UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

FACULTY OF ENGINEERING, BUILT ENVIRONMENT AND INFORMATION TECHNOLOGY

DEPARTMENT OF ELECTRICAL, ELECTRONIC AND COMPUTER ENGINEERING
`http://www.up.ac.za/eece`

EAI732: Intelligent Systems

Assignment 4: Sampling and Sequential Methods

Compiled By
Sholto Armstrong
14036071

12th August, 2018

DECLARATION OF ORIGINALITY

UNIVERSITY OF PRETORIA

Full names of student:	Sholto Armstrong
Student number:	14036071
Topic of work:	Assignment 4

Declaration

1. I understand what plagiarism is and am aware of the University's policy in this regard.
2. I declare that this assignment report is my own original work. Where other people's work has been used (either from a printed source, Internet or any other source), this has been properly acknowledged and referenced in accordance with departmental requirements.
3. I have not used work previously produced by another student or any other person to hand in as my own.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as his or her own work.

SIGNATURE: ______________________ DATE: 13 June 2018

TABLE OF CONTENTS

1 Introduction

It is often the case that one is presented with a series of measurements, from which a systems internal state is to be inferred. When approaching this problem, one can attempt to use each measurement alone in order to determine the internal state. However, this is not trivial as there is often noise involved. Furthermore, one is not always able to measure all the components of the internal state. This is due to the fact that the measurements are usually a function of various components in the current state. Sequential methods allow the knowledge of the state at previous time steps to influence the estimate of a systems current state. This often leads to more accurate estimates of the systems state. In this paper, two successful algorithms, which solve this problem, are to be explored. The first algorithm, which is widely used across multiple problems, is the Kalman filter[1]. This algorithm attempts to successively approximate the current state of the system. This is done by first predicting the systems current state and then evaluating what the measurement of the current state would be. It then attempts to correct errors in its approximation. The second algorithm which we will consider is the Particle filter[2]. This algorithm randomly places a number of particles on the state space. These are then used to estimate the posterior probability of a state given the current measurement. In this report, both the particle filter and the Kalman filter will be implemented from first principles in order to gain an understanding of how these algorithms work. They will then be tested on various problems to evaluate their performance and to explore their limitations.

2 Extended Kalman Filter

In order to derive the Kalman filter, we first need to formulate the problem. We begin by defining the state vector of a system at time k as $\mathbf{x}_k$ and the measured value as $\mathbf{z}_k$. Here $\mathbf{x}_k$ is an $n \times 1$ vector and $\mathbf{z}_k$ is an $m \times 1$ vector. Equation (2.1) shows how $\mathbf{x}_k$ is related to $\mathbf{x}_{k-1}$ and Eq. (2.2) shows how $\mathbf{z}_k$ is obtained from $\mathbf{x}_k$. Here $\mathbf{f}(\mathbf{x})$ is the a process function, which returns an $n \times 1$ vector, and $\mathbf{h}(\mathbf{x})$ and observation function which returns an $m \times 1$ vector. The variables $\mathbf{w}_{k-1}$ and $\mathbf{v}_k$ are zero-mean noise vectors with the covariances of $\mathbf{Q}_k$ ($n \times n$) and $\mathbf{R}_k$ ($m \times m$) respectively.

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{w}_{k-1} \tag{2.1}$$

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k \tag{2.2}$$

As shown in [3], we begin by deriving the prediction step of the algorithm. For this we assume an initial estimate of the state vector is known $\mathbf{x}_0^a$. We also make a crucial assumption that the error of this prediction is Gaussian distributed with a covariance of $\mathbf{P}_0$. We denote the forested value of the state vector at time k as $\mathbf{x}_k^f$, which is the expected value of $\mathbf{x}_k$ given the previous measurements ($\mathcal{D}_{k-1}$). In order to make calculation simpler, we obtain a first Taylor series expansion for $\mathbf{f}(\mathbf{x})$ and $\mathbf{h}(\mathbf{x})$ about $\mathbf{x}_k^a$ and $\mathbf{x}_k^f$ respectively. This is shown in Eqs. (2.3) and (2.4).

$$\mathbf{f}(\mathbf{x}) \approx \mathbf{f}\left(\mathbf{x}_{k-1}^a\right) + \mathbf{J}_\mathbf{f}\left(\mathbf{x} - \mathbf{x}_{k-1}^a\right) \tag{2.3}$$

$$\mathbf{z}_k \approx \mathbf{h}\left(\mathbf{x}_{k-1}^f\right) + \mathbf{J}_\mathbf{h}\left(\mathbf{x} - \mathbf{x}_{k-1}^f\right) \tag{2.4}$$

Here we have defined $\mathbf{J}_\mathbf{f}$ in Eq. (2.5) and $\mathbf{J}_\mathbf{h}$ in Eq. (2.6).

$$\mathbf{J}_\mathbf{f} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} \tag{2.5}$$

$$\mathbf{J_h} = \begin{pmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_n}{\partial x_1} & \frac{\partial h_n}{\partial x_2} & \cdots & \frac{\partial h_n}{\partial x_n} \end{pmatrix} \tag{2.6}$$

We also define $e_k = x_k - x_k^a$ and $e_k^f = x_k - x_k^f$. We can now use Eq. (2.3) to write the expected value of $\mathbf{f}(\mathbf{x})$ given $\mathcal{D}_{k-1}$. This is done in Eq. (2.7).

$$\begin{aligned} E\left(\mathbf{f}(\mathbf{x})|\mathcal{D}_{k-1}\right) = \mathbf{x}_k^f &\approx \mathbf{f}\left(\mathbf{x}_{k-1}^a\right) + \mathbf{J_f}E\left(e_{k-1}|\mathcal{D}_{k-1}\right) \\ &= \mathbf{f}\left(\mathbf{x}_{k-1}^a\right) \end{aligned} \tag{2.7}$$

From this we can use Eq. (2.1) and Eq. (2.3) to write e_{k-1}^f in terms of e_{k-1}. This is done in Eq. (2.8).

$$\begin{aligned} e_k^f &= x_k - x_k^f \\ &= \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{w}_{k-1} - \mathbf{f}\left(\mathbf{x}_{k-1}^a\right) \\ &\approx \mathbf{J_f}e_{k-1} + \mathbf{w}_{k-1} \end{aligned} \tag{2.8}$$

We can now determine $\mathbf{P}_k^f$ as shown in Eq. (2.9)

$$\begin{aligned} \mathbf{P}_k^f &= E\left(e_k^f(e_k^f)^T\right) \\ &= \mathbf{J_f}E\left(e_{k-1}e_{k-1}^T\right)\mathbf{J_f}^T + E\left(\mathbf{w}_{k-1}\mathbf{w}_{k-1}^T\right) \\ &= \mathbf{J_f}\mathbf{P}_{k-1}\mathbf{J_f}^T + \mathbf{Q}_{k-1} \end{aligned} \tag{2.9}$$

Now that we are able to predict the state vector of a system with its covariance, we would like to correct this prediction by taking into account the measurement received at time step k $\mathbf{z}_k$. This can be done by calculating the difference in the expected value of $\mathbf{z}_k$ and the actual received measurement. This error is then added to the current predicted state vector $\mathbf{x}_k^f$ with a scalar gain of $\mathbf{K}_k$, called the Kalman gain. This is similar to the step in a gradient decent process. However, it is possible to calculate the exact value of $\mathbf{K}_k$, which makes the system converge in one step for a linear $\mathbf{f}$ and $\mathbf{h}$. This update process is shown in Eq. (2.10).

$$\begin{aligned} \mathbf{x}_k^a &= \mathbf{x}_k^f + \mathbf{K}_k(\mathbf{z}_k - E(\mathbf{h}(\mathbf{x}_k)|\mathcal{D}_k)) \\ &\approx \mathbf{x}_k^f + \mathbf{K}_k(\mathbf{z}_k - \mathbf{h}(\mathbf{x}_k^f)) \end{aligned} \tag{2.10}$$

We can now determine $\mathbf{P}_k$ as shown in [3] the result is summarised in Eq. (2.11).

$$\begin{aligned} \mathbf{P}_k &= E\left(e_k E_k^T\right) \\ &= \mathbf{P}_k^f - \mathbf{K}_k\mathbf{J_h}\mathbf{P}_k^f - \mathbf{P}_k^f\mathbf{J_h}^T\mathbf{K}_k^T + \mathbf{K}_k\mathbf{J_h}\mathbf{P}_k^f\mathbf{J_h}^T\mathbf{K}_k^T + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^T \end{aligned} \tag{2.11}$$

The Kalman gain is then determined by minimising the trace of $\mathbf{P}_k$. The result is shown in Eq. (2.12)

$$\mathbf{K}_k = \mathbf{P}_k^f\mathbf{J_h}^T\left(\mathbf{J_h}\mathbf{P}_k^f\mathbf{J_h}^T + \mathbf{R}_k\right)^{-1} \tag{2.12}$$

We can now simplify $\mathbf{P}_k$ by substituting Eq. (2.12) into Eq. (2.11). The result is shown in Eq. (2.13)

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{J_h})\mathbf{P}_k^f \tag{2.13}$$

2.1 Implementation

The two steps of the algorithm are shown in Algorithms 1 and 2. These are run after one another in order to produce a single prediction of the systems state. To initialise the algorithm a suitable $\mathbf{x}_0^a$ and $\mathbf{P}_0$ are selected. These are problem specific and should be selected depending on the criteria of the problem. If an initial estimate of $\mathbf{x}_0^a$ is known, as is the case where the starting position of a robot is known, then this initial estimate should be used. However, if this is not the case $\mathbf{x}_0^a$ should be set to the expected value of $\mathbf{x}_0$. The covariance $\mathbf{P}_0$ should be initialised to the certainty of the initial state vector $\mathbf{x}_0^a$. If $\mathbf{x}_0^a$ is known to a large degree, one could set the covariance as small. In contrast, one could set the covariance large to effectively create a uniform distribution over the posterior of $\mathbf{x}_0^a$. It is common practice to set $\mathbf{P}_0 = p \times \mathbf{I}$ as correlations in $\mathbf{x}_0^a$ is seldomly known.

Algorithm 1 EKF - Prediction

```
1: procedure EKF-PREDICTION($\mathbf{x}_{k-1}^a$, $\mathbf{P}_{k-1}$)
2:     $\mathbf{x}_k^a \leftarrow \mathbf{f}\left(\mathbf{x}_{k-1}^a\right)$
3:     $\mathbf{P}_k^f \leftarrow \mathbf{J}_\mathbf{f}\mathbf{P}_{k-1}\mathbf{J}_\mathbf{f}^T + \mathbf{Q}_{k-1}$
4: return $\mathbf{x}_k^a$, $\mathbf{P}_k^f$
5: end procedure
```

Algorithm 2 EKF - Correction

```
1: procedure EKF-CORRECTION($\mathbf{x}_k^f$, $\mathbf{P}_k^f$, $\mathbf{z}_k$)
2:     $\mathbf{K}_k \leftarrow \mathbf{P}_k^f\mathbf{J}_\mathbf{h}^T\left(\mathbf{J}_\mathbf{h}\mathbf{P}_k^f\mathbf{J}_\mathbf{h}^T + \mathbf{R}_k\right)^{-1}$
3:     $\mathbf{x}_k^a \leftarrow \mathbf{x}_k^f + \mathbf{K}_k(\mathbf{z}_k - \mathbf{h}(\mathbf{x}_k^f))$
4:     $\mathbf{P}_k \leftarrow (\mathbf{I} - \mathbf{K}_k\mathbf{J}_\mathbf{h})\mathbf{P}_k^f$
5: return $\mathbf{x}_k^a$, $\mathbf{P}_k$
6: end procedure
```

One can improve the accuracy this algorithm by iterating the correction step until convergence. This serves to reduce the errors caused by the first order approximations of the functions $\mathbf{f}(\mathbf{x})$ and $\mathbf{h}(\mathbf{x})$. The initial prediction step of this algorithm remains the same, however the correction step is replaced with Algorithm 3. This algorithm is named the Iterated Extended Kalman Filter.

Algorithm 3 IEKF - Correction

```
1: procedure IEKF-CORRECTION($\mathbf{x}_k^f$, $\mathbf{P}_k^f$, $\mathbf{z}_k$, $\alpha$)
2:     $\mathbf{x}_k^a \leftarrow \mathbf{x}_k^f$
3:     repeat
4:         $\mathbf{x}_k^h \leftarrow \mathbf{x}_a^k$
5:         $\mathbf{x}_k^a$, $\mathbf{P}_k \leftarrow$ EKF-Correction$\left(\mathbf{x}_k^a, \mathbf{P}_k^a, \mathbf{z}_k\right)$
6:     until $||\mathbf{x}_k^h - \mathbf{x}_k^a|| < \alpha$
7: return $\mathbf{x}_a^f$, $\mathbf{P}_k$
8: end procedure
```

3 Particle Filter

In Section 2, we make the assumption that $p(\mathbf{x}_k|\mathcal{D}_k) = \mathcal{N}(\mathbf{x}_k|\mathbf{x}_k^a|\mathbf{P}_k)$. However, it is often the case that this distribution is multi-modal and hence cannot be properly approximated by a Gaussian function. Due to this, a new algorithm is needed to determine $p(\mathbf{x}_k|\mathcal{D}_k)$. It is possible to approximate this distribution with a finite number(N_s) of elements(particles), whereby each element carries an associated weight proportional to how lightly it is. This is formalised in Eq. (3.1). Here, $\delta(x)$ is the Dirac delta function and is introduced due to the shift between continuous and discrete time.

$$p(\mathbf{x}_k|\mathcal{D}_k) \approx \sum_{i=1}^{N_s} w_k^i \delta(\mathbf{x}_k - \mathbf{x}_k^i) \tag{3.1}$$

Here we have defined w_k^i as shown in Eq. (3.2), where $q(x_k^i)$ is an importance function denoting the importance of the particle x_k^i.

$$w_k^i = \frac{\frac{p(x_k^i)}{q(x_k^i)}}{\sum_N^{i=1} \frac{p(x_k^i)}{q(x_k^i)}} \tag{3.2}$$

By definition, as $N_s \to \infty$, the approximation will tend towards the true posterior distribution[4]. As shown in [5], we can determine the current weight from the previous weight and a likelihood function. This is shown in Eq. (3.3).

$$w_k^i = \alpha w_{k-1}^i \frac{p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{x}_{k-1})}{q(\mathbf{x}_k|\mathbf{x}_{0:k-1}, \mathbf{z}_{0:k})} \tag{3.3}$$

This update step can be used to calculate the weights at each time step k. However, there is a problem with this, as after a few time intervals, a large proportion of the weights will tend to converge towards zero. This means that a large proportion of the computation will be spent updating weights which have little to no contribution on the probability distribution $p(\mathbf{x}_k|\mathcal{D}_k)$. Furthermore, this has a negative effect on the accuracy of this algorithm. This problem is known as the degeneracy phenomenon and can be minimised by selecting a good importance function $q(\mathbf{x})$. The most optimal choice of an importance function is derived in [6], which is shown in Eq. (3.4).

$$q(\mathbf{x}_k|\mathbf{x}_{k-1}^i, \mathbf{z}_{0:k})_{opt} = \frac{p(\mathbf{z}_k|\mathbf{x}_k, \mathbf{x}_{k-1}^i)p(\mathbf{x}_k|\mathbf{x}_{k-1}^i)}{p(\mathbf{z}_k|\mathbf{x}_{k-1}^i)} \tag{3.4}$$

It is, however, not always possible to evaluate this importance function. Due to this, $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ is often used as an importance function. With this, the update to the weights is simplified to $w_k^i = \alpha w_{k-1}^i p(\mathbf{z}_k|\mathbf{x}_k)$. In order to compare the Particle Filter to the Extended Kalman Filter, we can assume that $p(\mathbf{z}_k|\mathbf{x}_k) = \mathcal{N}(\mathbf{z}_k|\mathbf{h}(\mathbf{x}_k), \mathbf{R}) = \mathcal{N}(\mathbf{h}(\mathbf{x}_k)|\mathbf{z}_k, \mathbf{R})$ and that $p(\mathbf{x}_k|\mathbf{x}_{k-1}) = \mathcal{N}(\mathbf{x}_k|\mathbf{f}(\mathbf{x}_{k-1}), \mathbf{Q})$. This forms first algorithm to be implemented, known as the sequential importance sampling algorithm, shown in Algorithm 4. Note that special care was taken to evaluate the weight in the log domain. This is done to ensure computational stability as it is often the case that the likelihood function my be small. For this reason, the scaling of the weights is performed as shown in Algorithm 5.

Algorithm 4 The Sequential Importance Sampling Algorithm

```
1: procedure SIS($\mathbf{x}_{k-1}$, $\mathbf{w}_{k-1}$)
2:     $\mathbf{v} \sim \mathcal{N}(0, \mathbf{Q})$
3:     $\mathbf{x}_k \leftarrow \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{v}$
4:     $\mathbf{logw}_k \leftarrow \mathbf{logw}_{k-1} + ln(\mathcal{N}(\mathbf{h}(\mathbf{x}_k)|\mathbf{z}_k, \mathbf{R}))$
5:     $\mathbf{logw}_k \leftarrow ScaleLog(\mathbf{logw}_k)$
6:     $maxi \leftarrow argmax(\mathbf{logw}_k)$
7: return $\mathbf{x}_k^{(maxi)}$, $\mathbf{x}_k$, $\mathbf{logw}_k$
8: end procedure
```

Algorithm 5 Log Scaling Algorithm

```
1: procedure ScaleLog(logw)
2:     $b \leftarrow max(\mathbf{logw})$
3:     $\alpha \leftarrow b + ln(sum(exp(\mathbf{logw} - b)))$
4: return $\mathbf{logw}_k - \alpha$
5: end procedure
```

This algorithm is still subject to the degeneracy phenomenon. Due to this, a better approach is needed. A logical starting point is to try and detect when the degeneracy problem is occuring. A method introduced by [7], uses a measure known as the effective number of samples. This is shown in Eq. (3.5).

$$\begin{aligned} N_{eff} &= \frac{N_s}{1 + Var(w_k)} \\ &\approx \frac{1}{\sum_{i=1}^{N_s} (w_k^i)^2} \end{aligned} \tag{3.5}$$

Using Eq. (3.5), we can re-sample $\mathbf{X}$ from $p(\mathbf{x}_k|\mathcal{D}_k)$ every time $N_{eff} \leq N_T$. This serves two purposes. Firstly, it reduces the number of samples with negligible weight. Secondly, it produces more partials in regions of high probability. This allows computing resources to be focused on regions which are more important. This leads to the generic particle filter shown in Algorithm 6.

Algorithm 6 The Generic Particle Filter

```
1: procedure GPF($\mathbf{x}_{k-1}$, $\mathbf{w}_{k-1}$)
2:     $\mathbf{v} \sim \mathcal{N}(0, \mathbf{Q})$
3:     $\mathbf{x}_k \leftarrow \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{v}$
4:     $\mathbf{logw}_k \leftarrow \mathbf{logw}_{k-1} + ln(\mathcal{N}(\mathbf{h}(\mathbf{x}_k)|\mathbf{z}_k, \mathbf{R}))$
5:     $\mathbf{logw}_k \leftarrow ScaleLog(\mathbf{logw}_k)$
6:     $maxi \leftarrow argmax(\mathbf{logw}_k)$
7:     $\mathbf{holdx} \leftarrow \mathbf{x}_k^{(maxi)}$
8:     $N_{eff} \leftarrow \frac{1}{\sum_{i=1}^{N_s} (w_k^i)^2}$
9:     if $N_{eff} \leq N_T$ then
10:        $\mathbf{x}_k, \mathbf{logw}_k \leftarrow$ resample $\mathbf{x}_k, \mathbf{logw}_k$ according to $p(\mathbf{x}_k|\mathcal{D}_k)$
11:     end if
12: return holdx, $\mathbf{x}_k$, $\mathbf{logw}_k$
13: end procedure
```

It is also possible to re-sample on every iteration. With this version of the algorithm, one does not have to add $\mathbf{logw}_{k-1}$ to $ln(\mathcal{N}(\mathbf{h}(\mathbf{x}_k)|\mathbf{z}_k, \mathbf{R}))$ in order to obtain $\mathbf{logw}_k$. This algorithm is known as Sampling Importance Resampling and is shown in Algorithm 7.

Finally, jitter can be added to the above algorithms by adding additional zero-mean Gaussian noise to $\mathbf{X}_k$ with a covariance of $K\mathbf{I}$. This stops the particle filter becoming too certain of a point, which may lead to divergence[2].

Algorithm 7 The Sampling Importance Resampling Algorithm

```
1: procedure SIR($\mathbf{x}_{k-1}$, $\mathbf{w}_{k-1}$)
2:     $\mathbf{v} \sim \mathcal{N}(0, \mathbf{Q})$
3:     $\mathbf{x}_k \leftarrow \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{v}$
4:     $\mathbf{logw}_k \leftarrow ln(\mathcal{N}(\mathbf{h}(\mathbf{x}_k)|\mathbf{z}_k, \mathbf{R}))$
5:     $\mathbf{logw}_k \leftarrow ScaleLog(\mathbf{logw}_k)$
6:     $maxi \leftarrow argmax(\mathbf{logw}_k)$
7:     $\mathbf{holdx} \leftarrow \mathbf{x}_k^{(maxi)})$
8:     $N_{eff} \leftarrow \frac{1}{\sum_{i=1}^{N_s}(w_k^i)^2}$
9:     $\mathbf{x}_k, \mathbf{logw}_k \leftarrow$ resample $\mathbf{x}_k, \mathbf{logw}_k$ according to $p(\mathbf{x}_k|\mathcal{D}_k)$
10: return holdx, $\mathbf{x}_k$, $\mathbf{logw}_k$
11: end procedure
```

4 Datasets

The datasets, which will be used to test the algorithms discussed above, are described in this section.

4.1 Dataset A: Linear Combination of Sinusoids

The first problem to be tested is a simple non-linear problem, where the measured data is a noisy mixture of sinusoids. Here, the input sinusoids are to be inferred. The properties of this problem are listed as follows:

1. $n = 4$
2. $m = 2$
3. $N_d =$ Number of datapoints $= 100$
4. $\mathbf{f}(\mathbf{x}) = \begin{pmatrix} sin(x_3 + 4\frac{\pi}{N_d}) \\ cos(x_4 + \frac{\pi}{N_d}) \\ x_3 + 4\frac{\pi}{N_d} \\ x_4 + \frac{\pi}{N_d} \end{pmatrix}$
5. $\mathbf{h}(\mathbf{x}) = \begin{pmatrix} x_1 + 0.8x_2 \\ 4x_1 + x_2 \end{pmatrix}$
6. $\mathbf{J_f}(\mathbf{x}) = \begin{pmatrix} 0 & 0 & cos(x_3 + 4\frac{\pi}{N_d}) & 0 \\ 0 & 0 & 0 & -sin(x_4 + \frac{\pi}{N_d}) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
7. $\mathbf{J_h}(\mathbf{x}) = \begin{pmatrix} 1 & 0.8 & 0 & 0 \\ 4 & 1 & 0 & 0 \end{pmatrix}$

Furthermore, the following parameters were used for this dataset unless specified otherwise.

1. $\mathbf{P}_0 = 0.5 * \mathbf{I}$
2. $N_s = 8000$
3. $\mathbf{X}_0^{pf} \sim \mathcal{N}(\mathbf{0}, \mathbf{P}_0)$
4. $\mathbf{Q} = 0.01\mathbf{I}$
5. $\mathbf{R} = 0.01\mathbf{I}$
6. $K = 0.4$
7. $\mathbf{x}_0^{kf} = \mathbf{0}$
8. $\mathbf{x}_0^{real} = \begin{pmatrix} 0 & -1 & 0 & \pi \end{pmatrix}^T$

4.2 Dataset B: Bearings Only Motion Tracking

The second problem, from [2], is a motion tracking problem, where one is only given the bearings of an object. From this, one must infer the objects x and y coordinates. The properties of this problem are listed as follows:

1. $n = 4$
2. $m = 1$
3. $N_d =$ Number of datapoints $= 24$
4. $\mathbf{f}(\mathbf{x}) = \begin{pmatrix} x_1 + x_2 \\ x_2 \\ x_3 + x_4 \\ x_4 \end{pmatrix}$
5. $\mathbf{h}(\mathbf{x}) = \begin{pmatrix} arctan2(x_3, x_1) \end{pmatrix}$
6. $\mathbf{J_f}(\mathbf{x}) = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
7. $\mathbf{J_h}(\mathbf{x}) = \begin{pmatrix} -\frac{x_3}{x_1^2 + x_3^2} & 0 & \frac{x_1}{x_1^2 + x_3^2} & 0 \end{pmatrix}$

Furthermore, the following parameters were used for this dataset unless specified otherwise.

1. $K = 0.2$
2. $\mathbf{P}_0 = \begin{pmatrix} 0.5^2 & 0 & 0 & 0 \\ 0 & 0.005^2 & 0 & 0 \\ 0 & 0 & 0.03^2 & 0 \\ 0 & 0 & 0 & 0.01^2 \end{pmatrix}$
3. $N_s = 4000$
4. $\mathbf{x}_0^{kf} = \begin{pmatrix} 0 & 0 & 0.4 & -0.05 \end{pmatrix}^T$
5. $\mathbf{X}_0^{pf} \sim \mathcal{N}(\mathbf{x}_0^{kf}, \mathbf{P}_0)$
6. $$\begin{aligned} \mathbf{Q} &= \begin{pmatrix} 0.5 & 0 \\ 1 & 0 \\ 0 & 0.5 \\ 0 & 1 \end{pmatrix} \cdot q\mathbf{I}_2 \cdot \begin{pmatrix} 0.5 & 0 \\ 1 & 0 \\ 0 & 0.5 \\ 0 & 1 \end{pmatrix}^T \\ &= q \begin{pmatrix} 0.25 & 0.5 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0 & 0 & 0.25 & 0.5 \\ 0 & 0 & 0.5 & 1 \end{pmatrix} \end{aligned}$$
7. $\mathbf{R} = \begin{pmatrix} 0.005^2 \end{pmatrix}$
8. $\mathbf{x}_0^{real} = \begin{pmatrix} -0.05 & 0.001 & 0.7 & -0.055 \end{pmatrix}^T$

4.3 Dataset C: Distance and Bearings Motion Tracking

The final problem considered is the same as discussed in Section 4.2. The only difference is that the perpendicular distance to the object is also measured. This value is squared to ease the computation. The parameters for both problems remain the same apart from the ones listed below:

1. $m = 2$
2. $\mathbf{h}(\mathbf{x}) = \begin{pmatrix} arctan2(x_3, x_1) \\ x_1^2 + x_3^2 \end{pmatrix}$
3. $\mathbf{J_h}(\mathbf{x}) = \begin{pmatrix} -\frac{x_3}{x_1^2+x_3^2} & 0 & \frac{x_1}{x_1^2+x_3^2} & 0 \\ 2x_1 & 0 & 2x_3 & 0 \end{pmatrix}$
4. $\mathbf{R} = \begin{pmatrix} 0.005^2 & 0 \\ 0 & 0.0001 \end{pmatrix}$

5 Evaluation Procedure

This section discusses the tests which are to be performed.

5.1 Experiment 1: Performance Test with Dataset A

5.1.1 *Aim*

To test whether of not the above mentioned algorithms work on a simple dataset, discussed in Section 4.1, and how well they perform.

5.1.2 *Procedure*

The following is performed in order to obtain the results:

1. Each of the algorithms are initialised with the parameters discussed in Section 4.1.
2. Noisy **X** and **Z** data-points are generated using Eqs. (2.1) and (2.2).
3. Each of the algorithms are run through with **Z**.
4. The error is calculated as $mean(||\mathbf{X}_{1:2} - \mathbf{X}^{est}_{1:2}||)$, where only the first two elements are considered as multiple correct answers exist for the last two parameters.
5. This is run 100 times in order to calculate various statistics.
6. A single example, with the associated **X** and $\mathbf{X}^{est}$ is plotted for each algorithm.

5.2 Experiment 2: Convergence Test with Dataset A

5.2.1 *Aim*

To visually display how the various algorithms converge to the correct output.

5.2.2 *Procedure*

The following is performed in order to obtain the results:

1. The GPF and EKF are initialised with the parameters discussed in Section 4.1.
2. Noisy **X** and **Z** data-points are generated using Eqs. (2.1) and (2.2).
3. Each of the algorithms are run through with **Z**.
4. **X** and $\mathbf{X}^{est}$ is plotted for each algorithm.

5.3 Experiment 3: Performance Test with Dataset B

5.3.1 Aim

To compare the overall performance of the various algorithms discussed above, on a non-linear non-Gaussian problem.

5.3.2 Procedure

The following is performed in order to obtain the results:

1. Each of the algorithms are initialised with the parameters discussed in Section 4.2.
2. Noisy **X** and **Z** data-points are generated using Eqs. (2.1) and (2.2).
3. Each of the algorithms are run through with **Z**
4. The error is calculated as $mean(||\mathbf{X} - \mathbf{X}^{est}||)$.
5. This is run 1000 times in order to calculate various statistics.

5.4 Experiment 4: Convergence Test with Dataset B

5.4.1 Aim

To gain insight into how both the Particle Filter and the Extended Kalman Filter converges to a solution.

5.4.2 Procedure

The following is performed in order to obtain the results:

1. IEKF and GPF with jitter are initialised with the parameters discussed in Section 4.2.
2. Noisy **X** and **Z** data-points are generated using Eqs. (2.1) and (2.2).
3. Each of the algorithms are run through with **Z**
4. The error is calculated as $mean(||\mathbf{X} - \mathbf{X}^{est}||)$.
5. This is run 1000 times in order to calculate various statistics.

5.5 Experiment 5: Starting Position Test with Dataset B

5.5.1 *Aim*

To compare the effects of the starting position on Dataset B.

5.5.2 *Procedure*

The following is performed in order to obtain the results:

1. IEKF and GPF with jitter are initialised with the parameters discussed in Section 4.2, apart from $\mathbf{x}_0^{kf} = \mathbf{x}_0^{real}$ and $\mathbf{x}_0^{pf}$ is adjusted accordingly. Furthermore we adjust the certainty of the estimate to account for these changes as follows:
$$\mathbf{P}_0 = 10^{-6} \cdot \mathbf{I}_4 + ((\mathbf{x}_0^{real}) - (\mathbf{x}_0^{kf}))^2 \odot \mathbf{I}_4$$
2. Noisy $\mathbf{X}$ and $\mathbf{Z}$ data-points are generated using Eqs. (2.1) and (2.2).
3. Each of the algorithms are run through with $\mathbf{Z}$
4. The error is calculated as $mean(||\mathbf{X} - \mathbf{X}^{est}||)$.
5. This is run 1000 times in order to calculate various statistics.
6. The initial position $\mathbf{x}_0^{kf}$ is shifted one axis at a time and plots are drawn of the accuracy.

5.6 Experiment 6: Number of Particles Test with Dataset B

5.6.1 *Aim*

To compare the effects of the number of particles on the Particle filter algorithms.

5.6.2 *Procedure*

The following is performed in order to obtain the results:

1. The Particle Filter algorithms are initialised with the parameters discussed in Section 4.2, apart from the number of partials are looped from $N_s = 100$ to $N_s = 10000$ using steps of 100.
2. Noisy $\mathbf{X}$ and $\mathbf{Z}$ data-points are generated using Eqs. (2.1) and (2.2).
3. Each of the algorithms are run through with $\mathbf{Z}$
4. The error is calculated as $mean(||\mathbf{X} - \mathbf{X}^{est}||)$.
5. This is run 1000 times in order to calculate various statistics.
6. Plots are drawn of the accuracy for each N_s.

5.7 Experiment 7: Performance Test with Dataset C

5.7.1 Aim

To compare the overall performance of the various algorithms discussed above, on a non-linear Gaussian problem.

5.7.2 Procedure

The following is performed in order to obtain the results:

1. Each of the algorithms are initialised with the parameters discussed in Section 4.3.
2. Noisy $\mathbf{X}$ and $\mathbf{Z}$ data-points are generated using Eqs. (2.1) and (2.2).
3. Each of the algorithms are run through with $\mathbf{Z}$
4. The error is calculated as $mean(||\mathbf{X} - \mathbf{X}^{est}||)$.
5. This is run 1000 times in order to calculate various statistics.

6 Results

This section summarises the results of the experiments performed.

6.1 Experiment 1: Performance Test with Dataset A

The results of experiment 1, detailed in Section 5.1, are shown in Fig. 1 and Table 1.

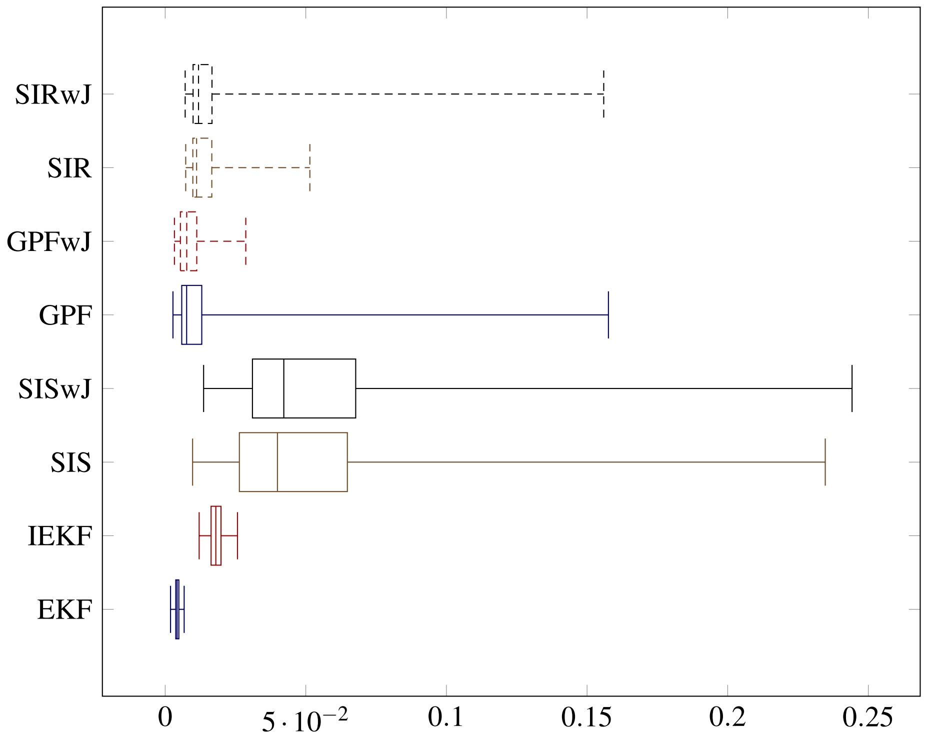

Figure 1. Results of the various algorithms tested on dataset A

Algorithm	EKF	IEKF	SIS	SISwJ	GPF	GPFwJ	SIR	SIRwJ
Mean RMSE$(1e^{-3})$	4.33	18.06	53.93	53.25	13.32	9.20	14.30	17.03

Table 1. Mean RMSE of the various algorithms on Dataset A

6.2 Experiment 2: Convergence Test with Dataset A

The results of experiment 2, detailed in Section 5.2, are shown in Figs. 2 and 3.

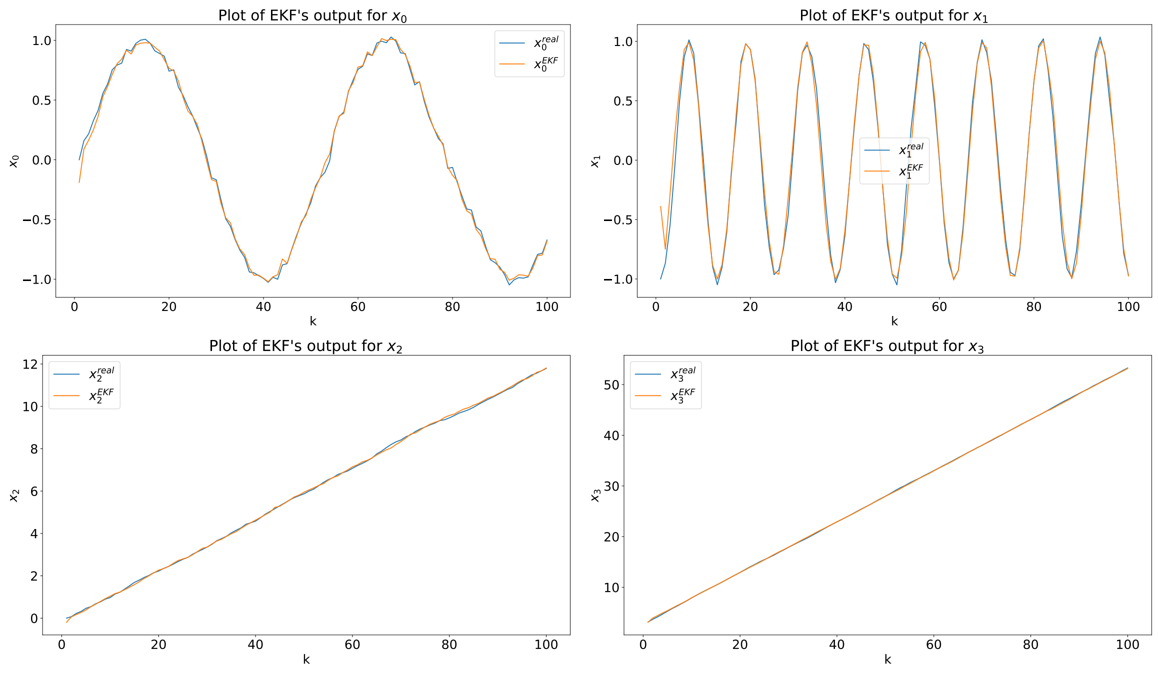

Figure 2. Results of EKF plotted with the actual X values produced from dataset A

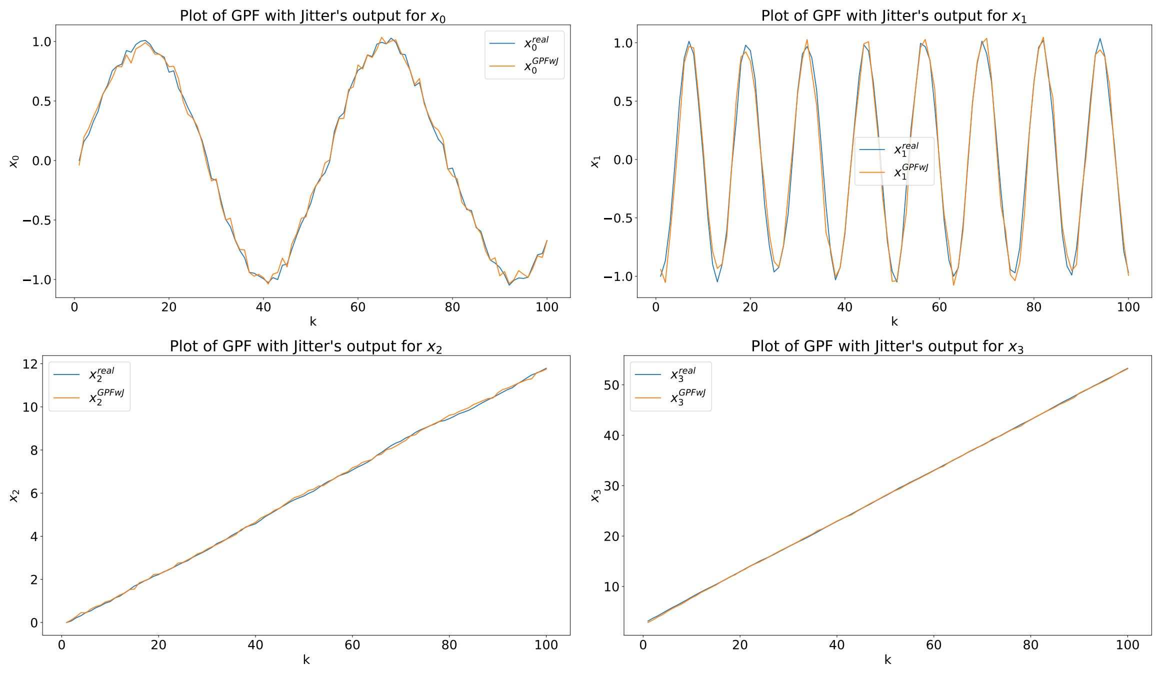

Figure 3. Results of PF plotted with the actual X values produced from dataset A

6.3 Experiment 3: Performance Test with Dataset B

The results of experiment 3, detailed in Section 5.3, are shown in Figs. 1 and 4 and Table 1.

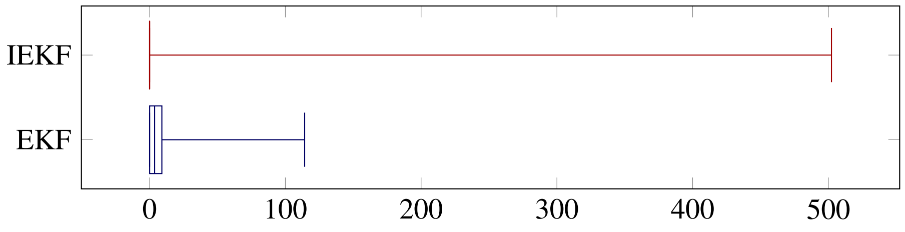

Figure 4. Results of the EKF based algorithms tested on dataset B

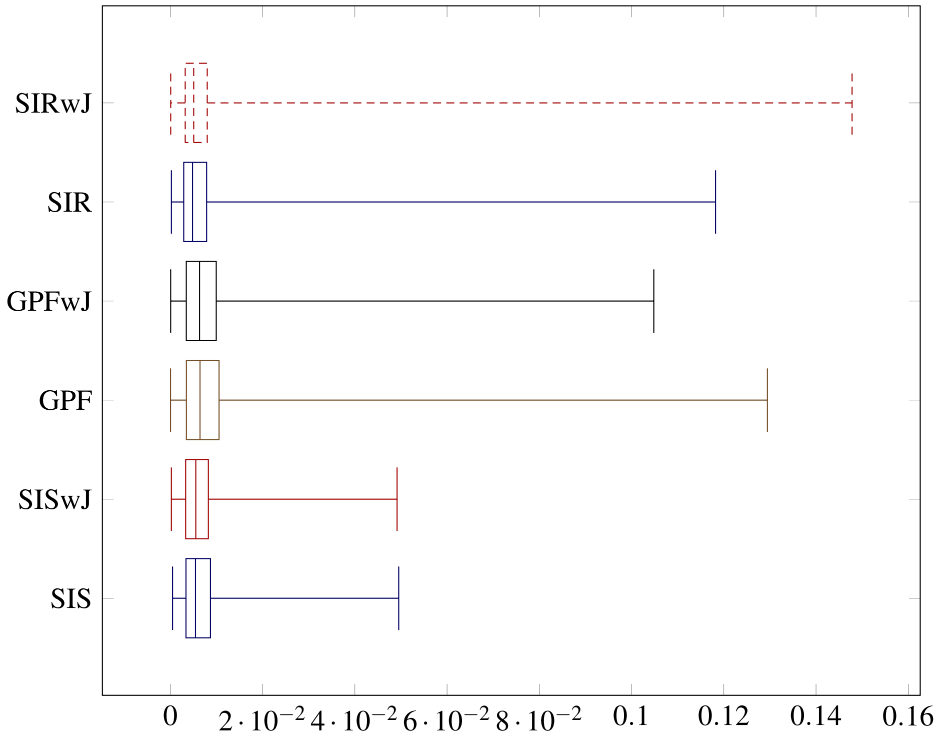

Figure 5. Results of the PF based algorithms tested on dataset B

Algorithm	EKF	IEKF	SIS	SISwJ	GPF	GPFwJ	SIR	SIRwJ
Mean RMSE$(1e^{-2})$	651.94	1461.18	0.68	0.67	0.84	0.82	0.68	0.71

Table 2. Mean RMSE of the various algorithms on Dataset B

6.4 Experiment 4: Convergence Test with Dataset B

The results of experiment 4, detailed in Section 5.4, are shown in Figs. 6 to 9. Note that the number of particles in a bit have been scaled to sum to 1. Furthermore, the probability distribution of the EKF has been scaled to the maximum value of the particle filter.

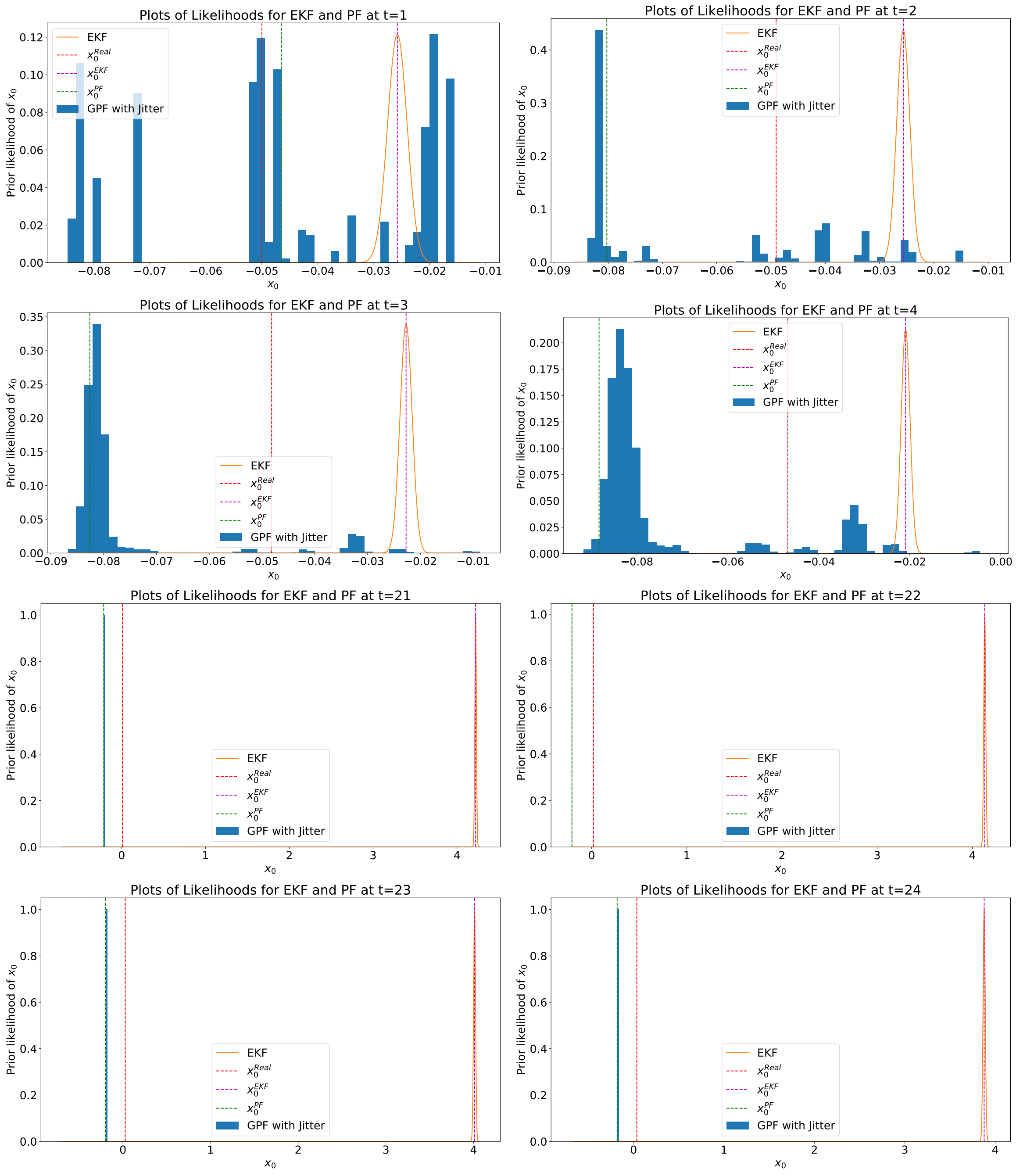

Figure 6. Plot of the various likelihood distributions for the first and last 4 time-steps over x_0

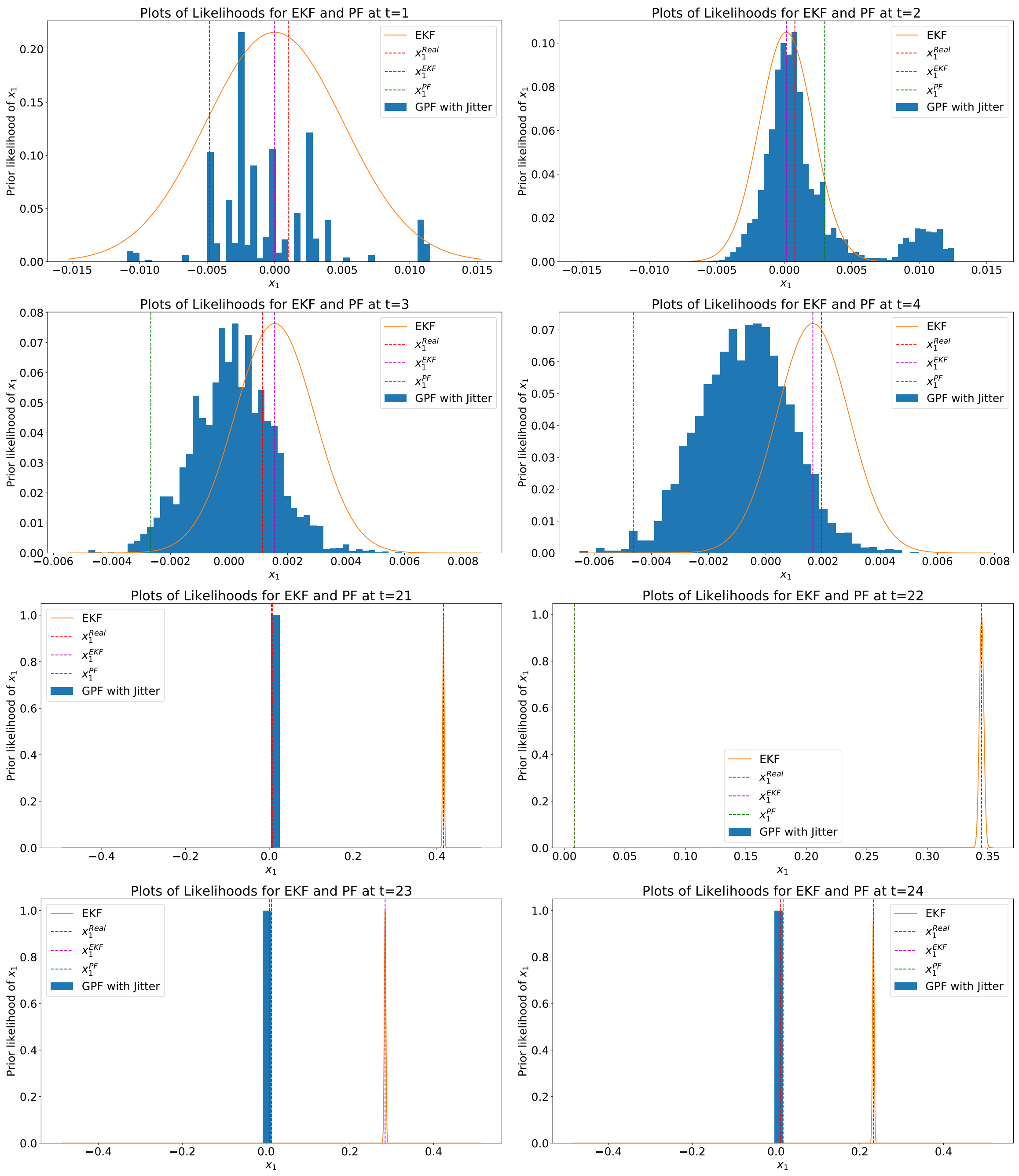

Figure 7. Plot of the various likelihood distributions for the first and last 4 time-steps over x_1

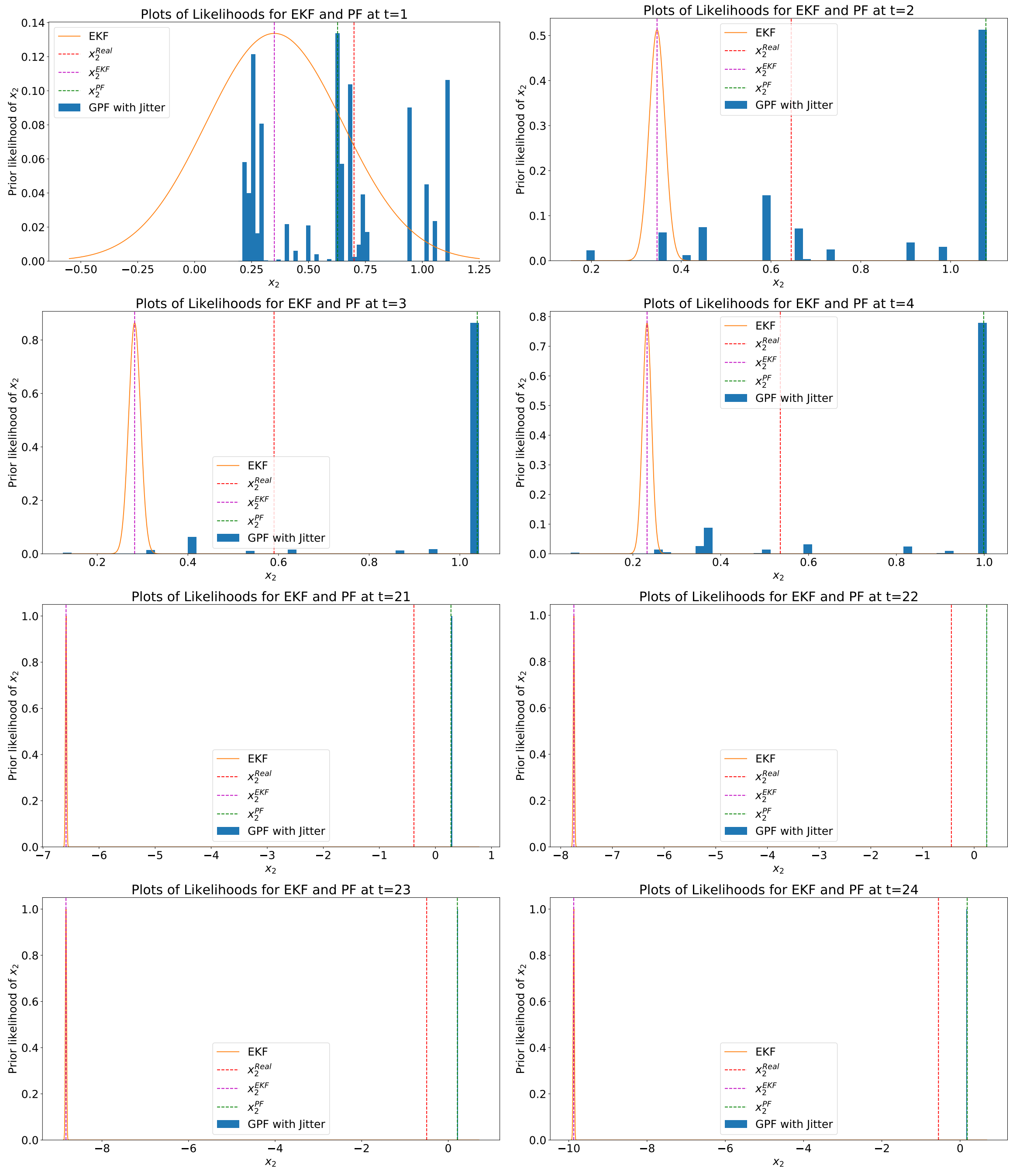

Figure 8. Plot of the various likelihood distributions for the first and last 4 time-steps over x_2

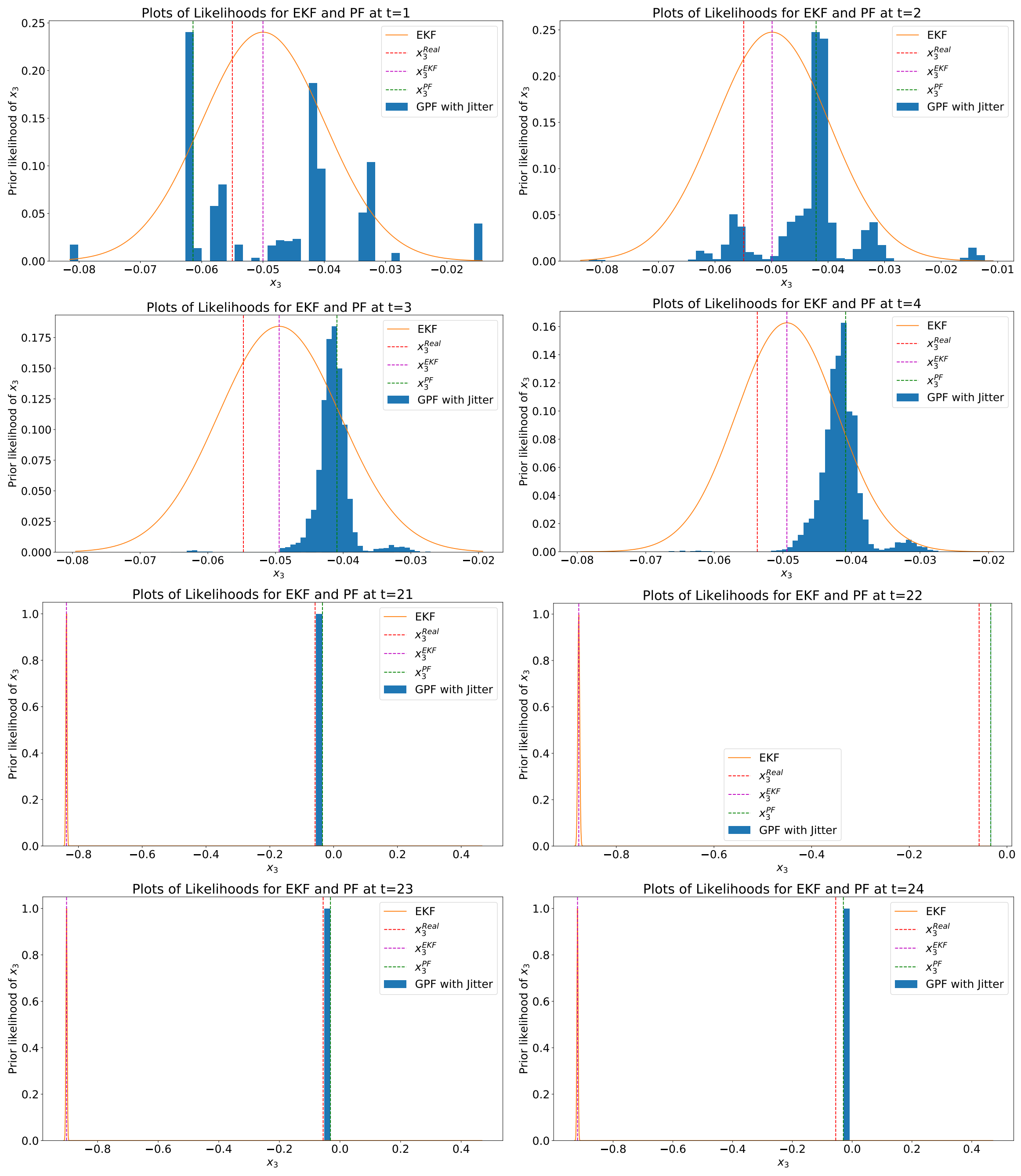

Figure 9. Plot of the various likelihood distributions for the first and last 4 time-steps over x_3

6.5 Experiment 5: Starting Position Test with Dataset B

The results of experiment 5, detailed in Section 5.5, are shown in Figs. 10 and 11.

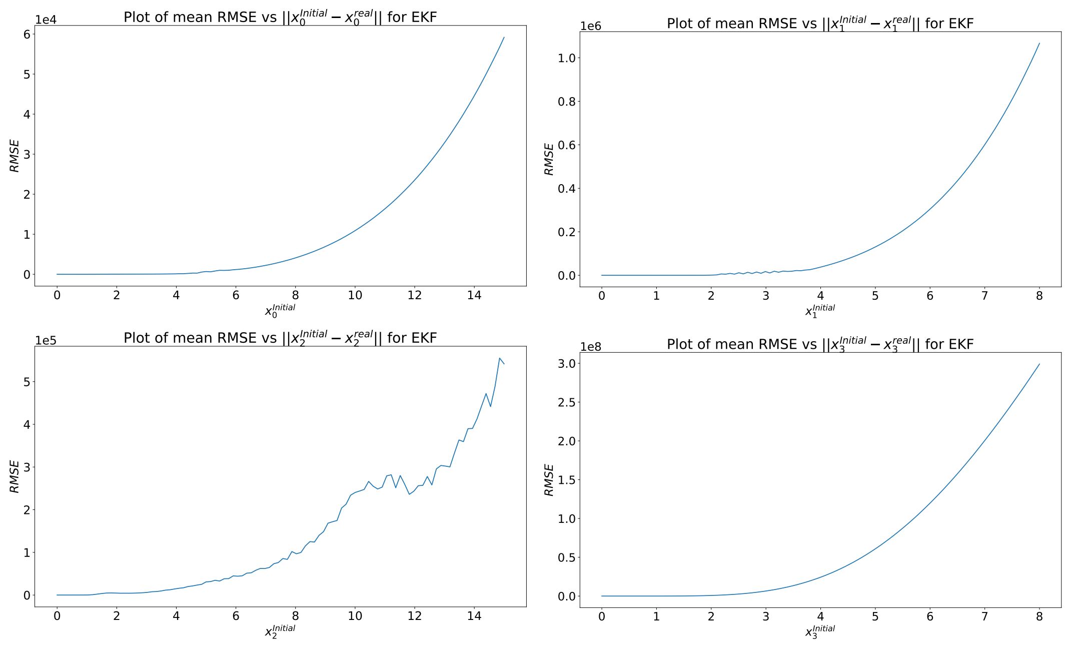

Figure 10. Plot of the RMSE, for EKF, vs the distance of the starting position from the real position over each dimension of x

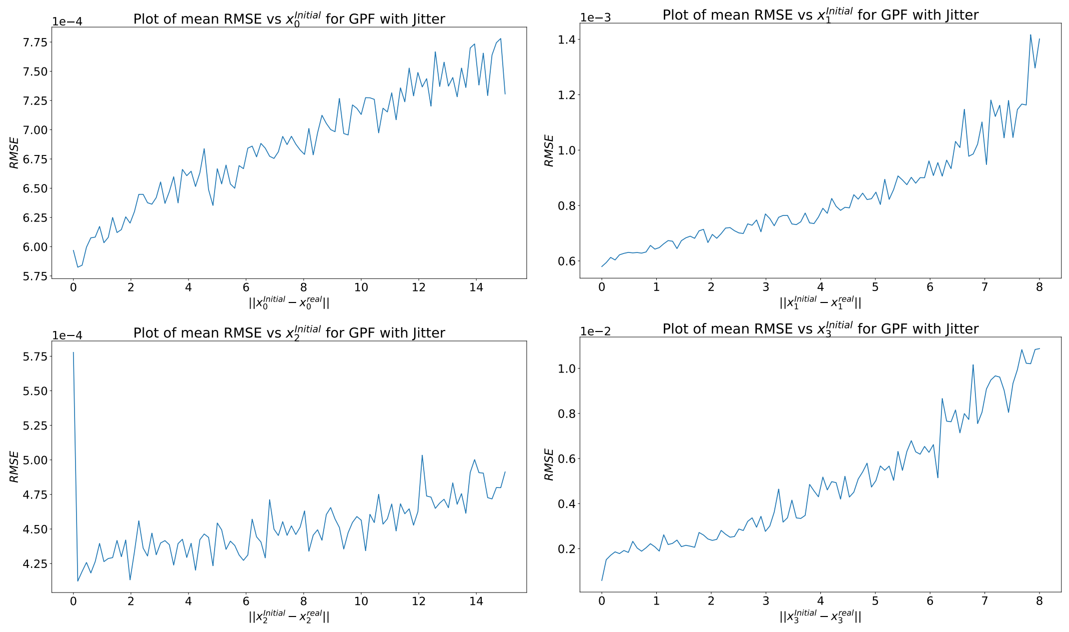

Figure 11. Plot of the RMSE, for PF, vs the distance of the starting position from the real position over each dimension of x

6.6 Experiment 6: Number of Particles Test with Dataset B

The results of experiment 6, detailed in Section 5.6, are shown in Fig. 12.

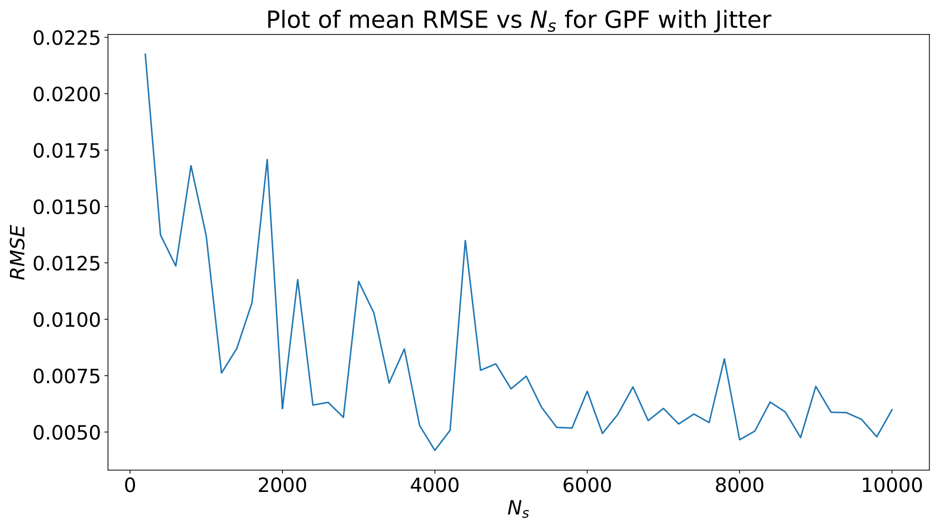

Figure 12. Plot showing the effect of N_s on the RMSE

6.7 Experiment 7: Performance Test with Dataset C

The results of experiment 7, detailed in Section 5.7, are shown in Fig. 13 and Table 3.

Algorithm	EKF	IEKF	SIS	SISwJ	GPF	GPFwJ	SIR	SIRwJ
Mean RMSE$(1e^{-3})$	5.43	0.31	0.89	1.38	0.14	0.23	0.16	0.44

Table 3. Mean RMSE of the various algorithms on Dataset C

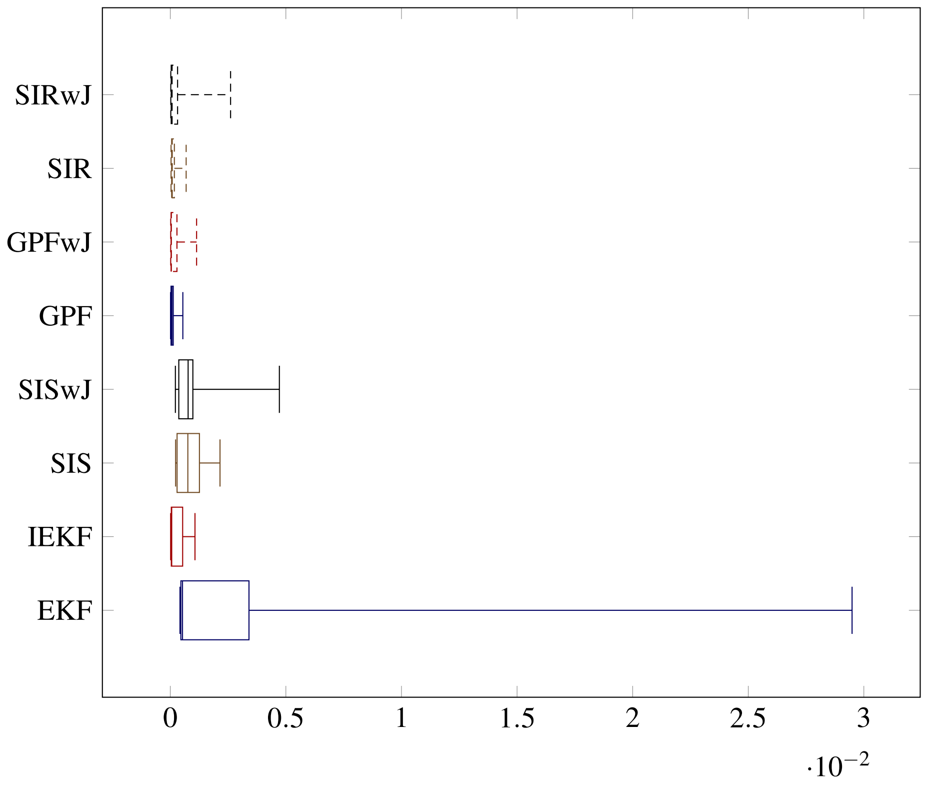

Figure 13. Results of the various algorithms tested on dataset C

7 Discussion

When looking at the results of dataset A, from Sections 6.1 and 6.2, one can see that both the EKF and the particle filters are able to perform extremely well with a simple non-linear dataset. In this situation the EKF-based algorithms outperform the particle filters. This is due to the simplicity of the measurement function, $h(x)$, as the EKF's first order approximation is accurate. Furthermore, this example only has one possible mode of answers. Due to this, the EKF is able to converge to the solution using its Gaussian approximation to its certainty. This is in contrast to dataset B as this dataset is multi-modal, as can be seen in Section 6.4. The particle filter is able to model this multi-modal posterior probability density, which the EKF is unable to do. This contributes to the errors of the EKF based algorithms shown in Fig. 4. One would expect to see a low error for a small fraction of the data, where the algorithm just happened to converge to the correct mode. Furthermore, one would expect to see a large RMSE for situations where the EKF converged to the wrong mode. Once the algorithm has converged to the wrong mean, further measurements could cause the posterior probability to spread out. This causes the algorithm to completely diverge to the wrong state vector, which creates large spikes in RMSE over multiple runs. The particle filter is able to recover if it has diverged to the wrong state vector as a single particle left in a region of lower probability is able to assist the algorithm recover from its mistake. This is the reason that the SIR particle filter has the largest upper-bound, as the re-sampling causes less particles to be in regions of lower probability. The SIS algorithm worked best in this situation as this system was highly multi-modal, hence many low probability particles contribute to the resiliency of the algorithm. In most cases the jitter helped with the resiliency as well for similar reasons.

For the datasets tested, IEKF did not appear to have much advantage over EKF and, in most cases, performed relatively badly. This is due to the fact that the multiple calls to the correction step of the algorithm can cause the algorithm to diverge. Hence, accuracy is gained at the cost of stability. IEKF

would be more suited for situations where the measurement function is not well locally approximated by the first order Taylor expansion.
The Particle filters are far more resilient across multiple functions than the Kalman filter as it is able to model more complex posterior likelihood functions. However, if the likelihood function isn't complex, as in dataset C, it may be better to use the Kalman Filter as it is faster than the particle filter for a single core processor. The most processor intensive part of the Kalman Filter is the matrix inversion in the correction step. Furthermore, efficient algorithms exist for the matrix inversion step and closed formed solutions exist for low dimensionality problems. However, it is often impossible to avoid problems with a complex posterior. Furthermore, particle filters are able to utilise multi-core processors and GPUs as each particle can be updated independently. Finally, particle filters have the advantage of concentrating the computation to the regions of the state space which are most important. This means that they are really effective in their use of computation resources.
One parameter which is important when using a particle filter is the number of particles. As shown in Section 6.6, the more particles, the lower the error. This follows a decaying exponential trend, which means that after a certain point, adding more particles will have a negligible effect on the error. A trivial approach to selecting the number of particles would be to determine how many particles can possible be processed with the time and processing power constraints provided. However, this could lead to a lot of waisted resources. An appropriate number of particles, to balance the processing power to accuracy trade-off, is problem dependant and is a function of the number of dimensions and the complexity of the functions $\mathbf{f}(\mathbf{x})$ and $\mathbf{h}(\mathbf{x})$. Due to this, the best approach would be to run the algorithm a few times, adjusting the number of partials. From this, one could plot the error vs N_s curve and select the number of particles based on the accuracy required. For dataset B, once can see that 4000 particles is sufficient as the RMSE, shown in Fig. 12, does not decrease much after 4000 particles.
The last aspect one could consider is the starting position. As shown in Section 6.5, with an accurate starting position, one is able to achieve accurate results. The further away one gets from the starting position, the less accurate the results are. The particle filter seems to be more tolerant to changes in this as it is more resilient. When considering the problem discussed in dataset B, one can see that the further the initial location is away from the starting position, the faster the object would have to move in-order to make a change in the measured angle. This could cause many modes as it is possible that the object is moving very slowly and close or fast and far away. Due to this, the particle filter will be better in this situation and the EKF will struggle to converge to the correct location. However, if the object is close and moving fast, there are less possibilities as it is unlikely that it is further away and moving even faster. Due to this, the EKF will perform better the closer and the faster the object is travelling.

8 Conclusion

In this paper we implemented two methods of handling sequential data, where an observation is made and data is to be inferred about the internal state of the object. We first implemented the Extended Kalman Filter, which provides a numerical method for analysing such a problem. However, it comes at the cost of assuming that the posterior distribution of the state variable is Normally distributed. This assumption was shown to be insufficient for some situations discussed in this paper. After implementing the Extended Kalman Filter, the Particle Filter was implemented. This attempts to approximate the posterior distribution with a number of samples, called particles. This algorithm works extremely well for situations where the posterior distribution is not Gaussian, however it requires more computation power. Furthermore, if no particles are placed in a favourable position, the algorithm is unable to converge.

9 References

[1] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[2] N. J. Gordon, D. J. Salmond, and A. F. Smith, “Novel approach to nonlinear/non-gaussian bayesian state estimation,” in *IEE Proceedings F (Radar and Signal Processing)*, vol. 140, no. 2. IET, 1993, pp. 107–113.

[3] G. A. Terejanu *et al.*, “Extended kalman filter tutorial,” *Department of Computer Science and Engineering, University at Buffalo. Available online: https://homes. cs. washington. edu/˜ todorov/courses/cseP590/readings/tutorialEKF. pdf (accessed on 16 February 2017)*, 2008.

[4] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, “A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking,” *IEEE Transactions on signal processing*, vol. 50, no. 2, pp. 174–188, 2002.

[5] L. Turner and C. Sherlock, “An introduction to particle filtering,” 2013.

[6] A. Doucet, S. Godsill, and C. Andrieu, “On sequential monte carlo sampling methods for bayesian filtering,” *Statistics and computing*, vol. 10, no. 3, pp. 197–208, 2000.

[7] N. Bergman, “Recursive bayesian estimation,” *Department of Electrical Engineering, Linköping University, Linköping Studies in Science and Technology. Doctoral dissertation*, vol. 579, p. 11, 1999.