Smoothing methods for particle filters in tracking applications

Joel Nulsen[1], Paul Baxter[2] and Trevor Wood[2,*]

Abstract – **In the use of particle filters to estimate a target's location, smoothing can be used to refine state estimation using future data. In this paper we present two established smoothing algorithms, the Sequential Fixed-Lag Smoother (SFLS) and the Backwards Simulation Particle Smoother (BS-PS). While SFLS can run in real time (although with a fixed delay), BS-PS can be excessively computationally expensive. We propose a novel combination of BS-PS and SFLS which requires significantly less computation in exchange for a small reduction in accuracy. The performance and complexity of all four methods, including the particle filter with no smoothing and the combined smoother, are compared both on simulated truth data and on real data with known waypoints for an indoor tracking system using a combination of inertial sensor data and a building map.**

I. INTRODUCTION

In tracking problems where noisy measurements are available, or fusion of disparate incomplete data sources is required, discrete time Bayesian filtering techniques are often applied to give an estimation of the target state in the form of a probability distribution. In the filtering problem, this estimation is based on all measurements collected up to the current time. In sufficiently simple situations where the relevant equations are linear and all noise can be assumed to be white and Gaussian, analytic methods such as the Kalman filter exist to give optimal solutions – see for example Chapter 4 of [1].

However, many real-world problems have complex state spaces and nonlinear process equations which render such analytic approaches intractable. In these instances, sequential Monte Carlo methodologies known as particle filters are often used. Particle filters attempt to approximate the desired probability distribution using a set of N particles, each with an associated importance weight.

The state of particle i at time step k is written $\boldsymbol{x}_k^{(i)}$, for $i = 1, \dots, N$ and $k = 1, \dots, T$. At each time step, the process noise $\boldsymbol{v}_k^{(i)}$ is sampled from some known distribution (typically Gaussian) and the particle is propagated according to the forwards model

$$\boldsymbol{x}_{k+1}^{(i)} = \boldsymbol{f}\left(\boldsymbol{x}_k^{(i)}, \boldsymbol{v}_k^{(i)}\right), \tag{1}$$

where $\boldsymbol{f}$ is in general some nonlinear function taking values in the state space. The importance weights $w_k^{(i)}$ are typically assigned based on the observations made up to time k. The exact way in which this is done depends on the particular filtering algorithm being used, but they must satisfy the normalisation condition $\sum_{i=1}^{N} w_k^{(i)} = 1$. The approximation to the target state estimation distribution at time k is then given by $\sum_{i=1}^{N} w_k^{(i)}\, \delta\left(\boldsymbol{x}_k^{(i)}\right)$, with δ the Dirac delta function.

Most particle filters also employ resampling to maintain particle diversity when too many particles are given zero (or almost zero) weight. The filter draws N new particles from the old particle sates, with the probability of choosing a particular state given by its weight.

In this paper we focus on the smoothing problem of estimation based on future observations. Both the Sequential Fixed-Lag Smoother and the Backwards Simulation Particle Smoother work in conjunction with a particle filter by using the particle states at future times to select a subset (or equivalently re-weight) the particles at the present time step – they do not use the future observations directly. Because of this, their performance is heavily dependent on that of the underlying particle filter. Improvements in performance most often arise in situations where particles become erroneous, clearly diverging from the target state, and are subsequently discarded/de-weighted by the particle filter. Smoothing techniques can allow such particles to be eliminated sooner (SFLS) or even altogether (BS-PS).

Sections II and III present the SFLS and BS-PS algorithms along with notes on their performance and complexity. Section IV proposes and outlines a novel combination of these two algorithms. Section V describes the experimental setups used in detail, and finally Sections VI and VII present the results of these experiments in terms of accuracy and computational cost respectively. A brief discussion of suitability of the techniques discussed for various tracking applications along is given in Section VIII.

[1] Centre for Mathematical Sciences, University of Cambridge, Wilberforce Road, Cambridge, CB3 0WA
[2] Cambridge Consultants, Science Park, Milton Road, Cambridge, CB4 0DW
* Corresponding Author – trevor.wood@cambridgeconsultants.com

II. SEQUENTIAL FIXED-LAG SMOOTHER

A. *Algorithm and notes*

There are several SFLS algorithms in existence; a simple version is presented here, along with a brief statistical justification.

In the particle filter framework, a particle $\boldsymbol{x}_{k+1}^{(j)}$ can be said to be "descended" from a unique particle $\boldsymbol{x}_{k}^{(i)}$ if it was produced either by progressing $\boldsymbol{x}_{k}^{(i)}$ forwards according to (1), or by resampling and drawing the state occupied by $\boldsymbol{x}_{k}^{(i)}$. Following this thread backwards in time, we can define the "ancestor" of $\boldsymbol{x}_{k+1}^{(j)}$ at time t for all $0 \leq t \leq k$ inductively in the obvious way.

An SFLS has a fixed delay between observations and estimates; suppose the delay is L time steps. Then when the particle filter reaches step $k+L$, our smoothing algorithm updates the weights of particles at step k according to

$$w_{k|k+L}^{(i)} = \textstyle\sum_{j\leftarrow i} w_{k+L}^{(j)}, \tag{2}$$

where the notation $j \leftarrow i$ may be read as " j descended from i ", and $w_{k|k+L}^{(i)}$ is the importance weight of particle i at time k conditioned on observations up to time $k+L$. In other words, the updated weight of a particle at time k is determined by the number and weights of its descendants at time $k+L$.

In particular, those particles at time k with no descendants at time $k+L$ are downweighted to zero, and can be discarded, in effect, early. This is most pronounced in situations where the particle filter de-weights a large proportion of particles at each time step, and is very important in the combination of SFLS and BS-PS given in Section IV. At the same time, "fertile" particles which have many descendants after L time steps are given higher weights.

As mentioned earlier, SFLS can run at the same speed as the particle filter, but results are only available after a fixed time delay equivalent to the lag L. It also has a fixed memory requirement; a list of ancestors for each particle for the last L time steps has to be stored as a cyclic buffer. However, increasing the lag tends to give better results [2], so choosing the length of lag to be used is a balance of accuracy versus computational cost and result availability.

B. *Justification of algorithm*

For the standard "bootstrap" particle filter with resampling at every step and observations $\boldsymbol{z}_k$, the importance weights are [3]

$$w_k^{(i)} \propto p\left(\boldsymbol{z}_k \middle| \boldsymbol{x}_k^{(i)}\right), \tag{3}$$

where $p\left(\boldsymbol{z}_k \middle| \boldsymbol{x}_k^{(i)}\right)$ is the probability of observing $\boldsymbol{z}_k$ conditioned on the target state being $\boldsymbol{x}_k^{(i)}$. We seek the updated weights $w_{k|k+L}^{i}$ in terms of the w_{k+L}^{j}. The Chapman-Kolmogorov equations give (integrating over all intermediate states)

$$w_{k|k+L}^{(i)} \propto p\left(\boldsymbol{z}_{k+L} \middle| \boldsymbol{x}_k^{(i)}\right) \tag{4}$$

$$= \int \mathrm{d}\boldsymbol{x}_{k+1} \ldots d\boldsymbol{x}_{k+L}\, p\left(\boldsymbol{z}_{k+L}|\boldsymbol{x}_{k+L}\right) p\left(\boldsymbol{x}_{k+L}|\boldsymbol{x}_{k+L-1}\right) \ldots p\left(\boldsymbol{x}_{k+1}\middle|\boldsymbol{x}_k^{(i)}\right), \tag{5}$$

where $p\left(\boldsymbol{z}_{k+L} \middle| \boldsymbol{x}_k^{(i)}\right)$ is the probability of observing $\boldsymbol{z}_{k+L}$ at time $k+L$ conditioned on the target being in state $\boldsymbol{x}_k^{(i)}$ at time k, and $p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k)$ is the probability of propagating a particle in state $\boldsymbol{x}_k$ at time k to a state $\boldsymbol{x}_{k+1}$ at time $k+1$. We then make use of the operator approximation

$$\int \mathrm{d}\boldsymbol{x}_{k+1} \ldots d\boldsymbol{x}_{k+L}\, p\left(\boldsymbol{x}_{k+L}|\boldsymbol{x}_{k+L-1}\right) \ldots p\left(\boldsymbol{x}_{k+1}\middle|\boldsymbol{x}_k^{(i)}\right) \approx \sum_{j\leftarrow i} \tag{6}$$

The dummy labels on the LHS are all the intermediate states between times k and $k+L$, and the corresponding index on the RHS sums over the descendants of particle $\boldsymbol{x}_k^{(i)}$. This becomes exact in the limit $N \to \infty$. Substituting into (5) gives

$$p\left(\boldsymbol{z}_{k+L} \middle| \boldsymbol{x}_k^{(i)}\right) \approx \sum_{j\leftarrow i} p\left(\boldsymbol{z}_{k+L} \middle| \boldsymbol{x}_{k+L}^{(j)}\right) \tag{7}$$

$$\propto \sum_{j\leftarrow i} w_{k+L}^{j} \tag{8}$$

as required.

III. BACKWARDS SIMULATION PARTICLE SMOOTHER

A. *Algorithm*

Whereas the SFLS can effectively only see the future L time steps ahead, the backwards simulation particle smoother instead works with the complete output of the particle filter, at all time steps; such techniques are known as global smoothing. This has some immediately evident drawbacks. For example, the amount of data that needs to be stored is potentially very large, proportional to the dimensionality of the state space, the total number of time steps T, and the number of particles N used by the underlying particle filter. As we shall see, the computational complexity scales similarly. In addition, BS-PS can only be implemented after the particle filter has finished running, severely restricting the availability of results.

The BS-PS algorithm, as derived on page 167 of [1], produces any number S of trajectories through the state space, a single one of which we shall denote by $\{\tilde{\boldsymbol{x}}_k\}_{k=1}^{T}$. The process for generating such a path is as follows:

1. Choose $\tilde{\boldsymbol{x}}_T = \boldsymbol{x}_T^{(i)}$ with probability $w_T^{(i)}$
2. For $k = T-1, \dots, 1$
   a. Compute new weights by

$$w_{k|k+1}^{(i)} \propto w_k^{(i)} p\left(\tilde{\boldsymbol{x}}_{k+1} \middle| \boldsymbol{x}_k^{(i)}\right) \tag{9}$$

   b. Choose $\tilde{\boldsymbol{x}}_k = \boldsymbol{x}_k^{(i)}$ with probability $w_{k|k+1}^{(i)}$

Scaling the weights according to the forwards stepping probability in (9) helps to ensure that the trajectories are continuous paths through the state space – each backwards step will be individually reasonable. Further, particles with zero weight will not be included in the trajectory, so erroneous particles as described in Section I will be ignored.

B. *Complexity*

The calculation of individual stepping probabilities may be relatively simple. For example, in many particle filter applications, the forwards model (1) takes the slightly simpler form

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k) + \boldsymbol{v}_k, \tag{10}$$

and the process noise is assumed to be white Gaussian, i.e. $\boldsymbol{v}_k \sim \mathcal{N}(\boldsymbol{0}, Q)$ for a given covariance matrix Q. Denoting the probability of an event A by $\mathbb{P}(A)$, we have

$$\begin{aligned} p\left(\tilde{\boldsymbol{x}}_{k+1} \middle| \boldsymbol{x}_k^{(i)}\right) &= \mathbb{P}\left(\boldsymbol{v}_k = \tilde{\boldsymbol{x}}_{k+1} - \boldsymbol{f}\left(\boldsymbol{x}_k^{(i)}\right)\right) \\ &= \mathcal{N}\left(\tilde{\boldsymbol{x}}_{k+1} - \boldsymbol{f}\left(\boldsymbol{x}_k^{(i)}\right); \boldsymbol{0}, Q\right) \end{aligned} \tag{11}$$

where $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma)$ is the multivariate normal probability density function with mean $\boldsymbol{\mu}$ and covariance matrix Σ evaluated at the point $\boldsymbol{x}$.

However, as a whole BS-PS is computationally expensive. This is due to the fact that the above probability must be evaluated once at each time step for all N particles, for each backwards trajectory. Indeed, [1] gives the computational cost of the BS-PS algorithm as $O(STN)$. In theory, the complexity may be reduced using rejection sampling to generate the trajectory [4]. However, it was found that this approach can be significantly slower, e.g. in cases where the stepping probabilities $p\left(\tilde{\boldsymbol{x}}_{k+1} \middle| \boldsymbol{x}_k^{(i)}\right)$ are almost all either zero or extremely small, as in many applications.

The accuracy of the backwards smoother is much more dependent on the number of particles used by the underlying particle filter than it is on the number of trajectories produced [5]. As such, it is reasonable to take $S \ll N$, particularly if N is required to be very large. As a result, in many implementations it is the large value of N that is the primary cause of computational cost, rather than a large value of S.

While BS-PS was found to be an expensive algorithm for our example of the indoor tracker (see Section VIII), taking a long period of time to generate a usable number of trajectories, this may not always be the case. The high rate of particle de-weighting in the indoor tracker requires N to be very large, whence arises the cost of the algorithm. In applications where the particle filter does not require so many particles, BS-PS may be reasonably efficient.

The linear dependence of the complexity on T is, for the most part, unavoidable. Dividing a particularly long data set into chunks may allow for parallel processing, but the total computation required is the same. Further, BS-PS does not outperform the particle filter when $k \lesssim T$, since at these times it does not have much future data with which to work. Thus such a division of the data set would somewhat compromise the performance of the smoother.

It should also be noted that, while potentially very computationally costly, the input of particles at all time steps to the BS-PS algorithm does result in significantly increased performance, especially compared to a particle filter with no smoothing. Section IV presents a method for using BS-PS quickly in cases where N is required to be very large.

C. *Path vs. state*

There is a subtle qualitative difference between the output of the particle filter or SFLS as opposed to BS-PS. The former gives state estimations for each time step, from which one can extract an estimate of the path taken. By contrast, BS-PS gives multiple estimations for the path taken, from which a state estimate at each time step can be formed. This is relevant when it comes to comparing accuracy; there may be time steps where SFLS gives a better state estimation than BS-PS, but trajectories produced by BS-PS tend to better reflect the target path as a whole.

IV. COMBINATION OF SFLS AND BS-PS

As mentioned in Section II, SFLS discards particles with no descendants after the lag period L. In some cases this reduction in particle number can be very significant. For the simulated data experiment in Section VI.A, the particle filter used $N = 50{,}000$ particles at each time step, but the SFLS typically only kept between $N_k = 700$ and $N_k = 2{,}000$ of these (note that the number of particles discarded varies between time steps, hence the k suffix).

Turning to BS-PS, we find that while performance is typically higher than both the plain particle filter and the SFLS, it can be prohibitively computationally expensive. This expense arises from the typically quite large number of particles required by the filter. The question then naturally arises: what if both techniques were applied? That is, what would be the effect of running the particle filter with the SFLS (recall this does not slow down the process) and then using the remaining particles to generate backwards trajectories from BS-PS?

Depending on the proportion of particles discarded by the SFLS, this technique may speed up BS-PS very significantly. The above reduction in particle number resulted in an increase of speed by a factor of 50, as detailed in Section VIII. We find, however, that this combination has a slightly reduced accuracy in comparison to BS-PS running on the particle filter with no other smoothing.

V. EXPERIMENTAL SETUP

A. *Indoor tracker*

The performances and computational complexities of the methods presented were tested using a particle filter for tracking the location of a target indoors. Here, a particle state consists of x and y position coordinates, an orientation bias and a step length bias. While it is only the position that is estimated by the tracker, the hidden bias parameters feature in the update of a particle's position, and have noise added to them at each step.

For the most part, measurements take the form of step headings and times, but where possible the tracker can also use GPS measurements to improve location estimation – typically this only occurs when the target is outdoors. Information is also provided by making use of the building map along with an assumption that the user does not move through walls. Given an assumed average step length, these measurements can be used to generate a dead reckoning path. However, due to variation in step length and noise in the heading measurements, such paths tend to gradually bear less and less resemblance to the true path as time goes on. It was for this reason that a particle filter was used for the indoor tracking problem; it makes allowances for noise, and enables the effective use of map data to give much-improved results.

The high rate of de-weighting referred to in Section III.B comes from the fact that the particle filter assigns zero weight to any particles which cross a wall as their position is updated; in a narrow corridor, for instance, this can happen very often.

B. *Experiment with simulated data*

The first experiment used simulated truth data for a path around a building. Process noise was then added to the path, and finally noisy measurements were obtained, including GPS positions and times when the path was outside the building. White Gaussian noise was added to these measurements at each time step, with standard deviations given by baseline values multiplied by a scale factor. The baseline values were 0.15 radians for step orientation, 0.05 seconds for step time, 0.25 metres for GPS position, and 0.005 seconds for GPS time.

C. *Experiment with real data*

The indoor tracker particle filter was also tested using real inertial and GPS data, gathered by a unit which could be attached to a person's belt (see right). Similarly to the simulated data above, the raw data was processed to give a list of step bearings and times. While the unit is capable of receiving GPS signals while outside, for the purposes of this experiment this functionality was switched off.

In order to be able to measure accuracy, four waypoints were chosen around the path. At each waypoint, a button on the unit was pressed, and the time of this button press was stored. The particle states at these times were then recorded, from which the error was calculated.

D. *Error measures*

The weighted RMSE's for the particle filter and the SFLS at time k are given by

$$E_k = \sqrt{\sum_{i=1}^{N} w_k^{(i)} \left|\boldsymbol{x}_k^{(i)} - \boldsymbol{y}_k\right|^2} \quad \text{and} \quad E_k = \sqrt{\sum_{i=1}^{N_k} w_{k|k+L}^{(i)} \left|\boldsymbol{x}_k^{(i)} - \boldsymbol{y}_k\right|^2} \tag{7}$$

respectively, where $\boldsymbol{y}_k$ is the known position of the target at time k. The trajectories produced by BS-PS do not have weights associated with them, so the standard measure

$$E_k = \sqrt{\sum_{i=1}^{S} \frac{1}{S} \left|\tilde{\boldsymbol{x}}_k^{(i)} - \boldsymbol{y}_k\right|^2} \tag{8}$$

was used instead, where S is the number of trajectories.

Another error measure used was cut-off weighted RMSE: if a particle's RMSE was above a certain threshold, in this case chosen to be 15m, it was deemed to be simply wrong, and the error was recorded as being exactly the threshold.

VI. RESULTS

A. *Simulated data*

First we investigate performance for a single level of measurement noise (scale factor 2). All four methods (particle filter, SFLS, BS-PS and BS-PS on SFL smoothed data) were run 20 times, with the particle filter using 50,000 particles at each time step, SLFS using a lag of 30 time steps, and the BS-PS routines both producing 100 trajectories each time.

Figure 1 shows the average weighted RMSE at each time step, for a single noise level. Possibly the most striking features of this figure are the large peaks between 200 and 300 seconds in the error for the particle filter, and (less pronounced) for SFLS. These represent a period when a large number of particles typically became erroneous, and diverged significantly from the target path. In both cases, the error was subsequently corrected, although the SFLS tended to correct it sooner and adjust the particle weights to improve accuracy further. Indeed, the amount of time by which the SFLS corrected the error sooner was almost precisely the length of the lag used. By contrast, the peaks are absent for both ordinary BS-PS and BS-PS on SFL smoothed data (combined smoother); since the erroneous particles were later discarded, they did not feature in the trajectories that BS-PS produced.

We also note that BS-PS and the combined smoother have very similar levels of RMSE, and that they both consistently outperform the particle filter and the SFL smoother, though not by a great amount for the first half of the data set. It appears that it is precisely in those situations where particles diverge, only to re-converge on the correct state later, that backwards simulation has the greatest impact on accuracy.

Figure 3 gives more of an overview of the results. In order to prevent the peak in the particle filter error from skewing the results unreasonably, the cut-off weighted RMSE was averaged over the time period in Figure 1 for each run, and this is displayed in a box and whisker plot. Error measures which were deemed to be anomalous (outside $\sim 2.7\sigma$ assuming a normal distribution) are shown separately as crosses. We see clearly that smoothing has improved the estimation of the target state. BS-PS has very significantly improved the accuracy of the estimation, though slightly less so when running in combination with SFLS.

Finally, Figure 2 shows how weighted RMSE varies as measurement noise is increased; this was averaged over 5 runs for each noise level (scale factors 0.1, 0.2, 0.5, 1, 2 and 5), with the BS-PS producing 50 trajectories each time due to computational constraints. The results show that the conclusions about relative performance of the four algorithms hold for all but the highest noise level. At the highest noise level, the standard deviation for RMSE is the highest, so the relative increase in error for BS-PS is likely to be an artefact of the small number of tests. Figure 2 also serves to illustrate the dependency of the smoothing algorithms on the particle filter; as noise is increased, the particle filter performs worse, and then so too do all of the smoothing algorithms.

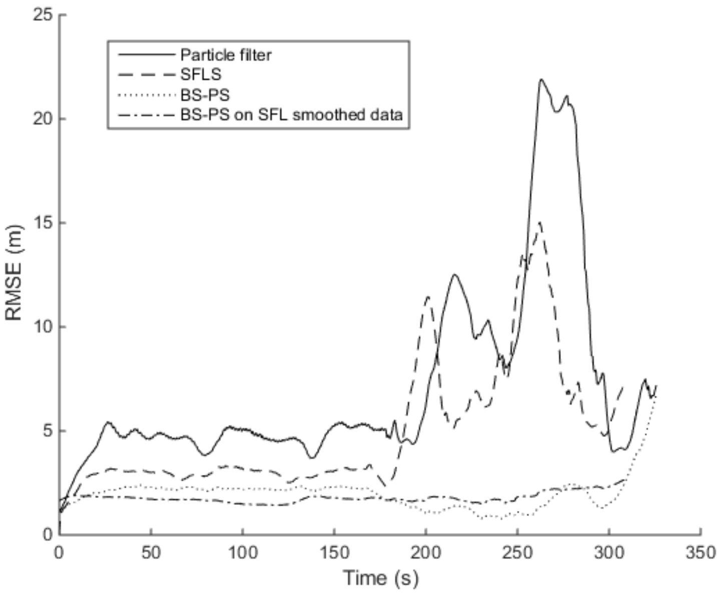

Figure 1: average weighted RMSE over the course of the data set

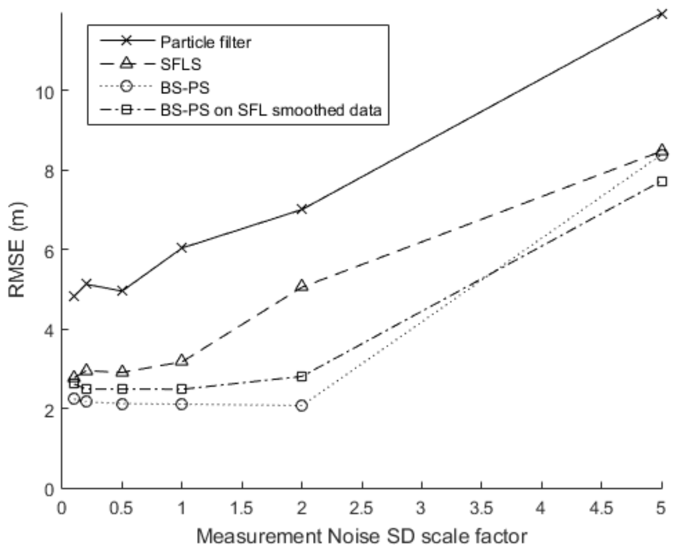

Figure 2: average weighted RMSE with measurement noise level

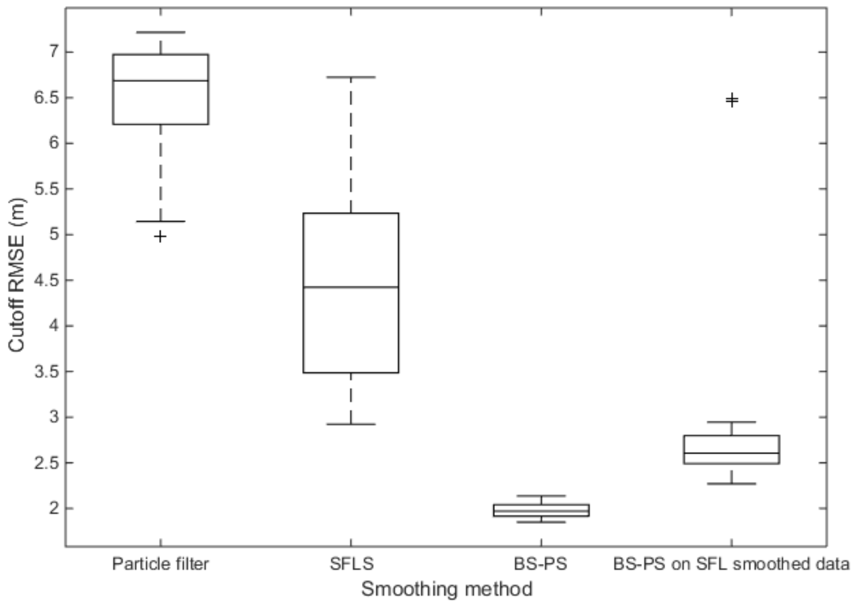

Figure 3: cut-off weighted RMSE for each run

B. Real data

Figure 4 shows the average weighted RMSE at the four predetermined waypoints along the path taken. In order to eliminate the effects of randomness as much as possible, 5 data sets for the same route were collected, and all four methods were run 5 times on each data set. Once again the particle filter used 50,000 particles and the SFLS had a lag of 30 time steps, and the BS-PS routines here produced 50 trajectories each time.

While the results are distinctly mixed, we see that smoothing in general does significantly improve accuracy. The unsmoothed particle filter gives the worst or second worst performance at all waypoints. The only absolutely consistent relationship in accuracy is between the particle filter and SFLS; here smoothing always improves accuracy, although only marginally at waypoint 2. It is also worth noting that the combined smoothing method outperforms ordinary BS-PS at waypoints 1, 3 and 4.

Figure 5 shows the positions of the particles at waypoint 4 in a single run for all four methods, as well as the true position of the waypoint (red circle). It clearly illustrates the ability of smoothing to refine state estimation through the elimination of erroneous particles, as well as the ability of BS-PS to further refine estimations given by SFLS. Note, however, that while it certainly tightens the estimation of SFLS, it does not necessarily improve it; it is most effective in situations when SFLS is incapable of reducing the estimation to a single particle cloud by itself.

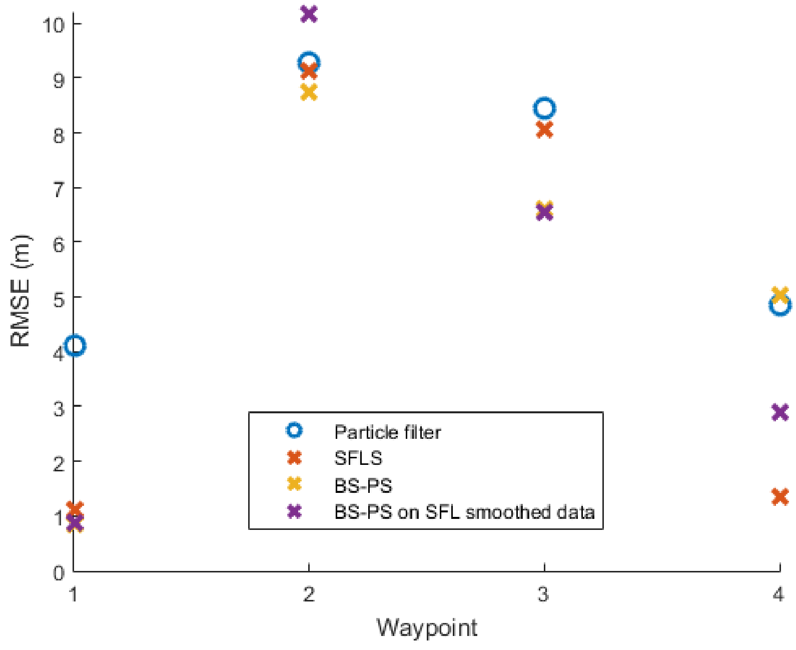

Figure 4: average weighted RMSE at each waypoint

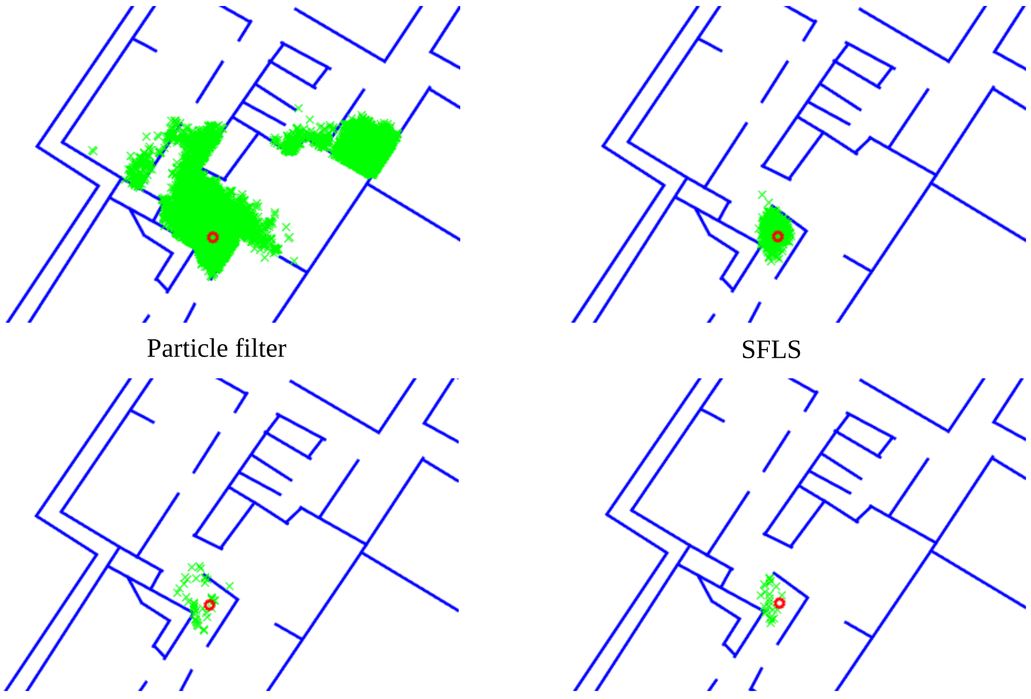

Figure 5: particle positions and true location at waypoint 4

VII. COMPUTATIONAL COMPLEXITY

Both the particle filter and SFLS are capable of following data in real time, though with a fixed delay in the case of the SFLS. In both of the experiments presented in Section V, however, data was collected in advance rather than streamed into the filter. The real time capabilities of these methods instead manifested themselves as the processing time being significantly less than the length of the data set itself. Here we focus instead on the computational cost of implementing BS-PS.

Processing was done using an Intel® i7-4770 3.40GHz CPU.

A. *Simulated data*

The simulated data set was approximately 600 seconds long (although this is only an estimate based on how long each step would take), which the particle filter and SFLS could process in 150 seconds.

To produce 100 trajectories for this data, the BS-PS routine running on the plain particle filter took an average of ~ 45 minutes ($2{,}700$ seconds).

However, BS-PS running on the SFL smoothed data, which has slightly improved performance compared to BS-PS on the plain particle filter, took an average of just 54 seconds, running in less than one tenth of the length of the data set. The factor 50 reduction in time taken is consistent with what one might expect given that SFLS typically only kept $700 < N_k < 2{,}000$ particles at each time step. The memory requirements (purely for running BS-PS) were also much reduced, from 872MB to 30MB.

B. *Real data*

The real data set was roughly half the length of the simulated one, lasting 308 seconds. 50 BS-PS trajectories took an average of ~ 15 minutes (900 seconds) to calculate using the plain particle filter output. By contrast, running this on the SFL smoothed data took an average of just 9 seconds. This further reduction in runtime, now by a factor of 100, came about primarily because SFLS kept even fewer particles for the real data than it did for the simulated data, typically in the range $300 < N_k < 1{,}200$ out of the total $50{,}000$. The memory required was reduced from 443MB to 7.5MB.

VIII. CONCLUSION

A. *Comparison of Results for Different Techniques*

In this paper, we presented two different smoothing techniques for particle filters. Sequential fixed-lag smoothing provides an efficient way of improving state estimation, but can be outperformed by the more costly backwards simulation particle smoother.

We found that a very cost-effective way to utilise the increased performance of BS-PS in situations where the particle filter requires a large number of particles is to smooth the output of the particle filter using SFLS first, in order to reduce the number of particles, before proceeding. In our example of the indoor tracker, the reduction in processing time required was dramatic, and accuracy attained by BS-PS was also slightly improved in the real data example. The table below presents a summary of the results for both the simulated (SD scale 2) and real data.

The results also make it clear, however, that no single one of SFLS and BS-PS outperforms the other in a truly consistent manner. This raises the possibility of further research into the precise nature of the situations in which each smoothing algorithm is the optimal choice.

		Particle filter		SFLS		BS-PS		Combined smoothing	
		Simulated	Real	Simulated	Real	Simulated	Real	Simulated	Real
RMSE (m)	Mean	7.16	6.67	4.96	4.91	1.98	5.31	3.27	5.12
	95 percentile	24.1	9.45	15.8	9.26	3.04	13.29	4.16	10.72
Results availability		Real time		Fixed delay		After post-processing		After post-processing	
Processing time (s)		150	90	150	90	2850	990	204	99
Additional memory		N/A	N/A	72MB	72MB	872MB	443MB	102MB	79MB

B. *Relevance of Techniques for Different Applications*

As a tracking algorithm, particle filtering techniques are suitable for a range of applications beyond the indoor tracking example presented here. Tracking of aircraft, ships, submarines, road vehicles and people using tracking technologies such as radar, sonar and video may all benefit from particle filter approaches when the measurement is noisy and non-linear in the system state variables. The more interesting question is when SFLS, BS-PS or combined smoothing should be preferred to a standard particle filter in real applications.

In scenarios where tracking in "up-to-the-second" real-time is necessary, none of the smoothing approaches here will be suitable as they inherently require future information to improve the tracking estimate.

Both BS-PS and combined smoothing are post-processing approaches which require the whole data set to be available before processing begins. These techniques are more appropriate for scenarios where the problem is to reconstruct the path that a target took. Relevant scenarios might include post-analysis of the path of an uncooperative target, post-analysis of a military exercise or retrospective analysis of traffic patterns (including patterns of flow through a building or along a flight path). It would of course be possible to bring these approaches closer to real-time by periodically running the full algorithm on the data set collected up to a particular point in time. However, this would lead to a computational burden beyond that described in the table above.

SFLS is relevant for scenarios in-between real-time and post-processing by giving results at a fixed delay. This would be relevant for scenarios where decisions based on the tracker output do not need to be taken instantly and where a more accurate estimate at a slight delay is better than a less accurate estimate now. The results using SFLS reported above were obtained by introducing only a 30 second time delay, which might be acceptable in some operational scenarios.

REFERENCES

[1] Simo Särkkä, 2013. "Bayesian Filtering and Smoothing," Cambridge University Press.

[2] Dan Simon, 2006. Chapter 9.3, "Optimal State Estimation," Wiley-Blackwell.

[3] B. Ristic, S. Arulampalam and N. Gordon, 2004. Chapter 3.5, "Beyond the Kalman Filter," Artech House.

[4] R. Douc, A. Garivier, E. Moulines and J. Olsson, 2011. Page 9, "Sequential Monte Carlo smoothing for general state space hidden Markov models," *Annals of Applied Probability*, **21**(6), 2109-2145.

[5] F. Lindsten and T. B. Schön, 2013. Chapter 3.1, "Backward Simulation Methods for Monte Carlo Statistical Inference," Now Publishers.

[6] E. Taghavi, F. Lindsten, L. Svensson, and T. B. Schön, 2013. "Adaptive stopping for fast particle smoothing," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* Vancouver, Canada.