Special Problems Fall 2018

Part Two

Planar Robot Localization with the Particle Filter

Sanandeesh Kamat *ssk93*

November 29, 2018

Contents

This final paper replicates a subset of algorithms described in *Probabilistic Robotics* by Sebastian Thrun, Wolfram Burgard, and Dieter Fox (**Thrun et al. 2005**). This author is deeply indebted to Dr. Thrun and his colleagues for producing this invaluable exposition into applied Bayesian filtering.

1 Introduction

During a previous study (Summer 2018), the performance of the Extended Kalman Filter (EKF) in localizing a planar robot was demonstrated and evaluated. The follow on study described in this paper investigated the performance of the *Particle Filter*, instead, in the very same planar localization task. Again, *localization* is defined as sequentially estimating the robot pose $\mathbf{x}_t$ given a precise environmental landmark map m to interpret incoming measurements $\mathbf{z}_t$. Together, the EKF and Particle Filter are the most popular Bayeisan filters implemented in robotics today. Whereas the EKF is the signifies the quintessential *parametric* Bayesian filter, the particle filter signifies the quintessential *non-parametric* Bayesian filter.

Non-parametric filters do not assume any particular form of PDF (e.g. Gaussian). They approximate the posterior belief of state with a collection of values spanning the belief state space. Some non-parameteric filters represent the posterior belief with a state space discretized into evenly spaced regions of uniform cumulative probability. Such filters are called *histogram filters*, and are essentially the discrete versions of the continuous Bayes filter. Others represent the posterior belief with a collection of random samples generated by the posterior distribution function itself. Such filters are called *particle filters*, and will be the primary filter investigated in this paper. As will be shown, particle filters are easy to implement, and versatile in their application.

The finite number of paramters employed by non-parametric filters will dictate the quality of the approximated belief. The approximated belief will converge to the true belief as the number of parameters tends to infinity. Non-parametric filters are consequently capabable of representing global uncertainty, distinct hypotheses, and multi-modal beliefs. This capability is required during *kidnapping*; when the robot is unaccountably transported into a different state by unforseen environmental forces. The potential to represent any arbitrarily complex belief is upper-bounded by the resource requirements for supporting that many parameters. Algorithms have been devised to modulate the number of parameters according to both the anticipated complexity of the belief and resources available. Bayesian filters which employ such algorithms in real time are called *resource adaptive*.

2 Particle Filters

The particle filter represents posterior beliefs with a collection of random samples generated by the posterior distribution function itself. These random samples are called *particles* and are defined as

$$\chi_t := x_t^{[1]}, x_t^{[2]}, \cdots, x_t^{[M]}. \tag{1}$$

The M samples in χ_t can be thought of as induvidual realizations of true outcomes (as if the true outcome might as well be an element of χ_t). Let's say that the outcome $\mathbf{x}_t$ which truly occurs is generated from the same posterior from which the particle filter is sampling,

$$x_t^{[m]} \sim p(x_t | z_{t:1}, u_{1:t}). \tag{2}$$

We would then expect the particle set χ_t to reliably describe the ways in which this outcome $\mathbf{x}_t$ is likely to occur. As a Bayesian filter, the posterior belief $bel(\mathbf{x}_t)$ is generated recursively from the previous posterior belief $bel(\mathbf{x}_{t-1})$. In the case of particle filters, we have χ_t generated recursively from χ_{t-1}.

	ParticleFilter$(\chi_{t-1}, \mu_t, \mathbf{z}_t)$
	Initialize empty prior and posterior particle sets.
1	$\overline{\chi}_t = \chi_t = \emptyset$
	Prediction Step (Prior Particle Set)
2	for $m = 1 : M$
	Forward propogate the mth previous posterior particles.
3	$sample\left(x_t^{[m]}\right) \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
	Compute weight of the proposed prior particle.
4	$w_t^{[m]} = p(\mathbf{z}_t \mid x_t^{[m]})$
	Store prior particle and weight together.
5	$\overline{\chi}_t = \overline{\chi}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
6	endfor
	Measurement Correction Step (Posterior Particle Set)
7	for $m = 1 : M$
	Sample the mth posterior particle.
8	draw $i \in (1 \cdots N)$ with probability $w_t^{[i]}$
	Store posterior particle.
9	$\chi_t = \chi_t + x_t^{[i]}$
10	endfor
11	return χ_t

Table 1: The Bootstrap Particle Filter Algorithm

The outline of the particle filter algorithm is provided in Table 1. The inputs inlcude the previous posterior belief χ_{t-1}, the current control u_t, and the current measurement set z_t. The algorithm can be divided into two steps.

During the *prediction step*, the a priori particle set $\overline{\chi}_t$ is constructed. During the *meausrement correction step*, the a posteriori particle set χ_t is constructed. The interpretations of these particle sets $\overline{\chi}_t$ and χ_t are identical to those of $\overline{bel}(\mathbf{x}_t)$ and $bel(\mathbf{x}_t)$ from the general Bayes filter. As such, the particle filter is remarkably performing Bayesian estimation of a dynamic state without the parametric form of the posterior itself. These two general steps of the algorithm are detailed below.

- **Prediction Step**

  Within lines 2-6, every element of the particle set $\left(x_{t-1}^{[m]} \in \chi_{t-1}\right)$ is propoageted through time to the next time step according to the expectation of the state transition model, $x_t^{[m]} = E[g(x_{t-1}^{[m]}, u_t)]$ (i.e. the error free motion model). This resulting particle set is the proposed *prior* particle set $\left(x_t^{[m]} \in \overline{\chi}_t\right)$.

  To later incorporate the most recent measurements $\mathbf{z}_t$, line 5 computes for each prior particle $x_t^{[m]}$ the corresponding *importance factor*, $w_t^{[m]}$. The importance factor is equal to the joint probability of witnessing the observation set $\mathbf{z}_t$ from the vantage point of each proposed prior particle in $\overline{\chi}_t$, given by $w_t^{[m]} = p(z_t|x_t^{[m]})$.

- **Measurement Correction Step**

  Within lines 7-10, the posterior particle set χ_t of the same size as $\overline{\chi}_t$ is constructed by randomly sampling (with replacement) particles from $\overline{\chi}_t$ into χ_t. The trick is that the probability of sampling any particular $\left(x_t^{[i]} \in \overline{\chi}_t\right)$ is determined by the relative weight of that particle $w_t^{[i]}$. That is, prior particles with large weights are more likely to be sampled than prior particles with low weights. In essence, this reconfigures the distribution of the prior particle set $\overline{\chi}_t$ according to the measurement model $p(\mathbf{z}_t|x_t)$ to approximate the true posterior distribtion $p(x_t|z_{1:t}, u_{1:t})$. This approach of posterior estimation is referred to as *resampling* or *importance sampling*.

The point estimate of robot pose $\hat{\mathbf{x}}_t$ is the expectation over the particle set, which is approximated by the sample mean:

$$\hat{\mathbf{x}}_t = E[\chi_t] \tag{3}$$

$$\approx \frac{1}{M}\sum_{m=1}^{M} x_t^{[m]}. \tag{4}$$

The critical advantage of the particle filter (over the EKF) becomes obvious when one seeks the *true* distribution of a belief the form of which is both unknown and evidently non-Gaussian. Even if input parameters are truly Gaussian, non-linear state transition and measurment models prohibit the the posterior distribution from being Gaussian.

On one hand, the EKF precisely defines what is explicitly an *approximation* of the true posterior (e.g. the Gaussian assumptions, the model linearizations,...). On the other hand, the particle filter approximates (as $N \to \infty$) what is (theoretically) the *true* posterior. This subtle but profound distinction has significantly elevated the popularity of the particle filter. When deploying filters for real-time operation, the primary tradeoffs to consider involve computational capacity of the fielded systems. Fortunately, because processor capabilities have improved extraordinarily, such sampling based algorithms have become pleasantly feasible today.

2.1 Kernel Density Estimation

The particle set χ_t generated by the particle filter algorithm is only a discrete/sampled approximation to the true posterior. One seeks to model a smooth continous density function so that cumulative probability mass can be obtained over any arbitrary region of state space (for finite M). Or one may seek simply to effectively visualize the corresponding density. To achieve such ends, a common solution is to perform *kernel density esimation* (KDE) upon the raw particle set χ_t. The KDEstimator is

$$\hat{f}_h(\mathbf{x}) = \frac{1}{M}\sum_{i=1}^{M} K_h(\mathbf{x} - \mathbf{x}_t^{[i]}) \tag{5}$$

$$= \frac{1}{M\sqrt{(2\pi)^N |\mathbf{\Sigma}|}} \sum_{i=1}^{M} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{x}_t^{[i]})^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{x}_t^{[i]})} \tag{6}$$

As shown in equation 4, a common kernel density chosen is the Gaussian distribution. KDE simply replaces each particle in the particle set with a mini-Gaussian (Gaussian divided by M) centered at the location of the original particle. The resulting sum of Gaussians integrates to one. The covariance matrix Σ of each mini-Gaussian is usually diagonal and either pre-defined or adaptively proportional to the density of samples in that neighbourhood. Figure 3 shows an example of a raw particle set and the corresponding KDE.

3 Planar Robot Motion & Measurement Models

For convenience, this section briefly reviews the motion and measurement models used by the particle filter in this paper. These models have been already been thoroughly described in the previous study's paper, (Kamat, 2018). Please refer to the previous paper for a more detailed description of these models.

The robot pose $\mathbf{x}_t$ is defined as

$$\mathbf{x}_t = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix}, \tag{7}$$

where (x_t, y_t) are the cartesian coordinates and θ_t is the angular orientation of the robot. The landmark map $\mathbf{m}$ is defind as

$$\mathbf{m} = \begin{pmatrix} m_x^1 & \cdots & m_x^N \\ m_y^1 & \cdots & m_y^N \end{pmatrix}, \tag{8}$$

where m_x^i and m_y^i are the cartesian coordinates of the ith landmark. The measurement set $\mathbf{z}_t$ is defined as

$$\mathbf{z}_t = \begin{pmatrix} r_t^1 & \cdots & r_t^N \\ \phi_t^1 & \cdots & \phi_t^N \end{pmatrix}, \tag{9}$$

where r_t^i is the local range and ϕ_t^i is the local bearing to the ith landmark. The correspondence between measurements z_t^i and landmarks m^i is assumed to be known.

3.1 Velocity Motion Model

The motion model dictates the state transition probability $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t)$. The particle filter simply samples the *forward* model $\mathbf{x}_t = g(\mathbf{x}_{t-1}, \mathbf{u}_t)$ to generate random pose $\mathbf{x}_t$ realizations (just like the simulation software against which filters are tested). As such, the inverse model which directly estimates $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t)$ is unnecessary. Moreover, the inner loop of the particle filter (Table 1, line 3) will execute $\mathbf{x}_t = g(\mathbf{x}_{t-1}, \mathbf{u}_t)$. The noisy forward motion model is defined as

$$g(\mathbf{x}_{t-1}, \mathbf{u}_t) = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} = \begin{pmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{pmatrix} + \begin{pmatrix} -\frac{\hat{v}}{\hat{w}}\sin(\theta) + \frac{\hat{v}}{\hat{w}}\sin(\theta + \hat{w}\Delta t) \\ \frac{\hat{v}}{\hat{w}}\cos(\theta) - \frac{\hat{v}}{\hat{w}}\cos(\theta + \hat{w}\Delta t) \\ \hat{w}\Delta t + \hat{\gamma}\Delta t \end{pmatrix}, \tag{10}$$

where $\hat{v}$, $\hat{w}$, and $\hat{\gamma}$ are the Gaussian-distributed velocity motion commands from $\mathbf{u}_t$.

In the absence of measurements $\mathbf{z}_t$, uncertainty in the posterior $bel(\mathbf{x}_t)$ grows unbounded over time. In terms of the EKF, this would be propogating the Gaussian posterior $(\mu_{t-1}, \mathbf{\Sigma}_{t-1})$ through the linearized approximation of $g(\mathbf{x}_{t-1}, \mathbf{u}_t)$ repeatedly. In terms of the particle filter, this would be propogating the particle set χ_{t-1} through the *actual* $g(\mathbf{x}_{t-1}, \mathbf{u}_t)$ repeatedly. Shown in Figure 1 is

the evolution of a particle set without the benefit of any importance sampling (measurement correction).

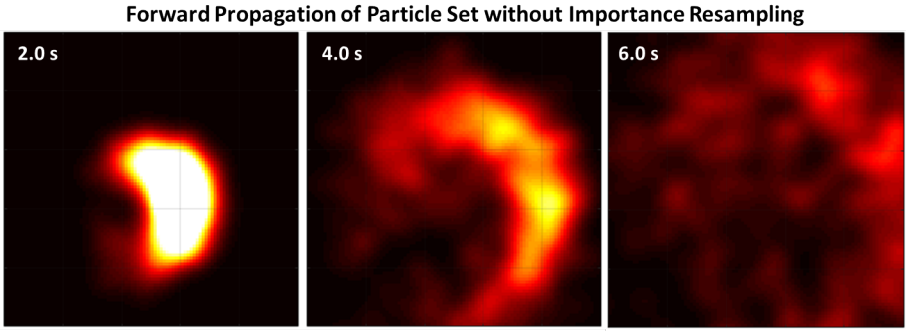

Figure 1: Forward Propogation of Particle Set without Importance Sampling

3.2 Feature Measurement Model

The measurement model dictates the measurement probability $p(\mathbf{z}_t|\mathbf{x}_t)$. As shown in Table 1 line 4, it is used to compute weights $w_t^{[i]}$ for each particle proposed in the prior particle set $\overline{\chi}_t$. The rational is that particles with higher weights possess a higher likelihood of representing the true pose $\mathbf{x}_t$ because they are consistent with the witnessed measurements. This rational is valid so long as landmark measurements $\mathbf{z}_t$ are conditionally independant of each other and induvidually correct on the average (i.e. *unbiased*).

The forward measurement model is defined as

$$h(\mathbf{x}_t) = \begin{pmatrix} r_r^i \\ \phi_t^i \end{pmatrix} = \begin{pmatrix} \sqrt{(m_{j,x} - x_t)^2 + (m_{j,y} - y_t)^2} \\ atan2(m_{j,y} - y_t, m_{j,x} - x_t) - \theta \end{pmatrix} + \begin{pmatrix} \epsilon_{\sigma_r^2} \\ \epsilon_{\sigma_\phi^2} \end{pmatrix} \tag{11}$$

where $\epsilon_{\sigma_r^2}$, $\epsilon_{\sigma_\phi^2}$ are the Gaussian-distributed measurement errors.

Unlike for the motion model $g(\mathbf{x}_{t-1}, \mathbf{u}_t)$, the particle filter (as it is implemented here...) does not directly sample the noisy measurement model $h(\mathbf{x}_t)$. Rather, it merely uses the error free measurement model,

$$E\left[h(\mathbf{x}_t)\right] = \begin{pmatrix} E[r_r^i] \\ E[\phi_t^i] \end{pmatrix} = \begin{pmatrix} \sqrt{(m_{j,x} - x_t)^2 + (m_{j,y} - y_t)^2} \\ atan2(m_{j,y} - y_t, m_{j,x} - x_t) - \theta \end{pmatrix}, \tag{12}$$

to compute the expected measurement for any proposed particle $x_t^{[i]}$ in the prior particle set $\overline{\chi}_t$.

Due to conditional independence between measurements of distinct landmarks, as well as independence between noise in r and θ, the prior particle

weights are computed as as follows,

$$
\begin{aligned}
w_t^{[i]} &= p(\mathbf{z}_t | x_t^{[i]}, m) && (13)\\
&= \prod_{k=1}^{K} p(\mathbf{z}_t^k | x_t^{[i]}, m) && (14)\\
&= \prod_{k=1}^{K} p(r_t^k, \phi_t^k | x_t^{[i]}, m) && (15)\\
&= \prod_{k=1}^{K} p(r_t^k | E[r_t^k], \sigma_r^2) p(\phi_t^k | E[\phi_t^k], \sigma_\phi^2). && (16)
\end{aligned}
$$

Figure 2 illustrates how the weights of particles decrease as a function of distance of the particle from the true robot pose.

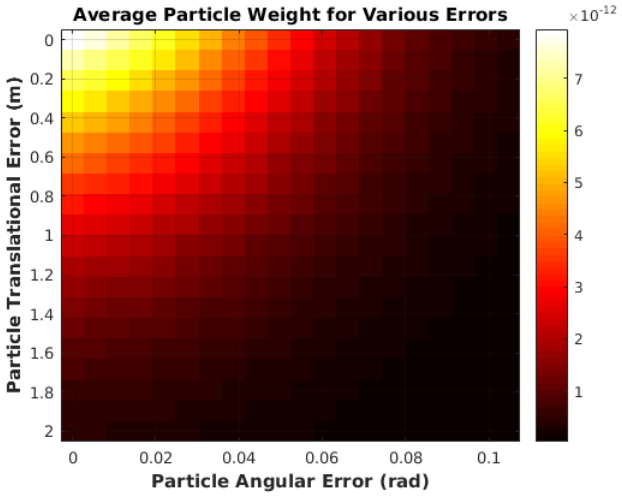

Figure 2: Particle weights as a function of particle error and measurement error

4 Monte Carlo Localization with the Particle Filter

4.1 Basic MCL for Global Localization

The application of particle filtering to robot localization is called *Monte Carlo Localization* (MCL). The MCL algorithm is obtained by inserting stochastic measurement and motion models, such as those described in Section 3, into the particle filter algorithm in Table 1. The modified particle filter algorithm is shown in Table 2.

	$\mathbf{MCL}(\chi_{t-1}, \mu_t, \mathbf{z}_t, \mathbf{m})$
	Initialize empty prior and posterior particle sets.
1	$\overline{\chi}_t = \chi_t = \emptyset$
	Prediction Step (Prior Particle Set)
2	for $m = 1 : M$
	Forward propogate the mth previous posterior particle.
3	$\begin{pmatrix} x_t^{[m]} \\ y_t^{[m]} \\ \theta_t^{[m]} \end{pmatrix} = g(\mathbf{x}_{t-1}^{[m]}, \mathbf{u}_t) = \begin{pmatrix} x_{t-1}^{[m]} \\ y_{t-1}^{[m]} \\ \theta_{t-1}^{[m]} \end{pmatrix} + \begin{pmatrix} -\frac{\hat{v}}{\hat{w}}\sin(\theta) + \frac{\hat{v}}{\hat{w}}\sin(\theta + \hat{w}\Delta t) \\ \frac{\hat{v}}{\hat{w}}\cos(\theta) - \frac{\hat{v}}{\hat{w}}\cos(\theta + \hat{w}\Delta t) \\ \hat{w}\Delta t + \hat{\gamma}\Delta t \end{pmatrix}$
	Compute weight of the proposed prior particle.
4	$w_t^{[i]} = \prod_{k=1}^{K} p(r_t^k \vert E[r_t^k], \sigma_r^2) p(\phi_t^k \vert E[\phi_t^k], \sigma_\phi^2)$
	Store prior particle and weight together.
5	$\overline{\chi}_t = \overline{\chi}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
6	endfor
	Measurement Correction Step (Posterior Particle Set)
7	for $m = 1 : M$
	Sample the mth posterior particle.
8	draw $i \in (1 \cdots N)$ with probability $w_t^{[i]}$
	Store posterior particle.
9	$\chi_t = \chi_t + x_t^{[i]}$
10	endfor
11	return χ_t

Table 2: The Monte Carlo Localization Algorithm

Only two modifications have been made! First, line 3 of Table 2 propagates each previous posterior particle randomly across time by the noisy velocity motion model, $g(\mathbf{x}_{t-1}^{[m]}, \mathbf{u}_t)$. Second, line 4 of Table 2 computes the weight of each propagated particle with respect to the most recent measurement set, $\mathbf{z}_t$. Here $p(r_t^k \vert E[r_t^k], \sigma_r^2)$ is the probability density of the actual measurment r_t^k from a Gaussian RV tuned to kth particle with mean $E[r_t^k]$ and variance σ_r^2. The remaining lines of Table 2 (i.e. the measurement correction step) are identical to those of Table 1. This is because the importance sampling phase is merely a

generic two line software operation. For example, the following Matlab commands

```
iResampled = randsample(indeces, numParticles, true, ParticleWeights);
ResampledPS = PriorParticleSet(:,iResampled);
```

resamples the particle set according the the particle weights. Unlike the EKF, which required numerous Jacobian calculations to incorporate the measurement/motion models, the particle filter relies on computer based random number generation to simulate them directly. In other words, whereas the EKF could be implemented by hand calculations, the particle filter fundamentally requires computers.

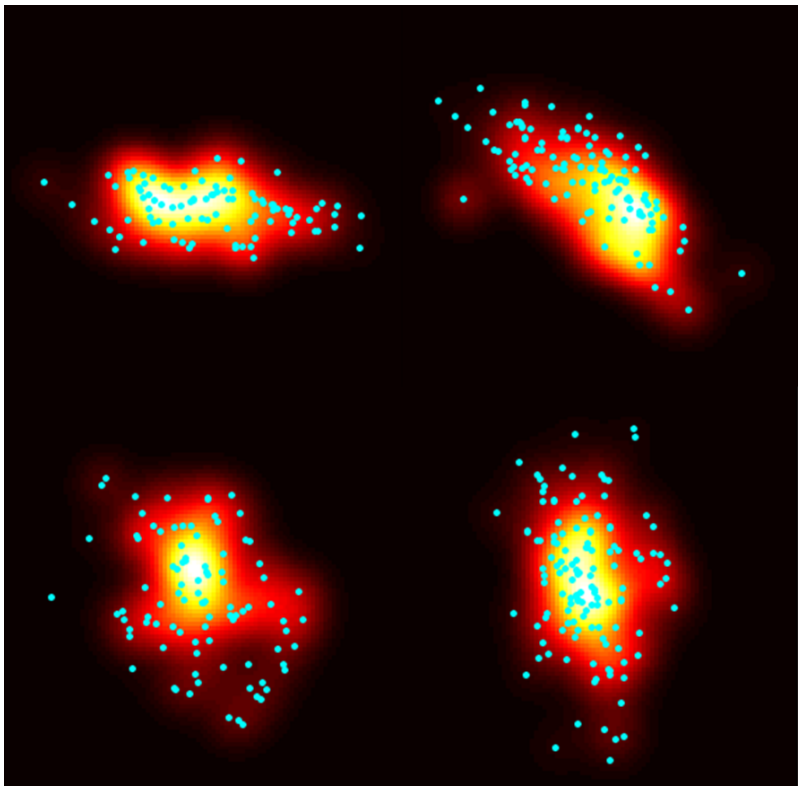

Figure 3: Example Posterior Particle Sets with Gaussian KDE

Shown in Figure 3 are example posterior particle sets with overlaid Gaussian KDEs for planar robot motion localization (the third angular dimension is missing). Visual inspection of these particle sets indicates that the true posteriors are non-Gaussian, but are *still uni-modal*. Up next, the particle filter's capacity to recover from sudden global ambiguities such as *kidnapping* will be tested as well.

4.2 Augmented MCL for Failure Recovery

As it has been presented so far, the MCL will eventually remove all particles except those close to a single pose $\mathbf{x}_t$. When this estimate $\mathbf{x}_t = E[\chi_t]$ approximates the truth, the precision is desirable. However, in the event of *global localization failures* (e.g. kidnapping), the narrow particle set χ_t will be unable to recover to the new true pose. The absence of any particles in the proximity of the correct pose is called the *paricle deprivation problem.*

This paper adresses particle deprivation by adding random particles to χ_t. The rationale is that the non-zero probability of global localization failure is captured by these random particles. The decision to infuse random particles into χ_t will be determined by some measurement-based metric. We seek to compare the most recent measurement probability,

$$p(\mathbf{z}_t|\mathbf{z}_{1:t-1}, u_{1:t}, \mathbf{m}), \tag{17}$$

with the expected (average) measurement probability. The most recent measurement probability is estimated by the sample average of the most recent weights.

$$p(\mathbf{z}_t|\mathbf{z}_{1:t-1}, u_{t-1}, \mathbf{m}) \approx \frac{1}{M}\sum_{i=1}^{M} w_t^m \tag{18}$$

Circumstances such as sensor noise and initially dispersed particles, which are *not* global localization failures, may cause low meaurement probabilities and trigger the infusion of random particles. Therefore, short term and long term *time averaged* measurement probabilities (w_{slow}, w_{fast}) are compared when considering random particles. The distribution of this new random particle can be tuned the measurement distribution $p(\mathbf{z}_t|\mathbf{x}_t)$. In other words, when the prior distribution fails, rely on the measurement likelihood!

	Augmented MCL$(\chi_{t-1}, \mu_t, \mathbf{z}_t, \mathbf{m})$
	Initialize short-term & long-term particle weight averages
1	static w_{slow}, w_{fast}
	Initialize empty prior and posterior particle sets.
2	$\overline{\chi}_t = \chi_t = \emptyset$
	Prediction Step (Prior Particle Set)
3	for $m = 1 : M$
	Forward propogate the mth previous posterior particle.
4	$\begin{pmatrix} x_t^{[m]} \\ y_t^{[m]} \\ \theta_t^{[m]} \end{pmatrix} = g(\mathbf{x}_{t-1}^{[m]}, \mathbf{u}_t) = \begin{pmatrix} x_{t-1}^{[m]} \\ y_{t-1}^{[m]} \\ \theta_{t-1}^{[m]} \end{pmatrix} + \begin{pmatrix} -\frac{\hat{v}}{\hat{w}}\sin(\theta) + \frac{\hat{v}}{\hat{w}}\sin(\theta + \hat{w}\Delta t) \\ \frac{\hat{v}}{\hat{w}}\cos(\theta) - \frac{\hat{v}}{\hat{w}}\cos(\theta + \hat{w}\Delta t) \\ \hat{w}\Delta t + \hat{\gamma}\Delta t \end{pmatrix}$
	Compute weight of the proposed prior particle.
5	$w_t^{[i]} = \prod_{k=1}^{K} p(r_t^k \vert E[r_t^k], \sigma_r^2) p(\phi_t^k \vert E[\phi_t^k], \sigma_\phi^2)$
	Store prior particle and weight together.
6	$\overline{\chi}_t = \overline{\chi}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7	$w_{avg} = w_{avg} + \frac{1}{M} w_t^{[m]}$
8	endfor
9	$w_{slow} = w_{slow} + \alpha_{slow}(w_{avg} - w_{slow})$
10	$w_{fast} = w_{fast} + \alpha_{fast}(w_{avg} - w_{fast})$
	Measurement Correction Step (Posterior Particle Set)
11	for $m = 1 : M$
12	with probability $max(0.0, 1.0 - w_{fast}/w_{slow})$ do
	Generate random particle according to $\mathbf{z}_t$.
13	$x_t^{[]} = GeneratePose(\mathbf{z}_t)$
14	Store new particle.
15	$\chi_t = \chi_t + x_t^{[]}$
16	else
	Sample the mth posterior particle.
17	draw $i \in (1 \cdots N)$ with probability $w_t^{[i]}$
	Store posterior particle.
18	$\chi_t = \chi_t + x_t^{[i]}$
19	endwith
20	endfor
21	return χ_t

Table 3: The Augmented Monte Carlo Localization Algorithm

Table 3 describes the Augmented MCL algorithm which is is proposed to be resilient to global localization failures. The only alterations to the Prediction Step are that running-averaged short-term and long-term measurement probabilities w_{slow}, w_{fast} are updated from the latest w_{avg}. Here, the decay rates are specified as $0 \leq \alpha_{slow} \ll \alpha_{fast}$. The Measurement Correction Step includes the main augmentaion in lines 12 to 15. For each re-sampling iteration, with probability $max(0.0, 1.0 - w_{fast}/w_{slow})$, a random sample is infused. The inten-

tion is to infuse random particles in a quantity proportional to the probability of a global localization failure. When a kidnapping occurs, for example, the short-term w_{fast} will drop faster than the long term w_{slow}. During this window of time, random particles based on the most recent measurements will enter the posterior belief. The distribution from which these random samples arise is described next.

4.2.1 Randomly Sampling Poses

We need to generate a random robot pose $\mathbf{x}_t$ that corresponds to a single landmark measurement $\mathbf{z}_t^i = (r_t^i, \phi_t^i)^T$. Whereas previously random poses were sampled from the motion model (Prediction Step), here we sample them from the measurement model. This capability is crucial when prior pose information is absent/unreliable. The pose distribution we seek (expanded by Bayes theorem) is

$$p(\mathbf{x}_t|\mathbf{z}_t^i, \mathbf{m}) = \frac{p(\mathbf{z}_t^i|\mathbf{x}_t, \mathbf{m})p(\mathbf{x}_t|\mathbf{m})}{p(\mathbf{z}_t^i|\mathbf{m})}. \tag{19}$$

Assuming that the pose is uniformly distributed given the map (i.e. uninformative prior), the likelihood function $p(\mathbf{z}_t^i|\mathbf{x}_t, \mathbf{m})$ is all we need.

However, landmark measurement is not an invertible function! In other words, a particular landmark range r_t^i and bearing ϕ_t^i could have been witnessed from a continuum of poses. Even for an error-free measurement, the space of poses forms a ring around the landmark with a diameter equal to the measured range r_t^i. Each pose around the ring will have a unique pointing angle depending on the measured bearing ϕ_t^i. Table 4 describes this pose sampling algorithm. The unformly distributed angle generated in line 1 guesses the indeterminate parameter, where on the circle the pose lies.

	RandomPoseGenerator$(\mathbf{z}_t^i, \mathbf{m})$
	Random angle.
1	$\hat{\gamma} = UnifRand(0, 2\pi)$
	Random range and bearing.
2	$\hat{r} = r_t^i + \varepsilon_r$
3	$\hat{\phi} = \phi_t^i + \varepsilon_\phi$
	Random pose.
4	$x = m_{i,x} + \hat{r}\cos\hat{\gamma}$
5	$y = m_{i,y} + \hat{r}\sin\hat{\gamma}$
6	$\theta = \hat{\gamma} - \pi - \hat{\phi}$
7	return$(x, y, \theta)^T$

Table 4: The Measurement-based Random Pose Sampling Algorithm

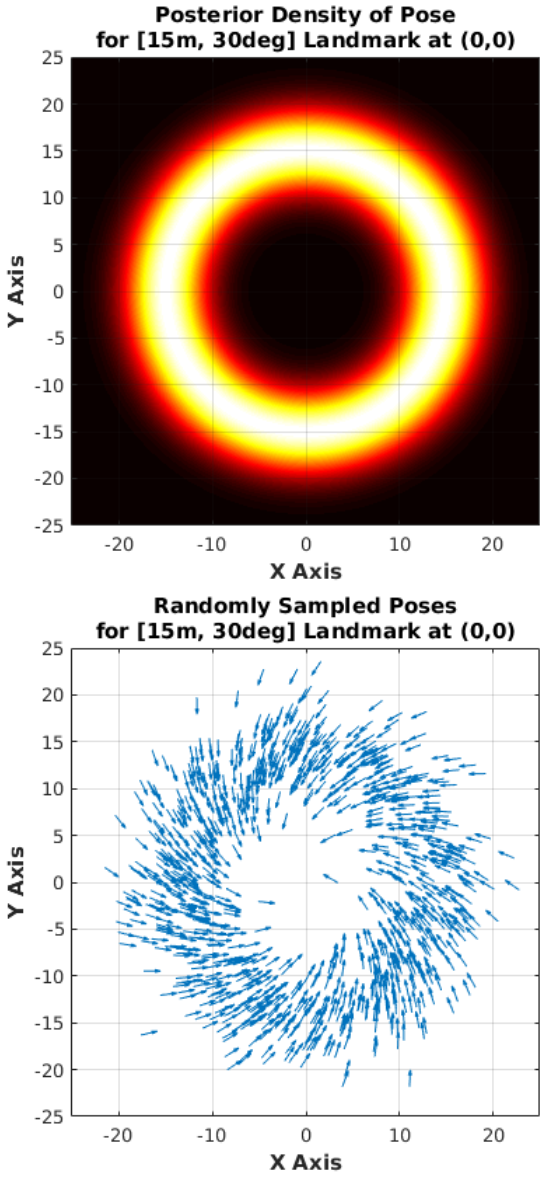

Figure 4: Example Posterior Particle Sets with Gaussian KDE

Figure 4 provides two representations of the landmark-based pose distribution. In Figure 4A, the probability of each pose is computed. Note that the third dimension of this figure, p_θ, would protrude like spiral. In Figure 4B, numerous poses are randomly sampled from the algorithm in Table 4. Here, the sample pose angles θ are visible to the reader.

5 Simulation Results

5.1 Scenario Design

The goal of these simulations is to learn how various parameters influence the Monte Carlo Localization (MCL) performance. These parameters include the motion error $\alpha_1 \cdots \alpha 4$, measurement error $\sigma_r^2, \sigma_\phi^2$, and number of particles M in χ_t. Therefore, the simulations were broken into three categories.

- Simulations which modulate the *motion error parameters* $\alpha_1 \cdots \alpha 4$, and leave the remaining parameters constant.
- Simulations which modulate the *measurement error parameters* $\sigma_r^2, \sigma_\phi^2$, and leave the remaining parameters constant.
- Simulations which modulate the *number of particles* M, and leave the remaining parameters constant.

Table 5 describes all of the parameters represented in the simulations, including those which were not probed for analysis (no range of values).

	Default	**Range**
Time Step	$Ts = 0.1s$	
Motion Commands	$v = 2\text{m/s}$ $w = 0.2\text{rad/s}$	
Map Configuration	$N = 10$ $R = 50\text{m}$	
Motion Error	$\alpha_{1,2} = 0.5$ $\alpha_{3,4} = 0.5$ $\alpha_{5,6} = 0.5$	$\alpha_{1,2} = 0.1 \cdots 5.0$ $\alpha_{3,4} = 0.1 \cdots 5.0$
Measurement Error	$\sigma_R^2 = 0.5$ $\sigma_\phi^2 = 0.05$	$\sigma_R^2 = 0.1 \cdots 5.0$ $\sigma_\phi^2 = 0.01 \cdots 1.0$
Number of Particles	$M = 500$	$M = 50 \cdots 1000$

Table 5: Selected Simulation Parameters

Note that the selected parameters in Table 5 precisely match those of the previous EKF-localization study (Kamat, 2018) for comparison. As before, the environment consisted of N landmarks uniformly spaced along the perimeter of a circle of radius R centered on the origin. Figure 5 illustrates three example maps of different N, R configurations.

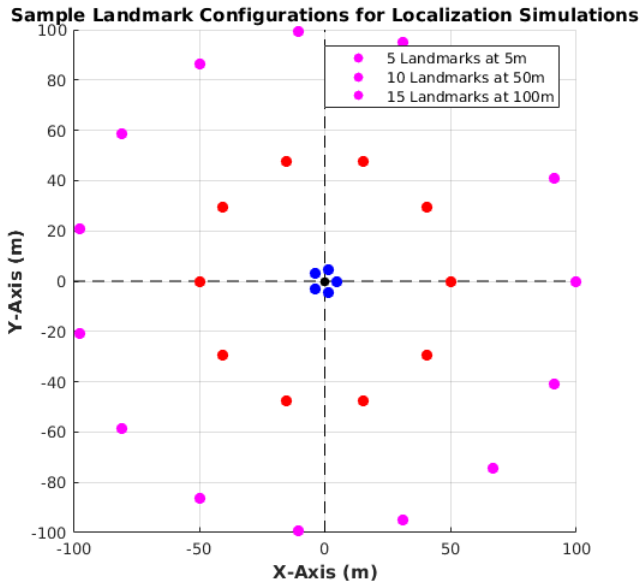

Figure 5: Sample Landmark Configurations for Localization Simulations

5.2 MCL for Global Localization

Figure 6 provides the raw robot localization results for 10 simulations run at the default parameters (Table 5). Figure 6A shows the true paths (black dashed) and the estimated paths (green solid) taken by the robot. Figure 6B,C plot the position and angular localization errors, respectively, versus time. These outputs reveal a strong localization capability, with $\mu_{PosErr} = 0.4$m and $\mu_{\theta Err} = 1.1$deg. Whereas the EKF-Localization (Kamat, 2018) localization errors were tumultuous and required transient 'settling down' time, these results are both accurate and precise from the start.

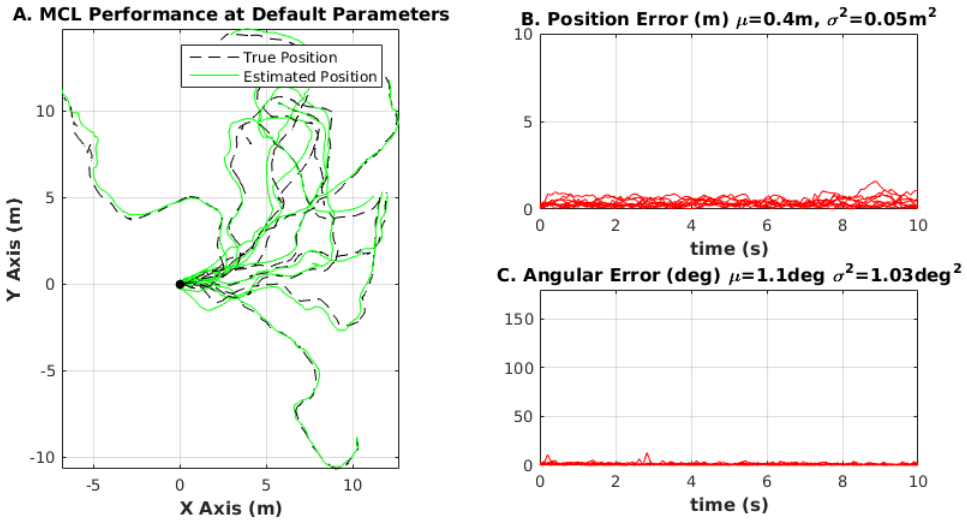

Figure 6: MCL Simulations at Default Parameters (Table 5)

5.2.1 Effects of Motion Error on Performance

This section describes how the overall MCL performance $(\mu_{PosErr}, \sigma^2_{PosErr})$ and $(\mu_{\theta Err}, \sigma^2_{\theta Err})$ was affected by wide modifications to the motion error parameters $\alpha_{1,2,3,4}$. Recall from Thrun et al. 2005 that $\alpha_{1,2}$ and $\alpha_{3,4}$ scale the variances of zero-mean Gaussian errors in tranlational and rotational motion commands, respectively. That is,

$$\begin{pmatrix} \hat{v} \\ \hat{w} \end{pmatrix} = \begin{pmatrix} v \\ w \end{pmatrix} + \begin{pmatrix} \epsilon_{\alpha_1 v^2 + \alpha_2 w^2} \\ \epsilon_{\alpha_3 v^2 + \alpha_4 w^2} \end{pmatrix}, \tag{20}$$

where ϵ_{b^2} is Norm$(0, b^2)$. Figure 7 top row and bottom row describe the position error $(\mu_{PosErr}, \sigma^2_{PosErr})$ and angular error $(\mu_{\theta Err}, \sigma^2_{\theta Err})$, respectively. As expected, increasing the tranlational motion error parameters $\alpha_{1,2}$ increases μ_{PosErr} from 0 to 3m, but does not influence $\mu_{\theta Err}$. Unexpectedly, increasing the angular motion error parameters $\alpha_{3,4}$ drove neither μ_{PosErr} nor $\mu_{\theta Err}$ up. In fact, $\mu_{\theta Err}$ was highest (3 deg) for the lowest $\alpha_{3,4}$ (0.1). The error variances $\sigma^2_{PosErr}, \sigma^2_{\theta Err}$ were primarily unaffected, which indicates *consistency* in performance.

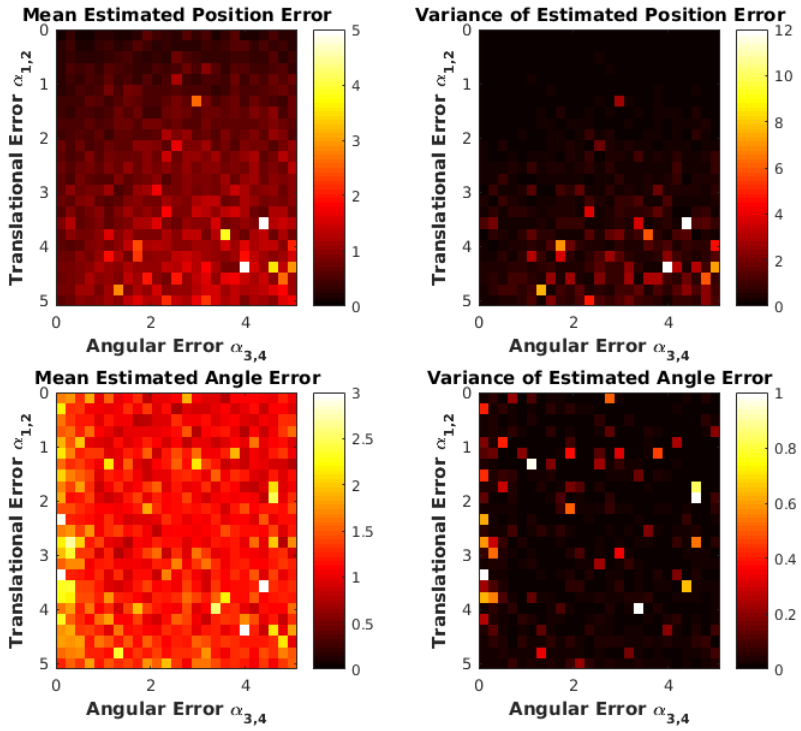

Figure 7: Mean and Variance of MCL Errors over Motion Error Parameters

5.2.2 Effects of Measurement Error on Performance

This section describes how the overall MCL performance $(\mu_{PosErr}, \sigma^2_{PosErr})$ and $(\mu_{\theta Err}, \sigma^2_{\theta Err})$ was affected by wide modifications to the measurement error parameters, $\sigma_r^2, \sigma_\phi^2$. Recall from Thrun et al. 2005 that σ_r^2, and σ_ϕ^2 are the variances of zero-mean Gaussian errors in range and bearing measurements, respectively. That is,

$$\begin{pmatrix} z^i_{t,r} \\ z^i_{t,\Phi} \end{pmatrix} = \begin{pmatrix} z^i_{t,r} \\ z^i_{t,\Phi} \end{pmatrix} + \begin{pmatrix} \epsilon_{\sigma_r^2} \\ \epsilon_{\sigma_\Phi^2} \end{pmatrix}, \tag{21}$$

where ϵ_{b^2} is Norm$(0, b^2)$. Figure 8 top row and bottom row describe the position error $(\mu_{PosErr}, \sigma^2_{PosErr})$ and angular error $(\mu_{\theta Err}, \sigma^2_{\theta Err})$, respectively. As expected, increasing the range measurement variance σ_r^2 increases μ_{PosErr} steadily from 0 to 2m, but does not influence $\mu_{\theta Err}$. Similarly, as expected, increasing the angular measurement variance σ_ϕ^2 increases $\mu_{\theta Err}$ steadily from 0 to 12 deg, but does not influence μ_{PosErr}. Similar to those of Figure 7, the error variances $\sigma^2_{PosErr}, \sigma^2_{\theta Err}$ were primarily unaffected. Notice that there are matching patterns of error exhibited between Figures 7 and 8! This would be because translational errors affect translational estimates, and angular errors affect angular estimates.

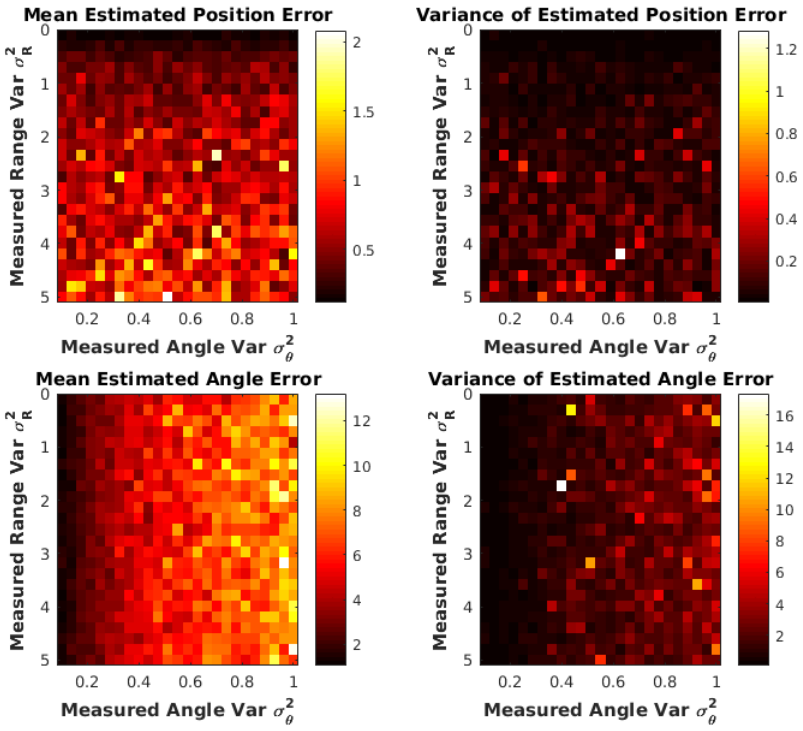

Figure 8: Mean and Variance of MCL Errors over Measurement Error Parameters

5.2.3 Effects of Number of Particles on Performance

This section describes how the overall MCL performance $(\mu_{PosErr}, \sigma^2_{PosErr})$ and $(\mu_{\theta Err}, \sigma^2_{\theta Err})$ was affected by wide modifications to the number of particles N used to represent the posterior belief. Figure 9 top row plots the position error $(\mu_{PosErr}, \sigma^2_{PosErr})$ and angular error $(\mu_{\theta Err}, \sigma^2_{\theta Err})$ versus N on the left and right y-axis, respectively. As expected, both position and angular error decreased as a function of N. The improvement in both of these errors continue until $N = 400$, and then remain flat. However, it does not appear that the variance in these errors changes as a function of N. A common theme across Figures 7, 8, and 9 is that error variances $\sigma^2_{PosErr}, \sigma^2_{\theta Err}$ do not change commensurately with widely changing simulation parameters.

Figure 9 bottom row plots the mean execution time of the particle filter as a function of N. As expected from the 'for-loop' implementation of the particle filter, the mean execution increases approximately linearly with N. Parallelization of the update/correction operations would have avoided this increase in execution time altogether. The take away message for a designer would be to either parallelize the particle filter operations, or set $N \leq 400$.

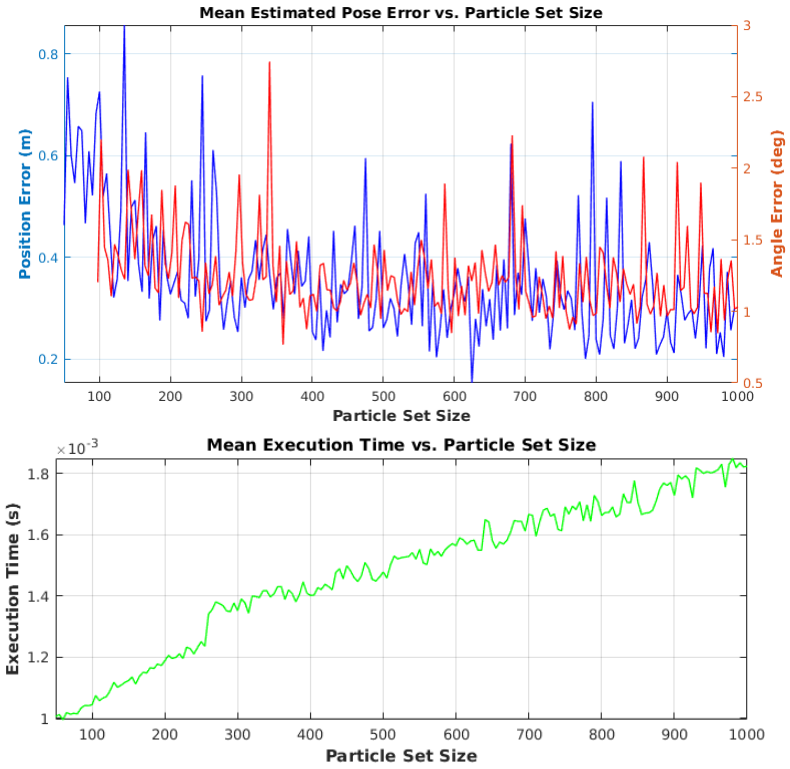

Figure 9: Mean Pose Error and Execution Time vs. Number of Particles

5.3 MCL for Failure Recovery

Recall that the *paricle deprivation problem* occurs when particles are tightly collected near the true robot state $\mathbf{x}_t$, the robot experiences a sudden kidnapping to another state $\mathbf{x'}_t$, and is subsequently unable to recover the true new pose. The ability to recover the true pose after an unaccountable kidnapping is known as *failure recovery*. This study simulated robot kidnapping by simply moving the robot (from wherever it is) at exactly $t = 5$s back to the origin. The basic particle filter, as described in Table 2, was expected to be unable to recover the true pose and the augmented particle filter, as described in Table 3, was expected to successfully to recover the true pose.

5.3.1 Basic MCL Performance (Baseline)

This section describes how the basic MCL performs after kidnapping at $t = 5$s. Figure 10A shows the true paths (black dashed) and the estimated paths (green solid) taken by the robot. All of the dashed lines return back to the origin for the kidnapping, but the green lines continue drifting outwards due to the particle deprivation near the origin. Figure 10B,C plot the mean position error and

angular error , respectively, versus time. At $t = 5$s, the position error jumps to 10m, but remains flat for the remainder of the simulation. This implies that the basic particle filter was still producing the *best* possible position given the 10m offset introduced by kidnapping. Similarly, the angular error jumps at $t = 5$s to various values > 50deg. Unexpectedly, many of these trails' angular errors returned to zero after 1 second. This implies that the basic particle is able to recover the correct angle but not the correct position after kidnapping.

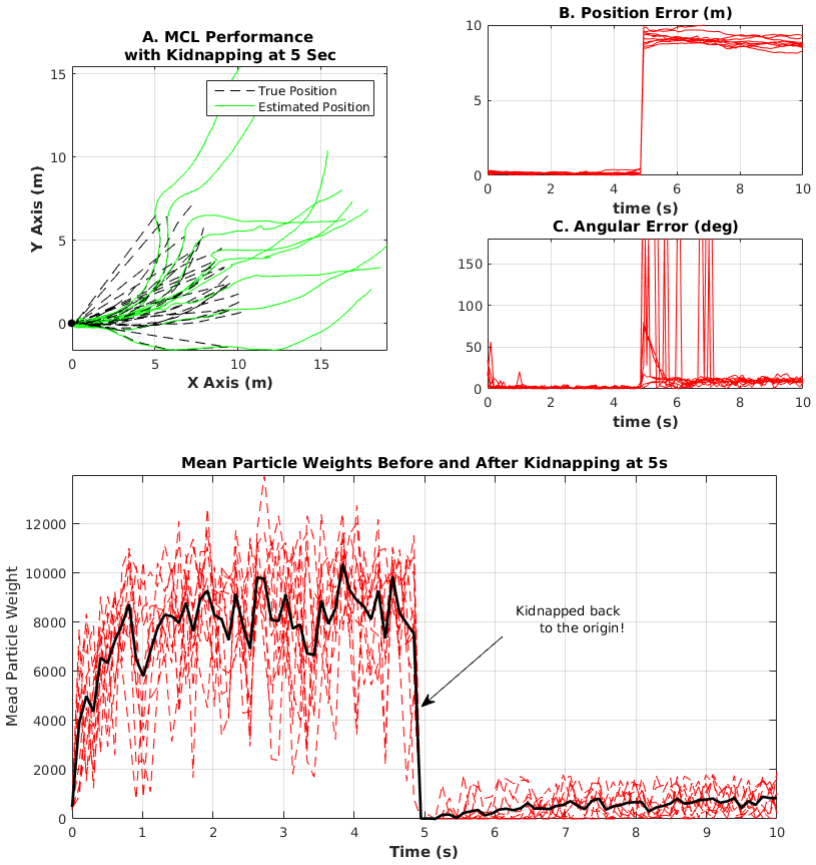

Figure 10: Basic MCL Performance after Kidnapping at $t = 5$s

Figure 10D shows the mean particle weights w_t (across the whole particle set χ_t) for each trial over time. As expected, the values drop precipitously at $t = 5$s close to zero and remains there. This figure, in particular, highlights the insufficient performance of basic MCL against kidnapping.

5.3.2 Augmented MCL Performance (Experimental)

This section describes how the augmented MCL performs after kidnapping at $t = 5$s. As described in Table 3, this particle filter begins replacing in new random particles in volume proportional to the drop in mean particle weight. These random particles are initialized at the measured range from but at a uniformaly random angle around the nearest landmark. In these cases, the nearest landmark was on the positive X axis. In Figure 11A, unlike in Figure 10A, the green lines (position estimates) make their way to the true position within 1 second. The reason the green lines jump to the far to the right before correcting is that the pose estimate is defined as the sample mean of poses over the particle set χ_t. Since the new particles uniformly surround the nearest landmark, the localization estimates will temporarily be *at* the nearest landmark. However, due to importance resampling, the new particles close to the true pose will persist over time, while those far from the true pose will not.

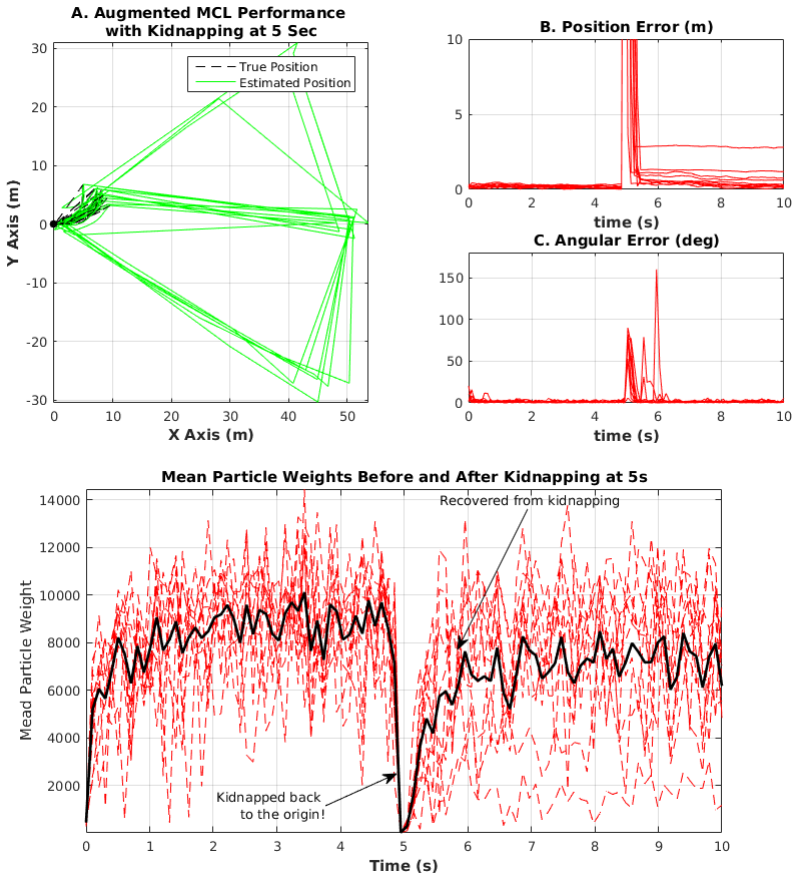

Figure 11: Augmented MCL Performance after Kidnapping at $t = 5$s

Figure 11D shows that the mean particle weights w_t do recover after kidnapping. Hence, the particle deprivation problem is overcome!

6 Conclusion

This study introduced the particle filter solution for planar robot localization using feature based maps (i.e. Monte Carlo Localization). The effort was based entirely on Chapters 4, 5, 6, and 8 of Thrun et al. 2005. The core idea of the particle filter is that posterior densities of state can be represented by a set of discrete random samples of state (i.e. non-parametric representation). The distribution of these samples is intended to match that of true state. This is made possible by having the particles *forward propogate* according to the stochastic motion model and then *re-sampling* these particles according to the stochastic measurement model. These two steps represent the prediction update and measurement correction steps from the generic Bayesian filtering framework.

The educational purpose of this study was to implement an MCL simulation capability with Matlab software, and to comprehensively probe the results. These results included...

- Section 5.2 *Basic MCL*

  How the basic MCL performance responds to wide modifications of various simulation parameters including motion error $\alpha_{1,2,3,4}$, measurement error $\sigma^2_{r,\phi}$, and particle set size N. Overall, the mean and variance of the localization errors, $\mu_{PosErr}, \sigma^2_{PosErr}$ and $\mu_{\theta Err}, \sigma^2_{\theta Err}$, were very low! This is in contrast to th EKF-localization trails (Kamat, 2018) which exhibited far more variance and an initial settling-down phase.

- Sections 5.3 *Augmented MCL*

  How the MCL algorithm can be augmented to overcome particle deprivation after kidnapping. These results indicated that the simple infusion of random particles tuned to the closest landmark can serve as a 'lifeline' when the bulk of the particle set is nowhere near the true location.

A critical reason for the high performance shown here is likely because the distributions which governed the true motion and measurements (in simulation), were *exactly* those which the particle filter applied during forward propogation and importance re-sampling. This puts the robot at an advantage unlikely to occur in practice. Therefore, it would be beneficial to re-run these simulations with particle filters applying distributions different from those applied by the outer simulation. Moreover, it would also be beneficial to re-run these simulations with the assumption of known landmark correspondences lifted.

7 References

1. Kamat. (2018). *Planar Robot Localization with the Extended Kalman Filter (EKF).* An unpublished school report. Available from

   `https://drive.google.com/file/d/1_xD-lSe36zxA_PwHaFMd2DyPvIGQStg5/view`

2. Mathworks. (2011). *Statistics and Machine Learning Toolbox (r2018a).* Retrieved April 15, 2018 from https://www.mathworks.com/help/stats/

3. Sarkka S (2013). *Bayesian Filtering and Smoothing.* Cambridge University Press.

4. Thrun S, Burgard W, Fox D. (2005). *Probabilistic Robotics.* Intelligent Robotics and Autonomous Agents. MIT Press.

8 Appendix: MCL Matlab Code

```
function [ParticleSet, muPose] = MonteCarloLocalization_Optimized(ParticleSet, v, w, Ts,...
                                                                  obsLocal, map, alpha, varObs)
    % Forward Propagate the Whole Particle Set
    numParticles = size(ParticleSet, 2);
    numLM        = size(map, 2);
    ParticleSet = SampleMotionModel(ParticleSet, v, w, Ts, alpha);
    % Iterate over each landmark and compute the weights across particles.
    ParticleWeights = zeros(1, numParticles);
    for iLM = 1:numLM
        % Generate Weights for Each Particle from the Measurements
        ParticleWeights = ParticleWeights + MeasurementModel(ParticleSet, obsLocal(:,iLM),
                                                             map(:,iLM), varObs);
    end
    % Normalize Weights
    ParticleWeights = NormalizeWeights(ParticleWeights);
    % Resample the Particle Set (Post-Measurement Posterior)
    [ParticleSet] = ResampleParticleSet(ParticleSet, ParticleWeights);
    % Estimate Pose as the Center of Mass
    muPose = mean(ParticleSet,2);
    return;
end


%% Normalize Weights
function [weightsNorm] = NormalizeWeights(weightsInput)
    weightsNorm = weightsInput./sum(weightsInput);
    weightsNorm((weightsNorm < 0)|isnan(weightsNorm)) = 0;
    return;
end

%% Generate the Next Robot Pose with the Noisy Velocity Motion Model
function [psOutput] = SampleMotionModel(psInput, v, w, Ts, alpha)
    zeroVec = zeros(1,size(psInput, 2));
    % Generate Noisy Velocity Commands
    vHat   = v + normrnd(zeroVec, alpha(1)*v^2 + alpha(2)*w^2);
    wHat   = w + normrnd(zeroVec, alpha(3)*v^2 + alpha(4)*w^2);
    gamHat = 0 + normrnd(zeroVec, alpha(5)*v^2 + alpha(6)*w^2);
    % Update Pose
    theta = psInput(3,:);
    psOutput = psInput + [-(vHat./wHat).*sin(theta)+(vHat./wHat).*sin(theta+wHat*Ts);
                          (vHat./wHat).*cos(theta)-(vHat./wHat).*cos(theta+wHat*Ts);
                           wHat*Ts + gamHat*Ts];
    psOutput(3,:) = mod(psOutput(3,:), 2*pi);
    return;
```

```
end


%% Generate Weights for Each Particle to a Single Landmark
function [Weights] = MeasurementModel(psInput, obsLocal, lm, varObs)
    % Expected Ranges/Angle from Each Particle to the Given Landmark
    deltaXs = lm(1)-psInput(1,:);
    deltaYs = lm(2)-psInput(2,:);
    ExpRanges = sqrt(deltaXs.^2 + deltaYs.^2);
    ExpAngles = atan2(deltaYs, deltaXs)-psInput(3,:);
    % Compute Weight for Each Particle to this Landmark
    Weights = pdf('Normal', obsLocal(1), ExpRanges, varObs(1)).*pdf('Normal',
                                                    obsLocal(2), ExpAngles,varObs(2));
    return;
end

%% Resample the Particle Set (Post-Measurement Posterior)
function [ResampledPS] = ResampleParticleSet(ParticleSet, ParticleWeights)
    numParticles = size(ParticleSet, 2);
    indeces = 1:numParticles;
    s = RandStream('mlfg6331_64');
    try
        iResampled = randsample(indeces, numParticles, true, ParticleWeights);
    catch ME
        rethrow(ME);
    end
    ResampledPS = ParticleSet(:,iResampled);
    return;
end
```