Particle-inspired motion updates for grid-based Bayesian trackers

Jason M. Aughenbaugh, Brian R. La Cour
Applied Research Laboratories
The University of Texas at Austin
P.O. Box 8029, Austin, Texas 78713-8029 USA
{Jason, blacour}@arl.utexas.edu

Abstract – The computational cost of the motion update has limited the application of grid-based Bayesian trackers. Drawing inspiration from particle filters, an algorithm for efficient grid-based motion updates is developed. The algorithm’s complexity is linear in the number of grid cells and independent of the time increment for the motion update. It has the flexibility to model any Markov motion process. The accuracy of the algorithm and its sensitivity to implementation parameters is assessed, and trade-offs between accuracy and computational cost are explored.

Keywords: Bayesian tracking, particle filters

1 Introduction

Bayesian inference is a general framework for performing optimal target tracking in which knowledge about the target state is modeled using probabilities and measurements are modeled using likelihood functions. This allows for the fusion of disparate types of information, such as measurements from various types of sensors.

There has been extensive work on representations of Bayesian filters, including variations of Kalman filters [1], grid-based methods [2], and particle filters [3]. Under the strict conditions of linearity in the measurements, Gaussian measurement errors, and Gaussian process noise, Kalman filters have analytic solutions for both the measurement update (in which new measurements are incorporated into the estimate of the target’s state) and the motion update (also known as the system update or the prediction step in which the estimate from one time is moved forward to a future time via a model of the target’s motion). Grid-based methods and particle filters relax these assumptions and use non-parametric models that, in theory, can approximate any measurements and any posterior distribution. A historical overview of Bayesian tracking methods is given in [4], and technical details are summarized in [5].

Particle filters have been favored over grid-based methods due to the high computational costs of existing grid-based methods. However, researchers have begun to tackle some of these computational challenges. The efficiency and accuracy of the measurement updates can be improved through intelligent likelihood sampling [4]. Multi-target problems can be addressed by using independent moving grids for each target [6]. The use of higher order polynomials in place of piecewise constant approximations can improve overall efficiency of the probability models [7].

The motion update, which is a straightforward and inexpensive operation in particle filters, remains a challenge for grid-based methods. The purpose of this paper is to present an efficient motion update algorithm for grid-based methods. In Sec. 2, grid-based Bayesian tracking is reviewed. A hybrid grid-particle motion updating approach with run time linear in the number of grid cells and independent of the update time increment is introduced in Sec. 3. In Sec. 4, the performance of the method is explored, and the methods and results are discussed in Sec. 5.

2 Grid-based tracking overview

Grid-based Bayesian tracking has a long history, including recent advances [8, 9, 10, 2, 11, 12, 13, 4]. For an inherently discrete state space, a grid representation is optimal. For a continuous state, the grid represents a piece-wise approximation of the posterior density. We restrict ourselves here to a single target and the kinematic state, such that \( s = \{ x, y, \dot{x}, \dot{y} \} \).

2.1 Basics of Bayesian updating

We let \( \rho_t(s|y) \) represent the probability density function (PDF) at time \( t \) over the continuous target state \( s \), posterior to receiving measurement \( y \). The motion updated prior is denoted \( \rho_t^- (s) \), and the likelihood is given by \( L(y|s) \). The posterior PDF is found using Bayes’ Rule, as

\[
\rho_t(s|y) = \frac{L(y|s)\rho_t^- (s)}{\int L(y|s')\rho_t^- (s')ds'}. \tag{1}
\]
For a grid representation, we refer to each grid cell \( C_k \), each of which contains at least one state. In a piecewise-
constant model, the PDF is approximated as \( \rho_t(s) \approx \sum_k P_t(C_k)1_{C_k}(s) \), where \( 1_{C_k}(s) \) is an indicator function equal to 1 when \( s \in C_k \) and zero otherwise, and \( P_t(C_k) \) is the constant value across the \( k \)th grid cell at time \( t \). The likelihood function is integrated appropriately to capture the
local behavior over a cell, such that Bayes’ Rule for the measurement update of each cell becomes

\[
P_t(C_k) = \frac{P_t^{-1}(C_k) \int_{C_k} L(y|s')ds'}{\sum_i P_t^{-1}(C_i) \int_{C_i} L(y|s')ds'}. \tag{2}
\]

### 2.2 Motion updating overview

Given the state at time \( t_1 \), the state is uncertain at time \( t_2 > t_1 \) but can be modeled by making assumptions about the target’s motion. In general form, the model assumptions are captured as transition probabilities, such that \( q_{t_2, t_1}(s'|s) \) defines the probability of transitioning from state \( s \) to state \( s' \) in the time period from \( t_1 \) to \( t_2 \), such that

\[
\rho_{t_2}(s') = \int q_{t_2, t_1}(s'|s) \rho_{t_1}(s)ds. \tag{3}
\]

The discrete form of (3) expands to

\[
P_{t_2}^{-1}(C_k) = \sum_{i=1}^{K} \int_{s \in C_i} q_{t_2, t_1}(s'|s) \rho_{t_1}(s)ds \int_{s' \in C_k} q_{t_2, t_1}(s'|s) \rho_{t_1}(s)ds'. \tag{4}
\]

A basic numerical approach has a complexity of \( O(K^2) \) due to the double loop over cells (i.e., for each destination cell, consider each originating cell). Several approximate methods have been proposed to reduce this complexity. Stone et al. propose using a Markov chain model based on a generalized random walk that can be visualized using “velocity sheets” [2]. The full implementation and performance details of this model are not available in the open literature, so no direct comparisons are made in this paper.

A linear (in the number of grid cells) time approximate solution to the motion update is given by Kreucher et al. in [6]. Their approach is to rewrite (3) and then expand the solution using a second order Taylor series. They assume a nearly constant velocity motion and a small time step \( \Delta t = t_2 - t_1 \). As they note, this results in the Fokker-Planck equation, which can be solved using a backward Euler method that leads to a system of equations that can be solved efficiently using Thomas’ algorithm, assuming independence between the \( x \) and \( y \) motions.

The limitation of this method is the requirement of a small \( \Delta t \). One could apply this motion update successively over several small \( \Delta t \) in order to simulate a large \( \Delta t \). However, this will increase the computational cost. We instead draw inspiration from particle filters and propose a hybrid motion updating algorithm that is linear in the number of grid cells, makes no assumption of small \( \Delta t \), and only requires the motion Kernel be represented by a Markov process.

### 3 Proposed motion update algorithm

The hybrid method remains grid-based in its representation of the PDF over the state space. However, it performs the motion update by temporarily modeling the probability in each cell with particles.

#### 3.1 Particle filter motion updates

We assume that the target dynamics are given by a set of linear stochastic differential equations of the form

\[
ds(t) = A(t)s(t)dt + Q(t)\,dW(t), \tag{5}
\]

where \( A(t) \) and \( Q(t) \) are real matrices and \( W(t) \) is a \( d \times d \) dimensional Markov process. The matrix \( A \) defines the basic kinematic relation giving rise to dispersion, while \( Q \) describes the process noise. The difficulty of evaluating (3), even with the assumption of (5), depends on the manner in which the probability density over the states is modeled.

In a particle filter, the motion update is straightforward for any stochastic process. Each particle \( m \) has associated with it a weight \( w_m \) and a state \( s_m \). The motion update is performed by applying (5) to each particle individually. For example, letting \( w_m \) be a realization of \( W \) and assuming stationary kinematic and noise processes over the time update period, one can update each particle by

\[
s_m(t_2) = s_m(t_1) + (t_2 - t_1)A s_m(t_1) + Q \sqrt{t_2 - t_1}. \tag{6}
\]

#### 3.2 Basic algorithm

Each cell is its own mini state-space with a uniform distribution of probability across it and therefore can be modeled by equally weighted particles that are uniformly randomly distributed over it. Assuming there are \( M \) such particles, then the overall probability carried by each particle originating in cell \( C_i \) at time \( t_1 \) is \( w_i = P_{t_1}(C_i) / M \). The motion update is performed by applying (5) to each particle.

The next step is to map these dispersed and diffused particles back into individual grid cells, and then to calculate the total motion-updated probability in each destination cell \( C_j \) that arrived from source cell \( C_i \) by summation. No Kernel smoothing is used; all of the weight \( w_i \) associated with particle \( i \) is given to the destination cell. This process is repeated for each source cell in the grid in a readily parallelized for-loop.

The most straightforward way to map the updated states to a new cell would be to search over all of the cells for each particle, leading to complexity \( O(K^2 M) \). Fortunately, there is a way to do it in \( O(KM) \) time for constant, uniform grids, as described in the following.

---

2105
A constant, uniform (per dimension) grid is assumed. The grid cells are identified according to the state at their center, and a data structure is created that, in matrix notation, contains columns corresponding to the \( x \), \( y \), \( v_x \), and \( v_y \) dimensions, where each row is a different grid cell. The rows are sorted first by \( v_y \), then by \( v_x \), then by \( y \), and finally by \( x \). For example, the first 10 rows of a simple example are shown in Table 1.

<table>
<thead>
<tr>
<th>( cell _id )</th>
<th>( x )</th>
<th>( y )</th>
<th>( v_x )</th>
<th>( v_y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>20</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>10</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>20</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>20</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>10</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: Grid data structure, first 10 rows.

The basic algorithm above requires \( 2KM \) random variates, but this can be reduced to \( 2M \) with little loss in accuracy. The motion update involves a dispersion and a diffusion, designated by the first and second terms on the right side of (5) respectively. Assuming the diffusion process is independent of the state, the stochastic process applied to each particle is the same in all grid cells. This assumption holds for most applications of interest, although it may fail in applications such as ground vehicle tracking along roads.

The idea is to reuse a prototype cell across all cells for the diffusion process model. A random sample of \( M \) particles is created for a generic grid cell a vertex at \(( 0, 0, 0, 0 )\) in four-dimensional Cartesian space with sides of lengths of \( \Delta x \), \( \Delta y \), \( \Delta v_x \), and \( \Delta v_y \). We then sample uniformly (randomly) from each dimension, for example, in the interval \([-\Delta x, \Delta x]\) for the \( x \)-dimension. This gives a cloud of particles inside the generic cell. The diffusion portion of the motion update is now implemented by drawing a random normal variate \( w_i \) for each particle. The prototype particle cloud can be mapped to each actual grid cell by adding in the grid cell center, and then (6) can be applied to each particle.

The reuse of the variates generally will introduce some bias into the update, and one could encounter very “unlucky” sampling. However, for reasonable sample sizes, the trade-offs between accuracy and run time, discussed in Sec. 4.6, seem to warrant the reuse of variates.

3.4 Probability mass thresholding

As the tracker runs, it is likely that the probability of the target being in certain regions approaches zero. For example, assume that there are 100 cells. If the probability were spread uniformly, each cell would have a probability of 0.01. Now assume the prior probability associated with a particular cell is 0.00000001. No matter how this probability disperses and diffuses into other cells, its contribution is essentially negligible. In order to speed the overall motion update, one can skip application of the motion update to this source cell. In order to preserve the total probability, the probability in the prior cell is assumed to stay there during the motion update, though mass may be added if it arrives from other cells. The selection of a threshold for this “skipping” may be problem dependent, but in general a threshold a few orders of magnitude below the uniform distribution probability is adequate. The accuracy of this optimization is examined in Sec. 4.6.

4 Evaluation of performance

In order to test the performance of the proposed motion update algorithm, we consider a Gaussian input distribution. The analytic solution of the Kalman filter provides the exact answer to this motion update, but it does not represent the “correct” or ideal estimate that one would expect from a grid-based model, regardless of algorithm. Due to discretization in the grid, the analytic solution can only be approached as the number of grid cells goes to infinity. Consequently, comparing the simulation solution directly to the analytic solution confounds the discretization effects with the motion update algorithm.

The best one can do is use a discretized version of the analytic solution. Therefore, the analytically updated continuous distribution is found and then discretized according to the selected grid dimensions. This still only approximates
such that 

\[ s(t) = s(0) + \int_0^t e^{A(t-t')} Q \, dW(t'), \]  

(8)

where the distribution of \( s(0) \) is given by \( \rho_0(s) \).

Defining damping coefficient \( \gamma = 0.00004 \, \text{s}^{-1} \), diffusion coefficient \( \sigma = 0.01 \, (\text{m/s})^{1/2} \), and time period \( \tau \), the analytic solution for the Integrated Ornstein-Uhlenbeck process is

\[
\begin{align}
\mu_\tau &= A \mu_0 \\
\Sigma_\tau &= A \Sigma_0 A^T + Q, \\
A &= \begin{bmatrix}
I & (1 - e^{-\gamma \tau}) I \\
0 & e^{-\gamma \tau} I
\end{bmatrix}, \\
Q(\tau) &= \sigma^2 \begin{bmatrix}
\gamma^{-2} b_{11}(\tau) I & \gamma^{-2} b_{12}(\tau) I \\
\gamma^{-2} b_{21}(\tau) I & \gamma^{-1} b_{22}(\tau) I
\end{bmatrix}, \\
b_{11}(\tau) &= \tau - 2 \gamma^{-1} \left(1 - e^{-\gamma \tau}\right) + \frac{1}{2} \gamma^{-1} (1 - e^{-2\gamma \tau}), \\
b_{12}(\tau) &= b_{21}(\tau) = \left(1 - e^{-\gamma \tau}\right) - \frac{1}{2} (1 - e^{-2\gamma \tau}), \\
b_{22}(\tau) &= \frac{1}{2} (1 - e^{-2\gamma \tau}).
\end{align}
\]

4.1 Example scenarios

For this example, we model the target motion as an Integrated Ornstein-Uhlenbeck (IOU) process \[2\]. It can be shown that, if \( s(0) \) is independent of \( \{W(t) : t \geq 0\} \), then the solution to (5) is given by the Itô integral

\[
\begin{align}
s(t) = s(t) &+ \int_0^t e^{A(t-t')} Q \, dW(t') \\
\end{align}
\]

where the distribution of \( s(0) \) is given by \( \rho_0(s) \).

4.2 Accuracy metrics

Each scenario is repeated 25 times in order to average out any effects due to the stochastic nature of the algorithm. We will consider two basic metrics for assessing average model accuracy across these runs. First, we consider the root mean squared errors (RMSE) in the position estimate. We choose this metric because the mean of the distribution is often used to summarize knowledge about the target location. Second, we consider errors in the overall estimate of the distribution by using the average \( L^1 \) norm error between the simulated distribution and a discretized form of the analytic solution. This measure is useful because it is bounded (between zero and two), with zero indicating perfect agreement between the distributions and two indicating that the two distributions have completely disjoint regions of support.

We considered information theoretic measures such as the Kullback-Leibler (KL) divergence, which provides a pointwise measure of the similarity of two distributions. For this study, the KL divergences revealed rankings of the scenarios that were nearly identical to the \( L^1 \) norm rankings, so the KL divergence analysis is omitted for brevity. A pure statistical entropy measure was considered less informative than these pairwise tests because it only reflects the bulk similarity of the distributions with no regard for local deviations.

4.3 Accuracy results

For this analysis, we do not skip grid cells with low probability (see Sec. 3.4) in order to test the accuracy of the base algorithm without this additional tuning parameter. We do, however, reuse the random variates as described in Sec. 3.3. The sensitivity of performance to both the probability threshold and the variate reuse is considered in Sec. 4.6.

The errors in the mean position estimate between the discretized prior and the discretized motion updated prior are shown on the left side of Fig. 1. For visual clarity, we have omitted from the figure some of the combinations of grid resolution and number of particles, but these follow the same trends. We first note that the error increases with update step time for most methods. This has two causes. First, any inaccuracy in the model will be amplified more the longer the trajectory. Second, the inherent error between the discretization of the analytical solution and the fully discrete solution similarly increase the longer the trajectory, regardless of the algorithm used for the fully discrete solution.

We begin by discussing the performance for the larger time steps, in which there is a clear separation of the scenarios. Specifically, the number of particles seems to have a significant effect on the relative performance in each of the scenarios. Those with 100 particles performed the worst, followed by those with 1000 and then those with 10000.

This results directly from the reuse of the random variates discussed in Sec. 3.3. Sparse sampling alone should not lead to any bias in the overall updated distribution. How-
ever, with sparse sampling, it is possible that any single set of points would be skewed in some direction. By reusing these variates across all cells, any skew in the points would translate to a skew in the final distribution, thereby affecting the mean of the distribution. The effects of variate reuse are considered further in Sec. 4.6.

A follow-up question is whether the accuracy in the mean using any particular number of particles is sufficient. In the worst case, the error after 6000 seconds is about 350 m. During these 6000 seconds, the target has actually moved 12000 m, so the error is only about 3% of the translation. At the same time, the true standard deviation (from the analytic, undiscretized solution) of the estimate in each position dimension is 4729 m. These observations suggest that even the worst case error is reasonable, and that certainly the errors with finer grids or with more particles are sufficiently small for a long time update. With 1000 particles, the worst case error reduces to about 125 m. For 10000 particles, the worst case error is about 50 m.

For an update of 300 s with 100 particles, the worst case error was about 35 m, with the target traveling 600 m and having a true, updated standard deviation of 1018 m in each position dimension. For the 60 s update, the target travels 120 m, and the maximum error in the mean estimate is 14 m for the base methods, and the true standard deviation is 1000 m. With 1000 particles, the worst case lowers to 6 m for 60 s and 10 m for 300 s.

The results in Fig. 1 also suggest that there is an improvement in performance with finer grids. Since arguably all grids yield reasonable results in mean estimation for just 100 particles (and more convincingly for 1000 particles), no firm conclusion can be drawn about the overall importance of grid resolution.

The results for the $L^1$ norm between the discretized analytic solution and the simulated solution are shown on the right of Fig. 1. For small time steps, the differences in performance for different numbers of particles (with the same grid) are small, but they increase significantly for time steps larger than 3000 seconds. For small time steps, small biases in the sample variates in velocity have little effect, but for large time steps, the effect can be quite large. We have omitted most 1000-particle results from the plot for clarity because they fell between the 100- and 10000-particle results.

Across all time steps, divisions based on the grid resolution are clear. However, these divisions are not strictly in terms of total degrees of freedom in the grid.

At short time steps, the $31 \times 31 \times 61 \times 61$ grid (not shown for clarity in the figure) performs comparably with the $31 \times 31 \times 31 \times 31$ grid (for equal particle counts), suggesting that the investment of additional degrees of freedom in the velocity dimension is wasted. At the same time, the $61 \times 61 \times 15 \times 15$ grid performs much better than the $31 \times 31 \times 31 \times 31$ grid, despite having about the same number of degrees of freedom. Similarly, the $51 \times 51 \times 9 \times 9$ grid performs better than the $31 \times 31 \times 15 \times 15$ despite the two having similar numbers of degrees of freedom.

The explanation appears to be that for small time steps, resolution in position is more important than resolution in velocity. For example, as the time step tends to zero, the influence of the velocity becomes less and less important, and the final accuracy will depend more and more on the actual discretization of the position space.

Conversely, as the time step increases to 6000 seconds, the $L^1$ norms of the scenarios become ordered roughly by total degrees of freedom, with slight preference to-
wards additional degrees of freedom in velocity. For example, now the $61 \times 61 \times 15 \times 15$ performs worse than the $31 \times 31 \times 31 \times 31$ scenario (for the shown 100 and 10,000 particles cases as well as the not shown 1000 particles case), and the $31 \times 31 \times 61 \times 61$ scenario (not shown) outperforms the $61 \times 61 \times 31 \times 31$ scenario (not shown).

It is not surprising that the finest grid with the most particles gives the best accuracy, but this accuracy comes at the cost of higher run time, as shown in the next section.

4.4 Computational costs

It was claimed in Sec. 3 that the proposed algorithm is linear in the number of particles and grid cells. We consider only the run time for the unoptimized implementation that does not skip grid cells with low prior probability (see Sec. 3.4). If one is allowed to skip low probability grid cells, the computational cost is distribution dependent and often sublinear.

The claim of linearity is supported by the data shown in Fig. 2. The simulation was implemented in C++ and run on a four-core computer with 4 GB of RAM. In the upper graph, the average run time over ten runs is plotted versus the number of the grid cells for various grids. The number of particles used is fixed at 100 for all scenarios, and an update time of 180 s is used. The data are separated into two categories: one in which the number of position grid cells is held fixed (at $51 \times 51$) and the number of velocity cells is varied, and on in which the number of velocity cells is fixed (at $15 \times 15$) and the number of position grid cells is varied. Also shown is the best-fit linear regression line for the combined data sets. There is no systematic deviation from this line for either data set, supporting the claim of linearity in the total number of grid cells.

The run time is also nearly linear in the number of particles used with a fixed grid, as shown in the lower graph of Fig. 2. Close examination of the residuals reveals a slight sub-linearity. The most likely explanation for this is related to overhead in the parallelization of the algorithm; a parallel for-loop is used over the grid cells, which involves the overhead of managing the parallelization. For a given grid cell, the particles are handled in series. Thus, the more particles per grid cell, the less overhead per particle. This effect was not seen when varying the number of grid cells, because the overhead also scaled with the number of grid cells.

Although not shown in the preceding data, there is a slight run time dependence on the time step: the longer the time step, the faster the run time. This is in sharp contrast with a finite difference method such as [6], in which the run time increases with time step. As implemented, each particle is examined to see if it falls outside the bounds of the state-space in a loop over dimension. Once it falls outside in one dimension, the remaining dimensions do not need to be checked, saving computational expense. The longer the time step, the more particles leave the state space (due both translation and diffusion), thereby leading to smaller run times.

In order to fully assess the algorithm and implementation, one must consider both accuracy and speed, as follows.

4.5 Accuracy and run time trade-offs

The selection of the best set of parameters is a multi-objective optimization problem that will be scenario- and problem-specific. In this section, we assume that the analyst is most interested in trade-offs between run time and the overall quality of the approximated distribution. Although a point estimate will probably be used to assess target states, the goal of a full Bayesian approximation is to capture the full distribution, and if the overall distribution is accurate, the mean the distribution will also be reasonably accurate.

Rather than weighting the multiple objectives explicitly, we consider the Pareto efficient set over possible parameter values [14]. A particular choice of parameter values will be a member of the Pareto efficient set if there is no other choice of parameter values for which one can do better in any given performance objective without sacrificing at least one other performance objective. For example, if scenario A has a lower run time and a lower $L^1$ norm error than scenario B, clearly scenario A is better than scenario B. If instead scenario A had a lower run time but higher $L^1$ norm error than scenario B, the optimal choice is unclear without an explicit weighting of the two objectives. The Pareto efficient set contains all scenarios that would be optimal for any particular weighting of objectives.

A scatter plot of run time versus $L^1$ norm is shown in Fig. 3. The Pareto efficient set is shown with blue circles and labeled with the corresponding configuration parameters. The red Xs mark the non-efficient scenarios, which are not labeled to avoid clutter; these scenarios can be rejected.
as sub-optimal for this example problem.

Although the $31 \times 31 \times 61 \times 61$ grid with 100 particles is on the Pareto frontier, it provides almost no gain in accuracy compared to the $61 \times 61 \times 15 \times 15$ with 100 particles, despite a steep increase in run time. Consequently, it is probably not ideal. Similarly, the $61 \times 61 \times 61 \times 61$ grid with 1000 or 10,000 particles also yields small accuracy gains for a large increase in run time. Of the remaining options, the choice of the ideal model depends on the exact trade-offs between run time and accuracy.

The results are encouraging because they reveal that one can achieve an accuracy near the best (corresponding to $61 \times 61 \times 61 \times 61$ with 10,000 particles) in less than $1/100$ the run time (using $61 \times 61 \times 61 \times 61$ with 100 particles), and that this takes only about 1 second. In general, the 100 particle runs outperform the higher number of particle runs for all grids except the finest. Although the exact Pareto sets are scenario dependent, one can conclude that the proposed method yields sufficient accuracy in reasonable run time, and is therefore feasible for real-time implementation.

4.6 Sensitivity to optimizations

The reuse of random variates (Sec. 3.3) and the use of probability thresholds (Sec. 3.4) were introduced to improve performance, but they could also affect accuracy. In this section, these trade-offs are explored for the $51 \times 51 \times 9 \times 9$ and the $61 \times 61 \times 61 \times 61$ grids, with each scenario repeated with 25 different runs to average out any effects due to the stochastic nature of the algorithm. The results were similar for other grids and are omitted for brevity.

The results for the RMSE in position estimate (shown on the left side of Fig. 4) are clear: the regeneration of random variates improves the position estimation by as much as a factor of 10 for small time steps, with the difference decreasing as the time step increases. However, as noted in Sec. 4.3, the overall error is already inconsequential with the reuse of variates. Moreover, there is little or no gain in the average $L^1$ norm (shown on the right side of Fig. 4) when regenerating random variates for each cell compared to reuse. Considering that these insignificant reductions in error come at the cost of increasing run time by about a factor of 100 (e.g., increasing from 0.0291 to 2.135 for scenario $51 \times 51 \times 9 \times 9$ with 100 particles), the regeneration of random variates is not valuable.

There is no perceptible change in the $L^1$ norm accuracy when implementing with the probability threshold, and nearly no effect in the RMSE. However, the benefits are substantial. For example, in the $61 \times 61 \times 61 \times 61$ with 1000 particles scenario, run time is reduced from 18.07 s to 0.38 s.

It is important to note that the gains from probability thresholding are seen only when the probability mass is relatively well localized. For example, with a nearly uniform distribution, there is no savings because when the probability in all cells exceeds the threshold, none are skipped. In practice, targets (or even false alarms) will be reasonably well localized, and gains in run time can be significant after a few measurement updates.

5 Discussion and summary

The motion update algorithm for grid-based Bayesian trackers introduced above has several strengths. First, the run time is linear in the number of grid cells. Second, the run time is algorithmically independent of the system update time increment, and in practice the updates actually become slightly faster the longer the update increment. Third, the algorithm can implement any stochastic model by simply changing the random distribution from which the process noise is drawn. Finally, the basic algorithm is easily adapted from existing particle filter implementations.

One limitation of the algorithm is that the largest efficiency gains rely on a uniform rectilinear grid, as it is this structure that facilitates efficient mapping of particles to grid cells. Other grid structures will require the development of other intelligent mapping schemes.

The algorithm requires a choice of the number of particles for all updates and a probability threshold for skipping cells. One could make these choices adaptive depending on the scenario. For example, when the probability is diffuse, few cells will be skipped by the tolerance check, so one may want to reduce the number of particles to improve run time. Once probability is more localized, the run time will decrease, but the localized probability spikes might benefit from increased accuracy in the motion update, so an increase in particles may be warranted.

The crux of the motion update is calculating the transition probabilities $q_{t_k,t_1}(C_k|C_i)$ in (4), which are independent of the prior distribution. One could save these transition proba-
abilities in a look-up table, rather than recalculating them for every update. There are two potential limitations with this. First, it is not clear that this would be significantly faster than the proposed algorithm, and it would require memory to store the $K \times K$ matrix. Second, there is no guarantee that the time step $t_{j+1} - t_j$ would be constant, or that the diffusion process is stationary across time steps. Therefore, one would either need to perform motion updates continually for small updates (allowing for an artificially constant time step) or to save a separate transition matrix for all relevant time steps. Nevertheless, a look-up table implementation is an interesting topic for future work, and the algorithm proposed in this paper would provide an efficient means for populating such a matrix.

In general, the proposed algorithm is efficient, accurate, and flexible. It therefore has the potential to reinvigorates grid-based Bayesian tracking schemes, especially combined with other recent advances. Suggested future work includes the direct comparison to other motion update methods and continued development of efficient grid-based Bayesian tracking schemes.

Acknowledgments

This work was supported by the U.S. Office of Naval Research under contract no. N00014-06-G-0218-01. The authors thank Brian Yocom for his suggestions.

References


