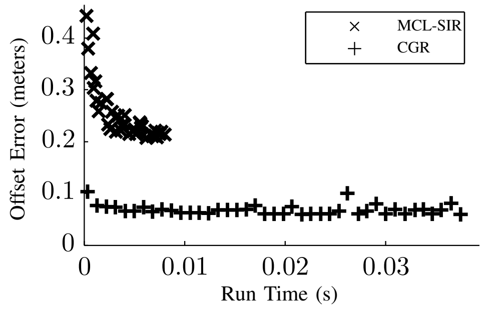


Fig. 5. Tradeoff between mean localization offset error and normalized computational run times.

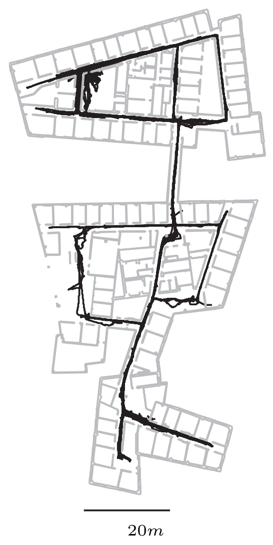


Fig. 6. Combined trace of robot location over all deployments

References

[1] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In *Proceedings of the National Conference on Artificial Intelligence*, pages 343–349. JOHN WILEY & SONS LTD, 1999.

[2] D. Salmond, N. Gordon, and A. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE Proc. F, Radar and signal processing*, volume 140, 1993.

[3] D. Fox. KLD-sampling: Adaptive particle filters and mobile robot localization. *Advances in Neural Information Processing Systems*, 2001.

[4] S. Lenser and M. Veloso. Sensor resetting localization for poorly modelled mobile robots. In *Int. Conf. on Robotics and Automation*, 2000.

[5] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, et al. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087, 1953.

[6] W.K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97, 1970.

[7] S. Duane, A.D. Kennedy, B.J. Pendleton, and D. Roweth. Hybrid monte carlo. *Physics Letters B*, 195(2):216 – 222, 1987.

[8] K. Choo and D.J. Fleet. People tracking using hybrid Monte Carlo filtering. In *Proc. of Int. Conf. on Computer Vision (ICCV)*. IEEE, 2001.

[9] M.K. Pitt and N. Shephard. Filtering via simulation: Auxiliary particle filters. *Journal of the American Statistical Association*, 1999.

[10] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *International Joint Conference on Artificial Intelligence*, volume 18. Citeseer, 2003.

[11] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *Robotics, IEEE Transactions on*, 23(1):34–46, 2007.

[12] Z. Chen, T. Kirubarajan, and M.R. Morelande. Improved particle filtering schemes for target tracking. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on*, volume 4, pages iv–145. IEEE, 2005.

[13] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. 2005.

[14] L. Zhang and B.K. Ghosh. Line segment based map building and localization using 2D laser rangefinder. In *IEEE Int. Conf. on Robotics and Automation*, 2000.

[15] M.A. Fischler and R.C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6), 1981.