Adapting Proposal Distributions for Accurate, Efficient Mobile Robot Localization

Patrick Beeson, Aniket Murarka, and Benjamin Kuipers
Department of Computer Sciences
The University of Texas at Austin
Email: {pbeeson, aniket, kuipers}@cs.utexas.edu

***Abstract*— When performing probabilistic localization using a particle filter, a robot must have a good proposal distribution in which to distribute its particles. Once weighted by their normalized likelihood scores, these particles estimate a posterior distribution over the possible poses of the robot.**

This paper 1) introduces a new action model (the system of equations used to determine the proposal distribution at each time step) that can run on any differential drive robot, even from log file data, 2) investigates the results of different algorithms that modify the proposal distribution at each time step in order to obtain more accurate localization, 3) investigates the results of incrementally adapting the action model parameters based on recent localization results in order to obtain proposal distributions that better approximate the true posteriors.

The results show that by adapting the action model over time and, when necessary, modifying the resulting proposal distributions at each time step, localization improves—the maximum likelihood score increases and, when possible, the percentage of wasted particles decreases.

I. Introduction

In mobile robotics research, progress has been made in many areas of spatial modeling; however, most research has been in the area of precise incremental localization with respect to a Cartesian frame of reference. For many researchers, this is the preferred method of robot localization to perform both *global localization* (when the robot is lost in a known environment [1]) and *local localization* (where the robot incrementally tracks small pose changes to remain localized—necessary when performing *simultaneous localization and mapping* (SLAM) [2]).

The most commonly used paradigm for incremental localization is probabilistic localization [3]. Here, the robot has some distribution of belief about its pose $\mathbf{x} = (x, y, \theta)$. After performing an action $\mathbf{a}$ and making an observation $\mathbf{o}$, it must then determine its new pose estimate $\mathbf{x}'$.

$$p(\mathbf{x}'|\mathbf{o}, \mathbf{x}, \mathbf{a}, \mathsf{m}) \propto p(\mathbf{o}|\mathbf{x}', \mathsf{m}) \cdot p(\mathbf{x}'|\mathbf{x}, \mathbf{a}) \quad (1)$$

$$posterior \propto likelihood \cdot proposal \quad (2)$$

The proposal distribution $p(\mathbf{x}'|\mathbf{x}, \mathbf{a})$ is determined by an *action model*. The action model is a system of equations that defines a probability distribution over resulting states $\mathbf{x}'$, given an initial state $\mathbf{x}$ and an action $\mathbf{a}$. One of the contributions of this paper is to introduce a new action model that models non-trivial uncertainty, even when using data from log files.

Using the observation the robot has at its current location, $\mathbf{o}$, and a map of the world, m (either computed *a priori* or via SLAM), the robot can create a likelihood distribution—a distribution over possible robot poses based on an observation model. Combining the likelihood and proposal distributions yields a posterior distribution that represents the actual uncertainty distribution of the robot's pose in the world.

The *particle filter* is a well-known method that efficiently approximates these distributions so that pose uncertainty can be quickly calculated by a robot moving through an environment. Particles are selected from the posterior distribution from the previous time step. The action model, then "moves" each particle to create the proposal distribution. Proposal particles are weighted according to their likelihood scores and normalized to sum to one. These weighted particles represent the new posterior distribution over poses.

Many action models are hand tuned to generate proposal distributions that overestimate the posterior. While this generally ensures high accuracy localization (the likelihood distribution definitely intersects the proposal), many particles end up with negligible likelihood scores. By tuning an action model, one can reduce these *wasted particles*. However, if improperly tuned, proposals can become too small. Underestimating proposals can cause inaccurate localization, which is a far more serious problem than having wasted particles.

This paper investigates two orthogonal methods for adapting the proposal in order to achieve better localization. The first method adapts the proposal at each time step. The idea is to change each proposal distribution based on the likelihood scores generated by an initial set of particles. This allows a proposal to grow or shrink in order to place its remaining particles in a region that will lead to higher *accuracy* localization.

The second method adapts the proposals at a slower rate. The idea is to change the parameters of the action model based on localization information from the recent past. This changes future proposal distributions, leading to proposals that better approximate the posteriors. This allows for fewer wasted particles, termed *efficiency* in this paper.

Together these methods lead to better localization by improving localization accuracy while attempting to decrease wasted particles. In other words, by using the adaptive methods investigated in this paper, the proposal distributions become better approximations of their respective posterior distributions.

II. Action Models

An action model is important as it determines the location, shape, and span of each proposal distribution. There has been previous research into different differential drive action models and into how to calibrate them.

Some research utilizes specific, contrived environments and constrained motion in order to disambiguate translational error from rotational error [4], [5]. In addition to constraining the environment and the motion taken, these methods are run offline, using the final localization error to find the parameters of a static action model. If the underlying source of error changes (e.g. motors wear out, new equipment changes the load, or the travel surface changes), the researcher must run these experiments again to determine new model parameters.

There has also been work tuning action model parameters in unconstrained environments. By using existing localization techniques to provide a *ground-truth* value for the pose, the robot can measure the error between its odometry readings and ground-truth. These errors may have random and systematic components. The error measurements are then used to calibrate the parameters of the action model.

Below, we discuss two approaches to action model tuning in unconstrained environments. The notation we use is as follows: if z is a scalar that represents an error-prone measured value, then $\hat{z}$ represents an estimate of the true value, and $\tilde{z}$ represents a distribution of uncertainty.

Roy and Thrun [6] use online localization techniques to calibrate the parameters of a sub-Cartesian action model. A sub-Cartesian action model uses the change in wheel shaft encoder counts between time $k-1$ and time k to calculate the incremental translational motion s_k and rotational motion ϕ_k. These values are used to estimate a change in Cartesian space (x, y, θ) [7].

In their model, Roy and Thrun state that odometry errors in both translation and rotation are linearly dependent on the distance traveled, $|s_k|$.

$$\hat{s}_k = s_k + |s_k| \cdot c_0^k \quad (3)$$

$$\hat{\phi}_k = \phi_k + |s_k| \cdot c_1^k \quad (4)$$

The estimation of $\hat{s}_k$ and $\hat{\phi}_k$ is done by tuning the linear coefficients c_0 and c_1. c_0^k and c_1^k are calculated by first finding the best linear coefficients for the current localization step, c_0' and c_1', and then performing a weighted average with the previous values of the parameters, c_0^{k-1} and c_1^{k-1}. This weighted average keeps the values of c_0 and c_1 from having large fluctuations.

$$c_0^k = \nu \cdot c_0^{k-1} + (1-\nu) \cdot c_0' \quad (5)$$

$$c_1^k = \nu \cdot c_1^{k-1} + (1-\nu) \cdot c_1' \quad (6)$$

Every pair of $\hat{s}_k, \hat{\phi}_k$ values corresponds to a change in the estimated pose $(\hat{x}, \hat{y}, \hat{\theta})$. The most common way to calculate the pose is to assume that both the translation and rotation maintained constant velocity across the measured interval. This arc is closely approximated by assuming the robot sequentially performs half of its rotation, all of its translation, then the other half of its rotation [7].

$$\hat{x}_k = \hat{x}_{k-1} + \hat{s}_k \cdot cos(\hat{\theta}_{k-1} + \frac{\hat{\phi}_k}{2}) \quad (7)$$

$$\hat{y}_k = \hat{y}_{k-1} + \hat{s}_k \cdot sin(\hat{\theta}_{k-1} + \frac{\hat{\phi}_k}{2}) \quad (8)$$

$$\hat{\theta}_k = \hat{\theta}_{k-1} + \hat{\phi}_k \quad (9)$$

The likelihood score of a pose can be evaluated based on how well the robot's current laser scan, if taken from that pose, matches the occupancy grid. The pose that yields the best match is seen as the ground-truth.

Roy and Thrun determine c_0' and c_1' by searching over possible values of these parameters. This search is essentially the same as sampling from a known proposal distribution. This proposal has a mean at c_0^{k-1}, c_1^{k-1}, but the size and shape of the proposal is not documented—although it is implied to be static. However this search region is defined, it needs to be general enough to find a useful *ground-truth* pose.

Eliazar and Parr [8] make some nice extensions to the model above. The actual implementation in Eliazar and Parr's publication uses a *Rao-Blackwellized particle filter*, but the concept can work with the simpler, single-map particle filter implementation described earlier.

First, Eliazar and Parr add another dimension of error that is normal to the direction of travel. This *drift* is represented by $\tilde{\delta}_k$. Secondly, instead of estimating scalar values of $\hat{s}_k$ and $\hat{\phi}_k$ (estimating only the mode of the uncertainty), this model tries to estimate the full distribution of error. They do this by assuming that $\tilde{s}_k$, $\tilde{\delta}_k$, and $\tilde{\phi}_k$ are Gaussians. This makes the change in Cartesian pose a distribution as well.

$$\tilde{x}_k = \tilde{x}_{k-1} + \tilde{s}_k \cdot cos(\tilde{\theta}_{k-1} + \frac{\tilde{\phi}_k}{2}) + \tilde{\delta}_k \cdot cos(\tilde{\theta}_{k-1} + \frac{\tilde{\phi}_k + \pi}{2}) \quad (10)$$

$$\tilde{y}_k = \tilde{y}_{k-1} + \tilde{s}_k \cdot sin(\tilde{\theta}_{k-1} + \frac{\tilde{\phi}_k}{2}) + \tilde{\delta}_k \cdot sin(\tilde{\theta}_{k-1} + \frac{\tilde{\phi}_k + \pi}{2}) \quad (11)$$

$$\tilde{\theta}_k = \tilde{\theta}_{k-1} + \tilde{\phi}_k \quad (12)$$

Additionally, the Gaussians $\tilde{s}_k$, $\tilde{\delta}_k$, and $\tilde{\phi}_k$ are dependent on both s_k and ϕ_k, generalizing from the more restrictive model above. The problem now becomes how to determine the correct parameters, $c_0 \ldots c_{11}$, that define these Gaussians.

$$\tilde{s}_k = \mathcal{N}(s_k \cdot c_0 + \phi_k \cdot c_1,\ {s_k}^2 \cdot c_2 + {\phi_k}^2 \cdot c_3) \quad (13)$$

$$\tilde{\delta}_k = \mathcal{N}(s_k \cdot c_4 + \phi_k \cdot c_5,\ {s_k}^2 \cdot c_6 + {\phi_k}^2 \cdot c_7) \quad (14)$$

$$\tilde{\phi}_k = \mathcal{N}(s_k \cdot c_8 + \phi_k \cdot c_9,\ {s_k}^2 \cdot c_{10} + {\phi_k}^2 \cdot c_{11}) \quad (15)$$

Due to the large number of parameters, the authors designed the algorithm to solve for six linear least squared systems. One system solves for the mean parameters of $\tilde{s}$ (c_0 and c_1). Using these values, another system solves for the variance parameters of $\tilde{s}$ (c_2 and c_3). Similarly, the other four linear systems solve for the parameters of $\tilde{\delta}$ and $\tilde{\phi}$ [8].

These parameters are found offline, after the data has been gathered; thus, as in Roy and Thrun's model, the search regions used during the data collection must be large enough to find some notion of ground-truth. Unlike Roy and Thrun's model, this model uses not only the best particle (the mode) from each time step but uses the weights of all the particles (from the likelihood scores) to help solve the least squared systems.

III. A New Action Model

Both of the action models detailed above rely on having sub-Cartesian odometry information (s, ϕ). However, many localization implementations may not have access to this information. Pre-compiled odometry servers, robot simulators, and shared log files [9] usually only provide Cartesian poses—(x, y, θ). This can be a problem for some researchers wanting to utilize the above models since there are many values of $\Delta(x, y, \theta)_k$ that yield no solution for s_k, ϕ_k (see Equations 7-9).

To overcome this problem, we use the approach proposed by Rekleitis [5]. Similar to the way Equations 7-9 approximate the constant velocity arc taken by the robot with a "turn-travel-turn" approach [7], we can break any change in pose into "turn-travel-turn" components.

$$\alpha_k = \arctan(\frac{\Delta y_k}{\Delta x_k}) - \theta_{k-1}; \quad \alpha_k \in [-\pi, \pi] \tag{16}$$

$$\rho_k = \sqrt{\Delta {x_k}^2 + \Delta {y_k}^2} \tag{17}$$

$$\beta_k = \Delta\theta_k - \alpha_k; \quad \beta_k \in [-\pi, \pi] \tag{18}$$

When sitting still ($\rho_k = 0$), then $\alpha_k = 0$ and $\beta_k = \Delta\theta_k$. Because robots do not always translate forward, but can move backwards, we also consider the case where $\alpha'_k = \alpha_k + \pi$, $\alpha'_k \in [-\pi, \pi]$, $\rho'_k = -\rho_k$, $\beta'_k = \beta_k + \pi$, $\beta'_k \in [-\pi, \pi]$. We only use these values when $|\alpha'_k| + |\beta'_k| < |\alpha_k| + |\beta_k|$.

Using α, ρ, and β, we can extend Eliazar and Parr's action model.

$$\begin{aligned}\tilde{s}_k = {} & \mathcal{N}(\alpha_k \cdot c_0 + \rho_k \cdot c_1 + \beta_k \cdot c_2, \\ & c_3 + {\alpha_k}^2 \cdot c_4 + {\rho_k}^2 \cdot c_5 + {\beta_k}^2 \cdot c_6)\end{aligned} \tag{19}$$

$$\begin{aligned}\tilde{\delta}_k = {} & \mathcal{N}(\alpha_k \cdot c_7 + \rho_k \cdot c_8 + \beta_k \cdot c_9, \\ & c_{10} + {\alpha_k}^2 \cdot c_{11} + {\rho_k}^2 \cdot c_{12} + {\beta_k}^2 \cdot c_{13})\end{aligned} \tag{20}$$

$$\begin{aligned}\tilde{\phi}_k = {} & \mathcal{N}(\alpha_k \cdot c_{14} + \rho_k \cdot c_{15} + \beta_k \cdot c_{16}, \\ & c_{17} + {\alpha_k}^2 \cdot c_{18} + {\rho_k}^2 \cdot c_{19} + {\beta_k}^2 \cdot c_{20})\end{aligned} \tag{21}$$

$$\begin{aligned}\tilde{x}_k = {} & \tilde{x}_{k-1} + \tilde{s}_k \cdot cos(\tilde{\theta}_{k-1} + \alpha_k) + \\ & \tilde{\delta}_k \cdot cos(\tilde{\theta}_{k-1} + \alpha_k + \frac{\pi}{2})\end{aligned} \tag{22}$$

$$\begin{aligned}y_k = {} & y_{k-1} + \tilde{s}_k \cdot sin(\tilde{\theta}_{k-1} + \alpha_k) + \\ & \tilde{\delta}_k \cdot sin(\tilde{\theta}_{k-1} + \alpha_k + \frac{\pi}{2})\end{aligned} \tag{23}$$

$$\tilde{\theta}_k = \tilde{\theta}_{k-1} + \tilde{\phi}_k \tag{24}$$

Notice, that we added a constant parameter in the variance formulas. There are a couple reasons for this. Any "constant" localization error due to the discretization of the map model (e.g. occupancy grids [10]) can be handled. Additionally, for testing purposes, it allows us a convenient way to create large, overestimating models or small, underestimating models by simply setting all uncertainty to be constant.

This is the action model we use below to compare changes to the straightforward particle filter localization algorithm.

IV. Building an Accurate Localization Algorithm

This section describes the different algorithms we utilize for improving localization accuracy. We then run experiments to show the validity of these algorithms under different conditions.

A. KLD-sampling

In recent work, Fox provides a method by which a robot that uses a particle filter to localize can be confident it has sufficiently sampled the proposal distribution [11]. We utilize this method in all of the tested algorithms below for two reasons. First, it provides an upper bound on the number of samples needed for a given proposal distribution. This eliminates many redundant particles when the proposal distribution is small. Second, it also provides a lower bound on the number of particles. This ensures that large proposals are adequately sampled, which permits a fair comparison of localization accuracy between various localization algorithms.

B. Shrinking the proposal

Suppose a proposal distribution overestimates the variance of the actual posterior distribution. Is there an intelligent way to shrink the proposal in order to use a small number of additional particles to obtain better localization? We incorporate a method by Grisetti, Stachniss, and Burgard [12] that shrinks the proposal distribution in such cases, therefore allowing more accurate localization.

Grisetti et al. observe that localization with laser range finders usually produces likelihood distributions that are much more highly peaked (smaller variance) than most proposal distributions. Given this assumption, it is possible to treat the proposal distribution as being constant under the likelihood distribution. Given a constant proposal, the posterior distribution is equal to the likelihood distribution. Thus, a new, better proposal would just be the estimated likelihood distribution. To obtain a closed form for the new proposal, the distribution around the likelihood peak is approximated with a Gaussian.

In our implementation the mean and covariance of the Gaussian are estimated by the weighted mean and covariance of the particles $\{\mathbf{x'}_j\}$ sampled from the original proposal and weighted by their normalized likelihoods. The equations are as follows [12]:

$$\mu = \frac{1}{\eta} \sum_j \mathbf{x'}_j \; p(\mathbf{o}|\mathbf{x'}_j, \mathsf{m}) \tag{25}$$

$$\Sigma = \frac{1}{\eta} \sum_j p(\mathbf{o}|\mathbf{x'}_j, \mathsf{m}) \; (\mathbf{x'}_j - \mu) \; (\mathbf{x'}_j - \mu)^T, \tag{26}$$

where $\eta = \sum_j p(\mathbf{o}|\mathbf{x'}_j, \mathsf{m})$.

For this new proposal, KLD-sampling should require far fewer particles than in the original proposal due to the smaller

variance. In the experiments below, we refer to this algorithm as shrink.

C. Growing the proposal

Suppose that a proposal underestimates the actual posterior distribution. If the maximum likelihood particle in the proposal has a high score, then localization will be largely unaffected. The more serious case occurs if all sampled particles have low likelihood scores, i.e. the proposal is not large enough to intersect the peak of the likelihood distribution. A similar scenario occurs when the proposal's mode is displaced from the actual posterior (e.g. wheel slippage causes the action model to create a proposal far from the robot's actual pose). Is there an intelligent way to grow the proposal in order to "locate" an area of high likelihood?

If all particles in a proposal distribution have low likelihood scores, there is no signal that tells us exactly how to grow the proposal to search for better likelihood scores. By *likelihood scores*, we mean the unnormalized values of $p(\mathbf{o}|\mathbf{x'}_j, \mathsf{m})$.

To handle such cases, we set a lower threshold on the maximum likelihood score. In the event that all likelihood scores fall below this threshold, a new proposal is created that doubles the standard deviation, σ, of all three dimensions, s, δ, ϕ, of the current proposal.

In our current implementation, we perform this doubling up to five times if necessary. The threshold is annealed to become more generous each time the proposal grows.

In the experiments below, we refer to this algorithm as grow. The proposal does not always grow, only when required, due to a lack of accurate localization using the current proposal.

D. Shrinking and Growing

The shrink and grow algorithms above should be considered mutually exclusive improvements to the vanilla particle filter algorithm, which uses a single, static proposal distribution.

In the experiments below, we also consider the shrink-and-grow algorithm that performs shrink on the original proposal, followed by grow if it is applicable (growing the original proposal, not the "smaller" proposal created by shrink). The shrink algorithm is also performed on any new proposal generated by the grow algorithm. This allows a proposal to grow until it intersects the likelihood distribution, then shrink down to fit the likelihood with a highly peaked Gaussian.

E. Experimental Setup

We want to see the influence the above three algorithms have on the accuracy of the particle filter. Here we describe our experimental setup.

First, the robot explores an environment to create a sensor log. The robot records an alternating sequence of time labeled odometry and range observations. A SICK LMS 200 laser range finder is used as the range sensing device. The log file used here has 3961 odometry readings (and 3961 laser readings) from a nine minute run. 418 time steps involve no motion by the robot.

The log file is then used to create a global metrical map via offline methods [13]. The global metrical map provides *ground-truth* of the actual location of the robot at every laser reading. The global map is represented as an occupancy grid that has 5 cm cells. For each localization algorithm, KLD-sampling is used to determine the number of particles each proposal needs.

We then ran localization experiments using three different sets of action model parameters. The parameter values of these three models are shown in Table I.

The large action model has a constant sized position and heading uncertainty that is centered on zero odometry error. The parameters, chosen beforehand, were selected in an attempt to overestimate the uncertainty at almost every time step.

The small action model also has constant uncertainty centered on the odometry estimate. These parameters were chosen to underestimate the posterior for a majority of the sensor trace—when the robot is making small movements, this model is sufficient for good localization.

The final action model is a least squares fitted model. These parameters were calculated by finding the error in s, δ, and ϕ between odometry and *ground-truth* (from the global map) and solving a series of 6 least squares systems to fit Equations 19-21 (explained earlier in the discussion of Eliazar and Parr's model [8]).

We tested 5 localization algorithms on these three models: raw odometry, vanilla particle filter, shrink, grow, and shrink-and-grow. For each time step in the log file, the robot is placed at *ground-truth*. Each of the 5 algorithms is run for a single localization step, and the localization results are recorded. Comparing localization algorithms without ground-truth is tricky, as one bad localization step due to unlucky sampling can affect all future localization. By restarting localization at *ground-truth* at each step, we can compare algorithms directly, eliminating the problem of cumulative error. At each step, the best (unnormalized) likelihood *score* of each algorithm is recorded: our *accuracy* metric.

The 5 localization algorithms, each run on the three models, gives us 15 datasets (each with 3960 localization steps) to analyze (L_1-L_5, S_1-S_5, F_1-F_5 in Table II). If the 5 localization algorithms were run separately, these datasets would be independent, but to save time, we ran all 5 localization algorithms at the same time: we ran shrink-and-grow and recorded the data for the other 4 algorithms. This means some of the algorithms had proposals that were supersets of others (e.g. L_5 had proposals that were supersets of the proposals in L_1-L_4). To be rigorous in our comparison, we created another 15 datasets from a different run (i.e. different sampled particles due to different random number generation). We compare these 15 datasets (L_6-L_{10}, S_6-S_{10}, F_6-F_{10}) with their counterparts.

F. Experimental Results

The complete results of our analysis are shown in Table III; however, for this paper we will only be focusing on

	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_{17}	c_{18}	c_{19}	c_{20}
large	0	1	0	1e-2	0	0	0	0	0	0	1e-2	0	0	0	1	0	1	3e-2	0	0	0
small	0	1	0	1e-6	0	0	0	0	0	0	1e-6	0	0	0	1	0	1	3e-6	0	0	0
least-squares fitted	-0.012	0.99	-0.012	2.6e-05	0	0.0052	0	0.0014	-0.0016	0.0019	1.4e-05	0	0.0012	0	0.98	-0.0048	0.98	4.1e-05	0	0.0093	0

TABLE I. *Action model parameters.* large is an overestimating model. small is an underestimating model. least squares fitted is a model fit using the *ground-truth* posterior distributions over the entire data trace.

$alg.\backslash^{model}$	large static run1	large static run2	small static run1	small static run2	LS fitted static run1	LS fitted static run2	large dynamic	small dynamic	LS fitted dynamic
odometry	L_1	L_6	S_1	S_6	F_1	F_6	LD_1	SD_1	FD_1
vanilla PF	L_2	L_7	S_2	S_7	F_2	F_7	LD_2	SD_2	FD_2
shrink	L_3	L_8	S_3	S_8	F_3	F_8	LD_3	SD_3	FD_3
grow	L_4	L_9	S_4	S_9	F_4	F_9	LD_4	SD_4	FD_4
shrink & grow	L_5	L_{10}	S_5	S_{10}	F_5	F_{10}	LD_5	SD_5	FD_5

TABLE II. *Dataset labels.* Left of double bar are labels for datasets used in Section IV. Right of double bar are labels for datasets used in Section V.

	L_6	L_7	L_8	L_9	L_{10}
L_1	$\times$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
L_2	$\leftarrow$	$\times$	$\uparrow$	$\uparrow$	$\uparrow$
L_3	$\leftarrow$	$\leftarrow$	$\times$	$\leftarrow$	$\uparrow$
L_4	$\leftarrow$	$\leftarrow$	$\uparrow$	$\times$	$\uparrow$
L_5	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\times$

	S_6	S_7	S_8	S_9	S_{10}
S_1	$\times$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
S_2	$\leftarrow$	$\times$	$\uparrow$	$\uparrow$	$\uparrow$
S_3	$\leftarrow$	$\leftarrow$	$\times$	$\uparrow$	$\uparrow$
S_4	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\times$	$\uparrow$
S_5	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\times$

	F_6	F_7	F_8	F_9	F_{10}
F_1	$\times$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
F_2	$\leftarrow$	$\times$	$\uparrow$	$\uparrow$	$\uparrow$
F_3	$\leftarrow$	$\leftarrow$	$\times$	$\times$	$\uparrow$
F_4	$\leftarrow$	$\leftarrow$	$\times$	$\times$	$\uparrow$
F_5	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\times$

TABLE III. *Localization accuracy of 5 different algorithms.* Paired t-test comparison of maximum likelihood score (accuracy) between different localization algorithms. Any algorithm that outperforms another with a statistical confidence of at least 99% ($p \leq 0.01$) should be pointed to by an arrow. "$\times$" means that $p > 0.01$.

certain aspects of the table. For example, the "raw odometry" localization algorithm can be ignored as it is not very accurate.

The table shows (via arrows) the algorithm that performed better: higher maximum likelihood scores. An algorithm is "better" with a 99% confidence ($p \leq 0.01$) using the standard paired t-test. "$\times$" represents little or no difference between algorithms for a particular action model ($p > 0.01$). We expected no significant difference between results collected from running the same algorithm on the same model (the diagonals).

Note that the shrink algorithm outperforms the vanilla particle filter in all cases: L_3 over L_7, L_8 over L_2, S_3 over S_7, S_8 over S_2, F_3 over F_7, F_8 over F_2. It also outperforms grow on the overestimating action model (L_3 over L_9, L_8 over L_4). This result is expected due to a more precise search space of the adapted proposal.

The grow algorithm also improves localization accuracy over the vanilla particle filter for every action model. As expected, it outperforms shrink on the underestimating model (S_4 over S_8, S_9 over S_3).

Finally, shrink-and-grow achieves better accuracy than any other algorithm.

V. Building an Efficient Localization Model

Overestimating models can provide quite good localization accuracy. We observe this by comparing localization accuracy across models in the previous section: L_5 is as good as F_{10} (i.e. $p > 0.01$), and L_{10} is as good as F_5. However, if the action model is always producing overestimating proposals, then many particles are wasted. By tuning the action model parameters online, the proposals should become better approximations of the posterior distributions. Thus, we expect to get more efficiency without losing accuracy.

A. Online Action Model Tuning

The previous section investigates methods for shrinking and growing the proposal distribution at each time step for accurate localization. Here we investigate a method that performs online action model tuning on a larger time scale. This process determines updated values for the parameters $c_0 \ldots c_{20}$ in Equations 19-21. By performing online parameter tuning, an action model can be customized to produce proposal distributions that closely resemble the posteriors. Additionally, online tuning of parameters should make localization robust to changes in the environment or to robot characteristics as the robot moves through the world.

We tune our parameters, $c_0 \ldots c_{20}$, using a series of least squares systems (described in Eliazar and Parr's work [8] and used for the least squares fitted action model). There are some additional issues that must be addressed in order to perform online parameter tuning.

1) Accumulating enough data: When tuning offline, all the data has been accumulated, and the linear systems are over-constrained. When tuning online, we must make sure to have enough data (α_k, ρ_k, β_k, s_k, δ_k, ϕ_k) to constrain the linear systems. The systems become constrained given four different values of α_k, ρ_k, and β_k; however, we have observed that more data than this is needed.

Because the lasers are updated at about 8 Hz, the robot can accumulate enough data to constrain the linear systems in about 0.5 seconds; however, the robot cannot move very far in this amount of time. We have found that if we tune the parameters this frequently, we can get very large parameter values due to the small, noisy values of α_k, ρ_k, β_k and s, δ, ϕ.

For the experiments below, we wait to tune the linear systems until the robot's cumulative displacement is at least 1 meter and its cumulative rotation is at least 10° from the last

tuning event. These two constants were chosen arbitrarily, and have not been validated against other possible values.

2) Estimating action and error values: Because shrink and shrink-and-grow end up sampling directly from a proposal (rather than sampling from the previous posterior distribution and moving the particles according to the action model), these algorithms do not have s, δ, ϕ values for their final posterior particles. To obtain these values, which are necessary to solve the least squared systems, we draw two samples: one sample from the previous posterior distribution (in our experimental platform this is simply the *ground-truth* from the previous time step) and one sample from the new posterior distribution. We can then use the action $\alpha_k, \rho_k, \beta_k$ and the two samples to find the s, δ, ϕ displacement for the new posterior particle. Those values are used in the linear least squares systems.

3) Using multiple samples per time step: Unlike Roy and Thrun [6], we do not simply want to tune our model using the maximum likelihood particles from the recent past. Instead, we would like to use multiple particles from each time step, as per Eliazar and Parr's calibration technique [8]. This forces the least squares systems to consider all the uncertainty from previous time steps, not just the modes.

To do this we simply perform N iterations of the above calculations to determine s, δ, ϕ values for N particles at each time step. In our current experiments, we set $N = 100$. Unlike Eliazar and Parr, because we are drawing these particles from the posterior, we do not use likelihoods as weights in the least squares systems.

4) Throttling the change in parameters: As noted in Roy and Thrun's paper, if the new parameters are simply plugged into the action model, the fluctuations of the mean and variance of proposals will be large. To restrain the change in values, we also update the parameters using a weighted average.

For each parameter, c_i, the new estimated value is combined with the previous value to get an updated value: $c_i^l = \nu \cdot c_o^{l-1} + (1-\nu) \cdot c_i'$. Here l occurs at a different timescale than k, which has been used throughout this paper.

Through experimentation, we found that $\nu = 0.1$ worked the best. Larger values were to restrictive, yielding parameter changes that were to slow given our limited sensor trace.

B. Experimental Setup

We want to compare the effects of dynamic action models, which change their parameters online, versus the static action models demonstrated in the previous section. To do this, we created 15 new datasets, running the 5 localization algorithms on the 3 action models above. The difference here is that these action models start with the parameters given in Table I, but are allowed to change. These datasets are shown in the rightmost portion of Table II: $LD_1, \ldots, FD_5$.

When gathering datasets, we recorded the mean likelihood score per particle, or average score, in addition to the maximum score. This average is calculated using all particles sampled in a localization step (i.e. all particles from the original proposal and all particles from any adaptive proposals are used). We will use this average as an *efficiency* metric. If the average is low, then many of the sampled particles were unnecessary due to low likelihood scores.

C. Experimental Results

The complete results of this experiment are shown in Tables IV and V. We are only concerned with certain aspects of these tables (e.g. the raw odometry always has a high average likelihood score since there is only 1 particle). Again, the arrows represent a significance of 99% ($p \leq 0.01$).

In the previous section, we compared the different localization algorithms against each other. There, the tables are easiest to read in terms of which algorithm outperformed another, and it was clear that shrink-and-grow was the most best algorithm for accuracy. In these tables, it is much easier to look at which action models benefitted from online parameter tuning (i.e. the diagonals).

For the large action model, we clearly see that using dynamic action model parameters results in more effective proposal distributions: less wasted particles. All algorithms (except for the single pose raw odometry) are more efficient when the action model is allowed to shrink down from an overestimating model (Table V, left). The shrink algorithm does lose accuracy (LD_3 vs. $L_{3,8}$); however, this is expected since the overestimating proposals cover a larger search area than the smaller proposals of the dynamic action model. Most important, shrink-and-grow remains accurate despite dynamically changing the parameters to gain efficiency.

The results presented in Tables III-V demonstrate that when starting with an overestimating action model, the overall best algorithm to use is shrink-and-grow with online action model tuning enabled.

For the small action model, we clearly see that allowing dynamic action model parameters decreases efficiency. This is expected, as larger proposals will undoubtedly mean some particles end up with low likelihood scores. However, this decrease in efficiency is overshadowed by an all around increase in accuracy (the diagonals all point to $SD_2 - SD_5$). When shrink is run with a dynamic action model, it becomes more accurate than both grow and shrink-and-grow run with static action models (SD_3 over both $S_{4,9}$ and $S_{5,10}$). Again shrink-and-grow improves its accuracy when the action model parameters are allowed to change online.

For localization, we prefer accuracy over efficiency (especially when the proposal is small enough to estimate using relatively few particles); thus, when starting with an underestimating action model, the overall best algorithm to use is again shrink-and-grow with online action model tuning enabled.

For the least squares fitted action model, we see that the vanilla particle filter and the grow algorithm perform worse with respect to accuracy when the action model parameters are allowed to change, although grow gains efficiency. We interpret these results to mean the initial parameters must have been slightly overestimating the best parameters in at least one dimension. Most important, there is no negative effect from the dynamic action model on the accuracy or efficiency of shrink-and-grow.

	$L_{1,6}$	$L_{2,7}$	$L_{3,8}$	$L_{4,9}$	$L_{5,10}$
LD_1	$\times$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
LD_2	$\leftarrow$	$\leftarrow$	$\uparrow$	$\times$	$\uparrow$
LD_3	$\leftarrow$	$\leftarrow$	$\uparrow$	$\leftarrow$	$\uparrow$
LD_4	$\leftarrow$	$\leftarrow$	$\uparrow$	$\leftarrow$	$\uparrow$
LD_5	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\times$

	$S_{1,6}$	$S_{2,7}$	$S_{3,8}$	$S_{4,9}$	$S_{5,10}$
SD_1	$\times$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
SD_2	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\uparrow$	$\uparrow$
SD_3	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$
SD_4	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\times$
SD_5	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$

	$F_{1,6}$	$F_{2,7}$	$F_{3,8}$	$F_{4,9}$	$F_{5,10}$
FD_1	$\times$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
FD_2	$\leftarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
FD_3	$\leftarrow$	$\leftarrow$	$\times$	$\times$	$\uparrow$
FD_4	$\leftarrow$	$\leftarrow$	$\uparrow$	$\uparrow$	$\uparrow$
FD_5	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\times$

TABLE IV. *Localization accuracy of dynamic vs. static action models.* Paired t-test comparison of maximum likelihood scores between dynamic vs. static action model tuning. The significance results are the same when comparing $LD_1 - LD_5$ to either datasets $L_1 - L_5$ or datasets $L_6 - L_{10}$. Any algorithm that outperforms another with a statistical confidence of at least 99% ($p \leq 0.01$) should be pointed to by an arrow. "$\times$" means that $p > 0.01$.

	$L_{1,6}$	$L_{2,7}$	$L_{3,8}$	$L_{4,9}$	$L_{5,10}$
LD_1	$\times$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$
LD_2	$\uparrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$
LD_3	$\uparrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$
LD_4	$\uparrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$
LD_5	$\uparrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$

	$S_{1,6}$	$S_{2,7}$	$S_{3,8}$	$S_{4,9}$	$S_{5,10}$
SD_1	$\times$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$
SD_2	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
SD_3	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
SD_4	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
SD_5	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$

	$F_{1,6}$	$F_{2,7}$	$F_{3,8}$	$F_{4,9}$	$F_{5,10}$
FD_1	$\times$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$
FD_2	$\uparrow$	$\times$	$\uparrow$	$\leftarrow$	$\uparrow$
FD_3	$\uparrow$	$\leftarrow$	$\times$	$\leftarrow$	$\leftarrow$
FD_4	$\uparrow$	$\uparrow$	$\uparrow$	$\leftarrow$	$\uparrow$
FD_5	$\uparrow$	$\leftarrow$	$\uparrow$	$\leftarrow$	$\times$

TABLE V. *Localization efficiency of dynamic vs. static action models.* Paired t-test comparison of average likelihood scores between dynamic vs. static action model tuning.

If the initial action model is already quite good, Tables III-V demonstrate that shrink-and-grow should be used for improved accuracy. Online tuning has no significant effects on accuracy or efficiency when using shrink-and-grow and an action model pre-tuned for a specific environment.

VI. Conclusion

In this paper, we provided a new action model that takes any change in x, y, θ and creates a non-trivial proposal distribution with Gaussians along the direction of travel, normal to the direction of travel, and in the orientation dimension.

We then used this action model to test the effects of different localization algorithms on localization accuracy. The most accurate algorithm was an algorithm introduced in this paper, shrink-and-grow, that adapts by shrinking and growing (if necessary) each individual proposal in order to better approximate the posterior distribution.

In order to prevent the algorithm from always having to grow or always having to shrink by a large amount, we allowed the action model parameters to change over time. By tuning the parameters online, we showed that the shrink-and-grow algorithm becomes more accurate for initially underestimating models. We also showed that the efficiency of shrink-and-grow improves when an overestimating action model tunes its parameters online.

Compiling all results, we found that the shrink-and-grow algorithm combined with a dynamic action model results in high accuracy localization while decreasing wasted particles when the initial parameter values start too high.

When performing online parameter tuning, shrink-and-grow did not significantly improve accuracy or efficiency over a least squares fitted static action model. We believe that this is due to the fact that the physical properties of the ground plane (thus the underlying systematic errors in the odometry) never changed throughout our test environment.

We expect future work will reveal that when moving between different surfaces (e.g. carpet vs. tile), online parameter tuning will show a significant improvement in both localization accuracy and efficiency over a static model tuned for a particular surface. In these scenarios, it is not just the parameters describing the variance of the proposal distributions that must change but also the parameters that describe the mean.

Acknowledgement

This work has taken place in the Intelligent Robotics Lab at the Artificial Intelligence Laboratory, The University of Texas at Austin. Research of the Intelligent Robotics lab is supported in part by grants from the National Science Foundation (IIS-0413257 and IIS-0538927), from the National Institutes of Health (EY016089), and by an IBM Faculty Research Award.

References

[1] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo localization: Efficient position estimation for mobile robots," in *Proc. of the 16th National Conf. on Artificial Intelligence (AAAI-99)*. AAAI/MIT Press, 1999.

[2] S. Thrun, "Robotic mapping: A survey," in *Exploring Artificial Intelligence in the New Millenium*, G. Lakemeyer and B. Nebel, Eds. Morgan Kaufmann, 2002.

[3] ——, "Probabilistic algorithms in robotics," *AI Magazine*, vol. 21, no. 4, pp. 93–109, 2000.

[4] J. Borenstein and L. Feng, "Measurement and correction of systematic odometry errors in mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 6, pp. 869–880, December 1996.

[5] I. Rekleitis, "Cooperative localization and multi-robot exploration," Ph.D. dissertation, School of Computer Science, McGill University, Montreal, Quebec, Canada, 2003.

[6] N. Roy and S. Thrun, "Online self-calibration for mobile robots," in *IEEE International Conference on Robotics & Automation (ICRA)*, 1999.

[7] C. M. Wang, "Location estimation and uncertainty analysis for mobile robots," in *IEEE International Conference on Robotics & Automation (ICRA)*, 1988.

[8] A. I. Eliazar and R. Parr, "Learning probabilistic motion models for mobile robots," in *International Conference on Machine Learning*, 2004.

[9] A. Howard and N. Roy, "The robotics data set repository (Radish)," 2003. [Online]. Available: http://radish.sourceforge.net/

[10] A. Elfes, "Occupancy grids: A probabilistic framework for robot perception and navigation," Ph.D. dissertation, Carnegie Mellon University, 1989.

[11] D. Fox, "Adapting the sample size in particle filters through KLD-sampling," *International Journal of Robotics Research*, vol. 22, no. 12, pp. 985–1003, 2003.

[12] G. Grisetti, C. Stachniss, and W. Burgard, "Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling," in *IEEE International Conference on Robotics & Automation (ICRA)*, 2005.

[13] J. Modayil, P. Beeson, and B. Kuipers, "Using the topological skeleton for scalable global metrical map-building," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.