On Particle Filters in Radar Target Tracking

E.F. Bauermeister

BRMETI001

Supervisor:

Assoc. Prof. D.W. O'Hagan

Department of Electrical Engineering, UCT

August 2016

A dissertation submitted to the Department of Electrical Engineering,

University of Cape Town,

in partial fulfillment of the requirements for the degree of

Master of Engineering specialising in *Radar.*

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the *IEEE Referencing* convention for citation and referencing. Each contribution to, and quotation in, this dissertation from the work(s) of other people, has been attributed, and has been cited and referenced.
3. This dissertation is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work.
5. This dissertation has been submitted to the *Turnitin* module and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor.

Signature of Author	Signed by candidate
Date	**2016 – 08 – 26**

Abstract

The dissertation focused on the research, implementation, and evaluation of particle filters for radar target track filtering of a maneuvering target, through quantitative simulations and analysis thereof. Target track filtering, also called target track smoothing, aims to minimize the error between a radar target's predicted and actual position. From the literature it had been suggested that particle filters were more suitable for filtering in non-linear/non-Gaussian systems. Furthermore, it had been determined that particle filters were a relatively newer field of research relating to radar target track filtering for non-linear, non-Gaussian maneuvering target tracking problems, compared to the more traditional and widely known and implemented approaches and techniques. The objectives of the research project had been achieved through the development of a software radar target tracking filter simulator, which implemented a sequential importance re-sampling particle filter algorithm and suitable target and noise models. This particular particle filter had been identified from a review of the theory of particle filters. The theory of the more conventional tracking filters used in radar applications had also been reviewed and discussed. The performance of the sequential importance re-sampling particle filter for radar target track filtering had been evaluated through quantitative simulations and analysis thereof, using predefined metrics identified from the literature. These metrics had been the root mean squared error metric for accuracy, and the normalized processing time metric for computational complexity. It had been shown that the sequential importance re-sampling particle filter achieved improved accuracy performance in the track filtering of a maneuvering radar target in a non-Gaussian (Laplacian) noise environment, compared to a Gaussian noise environment. It had also been shown that the accuracy performance of the sequential importance re-sampling particle filter is a function of the number of particles used in the sequential importance re-sampling particle filter algorithm. The sequential importance re-sampling particle filter had also been compared to two conventional tracking filters, namely the alpha-beta filter and the Singer-Kalman filter, and had better accuracy performance in both cases. The normalized processing time of the sequential importance re-sampling particle filter had been shown to be a function of the number of particles used in the sequential importance re-sampling particle filter algorithm. The normalized processing time of the sequential importance re-sampling particle filter had been shown to be higher than that of both the alpha-beta filter and the Singer-Kalman filter. Analysis of the posterior Cramér-Rao lower bound of the sequential importance re-sampling particle filter had also been conducted and presented in the dissertation.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Assoc. Prof. Daniel O'Hagan, for his willingness to undertake the task of supervising this dissertation. I would like to thank him for the excellent guidance and support given throughout the dissertation, his ideas and input into the dissertation topic, and his efforts in getting the dissertation into its final form.

I would like to thank the University of Cape Town (UCT) for the opportunity to complete this master's degree through their M.Eng (Radar) degree programme.

I would also like to acknowledge the Radar Masters programme convener, Assoc. Prof. Daniel O'Hagan, for the professional manner in which the programme had been conducted, and for the high academic quality he attained for the programme. An acknowledgement also goes to the various teaching assistants and academic support staff of the programme, who provided assistance when required.

I would like to thank my employer, the National Research Foundation (NRF) and the Square Kilometre Array – South Africa (SKA SA) for providing the funding to undertake this formal study through their Part-Time Education Assistance programme for employees. Furthermore, I thank them for the use of their resources in order to complete this dissertation, and the special leave arrangements in order to attend the required lectures.

Table of Contents

Index of Figures

Index of Tables

List of Symbols

α	position gain (alpha-beta filter)
α	reciprocal of the maneuver (acceleration) time constant (Singer-Kalman filter)
β	velocity gain (alpha-beta filter)
γ	acceleration gain (alpha-beta-gamma filter)
δ	jitter (radar measurements)
δ	perturbation (non-recursive Gauss-Newton filter)
ζ	normalized delta-time (radar measurements)
η	normalized time (radar measurements)
λ	acceleration gain constraint (alpha-beta-gamma-lambda filter)
λ	forgetting factor (recursive Gauss-Newton filter)
θ	azimuth angle (radar coordinates)
μ	damping factor (recursive Gauss-Newton filter)
μ	mean (general)
μ_X	population mean
ρ	slant range (radar coordinates)
σ	precision (standard deviation of measurement error)
σ	standard deviation (general)
σ_X	population standard deviation
σ^2	variance (general)
σ_X^2	population variance
τ	expected update period (radar measurements)
τ	pulse width (pulse radar)
ϕ	elevation angle (radar coordinates)

ϕ	transition (Kalman filter)
$\bar{x}$	sample mean
S_x	sample standard deviation
S_x^2	sample variance

Nomenclature

APF	Adaptive Particle Filter
APF	Auxiliary Particle Filter
ASIR	Auxiliary Sequential Importance Re-sampling (particle filter)
AWGN	Additive White Gaussian Noise
BS	Batch Size
CDF	Cumulative Distribution Function
CL	Confidence Level
CRLB	Cramér-Rao Lower Bound
EKF	Extended Kalman Filter
EMP	Expanded Memory Polynomial (filter)
ENU	East-North-Up
EPF	Evolutionary Particle Filter
FMP	Fading Memory Polynomial (filter)
FPGA	Field-Programmable Gate Array
GMAB	Growing-Memory / Alpha-Beta (filter)
GNF	Gauss-Newton Filter
GPU	Graphics Processing Unit
GPGPU	General Purpose Graphics Processing Unit
IEEE	Institute of Electrical and Electronics Engineers
IEKF	Iterated Extended Kalman Filter
IEPF	Improved Evolutionary Particle Filter
IMM	Interacting Multiple Model (filter)
IMM-NN	Interacting Multiple Model Nearest Neighbour (filter)
IMM-PDAF	Interacting Multiple Model Probabilistic Data Association Filter

IPPF	Independent Partition Particle Filter
MCMC	Markov Chain Monte Carlo
MCMC-IEKPF	Markov Chain Monte Carlo Iterated Extended Kalman Particle Filter
MCMC-PF	Markov Chain Monte Carlo Particle Filter
MVA	Minimum Variance Algorithm
NED	North-East-Down
NN	Nearest Neighbour
NP	Number of Particles
NP-PHDF	Noise-estimate Particle Probability Hypothesis Density Filter
PCRLB	Posterior Cramér-Rao Lower Bound
PDA	Probabilistic Data Association
PDAF	Probabilistic Data Association Filter
PDF	Probability Density Function
PF	Particle Filter
PPPF	Polynomial Predictive Particle Filter
PRF	Pulse Repetition Frequency
PRI	Pulse Repetition Interval
radar	RAdio Detection And Ranging
RBPF	Rao-Blackwellized Particle Filter
RCS	Radar Cross Section
RGNF	Recursive Gauss-Newton Filter
RMSE	Root Mean Squared Error
RPF	Regularized Particle Filter
SIS	Sequential Importance Sampling (particle filter)
SIR	Sequential Importance Re-sampling (particle filter)

SMC	Sequential Monte Carlo
SNR	Signal-to-Noise Ratio
TBD	Track-Before-Detect
TWS	Track-While-Scan (radar)
UI	User Interface
UKF	Unscented Kalman Filter
UPF	Unscented Particle Filter
VRPF	Variable Rate Particle Filter

Chapter 1 : Introduction

1.1 Overview

A radio detection and ranging (radar) system is an electromagnetic sensor that detects the location of objects through the reception of reflected electromagnetic energy [1]. The basic functional components of a general radar system are:

1. antenna;
2. transmitter;
3. receiver;
4. signal processor.

Each of these components are complex systems in themselves that have various subsystems that are not mentioned here. One of the components in some radar system's signal processor is the target tracking subsystem, which has the function of detecting and tracking targets. The target tracking subsystem functions through the implementation of a target tracking algorithm. The target tracking algorithm in the radar system's signal processor consists of four basic components [2], namely:

1. target detection;
2. target resolution;
3. target-to-track association;
4. target track filtering.

Target detection involves the conversion of radar measurements into target echoes, usually based on a detection threshold determined by a false alarm rate and probability of detection. Target resolution involves the resolving of closely spaced objects and clutter into individual targets, through techniques like Doppler processing (velocity resolution) or pulse compression (range resolution). Target-to-track association has the objective of assigning individual target measurements to potential tracks. Target track filtering has the objective of estimating a target's position, velocity and acceleration based on the current measurements. Target track filtering aims to minimize the error between the target's predicted and actual position, velocity and acceleration. A number of radar target tracking filter algorithms exist, with the most common and widely studied

being:

1. the alpha-beta filter and the Kalman filter with its numerous variations and extensions [3], [4], [5];
2. the Gauss-Newton and polynomial filters with its numerous variations and extensions [6], [7], [8].

However, the alpha-beta filter and the general Kalman filter only perform well in tracking targets with linear or non-maneuvering trajectories [9], [10], [11]. The Gauss-Newton filter has been shown to effectively track highly maneuvering targets under certain conditions [8], [12], [13]. The polynomial filter has been shown to be equivalent to one of the alpha-beta filter extensions, namely the alpha-beta-gamma filter [7], and thus has limitations in tracking non-linear/non-Gaussian systems [14]. The particle filter, as suggested by [15], is an optimal algorithm for maneuvering target tracking problems that performs better than the general Kalman filter and even the non-linear Extended Kalman Filter (EKF). This view is supported by the work of [16] on tracking targets with very low signal-to-noise ratio (SNR), and by the work of [17] on tracking targets in marine environments with high levels of sea clutter.

1.2 Statement of Research Problem

The dissertation will focus on the research, implementation, and evaluation of a particle filter for radar target track filtering of a maneuvering target, through quantitative simulations and analysis thereof.

1.3 Delineation of Research

Although a number of non-linear radar target tracking filters exist [9], this research will focus specifically on the evaluation of the particle filter and only for tracking a single target. The research conducted in this project is contained within these boundaries for the following reasons:

1. non-linear tracking is an active field of research, particularly relating to radar applications;
2. particle filter based radar target track filtering is a relatively newer field of research, particularly relating to radar applications;
3. particle filter based radar target track filtering is suggested to be more suited for non-linear, non-Gaussian maneuvering target tracking problems;
4. initially evaluation of single target tracking is done to determine the feasibility of extending

the research to multiple target scenarios in the future.

The target will be tracked after the detection phase of the radar system's signal processor, as track-before-detect (TBD) target tracking filters is a separate field of study that deals with tracking targets before they are declared as such. Furthermore, the particle filter will be evaluated through quantitative simulations and analysis thereof. It is beyond the scope of the present dissertation to achieve practical implementation of the tracking technique to a tracking radar system, but to do so is an area of future work.

1.4 Objectives of Research

The objectives of the research conducted for the dissertation are as follows:

1. develop a generic simulation framework for evaluating radar target tracking filters in Scilab 5.5.0 (http://www.scilab.org/), an open source MATLAB clone, with the following components:
    1. target model;
    2. noise model;
    3. target tracking filter model;
    4. display and data capture capability.
2. evaluate a particle filter algorithm for use as a radar target tracking filter through simulation and analysis thereof according the predefined metrics;
3. identify from the literature review any limitations in the field of research and extend the field of research by addressing these research limitations.

1.5 Research Methodology

The research methodology used in conducting the research for the dissertation consisted of the following components:

1. a literature review and critique;
2. software implementation of a particle filter algorithm;
3. quantitative (Monte Carlo) simulations;
4. analysis and evaluation of simulation results through defined performance metrics.

1.6 Dissertation Layout

The dissertation is divided into seven primary sections that are reflected as chapters and two secondary sections containing the references and the appendices. The layout of the dissertation is described to give an overview of the contents of each section:

- **Chapter 1 : Introduction**

  This chapter serves as an introduction to the dissertation and the research conducted for the dissertation.

- **Chapter 2 : Literature Review**

  The literature review and critique that has been undertaken is presented in this chapter.

- **Chapter 3 : Target Tracking Filter Theory**

  An overview of the theory of radar target tracking filters is presented in this chapter through the presentation of the most prominent target tracking filters from literature.

- **Chapter 4 : Particle Filter Theory and Operation**

  The theory and operation of the particle filter is discussed in depth in this chapter due to its relevance to the research conducted for the dissertation.

- **Chapter 5 : Radar Target Tracking Filter Simulator**

  The design, implementation and operation of the radar target tracking filter simulator is discussed in this chapter. The radar target tracking filter simulator is used for the quantitative simulation, evaluation and analysis of the particle filter as a radar target tracking filter in Chapter 6.

- **Chapter 6 : Quantitative Simulation, Evaluation and Analysis of Particle Filter**

  The quantitative simulation, evaluation and analysis of the particle filter as a radar target tracking filter is discussed in this chapter, using the radar target tracking simulator as described in Chapter 5.

- **Chapter 7 : Conclusions and Recommendations**

  The conclusions and recommendations of future work based on the research conducted for the dissertation, are presented in this chapter.

- **References**

  The references cited in the dissertation are listed in this section, using the Institute of Electrical and Electronics Engineers (IEEE) Referencing convention for citation and referencing.

- **Appendices**

  Supplementary material in support of the research conducted for the dissertation, like source code etc., is given in this section.

1.7 Summary

This chapter has presented a general overview of the project, focusing on the research problem, research objectives, and research methodology. An overview of the layout of the dissertation has also been presented. Chapter 2 will present the literature review and critique that is relevant to the present project.

Chapter 2 : Literature Review

2.1 Introduction

The literature review that has been undertaken for the research conducted for the dissertation and a discussion thereof, is presented in this chapter. The literature review focuses on particle filters and their implementation as radar target tracking filters, due to their relevance to the research conducted for the dissertation.

2.2 Literature Review

2.2.1 Context and Concepts

A concept-centric literature review has been performed to identify the recurring themes, focus areas, and persistent constraints addresses in the literature of the research field. From an initial review of the research field, the following key concepts, with a description of each concept, were identified for use in the concept-centric literature review's concept matrix, as shown in *Table 2.1*.

Table 2.1: Key Concepts of Literature Review

Concept	Description
Accuracy	the primary metric used in evaluating radar target tracking filters
Normalized Processing Time	the secondary metric used in evaluating radar target tracking filters
Root Mean Squared Error (RMSE) Metric	the metric used to compute the accuracy of a radar target tracking filter
Simulated	the primary research methodology of computer simulation is used in the research
Alternative Proposed	an alternative or improved radar target tracking filter architecture is proposed
Bi-static/Multi-static Radar	the research focuses on target tracking using bi-static/multi-static radar systems

Concept	Description
Multiple Objects	the research focuses on the tracking of multiple objects/targets
Passive Radar	the research focuses on target tracking using passive radar systems
Image Processing	the research focuses on target tracking using image processing techniques
Compared to Other Types	the performance of different radar target tracking filters are compared to one another

2.2.2 Concept Matrix

The concept matrix used for the literature review is shown in *Table 2.2*. For each row in the table which corresponds to a specific literature reference, the matching concept addressed in the literature is indicated in the corresponding concept column with an "X".

Table 2.2: Literature Review Concept Matrix

Concepts / References	Accuracy	Normalized Processing Time	RMSE Metric	Simulated	Alternative Proposed	Bi-static/Multi-static Radar	Multiple Objects	Passive Radar	Image Processing	Compared to Other Types
Arulampalam *et al.* [15]	X		X	X						X
Abdoul-Moaty *et al.* [18]	X	X	X	X	X	X				X
Liu *et al.* [19]	X	X		X						
Tilton *et al.* [20]	X	X	X	X						X
Zhan *et al.* [21]	X	X	X	X						
Zhang & Chen [22]	X	X	X	X						
Zhang *et al.* [23]	X	X	X	X	X					X

Concepts / References	Accuracy	Normalized Processing Time	RMSE Metric	Simulated	Alternative Proposed	Bi-static/Multi-static Radar	Multiple Objects	Passive Radar	Image Processing	Compared to Other Types
Yin *et al.* [24]	X	X	X	X	X					
Oudjane & Musso [25]	X			X	X					X
Tobias & Lanterman [26]				X	X	X		X	X	
Wang *et al.* [27]	X			X	X					X
Zhao *et al.* [28]	X		X	X	X					X
Ishibashi *et al.* [29]	X		X	X	X		X			
Kazem & Salut [30]	X			X						
Godsill & Vermaak [31]	X			X	X					
Godsill *et al.* [32]	X		X	X						
Guo *et al.* [33]			X	X						
Ng *et al.* [34]	X			X	X		X			
Ng *et al.* [35]	X			X	X		X			
Ulker *et al.* [36]	X		X	X	X					
Van der Merwe *et al.* [37]	X		X	X	X					X
Cevher *et al.* [38]	X			X			X		X	
Maskell *et al.* [39]	X		X	X		X	X			
Sobhani *et al.* [40]	X			X		X	X			X
Li & Wang [41]	X		X	X				X		X
Li *et al.* [42]	X		X	X				X		X
Li *et al.* [43]	X		X	X				X		X

Concepts / References	Accuracy	Normalized Processing Time	RMSE Metric	Simulated	Alternative Proposed	Bi-static/Multi-static Radar	Multiple Objects	Passive Radar	Image Processing	Compared to Other Types
Benavoli & Di Lallo [44]	X		X	X				X		X
Herman & Moulin [45]	X			X				X	X	X
Foo [46]	X		X	X		X				X
Soysal & Efe [47]	X		X	X		X				X
Chen & Huang [48]	X		X						X	

2.2.3 Review of Research Field

The primary research focus related to radar target tracking filters, is the tracking accuracy that can be achieved with various target tracking filters (see "Accuracy" column in *Table 2.2*). Accuracy as related to tracking filters, is described in Section 3.2.1. In approximately half the cases in *Table 2.2*, various different types of target tracking filters have been compared in order to determine which had a higher degree of accuracy under certain conditions (see "Compared to Other Type" column in *Table 2.2*). In the other cases, target tracking filters of the same type have been compared to one another to determine which had a higher degree of accuracy under certain conditions, while proposing improvements to that specific type of target tracking filter.

The primary metric used when comparing various target tracking filters, is the root mean squared error (RMSE) metric (see "RMSE metric" column in *Table 2.2*). This metric quantifies how closely the target tracking filter will match the target's true path. A secondary metric that has been used in some instances is the processing time taken by various target tracking filters [18], [19], [20], [21], [22], [23]. Computational complexity is however not a common metric used in evaluating target tracking filters, largely due to it not being a central research focus, although it is an important metric for the practical implementation of target tracking filters. It nevertheless has been mentioned, but not investigated in some instances [24].

The most common research methodology that is used in performing target tracking filter research, is

through computer simulation (see "Simulated" column in *Table 2.2*). This allows for the efficient evaluation of various target tracking filters without the need to implement and verify them in a practical radar system, which can be performed at a later stage. The simulations are largely conducted in two-dimensional Cartesian coordinates, but three-dimensional Cartesian coordinates are also used depending on the radar system and environment being modeled.

Some alternative architectures to the generic particle filter has been proposed and evaluated with respect to their use as radar target tracking filters. A summary of these proposed alternatives with their relevant references are listed:

1. particle filter with added smoothing [18];
2. particle filter with progressive correction / Rao-Blackwellised particle filter [25];
3. particle filter with particle placement [26];
4. evolutionary particle filter / improved evolutionary particle filter [27];
5. adaptive particle filter [28].

As stated at the beginning of Section 2.2.3, the primary research focus related to radar target tracking filters, is the tracking accuracy that can be achieved with various target tracking filters, and this has been the aim of the reported research efforts. The research efforts have specifically tried to improve the accuracy of existing particle filters.

In certain cases the particle filter has been combined with other types of tracking filters to produce a new hybrid type of filter, for example the noise-estimate particle probability hypothesis density filter (NP-PHDF) [29], and the Markov chain Monte Carlo iterated extended Kalman particle filter (MCMC-IEKPF) [23]. Furthermore, alternative implementations of the generic particle filter has been proposed that are not related to radar target tracking filters, as listed:

1. variable rate particle filter [31], [34], [35], [36];
2. unscented particle filter (UPF) [37];
3. polynomial predictive particle filter [24].

The majority of the research focused on tracking single targets with conventional, mono-static, active, radar systems. In the case of multiple target tracking, the targets were either treated as individual targets [29], [34], [35], or treated as an image processing problem [38]. Some multiple target tracking approaches have relied on the fact that the radar systems used were either bi-static or

multi-static in order to resolve and track individual targets [39], [40].

Some research has focused specifically on target tracking in passive radar systems using particle filters. The central focus of some of this research has been on the use of the time-of-arrival measurements from the passive radar system in tracking the targets with particle filters [41], [42], [43]. The other research focused on target tracking with particle filters in general without considering the impact of it being used in a passive radar system [26], [44], [45].

Similarly, in a few cases where target tracking with particle filtering in multi-static radar systems were considered, the fact that the radar system was multi-static in nature had no direct bearing on the research conducted [18], [46]. An exception was in the case where the geometry of the multi-static radar system was used in the target tracking process [47].

Some researchers have treated the problem of radar target tracking as an image processing problem [26], [38], [45], [48]. Image processing techniques have been applied to combine simultaneous tracking and classifying of targets based on their radar cross section (RCS) [45], [48], or where limited data or data of low quality was available from the radar [26], [38].

2.2.4 Critique of Research Field

A number of limitations have been identified from the research discussed so far, which has been addressed in this project. These limitations and the relevant sections of the dissertation that addresses these limitations, are as follows:

1. The quality of the external model of the target, that is the exact motion model used to represent the target, used for evaluating the performance of the radar target tracking filter is lacking or not fully justified. The external model that has been used in the project, and the quality thereof, is discussed in Section 5.4.2 and Section 5.4.3.
2. The noise model used for evaluating the performance of the radar target tracking filter is not clearly defined or assumed to be Gaussian, in the the literature reviewed that was relevant to this dissertation. A performance comparison of particle filter based radar target track filtering for both Gaussian and non-Gaussian noise models are thus lacking in the field of research. The definition of the different noise models that have been used in the project is presented in Section 5.5.
3. The number of simulation iterations used for evaluating the performance of the radar target tracking filter, and which has a direct impact on determining the accuracy of the radar target

tracking filter, is not reported in the literature reviewed. The theoretical justification of the number of simulation iterations that has been used in the project is presented in Section 6.3.

4. The performance of particle filter based radar target track filtering under simulated single-target, track-after-detect scenarios with missed target detections and transient detections, is not always fully explored in the field of research, except in the case of multi-sensor, multi-target tracking scenarios. The performance of particle filter based radar target track filtering under the conditions of:
    1. missed target detections is discussed in Section 6.6.1.2, and
    2. transient detections is discussed in Section 6.6.1.3.
5. The computational complexity of particle filters for radar target track filtering is rarely discussed in the field of research. The results of the evaluation of the normalized processing time of particle filter based radar target track filtering is presented in Section 6.6.2.

2.3 Summary

In this chapter a concept-centric review of the literature has been presented as it relates to radar target tracking filters and specifically to the particle filter and its use as a radar target tracking filter. The central issues that have been identified from the literature review are:

1. the primary research focus in the field of radar target tracking filters;
2. the primary and secondary metrics used in evaluating radar target tracking filters;
3. the common research methodology used;
4. some alternative particle filter architectures;
5. the specific radar systems and architectures modeled;
6. the limitations in the field of research.

The identification of these issues are important as it relates to the objectives of the present research project as stated in Chapter Error: Reference source not found. In Chapter 3, an overview of the theory of radar target tracking filters will be presented through a discussion of the most prominent radar target tracking filters from literature.

Chapter 3 : Target Tracking Filter Theory

3.1 Introduction

An overview of the theory of radar target tracking filters is presented in this chapter through the presentation of the most prominent radar target tracking filters from literature. Target tracking filter algorithms in radar target tracking can broadly be classified into two groups [2], namely:

1. parametric estimation;
2. stochastic state estimation.

Parametric estimation tracking filter algorithms function by assuming a perfect model for the motion of the target being tracked. On the other hand, stochastic state estimation tracking filter algorithms function by assuming an imperfect model for the motion of the target being tracked. Stochastic state estimation tracking filter algorithms are better to use in real world applications like radar, since they are considered to be more consistent and thus more accurate in tracking maneuvering targets [2]. For this reason, only stochastic state estimation tracking filter algorithms will be considered here. It should be noted that this chapter does not present an exhaustive list of tracking filter algorithms, but provides an overview of the salient tracking filter algorithms.

3.2 Introductory Concepts

3.2.1 Accuracy and Precision

The difference between accuracy and precision can be explained as follows, with reference to *Figure 3.1* [49]. Accuracy can be defined as the difference between the true value and the measured value of a quantity. Precision can be defined as the repeatability of multiple measured values of the same quantity. Accuracy is quantified by the mean error, while precision is quantified by the standard deviation of the error. In *Figure 3.1(a)*, high accuracy and low precision target shooting is illustrated, which has a low error mean and a high standard deviation. In *Figure 3.1(b)*, low accuracy and high precision target shooting is illustrated, which has a high error mean and a low standard deviation. In *Figure 3.1(c)* high accuracy and high precision target shooting is illustrated, which has a low error mean and a low standard deviation.

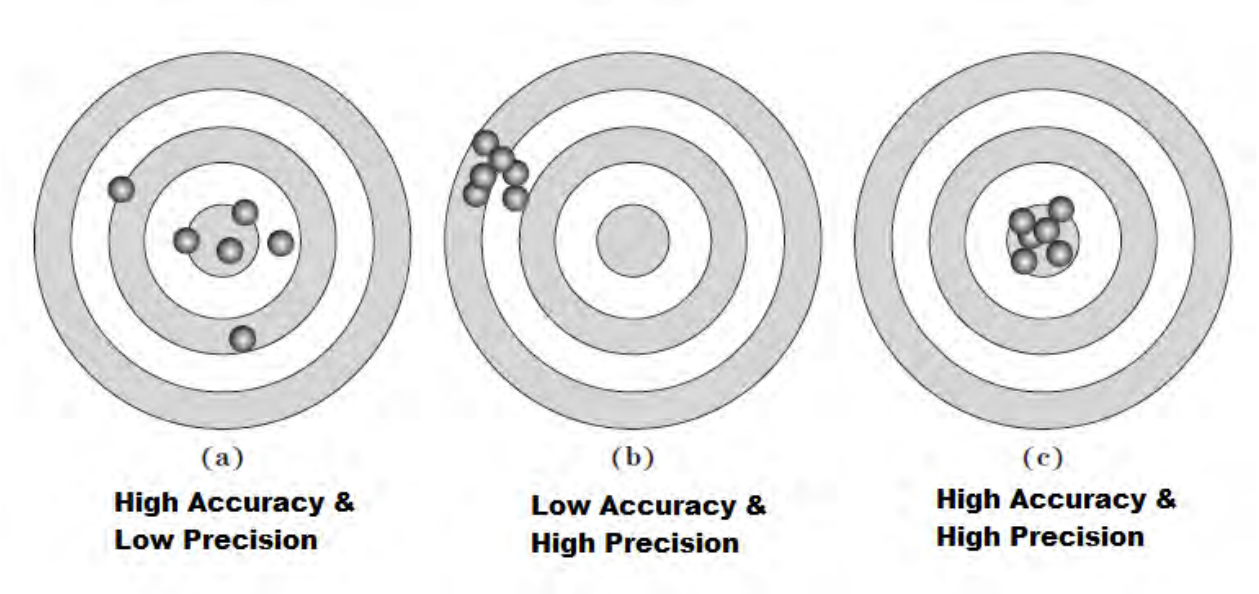

Figure 3.1: Accuracy and Precision

Accuracy is the primary metric in determining radar target tracking filter performance, since the purpose of a radar target tracking filter is to minimize the difference between the true position and the measured position of the radar target, *i.e.* the mean position error. A number of factors introduce errors in radar measurements that impact both the accuracy and the precision of radar measurements, namely:

1. noise and clutter;
2. target glint[1] and scintillation error[2];
3. multi-path signal propagation;
4. signal quantization and sampling rate;
5. gain and phase calibration;
6. antenna pointing errors and radar boresighting[3].

Since the factors that introduce noise into radar measurements are assumed to be random [49], repeated simulation of radar measurements with noise will produce a cumulative mean

[1] The inherent component of error in measurement of position and/or Doppler frequency of a complex target due to interference of the reflections from different elements of the target. Note that glint may have peak values beyond the target extent in the measured coordinate [50].

[2] Error in radar-derived target position or Doppler frequency caused by interaction of the scintillation spectrum with frequencies used in sequential measurement techniques [50].

[3] The process of aligning the electrical and mechanical axes of a directional antenna system, usually by an optical procedure [50].

measurement error close to zero, if the number of simulation iterations is large enough. The accuracy of the radar target tracking filter then becomes independent of the signal-to-noise ratio (SNR). However, the measurement precision, *i.e.* the standard deviation of the measurement error, is dependent on the SNR. An increase in SNR results in a decrease in precision, as expressed through the following proportional equation [49]:

$$\sigma \propto \frac{1}{\sqrt[4]{SNR}} \tag{3.1}$$

with SNR the signal-to-noise ratio.

Determining the number of simulation iterations for radar target tracking filter accuracy evaluation, will be discussed in Section 6.2.

3.2.2 Target Coordinate System

In general, a radar system produces measurement vectors that consists of range, r, azimuth angle, θ, and elevation angle, φ, components. The target tracking filter algorithm can be formulated to use a variety of coordinate system, for example Cartesian coordinates for the tracking of a target. Alternative coordinate systems that are used in radar systems for physical alignment are the North-East-Down (NED), with reference to *Figure 3.2* [6], or the East-North-Up (ENU) coordinate systems, with reference to *Figure 3.3* [6].

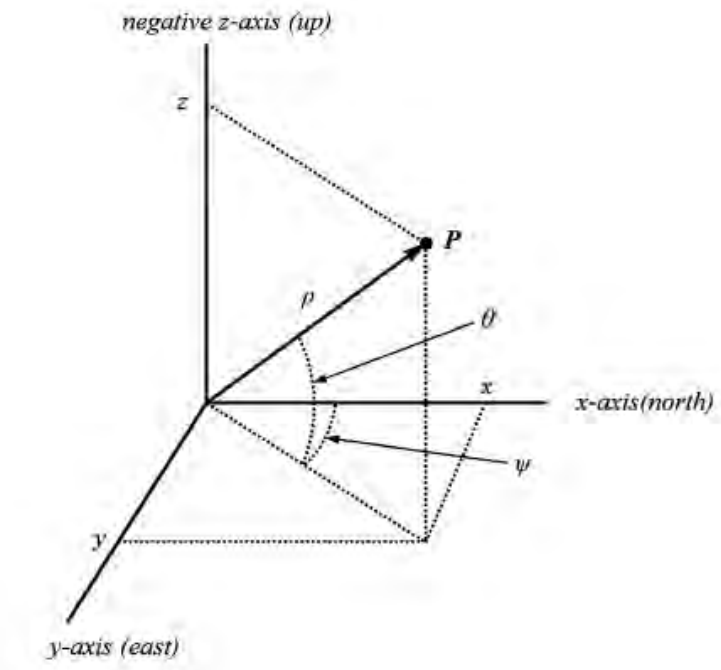

Figure 3.2: NED Coordinate System

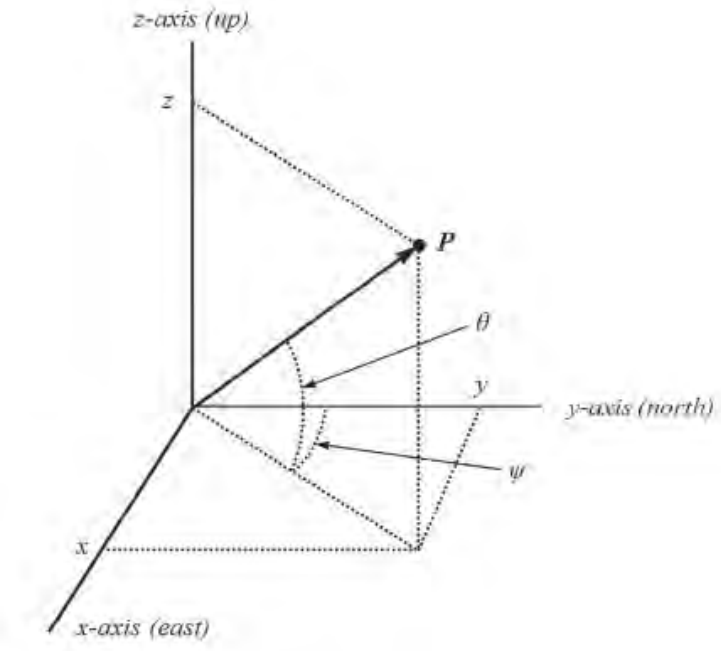

Figure 3.3: ENU Coordinate System

The radar measurement vectors can be converted to Cartesian coordinates for the target tracking filter using the following equations [2]:

$$x = r\cos\theta\cos\phi$$

$$y = r\sin\theta\cos\phi$$

$$z = r\sin\theta$$

with x, y, z *the Cartesian coordinates,* (3.2)

r *the target range,*

θ *the azimuth angle, and*

ϕ *the elevation angle.*

The Cartesian coordinates produced by the output of the target tracking filter can also be converted to radar measurement vectors using the following equations [2]:

$$r = \sqrt{x^2 + y^2 + z^2}$$

$$\theta = \tan^{-1}\left(\frac{y}{x}\right)$$

$$\phi = \cos^{-1}\left(\frac{z}{r}\right)$$

(3.3)

with r *the target range,*

θ *the azimuth angle,*

ϕ *the elevation angle, and*

x, y, z *the Cartesian coordinates.*

For other radar architectures, for example a passive bi-static radar that uses bi-static range, r, azimuth angle, θ, and bi-static Doppler frequency, f_d, measurements, equations have been developed to convert these measurements into Cartesian coordinates for input into a target tracking filter [44]. It should also be noted that in the literature the range, r, is also referred to and denoted as slant range, ρ.

3.3 Target Tracking Filters

3.3.1 Alpha-Beta Filter

The alpha-beta filter is mathematically defined by [1] as follows:

$$x_s(k) = x_p(k) + \alpha \times [x_m(k) - x_p(k)]$$

$$v_s(k) = v_s(k-1) + \left(\frac{\beta \times [x_m(k) - x_p(k)]}{T} \right)$$

$$x_p(k+1) = x_s(k) + [v_s(k) \times T]$$

with $x_s(k)$ *the filtered position,*

$v_s(k)$ *the filtered velocity,* (3.4)

$x_p(k)$ *the predicted position,*

$x_m(k)$ *the measured position,*

T *the detection interval,*

α *the position gain, and*

β *the velocity gain.*

From this description it can be seen that the existing target track position from the radar system is updated with a filtered position state and velocity state. This produces the predicted target track position. The position gain and velocity gain, α and β, determines the filtered position state and velocity state smoothed update value. According to [1], small position and velocity gains make small corrections in the target's predicted position. This results in the tracking filter being less sensitive to noise, but less responsive to fast maneuvers by the target being tracked and thus having a larger deviation from the assumed target model. The converse is thus also true; larger position and velocity gains will result in more tracking noise, but a faster response to fast maneuvers by the target being tracked. If the position gain and the velocity gain is equal to zero (*i.e.*, $\alpha = \beta = 0$), then the target tracking filter uses no information about the target's current position, only about the target's filtered previous position. If the position gain and the velocity gain is equal to one (*i.e.*, $\alpha = \beta = 1$), then no target filtering or smoothing is applied.

The computational efficiency of the alpha-beta tracking filter can be observed from its mathematical description as it only involves basic arithmetic computations. It also does not require large amounts of memory as it only needs access to the previous position prediction and current filtered position and velocity values to be able to predict the target's next position.

A number of variations and extensions to the alpha-beta filter has been developed to enhance its performance, while also increasing its computational complexity. The more prominent of these filters will briefly be mentioned here for completeness, namely:

1. alpha-beta-gamma filter [3]:

   alpha-beta filter with added acceleration;

2. alpha-beta-gamma-lambda filter [51]:

   alpha-beta-gamma filter with added acceleration gain constraint;

3. augmented alpha-beta filter [4]:

   1. also called a growing-memory / alpha-beta (GMAB) filter;
   2. alpha-beta filter with dynamically adjusting position and velocity gain.

3.3.2 Kalman Filter

The Kalman filter, as proposed by [52], is a recursive digital filter that can optimally estimate the future state of a system based on the current state data of the system that could, and in practice would, also contain a noise component, usually additive, zero-mean, white, Gaussian noise with a known covariance. As the Kalman filter is recursive, the filter require the storage of the previous state data of the system being modeled. Since the Kalman filter uses a linear system state equation to model the target, it does not perform well when tracking maneuvering targets [3]. The Kalman filter is mathematically defined by [1] as follows:

$$X(t_{k+1}) = \phi(t_k) \times X(t_k) + A(t_k) + A_p(t_k) \tag{3.5}$$

with $X(t_{k+1})$ *the filtered target state at time* t_{k+1},

$X(t_k)$ *the target state at time* t_k,

$\phi(t_k)$ *the transition matrix that moves the target linearly over an elapsed time,*

$A(t_k)$ *the target state change due to unknown acceleration, and*

$A_p(t_k)$ *the target state change due to known acceleration.*

The target state at time t_k, namely $X(t_k)$, consists of a position and velocity component for the target being tracked. The target state change due to unknown acceleration, namely $A(t_k)$, is generally caused by the target maneuvering or atmospheric drag on the target. This component is zero-mean and is characterized by its covariance matrix, namely $Q(t_k)$. At each radar measurement the acceleration is sampled to produce a discrete covariance matrix, if it is assumed that the unknown target maneuver is a white-noise process. Thus in practice, the Kalman filter continuously computes the covariance matrix of the target's estimated position and dynamically updates the filtered target state $X(t_{k+1})$. The target state change due to known acceleration, namely $A_p(t_k)$, is generally caused by gravity or Coriolis acceleration and can thus be corrected since it is known.

Thus in short, the filtered target state can be produced from the current measured target state, the target's transition matrix modeling the target's motion, the target's unknown acceleration characterized by its covariance matrix, and the target's known acceleration.

A number of variations and extensions to the Kalman filter have been developed to improve its performance, while also increasing its computational complexity. Once again, the more prominent of these filters will briefly be mentioned here for completeness, namely:

1. extended Kalman filter [3]:

   Kalman filter that linearizes the target's dynamics;

2. unscented Kalman filter (UKF) [53]:

   extended Kalman filter that uses a deterministic sampling approach to determine the state distribution;

3. nearest neighbour (NN) Kalman filter [5]:

Kalman filter that uses nearest neighbour data association techniques;

4. interacting multiple model (IMM) filter [3], [5]:

   uses multiple dynamic target models based on a prediction of the target's expected behaviour pattern;

5. interacting multiple model nearest neighbour (IMM-NN) filter [5]:

   combines the interacting multiple model filter with the nearest neighbour Kalman filter approach for improved target tracking in clutter;

6. probabilistic data association filter (PDAF) [5]:

   Kalman filter that uses probabilistic data association techniques;

7. interacting multiple model probabilistic data association filter (IMM-PDAF) [5]:

   uses a probabilistic data association filter as a state estimator, implemented in an interacting multiple model filter structure.

3.3.3 Swerling Filter

The Swerling filter is merely mentioned here for completeness. The Swerling filter is a precursor to the Kalman filter and has been superseded by the Kalman filter due to its improved performance. However, it has been shown by [6], that the Swerling and Kalman filters are equivalent to each other as both the Swerling and Kalman filters produce identical numerical results. Finally, it should be noted that the Swerling filter has the advantage of being self-initializing, *i.e.* not requiring the user to set the initial parameter values used in the filter, compared to the Kalman filter that needs to be initialized correctly by the user prior to their use [6].

3.3.4 Gauss Filter

3.3.4.1 (Non-Recursive) Gauss-Newton Filter

The Gauss-Newton filter (GNF) is a non-recursive (batch) filter based on the minimum variance algorithm (MVA), as described by [6]. The term non-recursive in the context of the Gauss-Newton filter means that all observations are simultaneously combined using the MVA. The Gauss-Newton filter combines a least-squares implementation of the MVA with the Newton method of local linearization. The Gauss-Newton filter consists of two components, namely a differential equation

to model the motion of the target being tracked, and an observation equation whose elements consist of the observations that were made of the target being tracked. The differential equation is referred to as the filter model. The filter model and the observation equation can both be either linear or non-linear in nature. This leads to four different implementations of the Gauss-Newton filter for the four cases as stated below:

- **Case 1**

  A linear filter model and a linear observation equation.

- **Case 2**

  A linear filter model and a non-linear observation equation.

- **Case 3**

  A non-linear filter model and a linear observation equation.

- **Case 4**

  A non-linear filter model and a non-linear observation equation.

The linear observation equation is defined by [6] as follows:

$$Y_n = MX_n + N_n$$

with Y_n *the observation vector,*

M *the observation matrix consisting of ones and zeros,* (3.6)

X_n *the true state vector of the observed target, and*

N_n *the unknown error vector.*

The non-linear observation equation is defined by [6] as follows:

$$Y_n = G(X_n) + N_n$$

with Y_n *the observation vector,*

$G(X_n)$ *containing:*
- *the observation matrix* M *,*
- *a state vector of a known trajectory* $\bar{X}_n$*,*
- *the perturbation vector* δX_n*, and*

N_n *the unknown error vector.* (3.7)

The Gauss-Newton filter functions by taking the vector of all the observations up to time t_n and its resultant covariance matrix, and subtracting the estimate of the true state vector of the observed target, X_n, to produce the residual vector, δY_n. An improved estimate of the true state vector of the observed target, X_n, is produced from this vector and the filter model which is in the form of a matrix of partial derivatives. This process is repeated until the best estimate of the true state vector of the observed target, X_n, is produced and this then becomes the output of the filter.

Thus, the linear observation equation over time is defined as:

$$\begin{aligned} Y_n &= MX_n + N_n \text{ for time } t_n \\ Y_{n-1} &= MX_{n-1} + N_{n-1} \text{ for time } t_{n-1} \\ &\vdots \\ Y_{n-L} &= MX_{n-L} + N_{n-L} \text{ for time } t_{n-L} \end{aligned} \tag{3.8}$$

with L *the memory length (size) of the filter.*

The non-linear observation equation over time can be inferred from this definition.

3.3.4.2 Gauss-Aitken Filter

It should be noted that for the Gauss-Newton filter implementation as stated in **Case 1** in Section 3.3.4.1, *i.e.* a linear filter model and a linear observation equation, the filter is referred to as a Gauss-Aitken filter .

3.3.4.3 Recursive Gauss-Newton Filter

The recursive Gauss-Newton filter (RGNF) is a recursive form of the Gauss-Newton filter,

proposed by [8]. The term recursive in the context of the RGNF means that observations are recursively combined using the MVA. For example, two observations will be combined using the MVA to produce an estimate, and then a third observation will be combined with that estimate to produce a new estimate etc. The RGNF adds two extra parameters to the Gauss-Newton filter, namely a forgetting factor, λ, and a damping factor, μ. The use of the damping factor, μ, will change the speed of convergence of the filter to a guaranteed solution, while the forgetting factor, λ, influences how good an approximation the RGNF produces to the Gauss-Newton filter. These added parameters allow the RGNF to track targets in non-linear situations with Doppler only information [54]. The full derivation of the filter equations and algorithm is presented by [55].

3.3.5 Polynomial Filter

The polynomial filter is based on the linear combinations of a number of polynomial terms, specifically the orthogonal polynomials of Legendre and Laguerre, as developed by [6]. The use of Legendre polynomials leads to a polynomial filter with expanding memory, know as the expanded memory polynomial (EMP) filter, due to their uniform weight function. On the other hand, the use of Laguerre polynomials leads to a polynomial filter with fading memory, know as the fading memory polynomial (FMP) filter, due to their exponentially decaying weight function. Initialization in the context of the polynomial filter refers to the initial values used in the polynomial.

An observation vector, Y_n, is produced from the target detections of the radar system at time t_n, and it is submitted as input to the polynomial filter for fitting a polynomial by the least-squares method to this observation vector. This polynomial, *x(t)*, can then be used to provide smoothed estimates of the true state vector, *X(t)*, *i.e.* determine what the value of *x(t)* is at time t_n, and thus X^*. The true state vector and its smoothed estimate is defined as follows [6]:

$$\begin{aligned} X(t) &= \left(x(t)\ ,\ \dot{x}(t)\ ,\ \ddot{x}(t)\ ,\ \ldots\ D^m x(t)\right)^T \text{ in general,} \\ X(t_n) &= \left(x_0\ ,\ x_1\ ,\ x_2\ ,\ \ldots\ x_m\right)_n^T \text{ at time } t_n \text{, and} \\ X^* &= \left(x_0^*\ ,\ x_1^*\ ,\ x_2^*\ ,\ \ldots\ x_m^*\right)^T \text{ the smooted estimate of } X(t). \end{aligned} \tag{3.9}$$

3.3.5.1 Expanded Memory Polynomial Filter

The FMP filter is based on fitting a polynomial by the least-squares method to the following finite-dimensional observation vector [6]:

$$\begin{aligned} Y_n &= \left(y_n \ , \ y_{n-1} \ , \ \ldots \ y_0\right)^T \ \textit{at time} \ t_n \\ Y_{n+1} &= \left(y_{n+1} \ , \ y_n \ , \ y_{n-1} \ , \ \ldots \ y_0\right)^T \ \textit{at time} \ t_{n+1} \\ &\vdots \\ Y_{n+m} &= \left(y_{n+m} \ , \ y_{n+m-1} \ , \ y_{n+m-2} \ , \ \ldots \ y_0\right)^T \ \textit{at time} \ t_{n+m}. \end{aligned} \tag{3.10}$$

The properties of the EMP filter is that it is self-initializing, but it cannot track a target indefinitely due to its expanding memory nature.

3.3.5.2 Fading Memory Polynomial Filter

The FMP filter is based on fitting a polynomial by the least-squares method to the following infinite-dimensional observation vector [6]:

$$\begin{aligned} Y_n &= \left(y_n \ , \ y_{n-1} \ , \ y_{n-2} \ , \ \ldots \right)^T \ \textit{at time} \ t_n \\ Y_{n+1} &= \left(y_{n+1} \ , \ y_n \ , \ y_{n-1} \ , \ \ldots \right)^T \ \textit{at time} \ t_{n+1} \\ &\vdots \\ Y_{n+m} &= \left(y_{n+m} \ , \ y_{n+m-1} \ , \ y_{n+m-2} \ , \ \ldots \right)^T \ \textit{at time} \ t_{n+m}. \end{aligned} \tag{3.11}$$

The properties of the FMP filter is that it is not self-initializing, but it can track a target indefinitely due to its fixed-length memory nature.

3.3.5.3 Composite EMP/FMP Filter

The composite EMP/FMP filter combines the EMP and FMP filters to produce a self-initializing polynomial filter that can track an arbitrary target indefinitely. In short, at start-up the EMP filter is used for its self-initializing properties, and at some point in time the FMP filter is used for its ability to track an arbitrary target indefinitely. The switch over point is determined analytically in such a

way that there are no transients generated in the filter output [6].

3.3.5.4 Probabilistic Polynomial Filter

The probabilistic polynomial filter, as proposed by [7], takes the composite EMP/FMP filter and incorporates probabilistic data association (PDA) techniques into the filter structure. The result, as shown through simulation, is a target tracking filter that can operate in high clutter density environments with low detection probability in the order of $P_D = 0.6$.

3.3.6 Particle Filter

The particle filter is an implementation of the formal recursive Bayesian filter that uses sequential Monte Carlo methods. The recursive Bayesian filter updates the posterior probability density function, as constructed by a Bayesian estimator, as new measurements are received. The theory related to the particle filter will be fully discussed in the Chapter 4.

3.4 Filter Performance Metrics

3.4.1 Root Mean Squared Error

There is no single performance metric for assessing tracking filter algorithm performance and it is still seen as an open research problem [2]. For simulated single target tracking performance however, there are two metrics that can be used. The metrics are the root mean squared error (RMSE) for position and velocity. The RMSE can also be used as an absolute metric when comparing tracking filter algorithms relative to each other [11]. In general, the RMSE for the target's position can be calculated as [2]:

$$RMSE^{pos} = \sqrt{\frac{1}{M}\sum_{i=1}^{M}(true\ position_i - predicted\ \ position_i)^2} \qquad (3.12)$$

with M the number of detections.

In general, the RMSE for the target's velocity can be calculated as [2]:

$$RMSE^{vel} = \sqrt{\frac{1}{M}\sum_{i=1}^{M}(true\,velocity_i - predicted\,velocity_i)^2} \qquad (3.13)$$

with M the number of detections.

For radar measurements in a two-dimensional *(x, y)* Cartesian coordinate system, the RMSE for the target's position can be calculated as:

$$RMSE^{pos} = \sqrt{\frac{1}{M}\sum_{i=1}^{M}\left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2\right]}$$

with M *the number of detections,*

x_i, y_i *the true position coordinates, and*

$\hat{x}_i, \hat{y}_i$ *the estimated position coordinates.* (3.14)

For radar measurements in a three-dimensional *(x, y, z)* Cartesian coordinate system, the RMSE for the target's position can be calculated as:

$$RMSE^{pos} = \sqrt{\frac{1}{M}\sum_{i=1}^{M}\left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (z_i - \hat{z}_i)^2\right]}$$

with M *the number of detections,*

x_i, y_i, z_i *the true position coordinates, and*

$\hat{x}_i, \hat{y}_i, \hat{z}_i$ *the estimated position coordinates.* (3.15)

Furthermore, for multiple (batch) measurements of the RMSE of the target's position for example, the batch RMSE position value can be calculated as the average of all the individual RMSE values, as follows:

$$RMSE_{batch}^{pos} = \frac{1}{N}\sum_{i=1}^{N}\left(RMSE_i^{pos}\right)$$

with N *the number of measurements.* (3.16)

3.4.2 Normalized Processing Time

Another metric that is sometimes used to compare various target tracking filters, is normalized processing time [18]. Since the hardware on which a target tracking filter is simulated or

implemented varies between implementations, the processing time will also vary between implementations. However, a relative processing time comparison can be done between different target tracking filters simulated or implemented on the same hardware, by normalizing the various target tracking filter processing times. This is accomplished by simply dividing each target tracking filter's processing time by the quickest processing time of all the target tracking filters being compared.

3.5 Computational Complexity vs. Accuracy

There is a direct correlation between the computational complexity and the accuracy of target tracking filter algorithms as shown by [4]. The higher the computational complexity, the higher the accuracy of target tracking filter algorithm. However, the conditions under which the target tracking filter algorithm operates also have a direct impact on their degree of accuracy. It has been shown by [56] that although a specific filter can be computationally more efficient than another, albeit not as accurate, the environment in which a target tracking filter operates, has an impact of the computational efficiency and accuracy filter metrics.

3.6 Summary

In this chapter, the theory of the most prominent target tracking filters were presented. Additionally, issues relating to the metrics used in evaluating the performance of target tracking filters, as well as the relationship between computational complexity and accuracy of target tracking filters, have also been discussed. In Chapter 4, the theory and operation of the particle filter will be discussed in depth due to its relevance to the research conducted for the dissertation.

Chapter 4 : Particle Filter Theory and Operation

4.1 Introduction

The theory and operation of the particle filter is discussed in depth in this chapter due to its relevance to the research conducted for the dissertation. The discussion follows the core development of particle filters from the initial sequential importance sampling algorithm, through to the sequential importance re-sampling particle filter, which is the most common version of the particle filter [16]. The discussion also covers other variations of the core particle filter that was proposed and developed along the way, in order to address specific problems that have been found in the earlier versions of the particle filter. The discussion highlights the reasons why the sequential importance re-sampling particle filter has become the dominant particle filter in use. The sequential importance re-sampling particle filter is commonly referred to as a *particle filter* in the literature, due to its ubiquitous use [57]. However, a number of other names and terms that are synonymous with the term particle filter are used in the literature, some for example being:

1. (Bayesian) bootstrap filter [58];
2. interacting particle approximation [59];
3. survival-of-the-fittest [15];
4. condensation algorithm [60];
5. sequential Monte Carlo (SMC) methods [61].

A number of specialized or hybrid particle filters also exist, of which a few of the more prominent are briefly discussed at the end of the chapter. This is not an exhaustive list and discussion of these specialized or hybrid particle filters, but is mentioned for the sake of completeness in covering particle filter theory.

4.2 Particle Filter Algorithm Development Overview

An overview of the development of what is considered, for the purposes of this dissertation and highlighted in blue, to be the core particle filter algorithms, is shown in *Figure 4.1*. Where one particle filter was developed from another particle filter, the interconnection between them shows the problem that the derived particle filter was attempting to address. For each problem being

addressed, the techniques used in addressing the problem is shown below the technique in bullet-point format. Known problems inherent in a specific particle filter and the techniques that were used in attempting to address the problems, are also shown. The rest of this chapter will provide more detail of the core particle filter algorithms and conclude with a discussion of some specialized or hybrid particle filters.

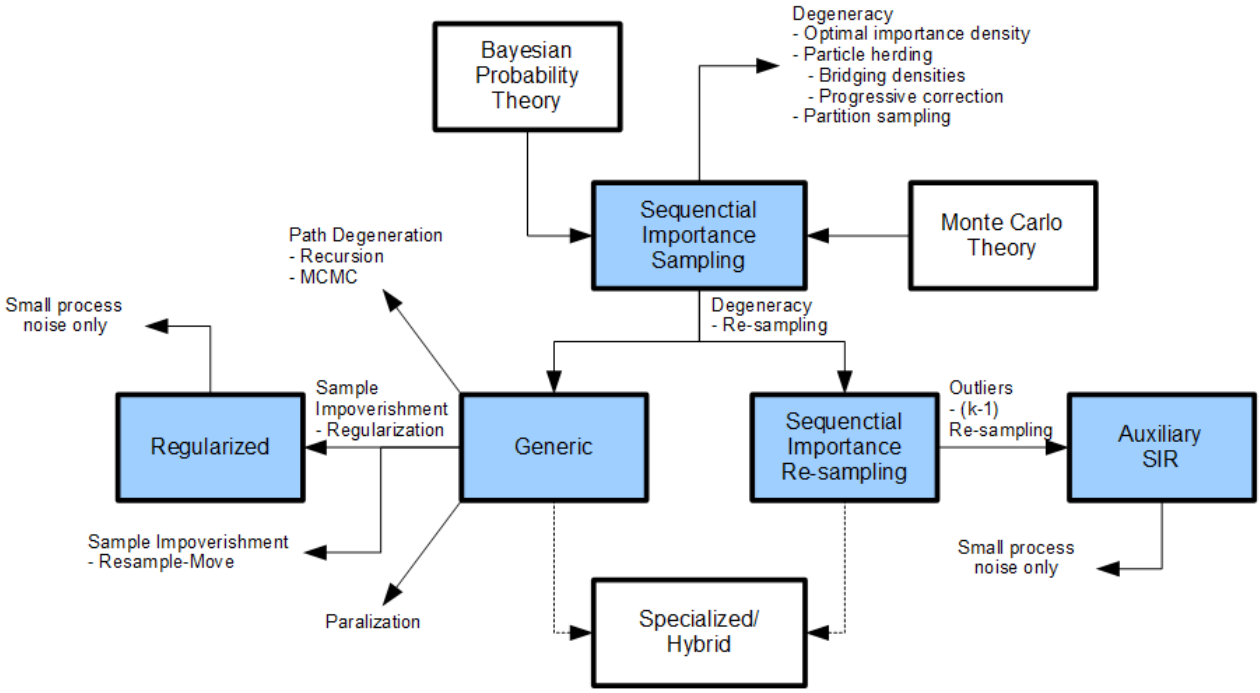

Figure 4.1: Particle Filter Algorithm Development Overview

4.3 Introductory Theoretical Concepts

To cover the theory of particle filters, two theoretical concepts will be discussed, namely Bayesian probability theory and Monte Carlo theory. These two concepts will be covered in an introductory manner as it is outside of the scope of this dissertation to cover these concepts in their entirety.

4.3.1 Bayesian Probability Theory

In Bayesian probability theory as applicable to dynamic state estimation, the posterior probability density function (PDF) of a process being modeled is constructed from the prior information, as for example past measurements [15]. It is assumed that the past measurements are made independent of each other for each group of measurements, and measurements between groups are also independent. In theory, an optimal posterior estimate of the process being modeled can be obtained

form the PDF, as well as a measure of the accuracy of the obtained estimate. The posterior PDF is defined to be [16]:

$$p(x_k|Y_k)=\frac{p(y_k|x_k)\,p(x_k|Y_{k-1})}{p(y_k|Y_{k-1})}$$

$$\begin{aligned}&\textit{with } p(y_k|x_k) \textit{ the likelihood function,}\\ &p(x_k|Y_{k-1}) \textit{ the prior PDF according to (4.2), and}\\ &p(y_k|Y_{k-1}) \textit{ the normalizing constant according to (4.3).}\end{aligned} \tag{4.1}$$

The prior PDF is defined to be [16]:

$$\begin{aligned}&p(x_k|Y_{k-1})=\int p(x_k|x_{k-1})\,p(x_{k-1}|Y_{k-1})\,dx_{k-1}\\ &\textit{with } p(x_k|x_{k-1}) \textit{ the transitional density.}\end{aligned} \tag{4.2}$$

The transitional density in (4.2) is defined by the noise present in the process being modeled and the system model of the process being modeled. The normalizing constant is defined to be [16]:

$$\begin{aligned}&p(y_k|Y_{k-1})=\int p(y_k|x_k)\,p(x_k|Y_{k-1})\,dx_k\\ &\textit{with } p(y_k|x_k) \textit{ the likelihood function, and}\\ &p(x_k|Y_{k-1}) \textit{ the prior PDF according to (4.2).}\end{aligned} \tag{4.3}$$

The likelihood function in (4.1) and (4.3) is defined by the measurement equation and the measurement noise sequence of the process being modeled. The prediction and update stages of Bayesian probability theory as applicable to dynamic state estimation, is defined by (4.2) and (4.1) respectively. The dynamic state estimation that is achieved using Bayesian probability theory is only a theoretical solution to state estimation, since generally no analytical expressions for these equations exists. A more complete analysis and discussion of this topic is presented by [62].

In the case of the particle filter, a suboptimal solution to the state estimation problem is achieved through approximating the equations for the prediction and update stages of Bayesian probability theory. This is achieved through the a combination of random samples, known as particles, and Monte Carlo theory which will be discussed in Section 4.3.2.

4.3.2 Monte Carlo Theory

Monte Carlo theory, is a generalized term for describing a computational algorithm that implements random sampling in order to produce a numerical result [62]. It was proposed by John von Neumann and Stanislas Ulam in the 1940s as a method to solve problems by modeling a problem through chance on a computer [63]. For example, Monte Carlo methods can be used to determine the statistical properties, like mean, variance, probability density function (PDF) using a histogram, etc., of any estimator of which the performance is to be determined. The Monte Carlo method, for a *99.75%* confidence level in the range $\pm 3\sigma$, can be expressed as the probability [64]:

$$P\left\{\left|\frac{1}{N}\sum_{j=1}^{N}\xi_j - m\right| < \frac{3\times\sigma}{\sqrt{N}}\right\} \approx 0.997$$

with N *the number of independent random variables,*

ξ *the random variable,*

m *the unknown quantity to calculate, and*

σ *the square root of the variance (i.e. standard deviation).* (4.4)

From this it follows that the arithmetic mean of the independent random variables will be approximately equal to the unknown quantity to calculate, namely *m*. Importantly, as the number of independent random variables *N* increases, the error in the estimate of the unknown quantity to calculate *m*, approaches zero.

4.4 Sequential Importance Sampling Particle Filter

The basis of the particle filter is the sequential importance sampling (SIS) algorithm. The SIS algorithm is a Monte Carlo technique that is used to implement a recursive Bayesian filter through Monte Carlo simulations [58]. The principle of operation of the SIS algorithm is the use of a number of random samples, or particles, to represent the posterior probability density function (PDF) of a process being modeled. Each random particle has a weight associated with it that is used to compute the estimate of the process being modeled, based on the weights of all the particles. The SIS algorithm recursively propagates these weights as each measurement of the process being modeled is sequentially processed. As the number of particles is increased, the estimated posterior

PDF becomes the optimal Bayesian estimate for the process being modeled. A generic description of the SIS particle filter, based on the SIS algorithm, is given as follows [15]:

$$
\begin{aligned}
&[\{x_k^i, w_k^i\}_{i=1}^{N_s}] = SIS[\{x_{k-1}^i, w_{k-1}^i\}_{i=1}^{N_s}, z_k] \\
&FOR\ i = 1 : N_s \\
&\text{- } Draw\ x_k^i \sim q(x_k | x_{k-1}^i, z_k) \\
&\text{- } Assign\ the\ particle\ a\ weight,\ w_k^i\ ,\ according\ to: \\
&\qquad w_k^i \propto w_{k-1}^i \frac{p(z_k | x_k^i)\, p(x_k^i | x_{k-1}^i)}{q(x_k^i | x_{k-1}^i, z_k)} \\
&END\ FOR
\end{aligned}
\tag{4.5}
$$

The SIS particle filter suffers from an unavoidable problem known as the degeneracy phenomenon. After a number of iterations, all but one of the particles will have a negligible weight associated with it. This results in all but one of the particles contributing almost zero to the approximation process. A number of solutions to the degeneracy phenomenon problem has been suggested and investigated, summarized as follows:

1. the use of an optimal importance density [65];
2. particle "herding" through the use of:
    1. progressive correction [25];
    2. bridging densities [66];
3. partition sampling:
    independent partition particle filter (IPPF) [35];
4. re-sampling [57].

The general solution to the problem of the degeneracy phenomenon was found to be the use of re-sampling, which led to the development of two new types of particle filter, will be discussed in Section 4.5 and Section 4.6.

4.5 Generic Particle Filter

The generic particle filter (GPF) was developed in an attempt to solve the unavoidable problem known as the degeneracy phenomenon, that is present in the SIS particle filter. The principle of operation of the GPF is the use of re-sampling when degeneracy is detected in the effective sample

size, N_{eff}, of the SIS particle filter, according to some threshold defined as N_T. The approximate effective sample size of the SIS particle filter is defined as [15]:

$$\widehat{N_{eff}} = \frac{1}{\sum_{i=1}^{N_s} (w_k^i)^2} \tag{4.6}$$

with w_k^i *the normalized particle weight, and*

N_s *the number of particles.*

A generic description of the GPF, based on the SIS algorithm, is given as follows [15]:

$$
\begin{aligned}
&[\{x_k^i, w_k^i\}_{i=1}^{N_s}] = GPF[\{x_{k-1}^i, w_{k-1}^i\}_{i=1}^{N_s}, z_k] \\
&FOR\ i = 1 : N_s \\
&\text{- } Draw\ x_k^i \sim q(x_k | x_{k-1}^i, z_k) \\
&\text{- } Assign\ the\ particle\ a\ weight,\ w_k^i\ ,\ according\ to: \\
&\qquad w_k^i \propto w_{k-1}^i \frac{p(z_k | x_k^i)\, p(x_k^i | x_{k-1}^i)}{q(x_k^i | x_{k-1}^i, z_k)} \\
&END\ FOR \\
&Calculate\ total\ weight:\ t = \sum [\{w_k^i\}_{i=1}^{N_s}] \\
&FOR\ i = 1 : N_s \\
&\text{- } Normalize:\ w_k^i = t^{-1} w_k^i \\
&END\ FOR \\
&Calculate\ \widehat{N_{eff}},\ according\ to\ (4.6) \\
&IF\ \widehat{N_{eff}} < N_T \\
&\text{- } Re\text{-}sample\ using\ (4.8): \\
&\qquad [\{x_k^i, w_k^i, -\}_{i=1}^{N_s}] = RESAMPLE[\{x_k^i, w_k^i\}_{i=1}^{N_s}] \\
&END\ IF
\end{aligned}
\tag{4.7}
$$

A generic description of the re-sampling process is given as follows [15]:

$$
\begin{aligned}
&[\{x_k^{j*}, w_k^j, i^j\}_{j=1}^{N_s}] = RESAMPLE[\{x_k^i, w_k^i\}_{i=1}^{N_s}, z_k] \\
&\textit{Initialize the CDF (cumulative distribution function):}\ c_1 = 0 \\
&\textit{FOR}\ i = 2 : N_s \\
&\text{- Construct CDF: } c_i = c_{i-1} + w_k^i \\
&\textit{END FOR} \\
&\textit{Start at the bottom of the CDF:}\ i = 1 \\
&\textit{Draw a starting point:}\ u_1 \sim U[0, N_s^{-1}] \\
&\textit{FOR}\ j = 1 : N_s \\
&\text{- Move along the CDF: } u_j = u_1 + N_s^{-1}(j-1) \\
&\text{- WHILE } u_j > c_i \\
&\quad \text{- } i = i + 1 \\
&\text{- END WHILE} \\
&\text{- Assign sample: } x_k^{j*} = x_k^i \\
&\text{- Assign weight: } w_k^j = N_s^{-1} \\
&\text{- Assign parent: } i^j = i \\
&\textit{END FOR}
\end{aligned}
\tag{4.8}
$$

Since $N_{eff} \leq N_S$, a small value of N_{eff} indicates a large degeneracy phenomenon. In re-sampling, particles with small weights are eliminated from the particle population, and the algorithm thus focuses on particles that have larger weights. However, it was found that the generic particle filter had three practical disadvantages due to the method of implementation of re-sampling, namely:

1. lack of algorithm parallelization;
2. particle path degeneration;
3. sample impoverishment due to lack of particle diversity.

A solution to the issue of particle path degeneration has been investigated through a number of

techniques, namely:

1. recursion;
2. Markov chain Monte Carlo (MCMC) re-sampling [66].

None of these techniques have found widespread practical use, likely due to the added computational complexity required in their implementation, or the subsequent development of the sequential importance re-sampling (SIR) particle filter, which will be discussed in Section 4.6. A solution to the issue of sample impoverishment due to the lack of particle diversity has been investigate through a number of techniques, namely:

1. the re-sample-move algorithm [66];
2. regularization.

The use of regularization in solving the problem of sample impoverishment due to the lack of particle diversity has proved to be the most effective, and this led to the development of the regularized particle filter, which will be discussed in Section 4.5.1.

4.5.1 Regularized Particle Filter

The regularized particle filter (RPF) was developed in an attempt to deal with the loss of particle diversity, or sample impoverishment, that is inherent in the generic particle filter [15]. The principle of operation of the RPF is that the re-sampling process is applied to a continuous approximation of the posterior density of the process being modeled, instead of a discrete approximation as is done in the case of the SIR particle filter. However, it was found that the RPF has two disadvantages when compared to the generic particle filter, namely:

1. the samples are no longer guaranteed to approximate the posterior density of the process being modeled;
2. the RPF only has improved performance when the noise in the process being modeled is small.

These disadvantages limit the practical use of the RPF.

4.6 Sequential Importance Re-sampling Particle Filter

The sequential importance re-sampling (SIR) particle filter was developed in an attempt to solve the unavoidable problem known as the degeneracy phenomenon, that is present in the SIS particle filter.

The principle of operation of the SIR particle filter is the use of re-sampling to solve the degeneracy phenomenon, as is used in the generic particle filter and its variants. The re-sampling process used in the SIR particle filter is identical to that which is used in the generic particle filter. The difference occurs in that the re-sampling process is performed at each iteration of the filter algorithm and thus no threshold value for the effective sample size is used to trigger the re-sampling process. A generic description of the SIR particle filter is given as follows [15]:

$$
\begin{aligned}
&[\{x_k^i, w_k^i\}_{i=1}^{N_s}] = SIR[\{x_{k-1}^i, w_{k-1}^i\}_{i=1}^{N_s}, z_k] \\
&FOR\ i = 1:N_s \\
&\text{- } Draw\ x_k^i \sim p(x_k | x_{k-1}^i) \\
&\text{- } Calculate\ w_k^i = p(z_k | x_k^i) \\
&END\ FOR \\
&Calculate\ total\ weight:\ t = \sum [\{w_k^i\}_{i=1}^{N_s}] \\
&FOR\ i = 1:N_s \\
&\text{- } Normalize:\ w_k^i = t^{-1} w_k^i \\
&END\ FOR \\
&\text{- } Re\text{-}sample\ using\ (4.8): \\
&[\{x_k^i, w_k^i, \text{-}\}_{i=1}^{N_s}] = RESAMPLE[\{x_k^i, w_k^i\}_{i=1}^{N_s}]
\end{aligned}
\tag{4.9}
$$

The SIR particle filter does not exhibit the same disadvantages that the generic particle filter and its variations have, but there are three other practical disadvantages to the SIR particle filter, namely:

1. algorithmic inefficiency [68];
2. some loss of particle diversity;
3. sensitivity to outliers.

The issue of sensitivity to outliers that was found to be present in the SIR particle filter, led to the development of the auxiliary sequential importance re-sampling particle filter, which will be discussed in Section 4.6.1.

4.6.1 Auxiliary Sequential Importance Re-sampling Particle Filter

The auxiliary sequential importance re-sampling (ASIR) particle filter was developed in an attempt to deal with the sensitivity of the SIR particle filter to outliers [15]. The ASIR particle filter is also commonly referred to as the auxiliary particle filter (APF) [66]. The principle of operation of the ASIR particle filter is that the re-sampling process happens at one time step before the current time step, *i.e.* at time step *(k – 1)*. This has the advantage that, in general, the ASIR particle filter is not so sensitive to outliers. However, it was found that the disadvantage of this re-sampling method is that the ASIR particle filter has degraded performance when the noise in the process being modeled is large. This disadvantage limits the practical use of the ASIR particle filter.

4.7 Specialized / Hybrid Particle Filters

A number of particle filters have been developed that are either specialized in their application or are a hybrid of particle filter theory and some other filter architecture, or uses the particle filter architecture in combination with some other filter theory. Although the term *particle filter* might appear in their name, not all of these specialized or hybrid filters may be considered to be particle filters as described before. For the sake of completeness in dealing with particle filter theory and operation, the most prominent of these from the literature will be discussed in chronological order.

4.7.1 Unscented Particle Filter

The unscented particle filter (UPF) use an unscented Kalman filter (UKF) to move the distribution of the particles towards the region of highest likelihood [37]. In effect a bank of UKFs are used to determine the importance proposal distribution of the process being modeled. Through simulation, the UPF has been shown to be more accurate than the generic particle filter. However, the UPF is computationally more expensive than most other particle filters due to the added UKF computational step.

4.7.2 “Likelihood” Particle Filter

A proposed alternative to the particle filters described up to this point, is the “likelihood” particle filter [15]. The principle of operation of the “likelihood” particle filter is the use of “likelihood” instead of the prior proportional density information of the process being modeled, in order to determine the posterior density function. The performance of the “likelihood” particle filter has been shown to be similar to that of the ASIR particle filter [15]. However, it was found that the

"likelihood" particle filter had a serious limitation when compared to the other particle filters that have been discussed. The "likelihood" particle filter is not generically applicable to problems due to the specific choice of the importance density for a specific problem and set of parameters. A similar approach was subsequently investigated and reported by [39] in the design of the proposal distribution based on "likelihood". In their approach they considered a proposal distribution suited to a specific problem, but have pointed out the practicality of actually generating such a problem specific proposal distribution.

4.7.3 Rao-Blackwellized Particle Filter

The Rao-Blackwellized particle filter (RBPF) uses a particle filter to estimate the marginal non-linear state probability density function (PDF), and conditional Kalman filters are then used for each particle in the particle filter in order to calculate the linear system state [67]. The RBPF can thus be considered a hybrid filter that is also know as the marginalized particle filter [68]. The performance of the RBPF compares well with other particle filters, but in general a dynamic model of the process being modeled is required to achieve these performance results. However, it has been applied to practical problems, like a navigation problem involving an unreliable aircraft radar altimeter with missing data [68].

4.7.4 Evolutionary Particle Filter / Improved Evolutionary Particle Filter

The evolutionary particle filter (EPF) and the improved EPF (IEPF) uses a particle selection technique based on an evolutionary algorithm, in order to minimize sample impoverishment due to lack of particle diversity, that is present in the generic particle filter. The disadvantage of the EPF is the added computational complexity that is added by the evolutionary computation step. An improvement to the EPF is the IEPF which uses the effective sample size, N_{eff} as used in the generic particle filter, refer to (4.6), to regulate the implementation of the evolutionary computation step [27]. However, the IEPF is still computationally more expensive than most other particle filters.

4.7.5 Variable Rate Particle Filter

The variable rate particle filter (VRPF) introduces the concept of variable rate re-sampling based on a dynamic model of the target being tracked, instead of deterministic re-sampling [31], [34], [35], [36]. The aim of the VRPF is to address the problem of computational inefficiency in the generic

particle filter. The disadvantage of the VRPF is that a dynamic model of the target being tracked needs to exist in order for the filter to be useful, as the re-sampling rate needs to be adjusted based on the dynamic model.

4.7.6 Markov Chain Monte Carlo Iterated Extended Kalman Particle Filter

The Markov chain Monte Carlo iterated extended Kalman particle filter (MCMC-IEKPF) uses an iterated extended Kalman filter (IEKF) to generate the proposal distribution for a Markov chain Monte Carlo particle filter (MCMC-PF) [23]. The presented results of the MCMC-IEKPF shows that it has higher accuracy that an MCMC-PF, but at a higher computational complexity. Furthermore, it is only evaluated for a specific class of problem, namely tracking targets that have been corrupted by glint noise.

4.7.7 Polynomial Predictive Particle Filter

The polynomial predictive particle filter (PPPF) combines a polynomial filter with a generic particle filter [24]. The polynomial filter is used to predict the path of the target being tracked through the extension of the system state dimension, which is then filtered with a generic particle filter. Improved accuracy is claimed for the PPPF compared to the generic particle filter, but no quantitative simulation results were presented.

4.7.8 Adaptive Particle Filter

An approach to particle filtering based on the probability of each particle is proposed by [28]. The particle filtering approach is called an adaptive particle filter (APF) and it is aimed at tracking sea-surface targets in the presence of glint noise. It employs a technique to adaptively estimate the number of particles to use in order to model the process being observed, in this case a ship tracking problem under glint noise.

4.7.9 Noise-estimate Particle Probability Hypothesis Density Filter

The noise-estimate particle probability hypothesis density filter (NP-PHDF) is a hybrid filter that uses a particle filter to estimate the noise parameters of a Kalman filter in order to initialize the Kalman filter [29]. In essence, the NP-PHDF cannot be considered a *particle filter* since the particle filter is only used to initialize a Kalman filter.

4.8 Summary

In this chapter, the theory of particle filters have been discussed due to its relevance to the research conducted for the dissertation. The discussion followed the development of various particle filters and their variants up to what is commonly referred to in the literature as the *particle filter*, which is more specifically known as the sequential importance re-sampling particle filter. The disadvantages related to the various particle f The design assumptions and constraints ilters that were developed have been highlighted, in order to show why the sequential importance re-sampling particle filter has become the dominant particle filter in use. A number of specialized or hybrid particle filters have also been discussed. In Chapter 5, the design, implementation and operation of the radar target tracking filter simulator, as used for the quantitative simulation of the particle filter as a radar target tracking filter, is discussed.

Chapter 5 : Radar Target Tracking Filter Simulator

5.1 Introduction

The design, implementation and operation of the radar target tracking filter simulator, as used for the quantitative simulation of the particle filter as a radar target tracking filter, is discussed in this chapter. The design and operation of the radar target tracking filter simulator is discussed through a presentation of the architecture of the radar target tracking filter simulator. The design assumptions and constraints that have been used in the design of the radar target tracking filter simulator are presented. Each of the components in the architecture of the radar target tracking filter simulator is also discussed in detail. Finally, the operation of the radar target tracking filter simulator is demonstrated through the use of images of the user interface and various simulator displays. The radar target tracking filter simulator had been implemented in Scilab 5.5.0 (http://www.scilab.org/), and the source code is included in Appendix A.

5.2 Architecture

For the quantitative simulations that have been done, a radar target tracking filter simulator has been developed, as described by [6]. The overall objective of the radar target tracking filter simulator was to allow for the simulation and data capture of various target tracking filters, target models, and noise models. The flow diagram of the radar target tracking filter simulator is shown in *Figure 5.1*.

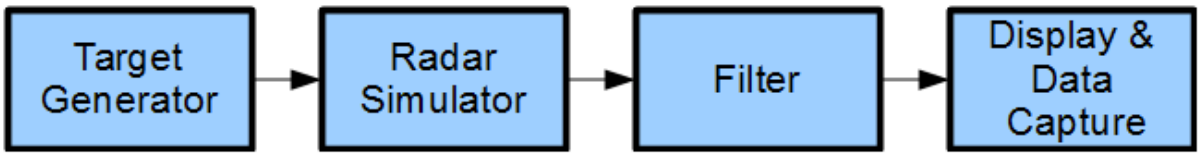

Figure 5.1: Radar Target Tracking Filter Simulator Flow Diagram

The architecture of the radar target tracking filter simulator consists of the four components as described below.

- **Target Generator**

  A data stream that represents a target with certain characteristics is generated by the target generator. An external model of the target is implemented by the target generator.

The target generator creates the exact, or true values of the target's position against which the output of the target tracking filter will be compared.

- **Radar Simulator**

  A mathematical model of the radar system is implemented in the radar simulator. The radar simulator takes the target generator output as input, and modifies it with the mathematical model of the radar system in order to produce an output data stream that represents the target detections from the modeled radar system. This is done by using a noise model to create the target detection errors that would be produced in an actual radar system. In the context of the radar simulator, the target detection errors that would be produced in an actual radar system, could be introduced by:

  - the physical radar system, *i.e.* mechanical design, electronics, etc.;
  - the environment in which the physical radar system operates, *i.e.* clutter, weather, etc.

- **Filter**

  The mathematical model of the target tracking filter is implemented in this component. The filter takes the radar simulator output as input, and generates the filtered target track as output.

- **Display & Data Capture**

  The various components of the radar target tracking filter simulator is displayed and the data captured for processing in this component. The output of the target tracking filter is compared against the output of the target generator in order to determine the accuracy of the target tracking filter.

5.3 Operating Modes

There are two modes of operation for the radar target tracking filter simulator, as described below.

- **Single Shot**

  In the single shot mode, the radar target tracking filter simulator is only executed once and the output is displayed. This mode is generally used for demonstration or testing purposes.

- **Batch**

  In the batch mode, the radar target tracking filter simulator is executed multiple times in succession and the aggregated output is displayed and stored to file. This mode is used for performing the quantitative (Monte Carlo) simulations of the target tracking filter.

5.4 Target Generator

In the radar target tracking filter simulator, a data stream that represents a target with certain characteristics is generated by the target generator. An external model of the target is implemented by the target generator. The purpose of the target generator is to create the exact, or true values of the target's position against which the output of the target tracking filter will be compared. The target generator's output is sent to the radar simulator component. A number of design issues relating to the target's characteristics and the external model used in the target generator will be discussed in Section 5.4.1, Section 5.4.2 and Section 5.4.3.

5.4.1 Target Maneuvering Sequences

There are four target maneuvering sequences that have been considered in the development of the external model of the radar target tracking filter simulator as described below [6].

- **Sequence 1**

  The target was in a straight-line motion, and is still in a straight-line motion.

- **Sequence 2**

  The target was in a straight-line motion, and is now performing a maneuver.

- **Sequence 3**

  The target was performing a maneuver, and is still performing a maneuver.

- **Sequence 4**

  The target was performing a maneuver, and is now in a straight-line motion.

These sequences are generally present in, what is termed, an arbitrary target.

5.4.2 Arbitrary Target

In target tracking filter engineering, the performance of a target tracking filter needs to be evaluated

for, what is termed, an arbitrary target or process [6]. An arbitrary target or process is defined to be non-polynomial. This means that it must not be possible to model the complete target track with a polynomial, be it a 0^{th} order polynomial or a 10^{th} order polynomial. However, an arbitrary target or process can be modeled piecewise by a number of polynomials. In practice it is easier to use sinusoidal functions to create the external model, which was done for the radar target tracking filter simulator.

5.4.3 External Model

The external model that had been chosen for implementation in the radar target tracking filter simulator, is a scenario of a fighter aircraft diving into an attack position on a ground target in a spiraling pattern and then exiting the area by climbing in a spiraling pattern [6]. This specific external model had been chosen for two reasons. Firstly, it is an arbitrary target as defined in Section 5.4.2, and thus a valid external model against which to evaluate the performance of a target tracking filter. Secondly, it represents a real-world scenario and would thus closely match the practical application of a target tracking filter. The external model was generated using the following generic equation:

$$z_{dive} = \cos\left(\frac{2\times\pi\times t}{m}\right)$$

$$z_{spiral} = a\times\sin\left(\frac{b\times 2\times\pi\times t}{m}\right)$$

$$z_{target} = z_{dive} + z_{spiral}$$

with t *the detection interval,*

m *the simulation length (x-axis),*

a *the maximum amplitude (peak) of the spiral relative to the average level, and*

b *the rate of the spiral relative to the simulation length, i.e. the number of spirals per simulation length.*

(5.1)

The aircraft's dive maneuver is modeled by z_{dive} and the aircraft's spiral maneuver is modeled by

z_{spiral}. The aircraft's total maneuver z_{target} is modeled as the sum of these two equations at a specific detection interval t for a simulation length m. An example of the exact, or true values of the target's position for this scenario as generated by the radar target tracking filter simulator, is show in *Figure 5.2*.

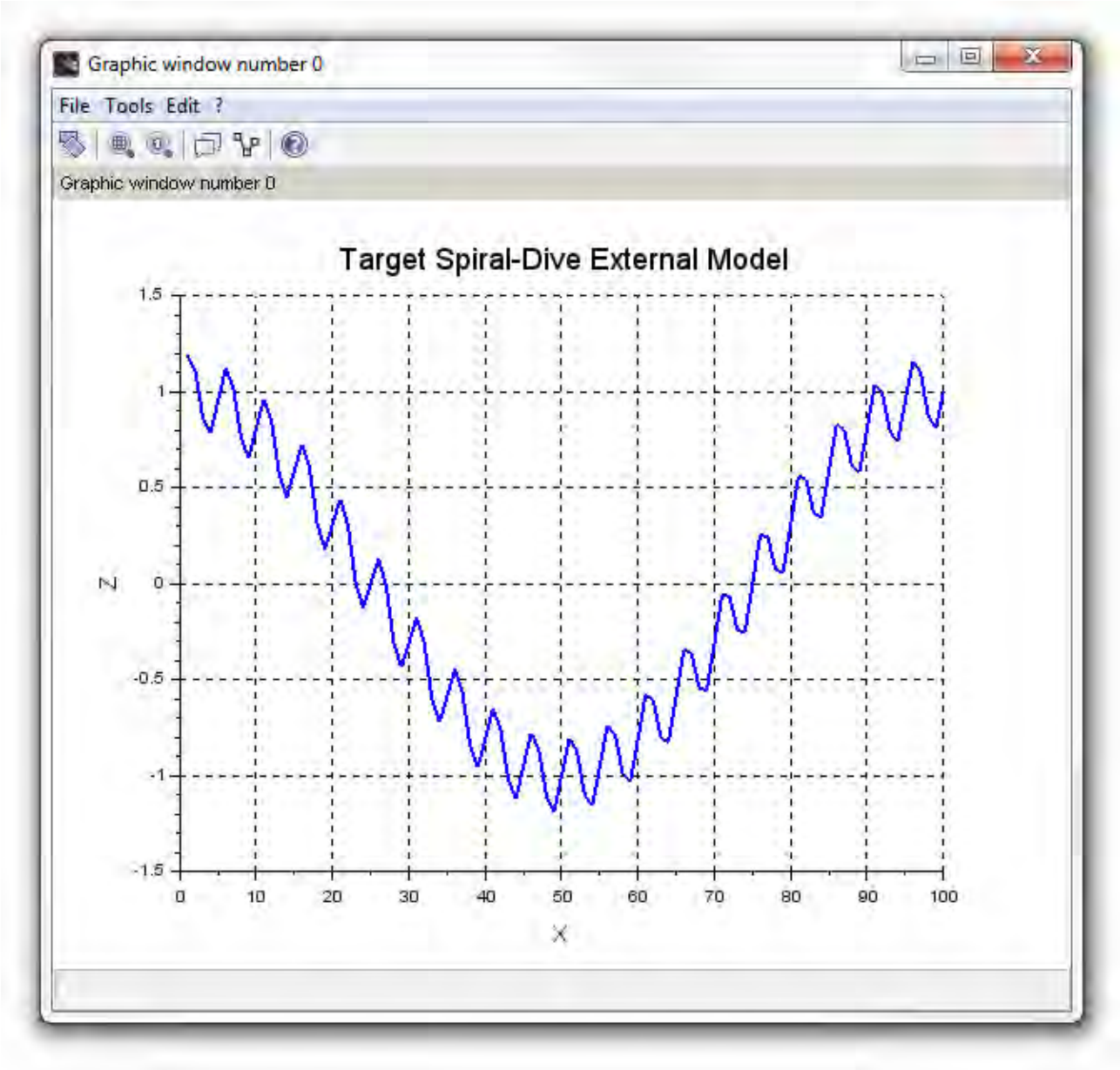

Figure 5.2: Target Spiral-Dive External Model

5.5 Radar Simulator

A mathematical model of the radar system is implemented in the radar simulator. The radar simulator takes the target generator output as input, and modifies it with the mathematical model of the radar system in order to produce an output data stream that represents the target detections from

the modeled radar system. A noise model is applied to the target generator output in order to create and simulate the target detection errors that would be produced in an actual radar system. In practice, these errors are produced through various mechanisms, like the physical characteristics of the radar system or environmental factors [1]. The radar target tracking filter simulator implements the following noise models for evaluation, namely:

1. Gaussian (or normal distribution);
2. non-Gaussian, termed Laplacian or double-exponential [62].

These two noise models are defined and discussed in Section 5.5.1 and Section 5.5.2.

5.5.1 Gaussian (or Normal Distribution) Noise Model

For the Gaussian noise model, a Gaussian or normal distribution random variable is used. The probability density function (PDF) of the Gaussian, or normal distribution, random variable is defined by [62] and [70], as follows:

$$p_X(x) = \frac{1}{\sigma \times \sqrt{2 \times \pi}} \times e^{\left\{-\frac{(x-\mu)^2}{2 \times \sigma^2}\right\}} \; ; \; -\infty < x < +\infty$$

with σ *the standard deviation of the Gaussian random variable,*

σ^2 *the variance of the Gaussian random variable,* (5.2)

x *the Gaussian random variable, and*

μ *the mean of the Gaussian random variable.*

5.5.2 Non-Gaussian (Laplacian or Double-Exponential) Noise Model

For the non-Gaussian noise model, a Laplacian or double-exponential random variable is used [62]. The probability density function (PDF) of the Laplacian, or double-exponential, random variable is define by [62] and [70], as follows:

$$p_X(x) = \frac{1}{\sigma \times \sqrt{2}} \times e^{\left\{-\frac{|x| \times \sqrt{2}}{\sigma}\right\}} \; ; \; -\infty < x < +\infty$$

σ *the standard deviation of the Laplacian random variable,*

x the Laplacian random variable, and (5.3)

μ *the mean of the Gaussian random variable.*

5.6 Tracking Filter

In the radar target tracking filter simulator, the mathematical model of the target tracking filter is implemented in the filter component. The filter component takes the radar simulator output as input, and generates the filtered target track as output. The radar target tracking filter simulator implements the following filter models for evaluation, namely:

1. sequential importance re-sampling (SIR) particle filter;
2. alpha-beta filter;
3. Singer-Kalman filter.

The filtered target track output is sent to the display and data capture component in the radar target tracking filter simulator.

5.7 Display & Data Capture

The various components of the radar target tracking filter simulator is displayed and the data captured for processing in this component. The output of the target tracking filter is compared against the output of the target generator in order to determine the accuracy of the target tracking filter. The radar target tracking filter simulator implements the following display and data capture functions:

1. selectable radar plots:
    1. true target track;
    2. target detections;
    3. filtered target track;
2. selectable auxiliary plots:

detection versus filter error plot;

3. simulation results:
    1. root mean squared error (RMSE) for the filtered target position;
    2. RMSE for the detected target position;
    3. total computation time of the filter component of the simulation.

For the batch simulation, by default only the cumulative RMSE and computation time is displayed and stored as it would take too long to plot each simulation run of the batch simulation in the various displays. However, the option to display each simulation run of the batch simulation is selectable via the user interface, as described in Section 5.8.

5.8 User Interface

The user interface (UI) is used to configure the radar target tracking filter simulator. The function of each component in the UI and a described thereof is presented as follows:

- **Filter Type:** radio buttons to select the type of filter to use. The available options are:
    - Particle:

        Particles [1 – 100,000]: number of particles to use in the particle filter;
    - Alpha-Beta:
        - Alpha: the position gain of the filter;
        - Beta: the velocity gain of the filter;
    - Singer-Kalman.
- **Noise Model:** radio buttons to select the noise model to apply to the target's true position in order to simulate the radar target detections. The available options are:
    - None;
    - Gaussian (refer to Section 5.5.1);
    - Laplacian (refer to Section 5.5.2).
- **Simulation Type:** radio buttons to select the type of simulation to perform. The available

options are:

- Single Shot (refer to Section 5.3);
- Batch (refer to Section 5.3):
  - Size [10 – 100,000]: select the number of simulation runs in the batch simulation;
  - Display Lock: check box to disable the display of each simulation run of the batch simulation.

- **Visible Plots:** check boxes to select the radar plots to display. The available options are:
  - True Track;
  - Detections;
  - Filtered Track.
- **Plot Type:** radio buttons to select the type of main plot. The available options are:

  2D Plot (Horizontal).
- **Auxiliary Plots:** check boxes to enable the auxiliary plots to display. The available options are:
  - Error Plot (refer to Section 5.9.1);
  - PCRLB Plot (refer to Section 5.9.2).
- **Detection Parameters:** check boxes to select the additional detection parameters that can be selected. The available options are:
  - Missed: check box to enable the simulation of missed target detections;

    [%]: the number of missed target detections to simulate expressed as a percentage of the total target detections;
  - Transient: check box to enable the simulation of a transient target detection;

    [%]: level of the detection transient to simulate expressed as a percentage of the average target detection level.
- **Target Model:** a drop-down list with the available target models that can be selected. The

available options are:

- Spiral-Dive (refer to Section 5.4.3);
- Random;
- Sinusoidal;
- Linear.

- **Run Simulation:** a push button that starts the simulator with the selected configuration.

The user interface (UI) of the radar target tracking filter simulator is shown on the left-hand side in *Figure* 5.3.

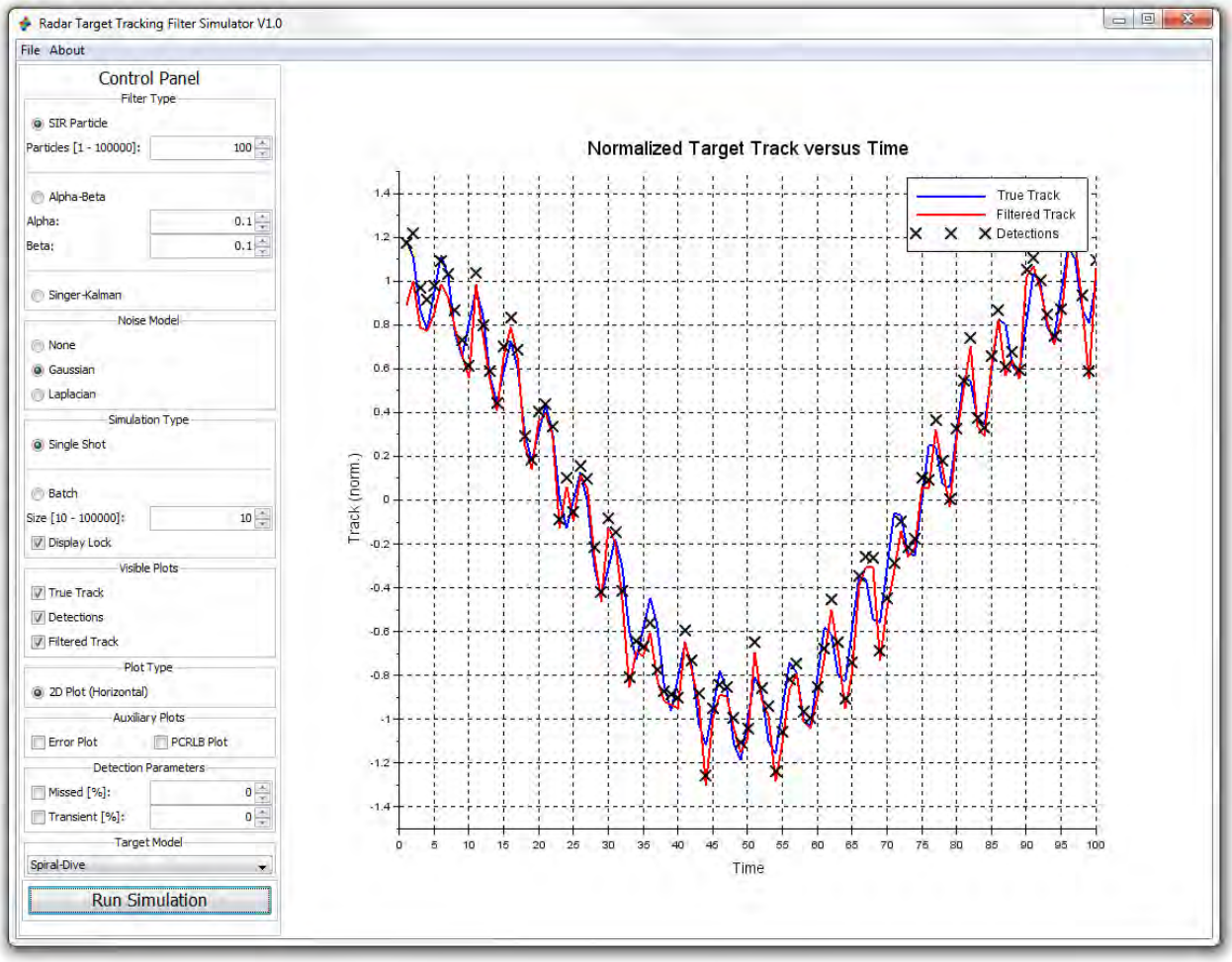

Figure 5.3: Radar Target Tracking Filter Simulator User Interface

The radar display of the radar target tracking filter simulator is shown on the right-hand side in *Figure 5.3*.

5.9 Auxiliary Plots

5.9.1 Error Plot

The error plot displays the detection errors as blue bar graphs and the filter errors as a green line. The purpose of this plot is to visually compare the performance of the radar target tracking filter with respect to the target detections. The error plot of the radar target tracking filter simulator is shown in *Figure 5.4*.

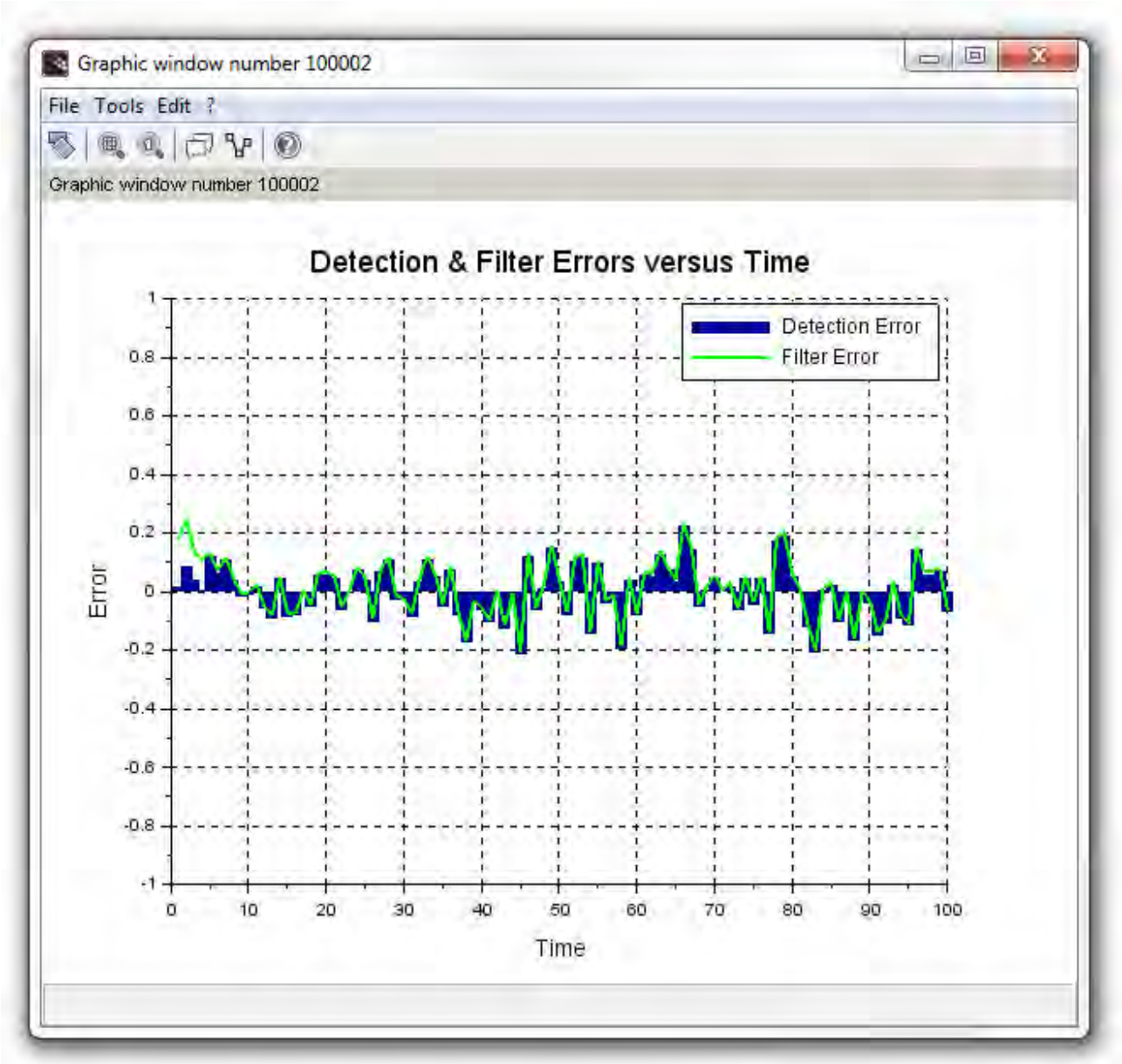

Figure 5.4: Radar Target Tracking Filter Simulator Error Plot

The error plot is a useful tool in identifying bias in the radar target tracking filter [6]. If the filter error line is offset from the nominal error level, then it would indicate bias in the radar target

tracking filter, or in the model of the radar system, for example if improper boresighting has been modeled. Furthermore, transients in the output of a radar target tracking filter can be caused by the initialization of certain types of the radar target tracking filters [2].

5.9.2 Posterior Cramér-Rao Lower Bound Plot

If the particle filter is selected as the type of filter to use in the simulator, then the posterior Cramér-Rao lower bound (PCRLB) plot for the particle filter can be enabled. The PCRLB plot and its significance to the particle filter is discussed in Section 6.7.2. The PCRLB plot of the particle filter in the radar target tracking filter simulator is shown in *Figure* 5.5.

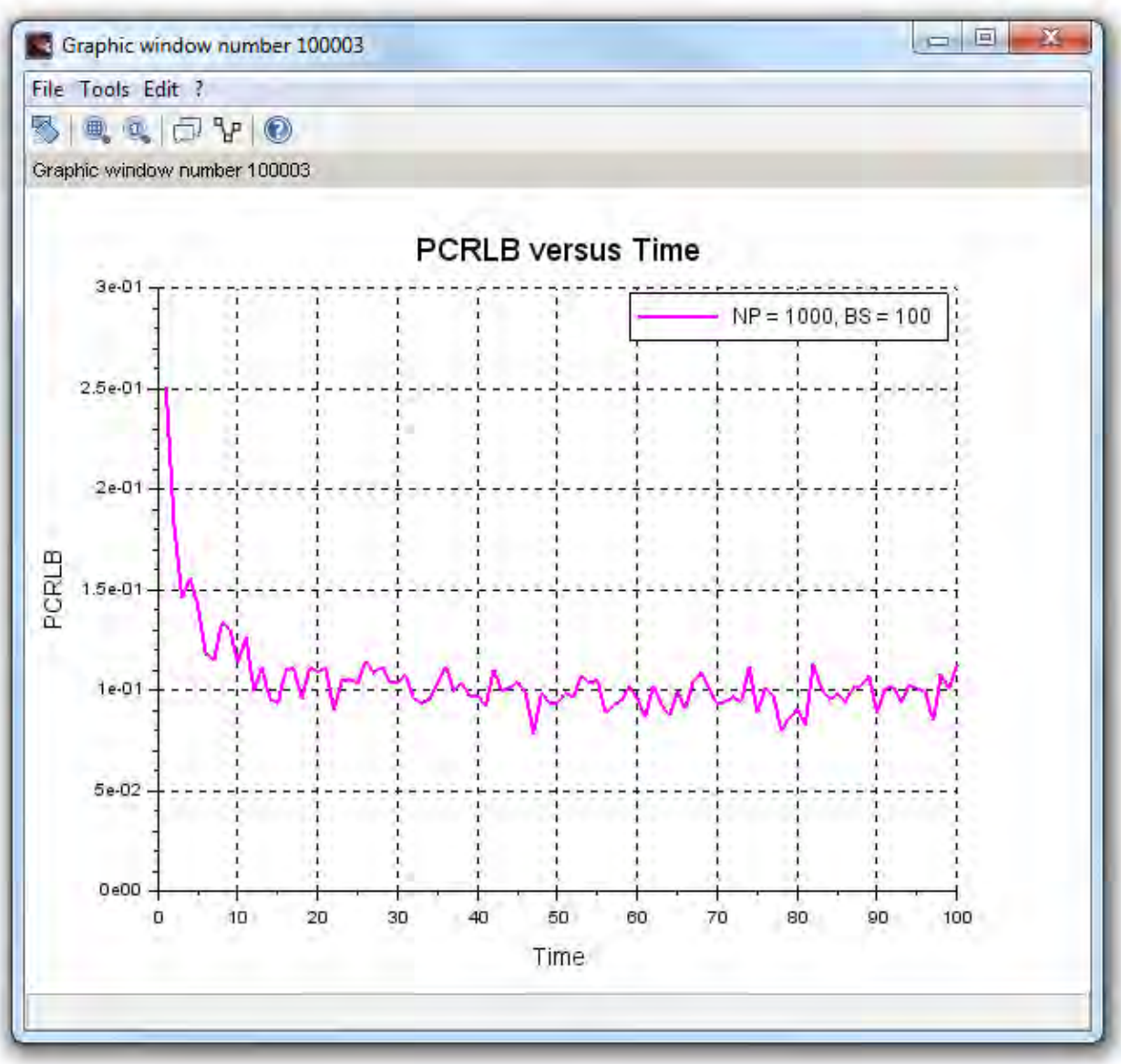

Figure 5.5: Radar Target Tracking Filter Simulator PCRLB Plot

The number of particles (NP) used in the particle filter and the batch size (BS) of the simulation is shown in the plot legend. The data of the PCRLB plot is saved in the data file as described in Section 5.10.

5.10 Simulation Results

Once the simulation has completed, a dialogue window is shown with the root mean squared errors for both the detected and filtered target positions, the total simulation time, as well as the location and file name of the file containing the saved simulation results. An example of the simulation results dialogue window is shown in *Figure 5.6*.

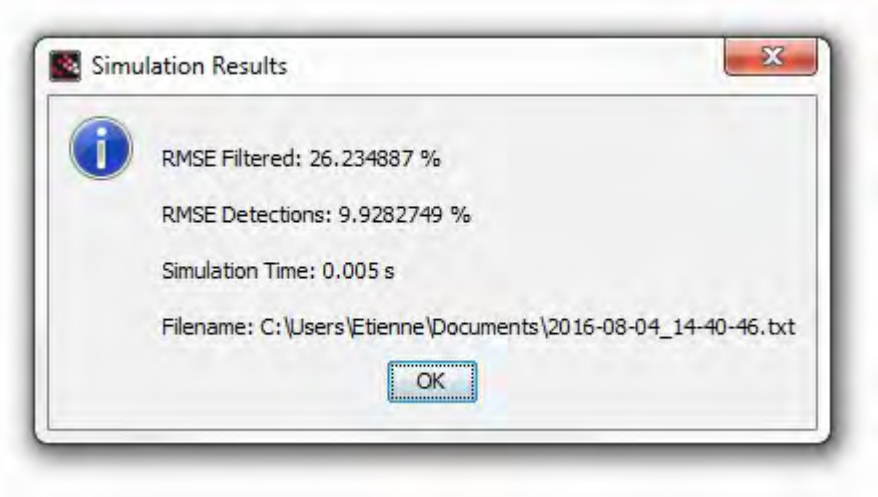

Figure 5.6: Simulation Results Dialogue Window

The simulation results are saved to an automatically named text file. The name of the results file is derived from the current date and time of the computer system on which the simulation software is executed, in the following format: *yyyy-mm-dd_hh-mm-ss.txt*. The location of the results file is derived from the current path from which the simulation software is executed. The configuration parameters of the simulation software is also saved in the results file for reference purposes. Additionally, the data from the simulation is saved in a separate, automatically named text file. The name of the data file is also derived from the current date and time of the computer system on which the simulation software is executed, in the following format: *yyyy-mm-dd_hh-mm-ss_data.txt*. The location of the data file is also derived from the current path from which the simulation software is executed. This allows for the simulation data to be further analyzed independently or stored for reference purposes.

5.11 Assumptions & Constraints

For the development of the radar target tracking filter simulator, there were a number of assumptions that have been made and constraints that have been placed on the simulator, as discussed below:

- Assumed the radar being simulated is properly boresighted.
  - In practice there will be alignment and pointing errors in the radar system that leads to bias or systematic errors in the target tracking filter [6].
  - Since these errors are unknown until the radar system is calibrated, they are not considered in the radar target tracking filter simulator.
- Assumed a constant target detection interval.
  - A constant detection interval, either as scan rate or pulse repetition frequency (PRF), is typical in a track-while-scan (TWS) radar or a tracking radar.
  - For a pulsed radar the PRF is defined as [2]:

$$PRF = \frac{1}{PRI}$$

with PRI the pulse repitition interval, and

the duty cycle as: (5.4)

$$d_t = \tau \times PRF = \frac{\tau}{PRI}$$

with τ the radar pulse width.

  - Since this is the type of radar system that will be used in practice, this assumption is used in the radar target tracking filter simulator.
- Assumed a nearly constant speed motion model for the target being tracked.
  - A nearly constant speed motion model characterizes the motion of a target performing maneuvers with control surfaces [2], for example an aircraft.
  - Since this is the type of target that would be tracked in practice, this motion model is used in the radar target tracking filter simulator.

- Assumed an external model based on polynomials.
    - The external model describes the physical process (target) that is being observed by the radar system as is used by the target generator in the radar target tracking filter simulator [6].
    - Polynomials can be used to approximate arbitrary external models without forcing functions over an appropriate time interval [6].
    - Since this is the type of external model that matches the type of target that would be tracked in practice, an external model based on polynomials is used in the radar target tracking filter simulator.
- Allow for the presence of missed detections.
    - Missed detections in radar measurements are caused by a number of factors including, target masking, signal amplitude fluctuations, etc. [2].
    - Since the exact quantity of missed detections present in typical radar measurements was unknown, the capability to define some level of missed detections has been included in the radar target tracking filter simulator.
- Allow for the presence of detection outliers.
    - Detection outliers in radar measurements are generally caused by interference in the environment or radar system [2].
    - Since the exact quantity of detection outliers present in typical radar measurements was unknown, the capability to define some level of detection outliers have been included in the radar target tracking filter simulator.
- Assumed no jitter was present in the radar measurements.
    - Irregular radar measurements cause by jitter occur in practical radar systems[69], but the jitter needs to be quantified.
    - The normalized time for the radar measurements is defined as [69]:

$$\eta = \frac{t - t_0}{\tau}$$

with t *the current time of a track,*

t_0 *the start time of a track, and*

τ *the expected update period.* (5.5)

- The normalized delta-time (caused by jitter) for the radar measurements is defined as [69]:

$$\zeta = \frac{\delta}{\tau}$$

with δ *the jitter, and*

τ *the expected update period.* (5.6)

- The normalized time including jitter for the radar measurements is thus defined as [69]:

$$\eta + \zeta = \frac{t + \delta - t_0}{\tau}$$

with t *the current time of a track,*

t_0 *the start time of a track,*

τ *the expected update period,*

δ *the jitter, and*

ζ *the normalised delta-time.* (5.7)

- The correct time can also be recovered by de-normalizing as follows [69]:

$$t = (\eta \times \tau) + t_0$$

$$t + \delta = (\eta + \zeta) \times \tau + t_0$$

$$\delta = \zeta \times \tau$$

with t *the current time of a track,*

t_0 *the start time of a track,* (5.8)

η *the normalized time,*

τ *the expected update period,*

δ *the jitter, and*

ζ *the normalised delta-time.*

- Since the quantity of jitter present in typical radar measurements was unknown, no jitter was included in the radar target tracking filter simulator.

5.12 Summary

In this chapter the quantitative simulations that have been performed in order to evaluate and analyze the operation of the particle filter as a radar target tracking filter, were discussed. The design and operation of the radar target tracking filter simulator that was developed in order to perform the quantitative simulations was discussed. The architecture of the radar target tracking filter simulator was presented through a detailed discussion of each component in the radar target tracking filter simulator architecture. Finally, the operation of the radar target tracking filter simulator was demonstrated through the use of images of the user interface and various simulator displays. In Chapter 6, the performance of the particle filter as a radar target tracking filter is evaluated and analyzed from the quantitative simulations conducted with the radar target tracking filter simulator described in this chapter.

Chapter 6 : Quantitative Simulation, Evaluation and Analysis of Particle Filter

6.1 Introduction

The quantitative simulation, evaluation and analysis of the performance of the sequential importance re-sampling (SIR) particle filter particle filter as a radar target tracking filter is discussed in this chapter. The quantitative simulations have been performed through the use of the radar target tracking filter simulator that had been developed for this dissertation, as described in Chapter 5. The performance of the particle filter is evaluated through a number of different filter and simulation configurations. A number of comparison simulations with other target tracking filters have also been performed.

6.2 Quantitative (Monte Carlo) Simulation Overview

A central issue in using the Monte Carlo method for quantitative simulation is determining the number of Monte Carlo simulation iterations to perform so that the result of the simulation is statistically significant and achieves a specific accuracy in the simulation result [71]. If an infinite number of Monte Carlo simulations were performed the population mean $\mu_{\bar{x}}$, the population standard deviation $\sigma_{\bar{x}}$, and the population variance $\sigma^2_{\bar{x}}$, would be calculated [72]. The population statistics are the *actual* statistics of the unknown quantity being calculated. For a finite number of Monte Carlo simulations being performed the sample mean $\bar{x}$, the sample standard deviation S_x, and the sample variance S^2_x, would be calculated [72]. Since it is impractical to perform an infinite number of Monte Carlo simulations, the technique that had been employed in this dissertation to determine the number of Monte Carlo simulations to perform, had been based on confidence intervals [73].

As stated before in Section 4.3.2, the Monte Carlo method, for a *99.75%* confidence level in the range $\pm 3\sigma$, can be expressed as the probability [64]:

$$P\left\{\left|\frac{1}{N}\sum_{j=1}^{N}\xi_j - m\right| < \frac{3\times\sigma}{\sqrt{N}}\right\} \approx 0.997$$

with N *the number of independent random variables,*

ξ *the random variable,*

m *the unknown quantity to calculate, and*

σ *the square root of the variance (i.e. standard deviation).* (6.1)

From this it follows that the arithmetic mean of the independent random variables will be approximately equal to the unknown quantity to calculate, namely *m*. Importantly, as the number of independent random variables *N* increases, the error in the estimate of the unknown quantity to calculate *m*, approaches zero. In essence, a confidence interval is a statistical measure that aims to provide information about a specific population statistic based on a sample statistic. There are generally two methods in which confidence intervals could be used in Monte Carlo simulations [73]. Firstly, the sample mean of a Monte Carlo simulation of a predefined number of iterations could be determined, and the confidence interval width could then be determined around this sample mean. Secondly, the number of Monte Carlo simulation iterations to be performed to reach a specific confidence interval width could be determined through the approximation of the population mean and the population standard deviation. The second method had been used in determining the number of Monte Carlo simulation iterations to perform for this dissertation and is discussed in Section 6.3.

6.3 Number of Monte Carlo Simulation Iterations

The upper and lower bounds of the confidence level (CL), is defined by [71] as follows:

$$(L,U)_{CL} = \mu_X \pm \left(z_c \times \frac{\sigma_X}{\sqrt{n}} \right)$$

with μ_X *the population mean,*

z_c *the confidence coefficient for each confidence interval,*

σ_X *the population standard deviation, and*

n *the number of simulation iterations.*

(6.2)

However, since the population mean $\mu_{\bar{x}}$, and the population standard deviation $\sigma_{\bar{x}}$, is not known, the upper and lower bounds of the confidence level (CL), is defined by [71] as follows:

$$(L,U)_{CL} = \bar{x} \pm \left(z_c \times \frac{S_x}{\sqrt{n}} \right)$$

with $\bar{x}$ *the sample mean,*

z_c *the confidence coefficient for each confidence interval,*

S_x *the sample standard deviation, and*

n *the number of simulation iterations.*

(6.3)

If it is assumed that the population standard deviation is approximately equal to the sample standard deviation, *i.e.* $\sigma_{\bar{x}} \approx S_x$, and the population mean is approximately equal to the sample mean, *i.e.* $\mu_{\bar{x}} \approx \bar{x}$, and it is further assumed that the confidence interval represents twice the maximum error, i.e. $error_{max} = \left(z_c \times \frac{S_x}{\sqrt{n}} \right)$, then the number of simulation iterations can be expressed as follows [71], [73]:

$$n = \left[\frac{100 \times z_c \times S_x}{E \times \bar{x}} \right]^2 \tag{6.4}$$

with z_c *a statistical constant for each confidence interval,*

S_x *the sample standard deviation,*

E *the error expressed as a percentage, and*

$\bar{x}$ *the sample mean.*

Since it was assumed that the population standard deviation was approximately equal to the sample standard deviation, and that the population mean was approximately equal to the sample mean, the population standard deviation and population mean must initially be approximated by running a single simulation a large number of times, in this case 10^7 times as suggested by [71]. From this single simulation, the approximate population standard deviation and the population mean was calculated as the sample standard deviation and the sample mean. Using these approximated population statistics and (6.4), the number of Monte Carlo simulation iterations for a given confidence level was calculated as shown in *Table 6.1*.

Table 6.1: Confidence Level versus Monte Carlo Simulation Iterations

Confidence Level (%)	Error (%)	z_c	n
99.75	0.25	3.0000	647533.856
99.00	1.00	2.5800	29932.252
98.00	2.00	2.3300	6103.119
96.00	4.00	2.0500	1181.103
95.50	4.50	2.0000	888.249
95.00	5.00	1.9600	690.991
90.00	10.00	1.6450	121.684
80.00	20.00	1.2800	18.419
68.00	32.00	1.0000	4.391
50.00	50.00	0.6745	0.818

The number of Monte Carlo simulation iterations performed to evaluate the particle filter was chosen to be *6,200* that presented a confidence level of *98%*, according to *Table 6.1*. This choice was based on the total number of simulations to perform, the time required to perform these

simulations, and the computational resources available to perform these simulations.

6.4 Quantitative Simulation Configurations

A number of different configurations of the quantitative simulations had been performed to evaluate different parameter settings of the particle filter as a radar target tracking filter, compared to a number of other radar target tracking filters, for various noise models, and under various simulated operational conditions. The different configurations are discussed in this section.

6.4.1 Filter Type

6.4.1.1 Sequential Importance Re-sampling Particle Filter

The SIR particle filter had been selected for simulation as it was shown to be the dominant particle filter from the literature. The SIR particle filter had been configured for different numbers of particles, namely *1*, *10*, *100*, and *1,000*, to determine the effect of the number of particles on the accuracy performance and normalized processing time of the filter. The SIR particle filter had been discussed in Section 4.6.

6.4.1.2 Alpha-Beta Filter

The alpha-beta filter with fixed gain had been selected for simulation as it presented a low computational complexity, low accuracy radar target tracking filter. The alpha-beta filter had been discussed in Section 3.3.1. The position gain and velocity gain values for the alpha-beta filter was selected through experimentation as *0.5* and *0.5* respectively, to produce a reasonably filtered track of the external model described in Section 5.4.3.

6.4.1.3 Singer-Kalman Filter

The Kalman filter selected for simulation and comparison was the Singer-Kalman filter as described by [74]. This specific implementation of the Kalman filter was selected as it is a special case of the Kalman filter as described in Section 3.3.2, that is governed by a target model that represents the motion of a manned maneuvering target. As such, it was deemed to be appropriate filter for tracking an arbitrary target with an external model as described in Section 5.4.3. The transition matrix for the Singer-Kalman filter is given by [74] as follows:

$$\phi(T,\alpha)=\begin{bmatrix} 1 & T & \frac{1}{\alpha^2}\left[-1+\alpha T+e^{-\alpha T}\right] \\ 0 & 1 & \frac{1}{\alpha}\left[1-e^{-\alpha T}\right] \\ 0 & 0 & e^{-\alpha T} \end{bmatrix} \tag{6.5}$$

with T the target position in time, and

α the reciprocal of the maneuver (acceleration) time constant.

The implementation of the Singer-Kalman filter was done as described by [75]. The Singer-Kalman filter is also referred to in the literature as the Singer-$\alpha\,\beta\,\gamma$-Kalman filter.

6.4.2 Noise Model

6.4.2.1 Gaussian Noise

The Gaussian noise model had selected as it represented the standard process noise model used in radar target tracking filter simulation, namely additive white Gaussian noise (AWGN). The Gaussian noise model had been discussed in Section 5.5.1.

6.4.2.2 Laplacian Noise

The Laplacian noise model had been selected as it represented a non-Gaussian noise model with which to evaluate the particle filter as a radar target tracking filter. The Laplacian noise model had been discussed in Section 5.5.2.

6.4.3 Detections

6.4.3.1 Ideal Detections

In an ideal detection scenario, ever transmitted radar pulse or waveform would generate a return from the target being tracked. Although this scenario might not represent a real-world condition, this ideal detection scenario had been used as a baseline against which to evaluate the SIR particle filter as a radar target tracking filter.

6.4.3.2 Missed Detections

In a non-ideal detection scenario, some target detections will be missed by the radar due to various

environmental, operational or target conditions. The missed detections scenario had been simulated by randomly inserting missed detections in the target detection data from a level of *1%* to *10%* of the total detections in increments of *1%*. The missed detections scenario had been used to evaluate the particle filter as a radar target tracking filter under these non-ideal, and thus more practical, real-world condition.

6.4.3.3 Transients

The evaluation of the particle filter as a radar target tracking filter in the presence of a transient condition in the target detection data had also been simulated. Transients can be caused by some environmental or operational conditions. The ability of a radar target tracking filter to handle these transient events without becoming unstable or oscillating is important in tracking filter engineering [6].

6.5 Quantitative Simulation Summary

A summary of the quantitative simulations that had been performed for the dissertation is shown in *Table 6.2*. The selection of the filter parameters have been described in Section 6.4.1. For each simulation configuration, *6,200* Monte Carlo simulation iterations were performed based on the selection method as described in Section 6.3.

Table 6.2: Quantitative Simulation Summary

Filter Type	Filter Parameters	Noise Model	Missed Detections (Increment)	Transient (Level)
SIR Particle	1 ; 10 ; 100 ; 1,000	Gaussian	0%	No
		Laplacian		
Alpha-Beta	0.5 ; 0.5	Gaussian	0%	No
		Laplacian		
Singer-Kalman	None	Gaussian	0%	No
		Laplacian		
SIR Particle	1 ; 10 ; 100 ; 1,000	Gaussian	1% – 10% (1%)	No
		Laplacian		
Alpha-Beta	0.5 ; 0.5	Gaussian	1% – 10% (1%)	No
		Laplacian		
Singer-Kalman	None	Gaussian	1% – 10% (1%)	No
		Laplacian		

Filter Type	Filter Parameters	Noise Model	Missed Detections (Increment)	Transient (Level)
SIR Particle	1 ; 10 ; 100 ; 1,000	Gaussian	0%	Yes (100%)
		Laplacian		
Alpha-Beta	0.5 ; 0.5	Gaussian	0%	Yes (100%)
		Laplacian		
Singer-Kalman	None	Gaussian	0%	Yes (100%)
		Laplacian		

6.6 Sequential Importance Re-sampling Particle Filter Evaluation

6.6.1 Accuracy

The accuracy of the sequential importance re-sampling (SIR) particle filter will be evaluated for an ideal detection, a missed detection, and a transient detection scenario. The metric used in evaluating the accuracy of the SIR particle filter is the root mean squared error (RMSE), that had been discussed in Section 3.4.1.

6.6.1.1 Ideal Detections

The accuracy performance using the RMSE metric, and the processing time performance of the SIR particle filter in Gaussian noise and for an ideal detection scenario, is shown in *Figure 6.1*, with PF denoting the particle filter, A-B denoting the alpha-beta filter, S-K denoting the Singer-Kalman filter, and PT denoting processing time.

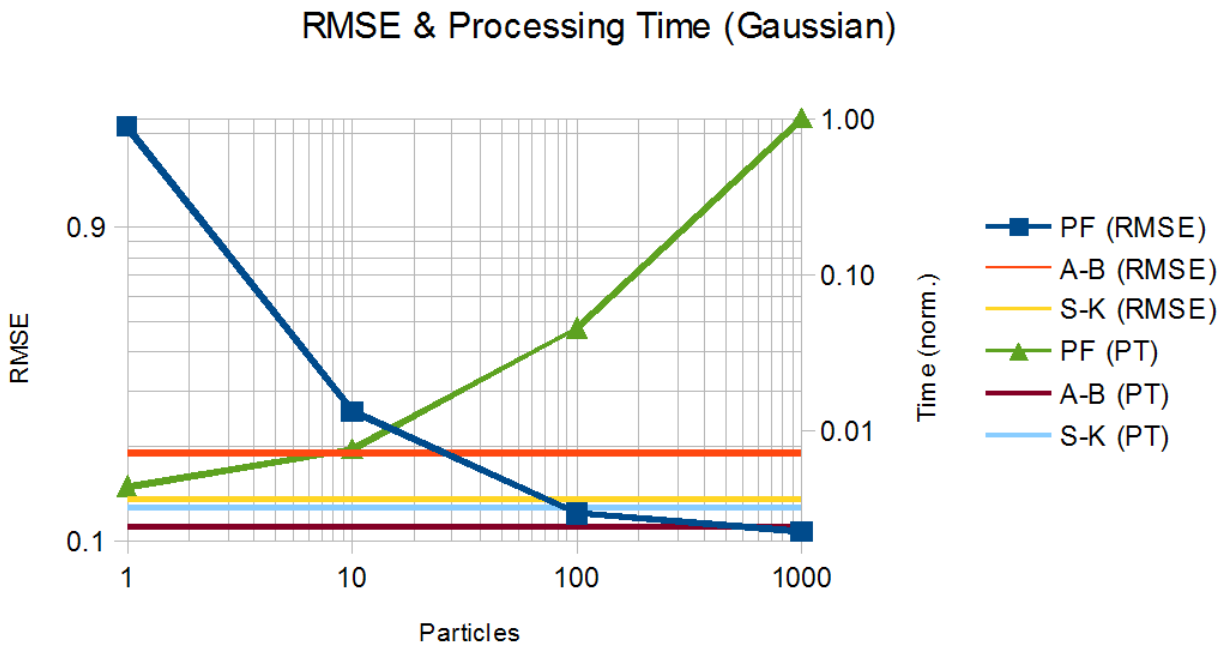

Figure 6.1: RMSE & Processing Time Comparison in Gaussian Noise

It can be observed that as the number of particles used in the SIR particle filter algorithm increases, the RMSE decreases, and thus the accuracy increases. For comparison, the accuracy performance of the alpha-beta filter and Singer-Kalman filter in Gaussian noise and for an ideal detection scenario, is also shown in *Figure 6.1*. The Singer-Kalman filter has better accuracy performance that the alpha-beta filter. For *100* particles used in the SIR particle filter algorithm, the SIR particle filter has better accuracy performance than both the alpha-beta filter and the Singer-Kalman filter. Increasing the number of particles used in the SIR particle filter algorithm to *1,000*, leads to a further RMSE decrease, and thus an accuracy increase.

The accuracy performance using the RMSE metric, and processing time performance of the SIR particle filter in Laplacian noise and for an ideal detection scenario, is shown in *Figure 6.2*, with PF denoting the particle filter, A-B denoting the alpha-beta filter, S-K denoting the Singer-Kalman filter, and PT denoting processing time.

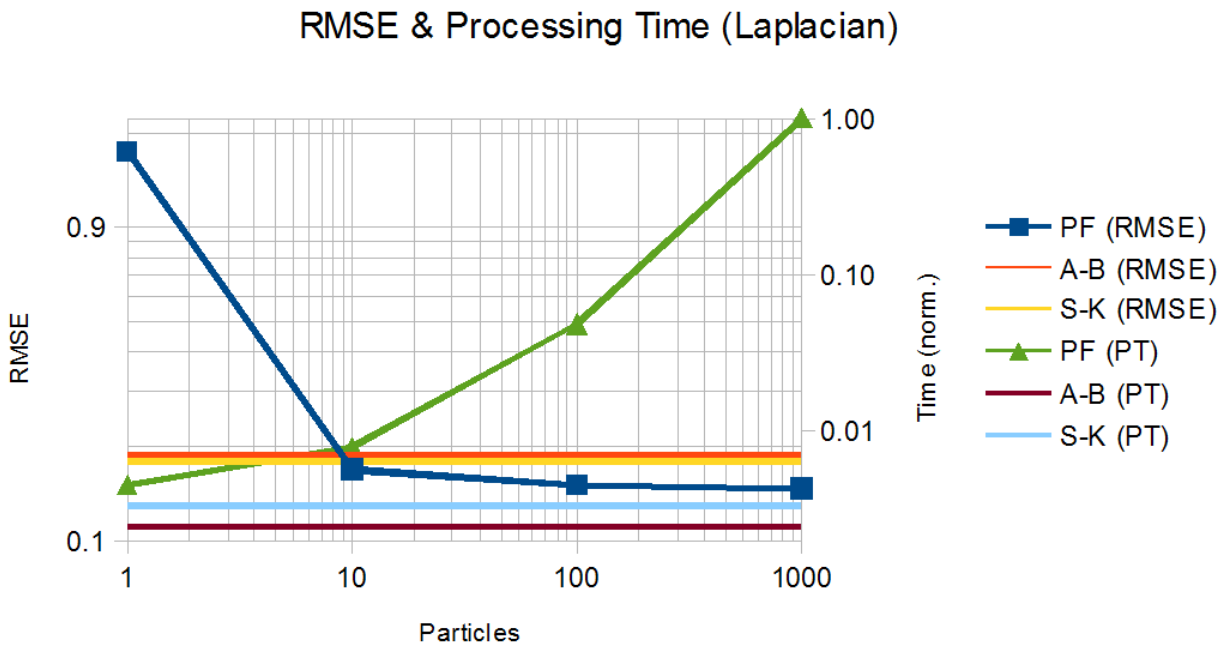

Figure 6.2: RMSE & Processing Time Comparison in Laplacian Noise

It can be observed that as the number of particles used in the SIR particle filter algorithm increases, the RMSE decreases, and thus the accuracy increases. For comparison, the accuracy performance of the alpha-beta filter and Singer-Kalman filter in Laplacian noise and for an ideal detection scenario, is also shown in *Figure 6.2*. The Singer-Kalman filter and the alpha-beta filter has similar accuracy performance in this scenario. For *10* particles used in the SIR particle filter algorithm, the SIR particle filter has better accuracy performance than both the alpha-beta filter and the Singer-Kalman filter. Increasing the number of particles used in the SIR particle filter algorithm to *100* and *1,000*, leads to further RMSE decreases, and thus accuracy increases.

From the results that had been obtained, the following three observations had been made, namely:

- the accuracy performance of the SIR particle filter is a function of the number of particles used in the SIR particle filter algorithm;
- the SIR particle filter has better accuracy performance in Laplacian noise than in Gaussian noise;
- the SIR particle filter has better accuracy performance than both the alpha-beta filter and the Singer-Kalman filter in both Gaussian noise and Laplacian noise.

6.6.1.2 Missed Detections

The missed target detection accuracy performance using the RMSE metric of the SIR particle filter in Gaussian noise, is shown in *Figure 6.3*. The missed target detection scenario was simulated from a level of *1%* to *10%* of the total detections in increments of *1%*. The ideal detections scenario as described in Section 6.6.1.1, is also shown as a baseline.

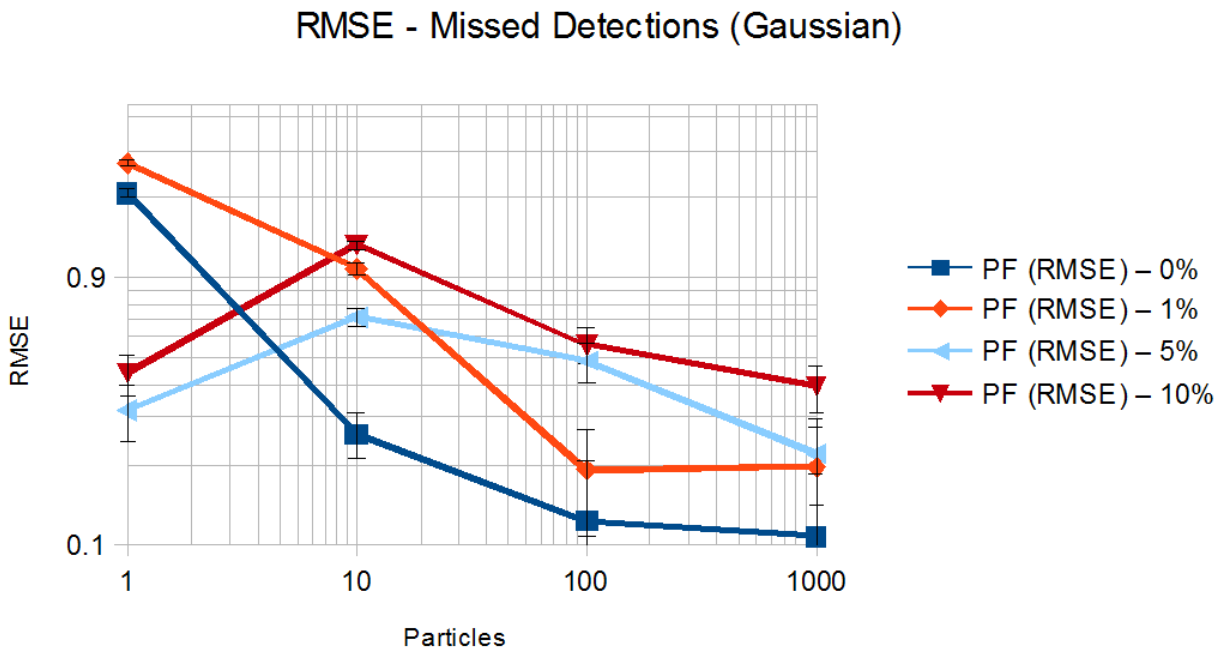

Figure 6.3: Particle Filter RMSE for Missed Detections in Gaussian Noise

In general, it can be observed that as the number of missed detections increases, the RMSE increases, and thus the accuracy decreases. The RMSE can be reduced, and thus the accuracy increased by increasing the number of particles used in the SIR particle filter algorithm.

The missed target detection accuracy performance using the RMSE metric of the SIR particle filter in Laplacian noise, is shown in *Figure 6.4*. The missed target detection scenario was simulated from a level of *1%* to *10%* of the total detections in increments of *1%*. The ideal detections scenario as described in Section 6.6.1.1, is also shown as a baseline.

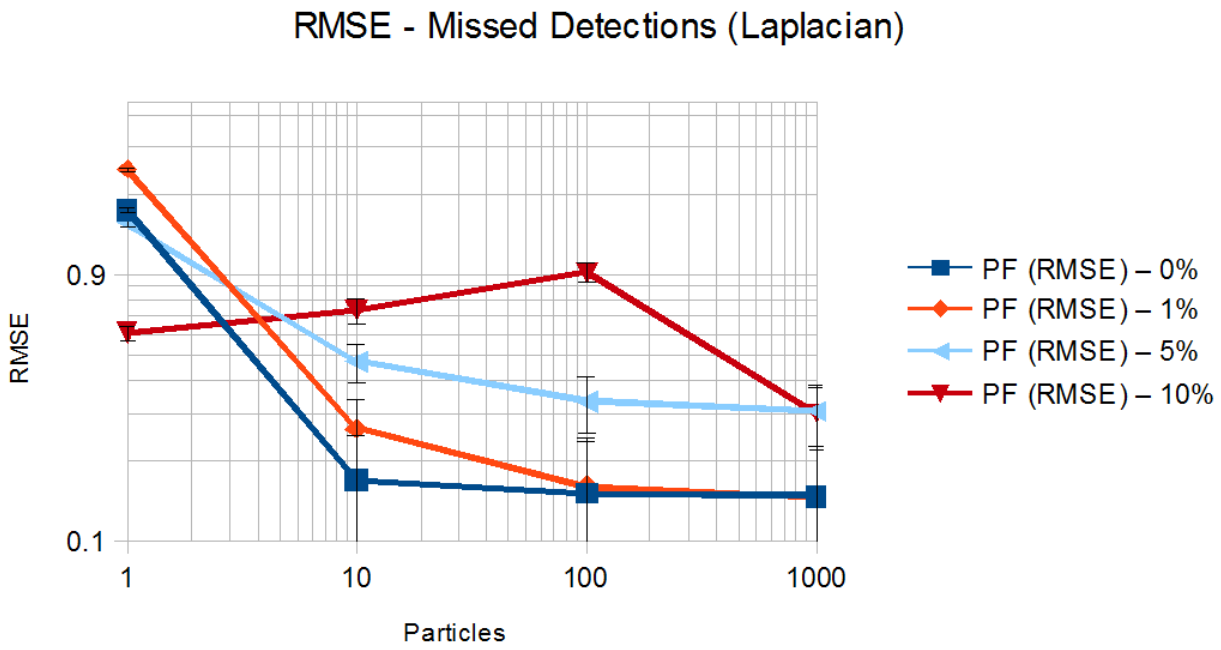

Figure 6.4: Particle Filter RMSE for Missed Detections in Laplacian Noise

In general, it can be observed that as the number of missed detections increases, the RMSE increases, and thus the accuracy decreases. The RMSE can be reduced, and thus the accuracy increased by increasing the number of particles used in the SIR particle filter algorithm. However, the increase in accuracy related to an increase in the number of particles used in the SIR particle filter algorithm for the Laplacian noise scenario, is not as great as for the Gaussian noise scenario.

For comparison, the missed target detection RMSE performance of the alpha-beta filter, the Singer-Kalman filter, and the SIR particle filter with *1,000* particle in Gaussian and Laplacian noise, is shown in *Figure 6.5*, with A-B denoting the alpha-beta filter, S-K denoting the Singer-Kalman filter, and PF ($N = 1000$) denoting the particle filter with *1,000* particles. In general it can be observed that as the number of missed detections increases, the RMSE increases, and thus the accuracy decreases. This is true for both the alpha-beta filter the Singer-Kalman filter, and the particle filter, and for both the Gaussian noise and the Laplacian noise scenarios. On average, the Singer-Kalman filter has better accuracy performance in the missed target detection scenario than the alpha-beta filter for both the Gaussian noise and Laplacian noise scenarios. The particle filter has better accuracy performance in the missed target detection scenario than both the alpha-beta filter and the Singer-Kalman filter, for both the Gaussian noise and Laplacian noise scenarios.

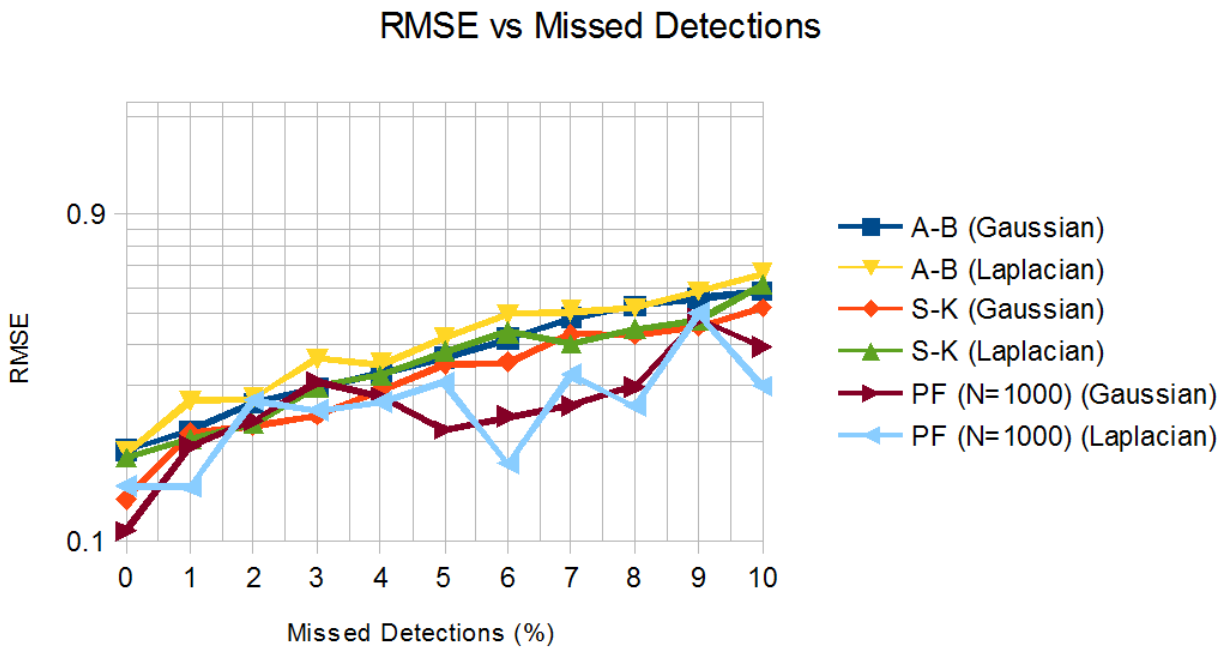

Figure 6.5: Alpha-Beta, Singer-Kalman and SIR Particle Filter RMSE for Missed Detections

From the results that had been obtained, the following two observations had been made, namely:

1. for the missed detection scenario, the accuracy performance of the SIR particle filter can be improved by increasing the number of particles used in the SIR particle filter algorithm;
2. for the missed detection scenario in Laplacian noise, the increase in the number of particles used in the SIR particle filter algorithm had less of an effect on improving the accuracy performance of the SIR particle filter than in the Gaussian noise scenario.

6.6.1.3 Transients

The accuracy performance using the RMSE metric, and the processing time performance of the SIR particle filter in Gaussian noise and for a transient detection scenario, is shown in *Figure 6.6*. It can be observed that as the number of particles used in the SIR particle filter algorithm increases from *10* particles onwards, the RMSE increases, and thus the accuracy decreases. This is the inverse as to what was observed in Section 6.6.1.1, for the ideal detection scenario. This phenomenon could be explained by the SIR particle filter's sensitivity to outliers, as had been reported in Section 4.6. Furthermore, for *10* particles used in the SIR particle filter algorithm, the SIR particle filter had better accuracy performance than had been reported in Section 6.6.1.1. This phenomenon could be explained by the transient countering the effect of the loss of particle diversity in the SIR particle filter, as had been reported in Section 4.6. This means that transients have both a positive and

negative effect on the SIR particle filter. For comparison, the accuracy performance of the alpha-beta filter and Singer-Kalman filter in Gaussian noise and for a transient detection scenario, is also shown in *Figure 6.6*, with PF denoting the particle filter, A-B denoting the alpha-beta filter, S-K denoting the Singer-Kalman filter, and PT denoting processing time. The Singer-Kalman filter again has better accuracy performance that the alpha-beta filter. For *10* particles used in the SIR particle filter algorithm, the SIR particle filter has better accuracy performance than both the alpha-beta filter and the Singer-Kalman filter.

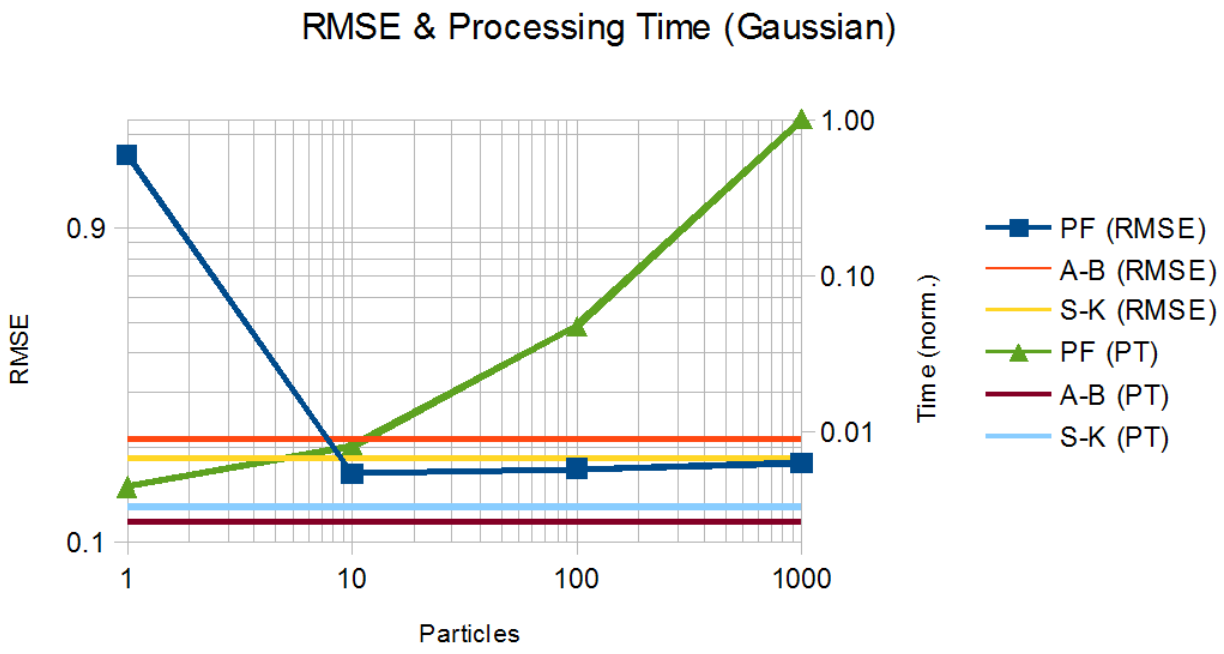

Figure 6.6: RMSE & Processing Time Comparison for Transient Detections in Gaussian Noise

The accuracy performance using the RMSE metric, and the processing time performance of the SIR particle filter in Laplacian noise and for a transient detection scenario, is shown in *Figure 6.7*. It can be observed that as the number of particles used in the SIR particle filter algorithm increases, the RMSE decreases, and thus the accuracy increases, but only up to approximately *100* particles. For comparison, the accuracy performance of the alpha-beta filter and Singer-Kalman filter in Laplacian noise and for a transient detection scenario, is also shown in *Figure 6.7*, with PF denoting the particle filter, A-B denoting the alpha-beta filter, S-K denoting the Singer-Kalman filter, and PT denoting processing time. The alpha-beta filter has better accuracy performance than the Singer-Kalman filter in this scenario, which is the inverse as to what was observed in Section 6.6.1.1. This phenomenon could be explained by the position gain and velocity gain values of the alpha-beta filter limiting the effect of the transient on the filtered target track. Furthermore, for *10* particles

used in the SIR particle filter algorithm, the SIR particle filter has worse accuracy performance than was reported in Section 6.6.1.1.

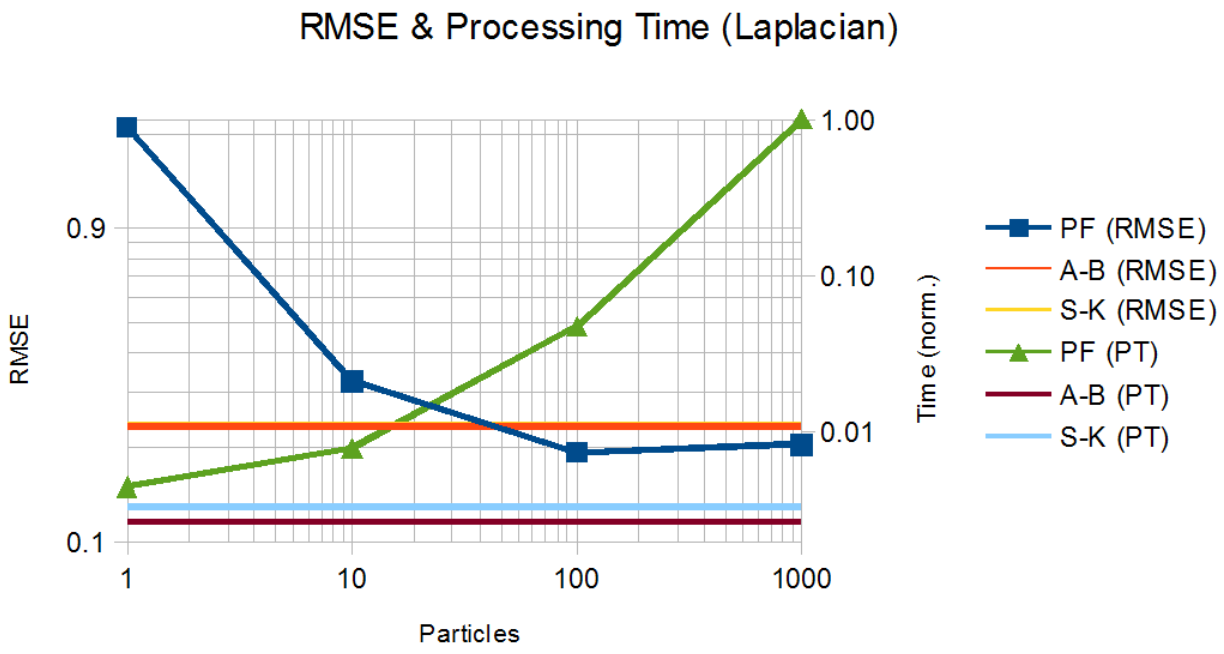

Figure 6.7: RMSE & Processing Time Comparison for Transient Detections in Laplacian Noise

From the results that had been obtained, the following four observations had been made, namely:

1. for the transient detection scenario in Gaussian noise:
    1. the accuracy performance of the SIR particle filter is generally better than for the ideal detection scenario;
    2. the accuracy performance of the SIR particle filter cannot be improved by increasing the number of particles used in the SIR particle filter algorithm above *10* particles;
2. for the transient detection scenario in Laplacian noise;
    1. the accuracy performance of the SIR particle filter is generally the same than for the ideal detection scenario, if the number of particles used in the SIR particle filter algorithm is above *100* particles;
    2. the accuracy performance of the SIR particle filter cannot be improved by increasing the number of particles used in the SIR particle filter algorithm above *100* particles.

6.6.1.4 Accuracy Performance Summary

The RMSE for the SIR particle filter, the alpha-beta filter and the Singer-Kalman filter, for an ideal detection, an average missed detection, and a transient detection scenario, for the simulations that had been performed is shown in *Table 6.3*. Although the SIR particle filter has better accuracy performance in all cases, the performance of the Singer-Kalman filter is very close to that of the SIR particle filter in the case of the average missed detection in the Gaussian noise scenario. This is due to the missed detections resulting in the SIR particle filter re-sampling process being performed less often. The SIR particle filter will then tend towards the SIS particle filter in behaviour, where no re-sampling is performed. As stated in Section 4.4, the SIS particle filter suffers from an unavoidable problem known as the degeneracy phenomenon.

Table 6.3: Root Mean Squared Error Summary

Filter Type	Noise Model	Ideal Detection	Missed Detections (Avg.)	Transient
Alpha-Beta	Gaussian	0.17077	0.36312	0.19129
	Laplacian	0.16901	0.39942	0.20833
Singer-Kalman	Gaussian	0.12104	0.31588	0.16677
	Laplacian	0.16151	0.34276	0.21131
SIR Particle NP = 1	Gaussian	1.88078	1.17837	1.54286
	Laplacian	1.57242	1.38101	1.89000
SIR Particle NP = 10	Gaussian	0.23392	0.46238	**0.14889**
	Laplacian	0.15247	0.39364	0.29299
SIR Particle NP = 100	Gaussian	0.11033	0.40900	0.15433
	Laplacian	0.13619	0.45492	**0.16334**
SIR Particle NP = 1000	Gaussian	**0.09666**	**0.31103**	0.16087
	Laplacian	**0.13223**	**0.30159**	0.18368
Confidence Interval: 98%				

6.6.2 Computational Complexity

The computational complexity of the SIR particle filter had been evaluated for an ideal detection, a missed detection, and a transient detection scenario. The metric used in evaluating the computational complexity of the SIR particle filter had been the normalized processing time in seconds, that had been discussed in Section 3.4.2. For the missed detections scenario, the average normalized processing time for a missed detection level of *1%* to *10%* of the total detections in

increments of *1%*, is presented. The processing time for each filter had been calculated on the actual processing time required to implement the filter algorithm only, and not on the computational load of the radar target tracking filter simulator. The processing time for each filter had been normalized on the longest processing time, namely that of the SIR particle filter with *1,000* particles used in the SIR particle filter algorithm. The normalized processing time for the SIR particle filter, the alpha-beta filter, and the Singer-Kalman filter, for an ideal detection, an average missed detection, and a transient detection scenario, for the simulations that had been performed is shown in *Table 6.4*.

Table 6.4: Normalized Processing Time Summary

Filter Type	**Noise Model**	**Ideal Detection**	**Missed Detections (Avg.)**	**Transient**
Alpha-Beta	Gaussian	0.00244	0.00272	0.00269
	Laplacian	0.00245	0.00275	0.00265
Singer-Kalman	Gaussian	0.00328	0.00365	0.00335
	Laplacian	0.00337	0.00367	0.00331
SIR Particle NP = 1	Gaussian	0.00446	0.00492	0.00450
	Laplacian	0.00458	0.00489	0.00449
SIR Particle NP = 10	Gaussian	0.00769	0.00837	0.00820
	Laplacian	0.00798	0.00846	0.00791
SIR Particle NP = 100	Gaussian	0.04612	0.04705	0.04811
	Laplacian	0.04907	0.04748	0.04771
SIR Particle NP = 1000	Gaussian	1.00000	1.00000	1.00000
	Laplacian	1.00000	1.00000	1.00000
Confidence Interval: 98%				

It can be observed that the higher the number of particles used in the SIR particle filter algorithm, the higher the normalized processing time of the SIR particle filter becomes. The lowest normalized processing time is that of the alpha-beta filter, followed by the Singer-Kalman filter, and then the SIR particle filter with *1* particle used in the SIR particle filter algorithm. These results had also been observed in *Figure 6.1* and *Figure 6.2* in Section 6.6.1.1, and *Figure 6.6* and *Figure 6.7* in Section 6.6.1.3. Although there are differences in the normalized processing time for each type of filter in the different noise and detection scenarios, the differences are not significant.

From the results that had been obtained, the following three observations had been made, namely:

1. the normalized processing time of the SIR particle filter is a function of the number of particles used in the SIR particle filter algorithm;
2. the normalized processing time of the SIR particle filter increases with an increase in the number of particles used in the SIR particle filter algorithm;
3. the normalized processing time of the SIR particle filter is higher than that of either the alpha-beta-filter or the Singer-Kalman filter, even for the case with *1* particle used in the SIR particle filter algorithm.

6.7 Particle Filter Analysis

6.7.1 (Parametric) Cramér-Rao Lower Bound

The parametric (or deterministic) Cramér-Rao lower bound, or more commonly referred to just as the Cramér-Rao lower bound (CRLB), is the lower bound of the mean squared estimation error for an estimate of a nonrandom parameter, which defines the absolute accuracy achievable of an unbiased estimation process [70]. The square root of the CRLB of an unbiased estimation process is the best achievable precision of the unbiased estimation process [49]. For an unbiased estimator, the CRLB can be expressed, given by [70], as follows:

$$E[(\hat{x}_i - x_i)^2] \geqslant F_{ii}^{-1}$$

with F_{ii} *the Fisher information matrix,*

$\hat{x}_i$ *the expected value, and*

x_i *the actual value.* (6.6)

The CRLB had been used as a metric in determining the estimation performance of particle filters in underwater navigation, [76] and [77], surface navigation [78], and missile tracking applications [79]. However, in these cases the theoretical CRLB could be derived since the model of the inertial navigation system, [76] and [78], or target dynamics [79] was know, or parametric data was available [77].

6.7.2 Posterior Cramér-Rao Lower Bound

The posterior (or Bayesian) Cramér-Rao lower bound (PCRLB) is the lower bound of the mean

squared estimation error for any estimate of a random parameter, which defines the absolute accuracy achievable of any estimation process [80]. The difference between the CRLB and the PCRLB is that the parameter being estimated is nonrandom in the case of the CRLB, and random in the case of the PCRLB. Furthermore, the estimation process in the case of the CRLB has to be unbiased, while in the case of the PCRLB, it can be either unbiased or biased. These differences makes the PCRLB more powerful that the CRLB as a lower bound of an estimator [80]. The PCRLB has been used as a metric in determining the estimation performance of particle filters in underwater navigation [81], and for nonlinear filtering applications, [82] and [83].

The theoretical derivation of the PCRLB is not trivial and some of the methods employed to derive the theoretical PCRLB is overly optimistic, as reported by [84]. Often there are no analytical solutions to the PCRLB as it involves the solving of complex, multi-dimensional integrals [80]. However, the PCRLB can be approximated through a numerical simulation-based approached where *a priori* information is available about a nonlinear estimation process [85]. If no *a priori* information about a nonlinear estimation process is available, then particle filters can even be used to approximate the PCRLB recursively for that nonlinear estimation process [85]. This approached has been proved to be sufficiently accurate to approximate the PCRLB for missile tracking applications [86]. The method used to numerically approximate the PCRLB, is known as Monte Carlo integration [80]. The process of Monte Carlo integration is given by [87] as follows:

$$\sqrt{\frac{1}{M}\sum_{i=1}^{M}(\hat{x}_t^i - x_t)^2} \geq \sqrt{tr\,P_t} \tag{6.7}$$

with M *the number of Monte Carlo integration iterations,*

t *the time in the Monte Carlo integration iteration* i*,*

$\hat{x}_t^i$ *the expected value,*

x_t *the actual value, and*

$tr\,P_t$ *the trace of the CRLB.*

The PCRLB is calculated by the radar target tracking filter simulator for a given simulation configuration using (6.7), and the PCRLB data is saved and a PCRLB plot can be generated as discussed in Section 5.9.2. The PCRLB versus time for the SIR particle filter, tracking the target as

described in Section 5.4.3 in Gaussian noise, and for a different number of particles (NP), is shown in *Figure 6.8*.

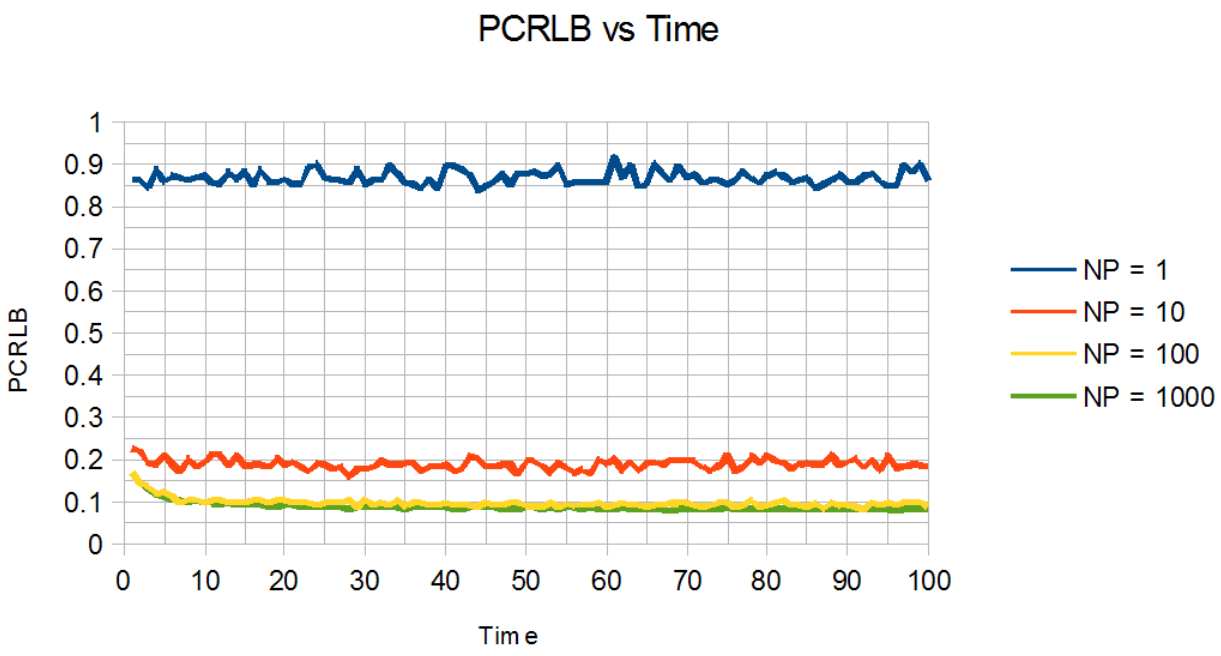

Figure 6.8: Posterior Cramér-Rao Lower Bound versus Time for Sequential Importance Re-sampling Particle Filter in Gaussian Noise

The PCRLB versus time for the SIR particle filter, tracking the target as described in Section 5.4.3 in Laplacian noise, and for a different number of particles (NP), is shown in *Figure 6.9*.

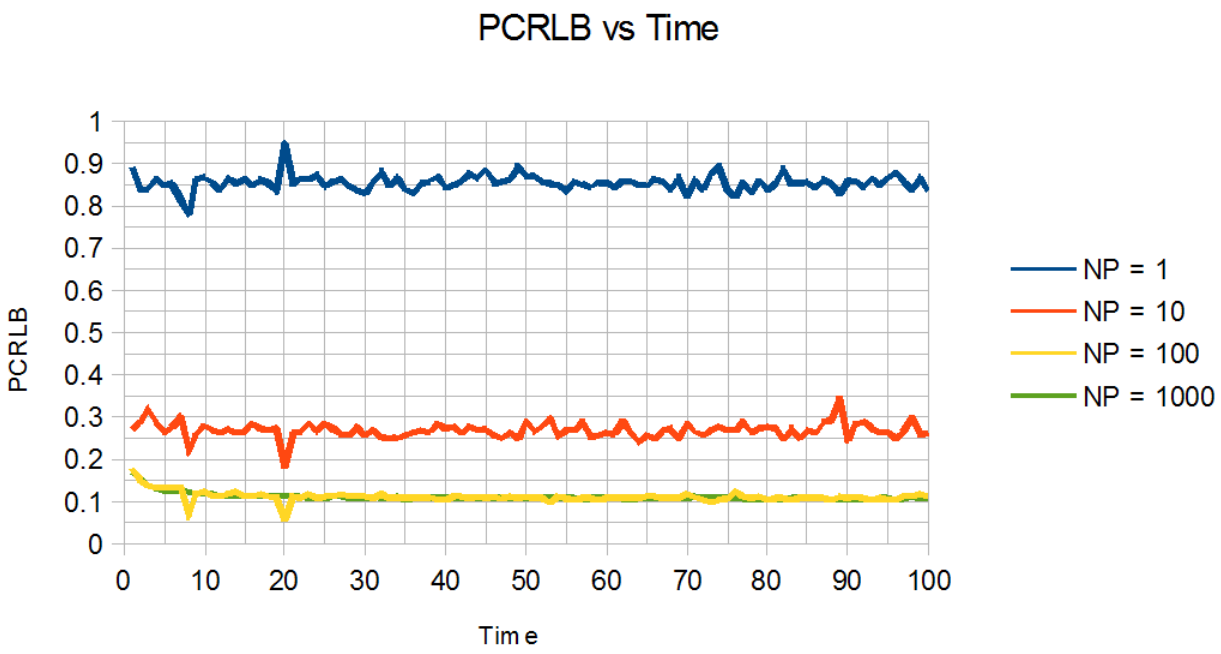

Figure 6.9: Posterior Cramér-Rao Lower Bound versus Time for Sequential Importance Re-sampling Particle Filter in Laplacian Noise

From these figures it can be observed that the PCRLB of the SIR particle filter is a function of the number of particles used in the SIR particle filter algorithm. As the number of particles used in the SIR particle filter increases, the calculated PCRLB decreases. The decrease in the PCRLB is not linear in relation to the increase in the number of particle used in the SIR particle filter algorithm. Thus, there is a optimal number of particles to use in the SIR particle filter algorithm where the increased normalized processing time of using a larger amount of particles, would not yield an equivalent increase in the performance of the SIR particle filter as a radar target tracking filter. The theoretical PCRLB will be reached when the number of particles used in the SIR particle filter algorithm equals infinity, *i.e.* $NP = \infty$.

6.8 Summary

In this chapter the performance of the SIR particle filter as a radar target tracking filter had been evaluated and analyzed through quantitative simulation. The SIR particle filter for different numbers of particles, had been compared to the alpha-beta filter and the Singer-Kalman filter. The accuracy performance of the filters had been analyzed with the RMSE metric, and the computational complexity of the filters had been analyzed with the normalized processing time metric. The error bounds of the SIR particle filter have also been analyzed through numerical

approximation of the PCRLB through Monte Carlo integration. In Chapter 7, the conclusions that can be drawn from the research conducted for the dissertation will be presented. Some recommendations of future work based on the research conducted for the dissertation will also be presented.

Chapter 7 : Conclusions and Recommendations

7.1 Introduction

An overview of the objectives of the research conducted for the dissertation, are presented in this chapter. The conclusions that had been drawn from the research conducted for the dissertation, are presented and discussed in this chapter. Finally, some recommendations of future work based on the research conducted for the dissertation is presented. These recommendations focuses mainly on the computational complexity, and the practical implementation of particle filters in radar systems. A number of extensions that can be made to the radar target tracking filter simulator are also suggested.

7.2 Research Objectives

The accomplishment of the research objectives as had been defined in Section 1.4, are presented as follows:

1. a generic simulation framework consisting of a target model, a noise model, a target tracking filter model, and a display and data capture module for evaluating radar target tracking filters, had been developed;
2. the evaluation of a particle filter algorithm, namely the sequential importance re-sampling (SIR) particle filter, for use as a radar target tracking filter had been performed through simulation and analysis thereof, according to the predefined metrics, namely the root mean squared error (RMSE) and the normalized processing time;
3. the limitations in the field of research had been identified from the literature review, and the field of research had been extend by addressing these research limitations as follows:
    1. the quality of the external model of the target used for evaluating the performance of the radar target tracking filter had been verified from literature;
    2. the noise model used for evaluating the performance of the radar target tracking filter had been clearly defined;
    3. the noise model used for evaluating the performance of the radar target tracking filter had been extended to include a non-Gaussian (Laplacian) noise model;

4. the number of simulation iterations used for evaluating the performance of the radar target tracking filter, and which had a direct impact on determining the accuracy of the radar target tracking filter, had been determined through a theoretically defensible approach to a specified level of confidence;
5. the performance of the SIR particle filter, the Singer-Kalman filter, and the alpha-beta filter as radar target tracking filters under simulated real-world detection scenarios, such as missed target detections and transient detections, had been evaluated;
6. the computational complexity of the SIR particle filter, the Singer-Kalman filter, and the alpha-beta filter as radar target tracking filters had been evaluated.

7.3 Conclusions

A number of conclusions can now be made based on the research conducted for the dissertation, namely:

1. the accuracy performance of the SIR particle filter is a function of the number of particles used in the SIR particle filter algorithm, with an increase in the number of particles used in the SIR particle filter algorithm leading to an increase in accuracy performance, except the increase in accuracy performance is not as large, with a confidence level of *98%*, in the case of the missed detections in the Laplacian noise scenario;
2. the SIR particle filter has better accuracy performance, with a confidence level of *98%*, in Laplacian noise than in Gaussian noise, and better accuracy performance than both the alpha-beta filter and the Singer-Kalman filter in both noise cases;
3. the normalized processing time of the SIR particle filter is a function of the number of particles used in the SIR particle filter algorithm, with an increase in the number of particles used in the SIR particle filter algorithm leading to an increase in normalized processing time;
4. the normalized processing time of the SIR particle filter is higher than that of either the alpha-beta filter or the Singer-Kalman filter, but with improved accuracy performance;
5. the posterior Cramér-Rao lower bound (PCRLB) of the SIR particle filter is a function of the number of particles used in the particle filter, with an increase in the number of particles used in the SIR particle filter algorithm leading to a decrease in the PCRLB.

7.4 Recommendations

A number of recommendations are now made with regard future work that can be done, based on the research that had been conducted for the dissertation.

7.4.1 Computational Complexity

The high computational complexity of particle filters remains a limiting factor in their practical use, due to the necessary computational requirements for solving real-time estimation problems [88]. The normalized processing time of the particle filter is proportional to the number of particles used in the particle filter algorithm. One effective method to dynamically control the normalized processing time of the particle filter is through the reduction of the number of active particles, as is done in the generic particle filter (GPF) [89]. However, this would lead to the problem of particle path degeneration, and sample impoverishment due to lack of particle diversity, as discussed in Section 4.5. The variable rate particle filter (VRPF) discussed in Section 4.7.5, makes use of variable rate re-sampling based on a specific dynamic model, to mitigate the effects of computational inefficiency in the particle filter. However, a dynamic model of the process being modeled needs to exist in order for the filter to be useful, as the re-sampling rate needs to be adjusted based on the dynamic model.

An alternative approach to research for the VRPF algorithm, would be the implementation of an adaptive re-sampling rate that would adjust the re-sampling rate based on the past behaviour of the process being modeled, instead of a dynamic model. The normalized processing time of this approach could be lower than that of the SIR particle filter, but its accuracy could also potentially be lower. These factors have to be determined and evaluated through quantitative simulations, as had been described in Chapter 6. Related to the VRPF algorithm, a number of techniques based on partial re-sampling has also been suggested to reduce the number of operations and amount of memory as required by SIR particle filters [90].

Efficient algorithms for the distributed implementation of particle filters have been studied as reported by [91]. Importantly, the exact posterior Cramér-Rao lower bound (PCRLB) for these efficient algorithms have been produced, and can thus be used in further research into the efficient distributed implementation of particle filters.

7.4.2 Practical Implementation

The practical implementation of particle filters for various applications have been studied, most

notably in the fields of econometrics, computer vision, and robotics [88]. As previously stated in Section 7.4.1, particle filters do have a high normalized processing time that is proportional to the number of particles used in the particle filter algorithm. However, the practical implementation of particle filters to reduce their normalized processing time is an active field of research.

A proposed solution involves the use of a many-core computing architecture, like a graphics processing unit (GPU) or general purpose GPU (GPGPU), and a distributed particle filter algorithm [88]. The research found that the type of estimation problem and the exchange scheme used in the distributed particle filter algorithm, determined both the computational performance gain and accuracy achieved with the distributed particle filter algorithm. The view of many-core computing in the practical implementation of particle filters is further reinforced by [89], where the focus is on the parallelization of existing particle filter algorithms on GPUs. It was found that for applications where the number of particles exceeded 10,000, the GPU implementation had faster execution time. However, in radar applications where the use of GPUs might not always be feasible due to size and power constraints, it is however suggested by [88] that embedded many-core architectures would be a feasible method to achieve performance portability.

The implementation of particle filters for radar target track filtering applications using a field-programmable gate array (FPGA), had been investigated [92]. It has been found that the numerical precision that could be achieved with the FPGA, limited the performance of the particle filter with respect to precision. Some alternative implementations of the SIR particle filter had been suggested and evaluated that showed some measure of performance improvement, but these implementations still required further refinement and more evaluation.

7.4.3 Radar Target Tracking Filter Simulator

The functionality of the radar target tracking filter simulator that had been produce for the research conducted for this dissertation, can be extended as follows:

1. other noise models can be implemented to evaluate the performance of particle filters in tracking targets in different environments, for example in sea or ground clutter [93];
2. the simulated target tracking environment can be extended from a two-dimensional to a three-dimensional Cartesian coordinate system, to evaluate three-dimensional tracking radar applications;
3. the efficiency of the radar target tracking filter simulator could be improved through the

application of software refactoring techniques [94];

4. the simulated target tracking environment can be extended for multiple target tracking scenarios.

References

[1] M.I. Skolnik, *Radar Handbook*, 3rd ed., New York: McGraw-Hill, 2008.

[2] W.D. Blair, "Radar Tracking Algorithms", in *Principles of Modern Radar, Volume I: Basic Principles*, M.A. Richards, J.A. Scheer & W.A. Holm eds., New Jersey: Scitech Publishing, 2010, pp. 713 – 772.

[3] K. Akcay, *Performance Metrics for Fundamental Estimation Filters*, Master's Thesis, The Graduate School of Natural and Applied Sciences, Middle East Technical University, Turkey, 2005.

[4] J.A. Lawton, R.J. Jesionowski, & P. Zarchan, "Comparison of Four Filtering Options for a Radar Tracking Problem" in *Journal of Guidance, Control, and Dynamics*, 21(4): 618 – 623, 1998.

[5] G. Soysal, & M. Efe, "Performance Comparison of Tracking Algorithms for a Ground Based Radar" in *Communications, Series A2 – A3: Physics, Engineering Physic, Electronics/ Computer Engineering, Astronomy and Geophysics*. 51(1): 1 – 16, 2007.

[6] N. Morrison, *Tracking Filter Engineering: The Gauss–Newton and Polynomial Filters*, Stevenage, United Kingdom: The Institution of Engineering and Technology, 2013.

[7] R. Nadjiasngar, M. Inggs, Y. Paichard & N. Morrison, "A New Probabilistic Data Association Filter Based on Composite Expanding and Fading Memory Polynomial Filters" in *Proceedings of the 2011 IEEE Radar Conference (Radar 2012)*, 23 – 27 May 2011, pp. 152 – 156.

[8] R. Nadjiasngar, S. Middleton & M. Inggs, "Doppler-Only Tracking with the Recursive Gauss-Newton-Filter" in *Proceedings of the IET International Conference on Radar Systems (Radar 2012)*, 22 – 25 October 2012, pp. 1 – 5.

[9] C.D. Karlgaard & H. Schaub, "Comparison Of Several Nonlinear Filters for a Benchmark Tracking Problem" in *Proceedings of the AIAA Guidance, Navigation and Control Conference*, 21 – 24 August 2006, pp. 1 – 17.

[10] Z. Tang, C. Sun & Z. Liu, "The Tracking Algorithm for Maneuvering Target Based on Adaptive Kalman Filter" in *The International Arab Journal of Information Technology*,

10(5): 453 – 459, 2013.

[11] M. Vinaykumar & R. Jatoth, "Performance Evaluation of Alpha-Beta and Kalman Filter for Object Tracking" in *Proceedings of the IEEE International Conference on Advanced Communication Control and Computing Technologies*, 8 – 10 May 2014, pp. 1369 – 1373.

[12] N. Morrison, R.T. Lord & M.R. Inggs, "The Gauss-Newton Algorithm Applied to Track-While-Scan Radar" in *Proceedings of the 2007 IET International Conference on Radar Systems*, 15 – 18 October 2007, pp. 1 – 5.

[13] N. Morrison, R.T. Lord & M.R. Inggs, "The Gauss-Newton Algorithm in Passive Aircraft Tracking using Doppler and Bearings" in *Proceedings of the 2007 IET International Conference on Radar Systems*, 15 – 18 October 2007, pp. 1 – 5.

[14] F. Carravetta, A. Germani & M. Raimondi, "Polynomial Filtering of Discrete-Time Stochastic Linear Systems with Multiplicative State Noise" in *IEEE Transactions on Automatic Control*, 42(8): 1106 – 1126, 1997.

[15] M.S. Arulampalam, S. Maskell, N. Gordon & T. Clapp, "A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking" in *IEEE Transactions on Signal Processing*, 50(2): 174 – 188, 2002.

[16] P. Losie, *Detection and Tracking of Stealthy Targets Using Particle Filters*, Master's Thesis, Electrical Engineering Department, California Polytechnic State University, USA, 2009.

[17] N. Hagarajan, *Target Tracking Via Marine Radar*, Master's Thesis, Graduate Faculty, The University of Toledo, USA, 2012.

[18] E.-S. Abdoul-Moaty, T.R. Abdoul-Shahid, A.E.-D.S. Hafez & M. Abd-El-Latif, "A Particle Filter for Multistatic Radar Tracking" in *Proceedings of the 2014 IEEE Aerospace Conference*, 1 – 8 March 2014, pp. 1 – 5.

[19] W. Liu, X. Chen & X. Chu, "Belief rule-based methodology and Particle filter for radar target tracking" in *Proceedings of the 2014 International Conference on Informative and Cybernetics for Computational Social Systems (ICCSS)*, 9 – 10 October 2014, pp. 85 – 90.

[20] A.K. Tilton, S. Ghiotto & P.G. Mehta, "A Comparative Study of Nonlinear Filtering Techniques" in *Proceedings of the 16th International Conference on Information Fusion (FUSION)*, 9 – 12 July 2013, pp. 1827 – 1834.

[21] K. Zhan, L. Xu, H. Jiang, L. Bai & M. Wu, "Particle filter based joint tracking and classification" in *Proceedings of the 2014 IEEE Chinese Guidance, Navigation and Control Conference (CGNCC)*, 8 – 10 August 2014, pp. 762 – 766.

[22] M. Zhang & W. Chen, "Variable Structure Multiple Model Particle Filter for Maneuvering Radar Target Tracking" in *Proceedings of the 2010 International Conference on Microwave and Millimeter Wave Technology (ICMMT)*, 8 – 11 May 2010, pp. 1754 – 1757.

[23] J. Zhang, H. Ji & Q. Xue, "New Approach Based on Particle Filter for Target Tracking with Glint Noise" in *Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics*, 11 – 14 October 2009, pp. 4791 – 4795.

[24] J.J. Yin, J.Q. Zhang & Y. Gao, "The Polynomial Predictive Particle Filter" in *Proceedings of the 2nd International Conference on Advanced Computer Control: Volume 4 (ICACC)*, 27 – 29 March 2010, pp. 527 – 531.

[25] N. Oudjane & C. Musso, "Progressive Correction for Regularized Particle Filters" in *Proceedings of the Third International Conference on Information Fusion: Volume 2 (FUSION 2000)*, 10 – 13 July 2000, THB2/10 – THB2/17.

[26] M. Tobias & A.D. Lanterman, "Techniques for birth-particle placement in the probability hypothesis density particle filter applied to passive radar" in *IET Radar, Sonar & Navigation*, 2(5): pp. 351 – 365, 2008.

[27] J. Wang, D. Dai, H. Dong, T. Quan, & Y. Jin, "Improved Evolutionary Particle Filter Algorithm Applied in Radar Tracking" in *Proceedings of the 2006 International Conference on Radar (CIE '06)*, 16 – 19 October 2006, pp. 1 – 4.

[28] H.-Y. Zhao, A.-H. Cai & S.-S. Zhang, "Research on Ship Tracking Based on Adaptive Particle Filter" in *Proceedings of the 2013 International Workshop on Microwave and Millimeter Wave Circuits and System Technology (MMWCST)*, 24 – 25 October 2013, pp. 233 – 236.

[29] M. Ishibashi, Y. Iwashita & R. Kurazume, "Noise-Estimate Particle PHD filter" in *Proceedings of the 2014 World Automation Congress (WAC)*, 3 – 7 August 2014, pp. 784 – 789.

[30] A. Kazem & G. Salut, "Track-while-scan Radar using deterministic particle filtering" in *Proceedings of the 3rd International Conference on Information and Communication*

Technologies – From Theory to Applications (ICTTA 2008), 7 – 11 April 2008, pp. 1 – 6.

[31] S. Godsill & J. Vermaak, "Variable rate particle filters for tracking applications" in *Proceedings of the 2005 IEEE/SP 13th Workshop on Statistical Signal Processing*, 17 – 20 July 2005, pp. 1280 – 1285.

[32] S.J. Godsill, J. Vermaak, W. Ng & J.F. Li, "Models and Algorithms for Tracking of Maneuvering Objects Using Variable Rate Particle Filters" in *Proceedings of the IEEE*, 95(5): 925 – 952, 2007.

[33] Y. Guo, D.L. Peng, H. Chen & A. Xue, "A Kernel Particle Filter Algorithm for Joint Tracking and Classification" in *Proceedings of the 15th International Conference on Information Fusion (FUSION)*, 9 – 12 July 2012, pp. 2386 – 2391.

[34] W. Ng, J. Li, S.K. Pang & S. Godsill, "On Tracking Applications using Variable Rate Particle Filters" in *Proceedings of the 2006 IEEE Nonlinear Statistical Signal Processing Workshop*, 13 – 15 September 2006, pp. 117 – 120.

[35] W. Ng, S.K. Pang, J. Li & S. Godsill, "Efficient variable rate particle filters for tracking manoeuvring targets using an MRF-based motion model" in *Proceedings of the 14th European Signal Processing Conference*, 4 – 8 September 2006, pp. 1 – 5.

[36] Y. Ulker, B. Gunsel & S. Kirbiz, "A Multiple Model Structure for Tracking by Variable Rate Particle Filters" in *Proceedings of the 19th International Conference on Pattern Recognition (ICPR 2008)*, 8 – 11 December 2008, pp. 1 – 4.

[37] R. van der Merwe, A. Doucet, N. de Freitas & E. Wan, "The Unscented Particle Filter" in *Advances in Neural Information Processing Systems 13*, T.K. Leen, T. Dietterich & V. Tresp eds., Cambridge, United States: MIT Press, 2001, pp. 584 – 590.

[38] V. Cevher, R. Velmurugan & J.H. McClellan, "A Range-Only Multiple Target Particle Filter Tracker" in *Proceedings of the 2006 IEEE International Conference on Acoustics, Speech and Signal Processing: Volume 4 (ICASSP 2006)*, 14 – 19 May 2006, pp. 4-905 – 4-908.

[39] S. Maskell, M. Briers, R. Wright & P. Horridge, "Tracking using a radar and a problem specific proposal distribution in a particle filter" in *IEE Proceedings Radar, Sonar and Navigation*, 152(2): 315 – 322, 2005.

[40] B. Sobhani, E. Paolini, M. Mazzotti, A. Giorgetti & M. Chiani "Multiple Target Tracking

with Particle Filtering in UWB Radar Sensor Networks" in *Proceedings of the 2015 International Conference on Localization and GNSS (ICL-GNSS)*, 22 – 24 June 2015, pp. 1 – 6.

[41] H. Li & J. Wang, "Particle filter for manoeuvring target tracking via passive radar measurements with glint noise" in *IET Radar, Sonar & Navigation*, 6(3): 180 – 189, 2011.

[42] H. Li, X. Ji & G. Zhao "TOA-based Target Tracking Using Improved Particle Filter in Passive Bistatic Radar With Glint Noise" in *Proceedings of the 6th International Congress on Image and Signal Processing: Volume 3 (CISP)*, 16 – 18 December 2013, pp. 1184 – 1188.

[43] H. Li, J. Wang & Y. Liu "Passive Coherent Radar Tracking Algorithm Based on Particle Filter and Multiple TDOA Measurements" in *Proceedings of the 2nd International Congress on Image and Signal Processing (CISP '09)*, 17 – 19 October 2009, pp. 1 – 4.

[44] A. Benavoli & A. Di Lallo, "Why should we use particle filtering in FM band passive radars?" in *Proceedings of the 5th European Radar Conference (EuRAD 2008)*, 30 – 31 October 2008, pp. 344 – 347.

[45] S. Herman & P. Moulin, "A Particle Filtering Approach to FM-Band Passive Radar Tracking and Automatic Target Recognition" in *Proceedings of the 2002 IEEE Aerospace Conference: Volume 4*, 9 – 16 March 2002, pp. 4-1789 – 4-1808.

[46] P.H. Foo, "Combining the interacting multiple model method with particle filters for manoeuvring target tracking with a multistatic radar system" in *IET Radar, Sonar & Navigation*, 5(7): 697 – 706, 2011.

[47] G. Soysal & M. Efe, "Data Fusion in a Multistatic Radar Network Using Covariance Intersection and Particle Filtering" in *Proceedings of the 14th International Conference on Information Fusion (FUSION)*, 5 – 8 July 2011, pp. 1 – 7.

[48] S. Chen & W. Huang, "Target Tracking Using Particle Filter with X-Band Nautical Radar" in *Proceedings of the 2013 IEEE Radar Conference (RADAR)*, 29 April – 3 May 2013, pp. 1 – 6.

[49] W.D. Blair, M.A. Richards & D.G. Long, "Radar Measurements", in *Principles of Modern Radar, Volume I: Basic Principles*, M.A. Richards, J.A. Scheer & W.A. Holm eds., New Jersey: Scitech Publishing, 2010, pp. 677 – 712.

[50] Institute of Electrical and Electronics Engineers, *IEEE Standard Radar Definitions*, IEEE Std 686™-2008, New York: IEEE, 2008.

[51] W.D. Blair, *Fixed-Gain Two-Stage Estimators for Tracking Maneuvering Targets*, (Weapons Systems Department report no. NSWCDD/TR-92/297), Dahlgren, USA: Naval Surface Warfare Center, 1992.

[52] R.E. Kalman, "A New Approach to Linear Filtering and Prediction Problems" in *Journal of Basic Engineering*, 82(1): 35 – 45, 1960.

[53] E.A. Wan & R. van der Merwe, "The Unscented Kalman Filter" in *Kalman Filtering and Neural Networks*, S. Haykin ed., New York: John Wiley & Sons, Inc., pp. 221 – 280, 2001.

[54] F.D-V. Maasdorp, *Doppler-only Target Tracking for a Multistatic Radar Exploiting FM Band Illuminators of Opportunity*, Doctoral Thesis, Department of Electrical Engineering, University of Cape Town, South Africa, 2015.

[55] R. Nadjiasngar, & M. Inggs, *The Recursive Gauss-Newton Filter*, 2011, Available: http://arxiv.org/pdf/1110.5212v1 [2016, February 15].

[56] D.M.A. Hussain, D. Hicks, D. Ortiz-Arroyo, S.A. Haq & Z. Ahmed, "A Case Study: Kalman & Alpha-Beta Computation under High Correlation" in *Proceedings of the International Multi-Conference of Engineers and Computer Scientists*, 19 – 21 March 2008, pp. 1 – 6.

[57] J. Carpenter, P. Clifford & P. Fearnhead, "An Improved Particle Filter for Non-linear Problems", in *IEE Proceedings – Radar, Sonar and Navigation*, 146(1): 2 – 7, 1999.

[58] N.J. Gordon, D.J. Salmond & A.F.M. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation" in *IEE Proceedings F – Radar and Signal Processing*, 140(2): 107 – 133, 1993.

[59] P. Biler, T. Funaki & W.A. Woyczynski, "Interacting Particle Approximation for Nonlocal Quadratic Evolution Problems" in *Probability and Mathematical Statistics*, 19(2): 267 – 286, 1999.

[60] I. Senji & Z. Kalafatić, "Tracking Objects Using Particle Filters" in *Proceedings of the 49th International Symposium ELMAR-2007 (Electronics in Marine)*, 12 – 14 September 2007, pp. 31 – 34.

[61] S. Dubuisson, *Tracking with Particle Filter for High-dimensional Observation and State Space*, New Jersey: Wiley, 2015.

[62] S.M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory*, New Jersey: Prentice Hall, 1993.

[63] C.M. Grinstead & J.L. Snell, *Introduction to Probability*, 2nd ed., Providence, Rhode Island: American Mathematical Society, 1997.

[64] I.M. Sobol, *A Primer for the Monte Carlo Method*, Boca Raton: CRC Press, 1994.

[65] P. Bunch & S. Godsill, "Particle Filtering with Progressive Gaussian Approximations to the Optimal Importance Density" in *Proceedings of the IEEE 5th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, 15 – 18 December 2013, pp. 360 – 363.

[66] S. Godsill & T. Clapp, "Improvement Strategies for Monte Carlo Particle Filters", in *Sequential Monte Carlo Methods in Practice*, A. Doucet, N. De Freitas & N. Gordon eds., New York: Springer-Verlag, 2001, pp. 139 – 158.

[67] F. Lindsten, P. Bunch, S.J. Godsill & T.B. Schön, "Rao-Blackwellized particle smoothers for mixed linear/nonlinear state-space models" in *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 26 – 31 May 2013, pp. 6288 – 6292.

[68] T. Schön, F. Gustafson & P.-J. Nordlund, "Marginalized Particle Filters for Mixed Linear/Nonlinear State-Space Models" in *IEEE Transactions on Signal Processing*, 53(7): 2279 – 2289, 2005.

[69] P. Reyneke, N. Morrison, D. Kourie & D. de Ridder, "Smoothing Irregular data using Polynomial Filters" in *Proceedings of the 52nd International Symposium ELMAR-2010 (Electronics in Marine)*, 15 – 17 September 2010, pp. 393 – 397.

[70] D.H. Johnson, *Statistical Signal Processing*, Houston: Rice University, 2013.

[71] M.R. Driels & Y.S. Shin, *Determining the Number of Iterations for Monte Carlo Simulations of Weapon Effectiveness*, (Department of Mechanical & Astronautical Engineering report no. NPS-MAE-04-005), Monterey, USA: Naval Postgraduate School, 2004.

[72] D.B. Thomas & W. Luk, “Estimation of Sample Mean and Variance for Monte-Carlo Simulations”, in *Proceedings of the 2008 IEEE International Conference on Electrical and Computer Engineering (ICECE) Field-Programmable Technology (FPT 2008)*, 8 – 10 December 2008, pp. 89 – 96.

[73] M.D. Byrne, “How Many Times Should a Stochastic Model be Run? An Approach Based on Confidence Intervals”, in *Proceedings of the 12th International Conference on Cognitive Modelling (ICCM 2013)*, 11 – 14 July 2013, pp. 445 – 450.

[74] R. A. Singer, “Estimating Optimal Tracking Filter Performance for Manned Maneuvering Targets” in *IEEE Transactions on Aerospace and Electronic Systems*, AES-6(4): 473 – 483, 1970.

[75] B.R. Mahafza, *Radar Systems Analysis and Design Using MATLAB*, Boca Raton: Chapman & Hall/CRC, 2000.

[76] R. Karlsson, F. Gustafsson & T. Karlsson, “Particle Filtering and Cramér-Rao Lower Bound for Underwater Navigation”, in *Proceedings of the 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '03): Volume 6*, 6 – 10 April 2003, pp. VI-65 – VI-68.

[77] R. Karlsson & F. Gustafsson, “Particle Filter for Underwater Terrain Navigation”, in *Proceedings of the 2003 IEEE Workshop on Statistical Signal Processing*, 28 September – 1 October 2003, pp. 526 – 529, 2003.

[78] R. Karlsson & F. Gustafsson, “Bayesian Surface and Underwater Navigation” in *IEEE Transactions on Signal Processing*, 54(11): 4204 – 4213, 2006.

[79] A. Farina, B. Ristic & D. Benvenuti, “Tracking a Ballistic Target: Comparison of Several Nonlinear Filters” in *IEEE Transactions on Aerospace and Electronic Systems*, 38(3): 854 – 867, 2002.

[80] E. Saritaş, *Parametric and Posterior Cramér-Rao Low Bounds for Extended Target Tracking in a Random Matrix Framework*, Master's Thesis, The Graduate School of Natural and Applied Sciences, Middle East Technical University, Turkey, 2015.

[81] Z. Zhao, H. Chen, G. Chen, C. Kwan & X.R. Li, “Comparison of Several Ballistic Target Tracking Filters” in *Proceedings of the IEEE 2006 American Control Conference*, 14 – 16 June 2006, pp. 2197 – 2202.

[82] P. Tichavský, C.H. Muravchik & A. Nehorai, “Posterior Cramér-Rao Bounds for Discrete-Time Nonlinear Filtering” in *IEEE Transactions on Signal Processing*, 46(5): 1386 – 1396, 1998.

[83] M. Orton & A. Marrs, “Particle Filters for Tracking with Out-of-Sequence Measurements” in *IEEE Transactions on Aerospace and Electronic Systems*, 41(2): 693 – 702, 2005.

[84] M. Hernandez, B. Ristic & A. Farina, “A Comparison of Two Cramér-Rao Bounds for Nonlinear Filtering with $P_d < 1$” in *IEEE Transactions on Signal Processing*, 52(9): 2361 – 2370, 2004.

[85] A. Tulsyan, B. Huang, R.B. Gopaluni & J.F. Forbes, “A particle filter approach to approximate posterior Cramér-Rao lower bound: The case of hidden states” in *IEEE Transactions on Aerospace and Electronic Systems*, 49(4): 2478 – 2495, 2013.

[86] M. Lei, B.J. van Wyk & Y. Qi, “Online Estimation of the Approximate Posterior Cramer-Rao Lower Bound for Discrete-Time Nonlinear Filtering” in *IEEE Transactions on Aerospace and Electronic Systems*, 47(1): 37 – 57, 2011.

[87] N. Bergman, *Recursive Bayesian Estimation: Navigation and Tracking Applications*, Doctoral Thesis, Department of Electrical Engineering, Linköpings University, Sweden, 1999.

[88] M. Chitchian, A.S. van Amesfoort, A. Simonetto, T. Keviczky & H.J. Sips, “Adapting Particle Filter Algorithms to Many-Core Architectures” in *Proceedings of the 2013 IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS)*, 20 – 24 May 2013, pp. 427 – 438.

[89] J. Gebart, *GPU Implementation of the Particle Filter*, Master's Thesis, Department of Electrical Engineering, Linköpings University, Sweden, 2013.

[90] M. Bolić, P.M. Djurić & S. Hong, “Resampling Algorithms for Particle Filters: A Computational Complexity Perspective”, in *EURASIP Journal on Applied Signal Processing*, 2004(15): 2267 – 2277, 2004.

[91] A. Mohammadi, *Distributed Implementations of the Particle Filter with Performance Bounds*, Doctoral Thesis, Faculty of Graduate Studies, York University, Canada, 2013.

[92] B. Ye & Y. Zhang, “Improved FPGA Implementation of Particle Filter for Radar Tracking

Applications" in *Proceedings of the 2009 2nd Asian-Pacific Conference on Synthetic Aperture Radar (APSAR)*, 26 – 30 October 2009, pp. 943 – 948.

[93] N.C. Currie, "Characteristics of Clutter", in *Principles of Modern Radar, Volume I: Basic Principles*, M.A. Richards, J.A. Scheer & W.A. Holm eds., New Jersey: Scitech Publishing, 2010, pp. 165 – 210.

[94] T. Mens & T. Tourwé, "A survey of software refactoring", in *IEEE Transactions on Software Engineering*, 30(2): 126 – 139, 2004.

Appendix A : Scilab “filter.sce” Source Code

```
ieee(1); //Floating point exception mode
funcprot(0); //Function protection mode
stacksize('max'); //Maximum stack size
xdel(winsid()); //Close all open figures
clearglobal(); //Kills all variables
clc(); //Clear command window

VERSION = 1;
REVISION = 0;
DATE = "2016-08-03"

FALSE = -1;
TRUE = 1;

//DEBUG = TRUE;
DEBUG = FALSE;
//DEBUG_NOISE = TRUE;
DEBUG_NOISE = FALSE;
//DEBUG_SPIRAL_DIVE = TRUE;
DEBUG_SPIRAL_DIVE = FALSE;

PARTICLE = 1;
PARTICLES_MIN = 1;
PARTICLES_MAX = 100000;
PARTICLES_STEPSIZE = 1;
ALPHA_BETA = 2;
KALMAN = 3;

NONE = 1;
GAUSSIAN = 2;
LAPLACIAN = 3;
```

```
SINGLE_SHOT = 1;
BATCH = 2;
BATCH_MIN = 10;
BATCH_MAX = 100000;
BATCH_STEPSIZE = 10;

PLOT_2D_HORZ = 1;
PLOT_2D_VERT = 2;
PLOT_3D = 3;

SPIRAL_DIVE = 1;
RANDOM = 2;
SINUSOIDAL = 3;
LINEAR = 4;

SPIRAL_AMPLITUDE = 0.20; //amplitude of aircraft spiral
SPIRAL_RATE = 20; //rate of aircraft spiral

global FILTER_TYPE;
FILTER_TYPE = PARTICLE;
global PARTICLES_PF; //Particle Filter: number of particles in particle filter
PARTICLES_PF = 10;
global ALPHA_ABF; //Alpha-Beta Filter: alpha parameter (position gain)
ALPHA_ABF = 0.1
global BETA_ABF; //Alpha-Beta Filter: beta parameter (velocity gain)
BETA_ABF = 0.1

global NOISE_MODEL;
NOISE_MODEL = NONE;

global SIMULATION_TYPE;
SIMULATION_TYPE = SINGLE_SHOT;
```

```
global BATCH_SIZE;
BATCH_SIZE = BATCH_MIN;
global DISPLAY_LOCK;
DISPLAY_LOCK = TRUE;

global TRUE_TRACK;
TRUE_TRACK = TRUE;
global DETECTIONS;
DETECTIONS = FALSE;
global FILTERED_TRACK;
FILTERED_TRACK = FALSE;

global ERROR_PLOT;
ERROR_PLOT = FALSE;
global PCRLB_PLOT;
PCRLB_PLOT = FALSE;

global PLOT_TYPE;
PLOT_TYPE = PLOT_2D_HORZ;

global TARGET_MODEL;
TARGET_MODEL = SPIRAL_DIVE;

global DETECT_MISSED;
DETECT_MISSED = FALSE;
global DETECT_MISSED_SIZE;
DETECT_MISSED_SIZE = 0;
global DETECT_TRANSIENT;
DETECT_TRANSIENT = FALSE;
global DETECT_TRANSIENT_SIZE;
DETECT_TRANSIENT_SIZE = 0;

////////////////////////////////////////////////
//Simulation parameters
```

```
////////////////////////////////////////////////
m = 100; //simulation length
n = 100; //number of radar detections in the simulation (number of target detections)

di_abf = 1; //Alpha-Beta Filter: detection interval

noise_scale = 0.4; //scaling factor for noise
Gaussian_noise_std_dev = 0.1; //standard deviation of the Gaussian noise
Laplacian_noise_variance = 0.1; //standard deviation of Laplacian noise

global t;
global x_true;
global x_detections;
global x_filtered;
global xd_errors;
global xf_errors;
global rmse_fil;
global rmse_det;
global time;
global filename_results;
global filename_data;
global batch_rmse_filtered;
global batch_rmse_detections;
global batch_simulation_time;
global pcrlb;
////////////////////////////////////////////////

////////////////////////////////////////////////
//Main window
////////////////////////////////////////////////
function uicontrol_filter()
   filter_ui = figure(...
      "tag",            "filter_ui",...
      "dockable",       "off",...
```

```
        "infobar_visible",  "off",...
        "toolbar_visible",  "off",...
        "toolbar",          "none",...
        "menubar_visible",  "on",...
        "menubar",          "none",...
        "default_axes",     "off",...
        "layout",           "border",...
        "visible",          "off"...
    );

    filter_ui.figure_id         = 100001;
    filter_ui.background        = -2;
    filter_ui.figure_position   = [0 0];
    filter_ui.figure_name       = "Radar Target Tracking Filter Simulator V"
            + string(VERSION) + "." + string(REVISION);
    filter_ui.axes_size         = [1120 840]; // 4:3 aspect ratio

////////////////////////////////////////////////////
//Drop-down menu items
////////////////////////////////////////////////////
    h1 = uimenu(filter_ui, "label", gettext("File"));
    uimenu(h1, "label", gettext("Close"), "callback", "filter_ui=
            get_figure_handle(100001); delete(filter_ui);", "tag", "close_menu");
    h2 = uimenu(filter_ui, "label", gettext("About"));
            uimenu(h2, "label",gettext("About"), "callback","filter_about();");
////////////////////////////////////////////////////

////////////////////////////////////////////////////
//frame_left parameters
////////////////////////////////////////////////////
    frame_left = uicontrol(filter_ui,...
        "tag", "frame_left",...
        "style", "frame",...
        "constraints", createConstraints("border", "left", [250, 0]),...
```

```
        "border", createBorder(...
            "titled",...
            createBorder("line", "lightGray", 1),...
            _("Control Panel"),...
            "center",...
            "below_top",...
            createBorderFont("", 16, "normal"),...
            "black"...
        ),...
        "backgroundcolor", [1 1 1],...
        "layout", "gridbag"...
    );

    frame_style = uicontrol(frame_left,...
        "style", "frame",...
        "backgroundcolor", [1 1 1],...
        "constraints", createConstraints("gridbag", [1, 10, 1, 1], [1, 1], "both", "center")...
    );
////////////////////////////////////////////////

////////////////////////////////////////////////
//frame_right parameters
////////////////////////////////////////////////
    frame_right = uicontrol(filter_ui,...
        "style", "frame",...
        "layout", "border",...
        "constraints", createConstraints("border", "center")...
    );

    plotid = newaxes(frame_right);
////////////////////////////////////////////////

////////////////////////////////////////////////
//Select filter type
```

```
////////////////////////////////////////////////
    frame_filter_type = uicontrol(frame_left,...
        "style", "frame",...
        "backgroundcolor", [1 1 1],...
        "constraints", createConstraints("gridbag", [1, 1, 1, 1], [0, 0], "both", "center"),...
        "border", createBorder(...
            "titled",...
            createBorder("line", "lightGray", 1),...
            _("Filter Type"),...
            "center",...
            "top",...
            createBorderFont("", 11, "normal"),...
            "black"...
        ),...
        "layout_options", createLayoutOptions("grid", [8, 1])...
    );

    frame_filter_type.layout = "grid";

    uicontrol(frame_filter_type,...
        "tag", "kalman",...
        "string", "Singer-Kalman",...
        "style", "radiobutton",...
        "backgroundcolor", [1 1 1],...
        "groupname", "filter_type",...
        "callback", "update_filter_type"...
    );

    uicontrol(frame_filter_type,...
        "string", "————————————",...
        "style", "text",...
        "foregroundcolor", [204/255 204/255 204/255],...
        "backgroundcolor", [1 1 1]...
    );
```

```
uicontrol(frame_filter_type,...
    "string", "————————————————",...
    "style", "text",...
    "foregroundcolor", [204/255 204/255 204/255],...
    "backgroundcolor", [1 1 1]...
);

uicontrol(frame_filter_type,...
    "tag", "beta_abf",...
    "style", "spinner",...
    "backgroundcolor", [1 1 1],...
    "value", BETA_ABF,...
    "min", 0,...
    "max", 1,...
    "sliderstep", [0.01],...
    "callback", "update_filter_type"...
);

uicontrol(frame_filter_type,...
    "string", "Beta:",...
    "style", "text",...
    "backgroundcolor", [1 1 1]...
);

uicontrol(frame_filter_type,...
    "tag", "alpha_abf",...
    "style", "spinner",...
    "backgroundcolor", [1 1 1],...
    "value", ALPHA_ABF,...
    "min", 0,...
    "max", 1,...
    "sliderstep", [0.01],...
    "callback", "update_filter_type"...
```

```
);

uicontrol(frame_filter_type,...
    "string", "Alpha:",...
    "style", "text",...
    "backgroundcolor", [1 1 1],...
    "verticalalignment", "middle",...
    "horizontalalignment", "left"...
);

uicontrol(frame_filter_type,...
    "string", "",...
    "style", "text",...
    "backgroundcolor", [1 1 1]...
);

uicontrol(frame_filter_type,...
    "tag", "alpha_beta",...
    "string", "Alpha-Beta",...
    "style", "radiobutton",...
    "backgroundcolor", [1 1 1],...
    "groupname", "filter_type",...
    "callback", "update_filter_type"...
);

uicontrol(frame_filter_type,...
    "string", "————————————",...
    "style", "text",...
    "foregroundcolor", [204/255 204/255 204/255],...
    "backgroundcolor", [1 1 1]...
);

uicontrol(frame_filter_type,...
    "string", "————————————",...
```

```
    "style", "text",...
    "foregroundcolor", [204/255 204/255 204/255],...
    "backgroundcolor", [1 1 1]...
);

uicontrol(frame_filter_type,...
    "tag", "particles_pf",...
    "style", "spinner",...
    "backgroundcolor", [1 1 1],...
    "value", PARTICLES_PF,...
    "min", PARTICLES_MIN,...
    "max", PARTICLES_MAX,...
    "sliderstep", [PARTICLES_STEPSIZE],...
    "callback", "update_filter_type"...
);

uicontrol(frame_filter_type,...
    "string", strcat(["Particles [", string(PARTICLES_MIN), " - ",
        string(PARTICLES_MAX), "]:"]),...
    "style", "text",...
    "backgroundcolor", [1 1 1],...
    "verticalalignment", "middle",...
    "horizontalalignment", "left"...
);

uicontrol(frame_filter_type,...
    "string", "",...
    "style", "text",...
    "backgroundcolor", [1 1 1]...
);

filter_type = uicontrol(frame_filter_type,...
    "tag", "particle",...
    "string", "SIR Particle",...
```

```
        "style", "radiobutton",...
        "backgroundcolor", [1 1 1],...
        "groupname", "filter_type",...
        "callback", "update_filter_type"...
    );

    filter_type.value = 1;
////////////////////////////////////////////////

////////////////////////////////////////////////
//Select noise model
////////////////////////////////////////////////
    frame_noise_model = uicontrol(frame_left,...
        "style", "frame",...
        "backgroundcolor", [1 1 1],...
        "constraints", createConstraints("gridbag", [1, 2, 1, 1], [0, 0], "both", "center"),...
        "border", createBorder(...
            "titled",...
            createBorder("line", "lightGray", 1),...
            _("Noise Model"),...
            "center",...
            "top",...
            createBorderFont("", 11, "normal"),...
            "black"...
        ),...
        "layout_options", createLayoutOptions("grid", [3, 1])...
    );

    frame_noise_model.layout = "grid";

    uicontrol(frame_noise_model,...
        "tag", "laplacian",...
        "string", "Laplacian",...
        "style", "radiobutton",...
```

```
        "backgroundcolor", [1 1 1],...
        "groupname", "noise_model",...
        "callback", "update_noise_model"...
    );

    uicontrol(frame_noise_model,...
        "tag", "gaussian",...
        "string", "Gaussian",...
        "style", "radiobutton",...
        "backgroundcolor", [1 1 1],...
        "groupname", "noise_model",...
        "callback", "update_noise_model"...
    );

    noise_model = uicontrol(frame_noise_model,...
        "tag", "none",...
        "string", "None",...
        "style", "radiobutton",...
        "backgroundcolor", [1 1 1],...
        "groupname", "noise_model",...
        "callback", "update_noise_model"...
    );

    noise_model.value = 1;
////////////////////////////////////////////////////

////////////////////////////////////////////////////
//Select simulation type
////////////////////////////////////////////////////
    frame_simulation_type = uicontrol(frame_left,...
        "style", "frame",...
        "backgroundcolor", [1 1 1],...
        "constraints", createConstraints("gridbag", [1, 3, 1, 1], [0, 0], "both", "center"),...
        "border", createBorder(...
```

```
        "titled",...
        createBorder("line", "lightGray", 1),...
        _("Simulation Type"),...
        "center",...
        "top",...
        createBorderFont("", 11, "normal"),...
        "black"...
    ),...
    "layout_options", createLayoutOptions("grid", [5, 1])...
);

frame_simulation_type.layout = "grid";

uicontrol(frame_simulation_type,...
    "tag", "display_lock",...
    "string", "Display Lock",...
    "style", "checkbox",...
    "backgroundcolor", [1 1 1],...
    "value", 1,...
    "callback", "update_simulation_type"...
);

uicontrol(frame_simulation_type,...
    "tag", "batch_size",...
    "style", "spinner",...
    "backgroundcolor", [1 1 1],...
    "value", BATCH_MIN,...
    "min", BATCH_MIN,...
    "max", BATCH_MAX,...
    "sliderstep", [BATCH_STEPSIZE],...
    "callback", "update_simulation_type"...
);

uicontrol(frame_simulation_type,...
```

```
    "string", strcat(["Size [", string(BATCH_MIN), " - ", string(BATCH_MAX), "]:"]),...
    "style", "text",...
    "backgroundcolor", [1 1 1],...
    "verticalalignment", "middle",...
    "horizontalalignment", "left"...
);

uicontrol(frame_simulation_type,...
    "string", "",...
    "style", "text",...
    "backgroundcolor", [1 1 1]...
);

uicontrol(frame_simulation_type,...
    "tag", "batch",...
    "string", "Batch",...
    "style", "radiobutton",...
    "backgroundcolor", [1 1 1],...
    "groupname", "simulation_type",...
    "callback", "update_simulation_type"...
);

uicontrol(frame_simulation_type,...
    "string", "————————————————",...
    "style", "text",...
   "foregroundcolor", [204/255 204/255 204/255],...
    "backgroundcolor", [1 1 1]...
);

uicontrol(frame_simulation_type,...
    "string", "————————————————",...
    "style", "text",...
   "foregroundcolor", [204/255 204/255 204/255],...
    "backgroundcolor", [1 1 1]...
```

```
    );

    uicontrol(frame_simulation_type,...
        "string", "",...
        "style", "text",...
        "backgroundcolor", [1 1 1]...
    );

    simulation_type = uicontrol(frame_simulation_type,...
        "tag", "single_shot",...
        "string", "Single Shot",...
        "style", "radiobutton",...
        "backgroundcolor", [1 1 1],...
        "groupname", "simulation_type",...
        "callback", "update_simulation_type"...
    );

    simulation_type.value = SINGLE_SHOT;
////////////////////////////////////////////////////

////////////////////////////////////////////////////
//Select visible plots
////////////////////////////////////////////////////
    frame_visible_plots = uicontrol(frame_left,...
        "style", "frame",...
        "backgroundcolor", [1 1 1],...
        "constraints", createConstraints("gridbag", [1, 4, 1, 1], [0, 0], "both", "center"),...
        "border", createBorder(...
            "titled",...
            createBorder("line", "lightGray", 1),...
            _("Visible Plots"),...
            "center",...
            "top",...
            createBorderFont("", 11, "normal"),...
```

```
            "black"...
        ),...
        "layout_options", createLayoutOptions("grid", [3, 1])...
    );

    frame_visible_plots.layout = "grid";

    uicontrol(frame_visible_plots,...
        "tag", "filtered_track",...
        "string", "Filtered Track",...
        "style", "checkbox",...
        "backgroundcolor", [1 1 1],...
        "value", 0,...
        "callback", "update_visible_plots"...
    );

    uicontrol(frame_visible_plots,...
        "tag", "detections",...
        "string", "Detections",...
        "style", "checkbox",...
        "backgroundcolor", [1 1 1],...
        "value", 0,...
        "callback", "update_visible_plots"...
    );

    uicontrol(frame_visible_plots,...
        "tag", "true_track",...
        "string", "True Track",...
        "style", "checkbox",...
        "backgroundcolor", [1 1 1],...
        "value", 1,...
        "callback", "update_visible_plots"...
    );
////////////////////////////////////////////////
```

```
////////////////////////////////////////////////
//Select plot type
////////////////////////////////////////////////
    frame_plot_type = uicontrol(frame_left,...
        "style", "frame",...
        "backgroundcolor", [1 1 1],...
        "constraints", createConstraints("gridbag", [1, 5, 1, 1], [0, 0], "both", "center"),...
        "border", createBorder(...
            "titled",...
            createBorder("line", "lightGray", 1),...
            _("Plot Type"),...
            "center",...
            "top",...
            createBorderFont("", 11, "normal"),...
            "black"...
        ),...
        "layout_options", createLayoutOptions("grid", [1, 1])...
    );

    frame_plot_type.layout = "grid";

//    uicontrol(frame_plot_type,...
//        "tag", "plot_3d",...
//        "string", "3D Plot",...
//        "style", "radiobutton",...
//        "backgroundcolor", [1 1 1],...
//        "groupname", "plot_type",...
//        "callback", "update_plot_type"...
//    );

//    plot_type = uicontrol(frame_plot_type,...
//        "tag", "plot_2d_vert",...
//        "string", "2D Plot Vert.",...
```

```
//        "style", "radiobutton",...
//        "backgroundcolor", [1 1 1],...
//        "groupname", "plot_type",...
//        "callback", "update_plot_type"...
//    );

    plot_type = uicontrol(frame_plot_type,...
        "tag", "plot_2d_horz",...
        "string", "2D Plot (Horizontal)",...
        "style", "radiobutton",...
        "backgroundcolor", [1 1 1],...
        "groupname", "plot_type",...
        "callback", "update_plot_type"...
    );

    plot_type.value = PLOT_2D_HORZ;
//////////////////////////////////////////////////

//////////////////////////////////////////////////
//Select auxiliary plots
//////////////////////////////////////////////////
    frame_auxiliary_plots = uicontrol(frame_left,...
        "tag", "frame_auxiliary_plots",...
        "style", "frame",...
        "backgroundcolor", [1 1 1],...
        "constraints", createConstraints("gridbag", [1, 6, 1, 1], [0, 0], "both", "center"),...
        "border", createBorder(...
            "titled",...
            createBorder("line", "lightGray", 1),...
            _("Auxiliary Plots"),...
            "center",...
            "top",...
            createBorderFont("", 11, "normal"),...
            "black"...
```

```
        ),...
        "layout_options", createLayoutOptions("grid", [1, 1])...
    );

    frame_auxiliary_plots.layout = "grid";

    uicontrol(frame_auxiliary_plots,...
        "tag", "pcrlb_plot",...
        "string", "PCRLB Plot",...
        "style", "checkbox",...
        "backgroundcolor", [1 1 1],...
        "value", 0,...
        "enable", "on",...
        "callback", "update_auxiliary_plots"...
    );

    uicontrol(frame_auxiliary_plots,...
        "tag", "error_plot",...
        "string", "Error Plot",...
        "style", "checkbox",...
        "backgroundcolor", [1 1 1],...
        "value", 0,...
        "callback", "update_auxiliary_plots"...
    );
////////////////////////////////////////////////

////////////////////////////////////////////////
//Select detection parameters
////////////////////////////////////////////////
    frame_detect_parameters = uicontrol(frame_left,...
        "style", "frame",...
        "backgroundcolor", [1 1 1],...
        "constraints", createConstraints("gridbag", [1, 7, 1, 1], [0, 0], "both", "center"),...
        "border", createBorder(...
```

```
        "titled",...
        createBorder("line", "lightGray", 1),...
        _("Detection Parameters"),...
        "center",...
        "top",...
        createBorderFont("", 11, "normal"),...
        "black"...
    ),...
    "layout_options", createLayoutOptions("grid", [2, 1])...
);

frame_detect_parameters.layout = "grid";

uicontrol(frame_detect_parameters,...
    "tag", "detect_transient_size",...
    "style", "spinner",...
    "backgroundcolor", [1 1 1],...
    "value", 0,...
    "min", 0,...
    "max", 100,...
    "sliderstep", [10],...
    "callback", "update_detect_parameters"...
);

uicontrol(frame_detect_parameters,...
    "tag", "detect_transient",...
    "string", "Transient [%]:",...
    "style", "checkbox",...
    "backgroundcolor", [1 1 1],...
    "value", 0,...
    "callback", "update_detect_parameters"...
);

uicontrol(frame_detect_parameters,...
```

```
        "tag", "detect_missed_size",...
        "style", "spinner",...
        "backgroundcolor", [1 1 1],...
        "value", 0,...
        "min", 0,...
        "max", 10,...
        "sliderstep", [1],...
        "callback", "update_detect_parameters"...
    );

    uicontrol(frame_detect_parameters,...
        "tag", "detect_missed",...
        "string", "Missed [%]:",...
        "style", "checkbox",...
        "backgroundcolor", [1 1 1],...
        "value", 0,...
        "callback", "update_detect_parameters"...
    );
////////////////////////////////////////////////

////////////////////////////////////////////////
//Select target model
////////////////////////////////////////////////
    frame_target_model = uicontrol(frame_left,...
        "style", "frame",...
        "backgroundcolor", [1 1 1],...
        "constraints", createConstraints("gridbag", [1, 8, 1, 1], [0, 0], "both", "center"),...
        "border", createBorder(...
            "titled",...
            createBorder("line", "lightGray", 1),...
            _("Target Model"),...
            "center",...
            "top",...
            createBorderFont("", 11, "normal"),...
```

```
            "black"...
        ),...
        "layout_options", createLayoutOptions("grid", [1, 1])...
    );

    frame_target_model.layout = "grid";

    target_models = strcat([gettext("Spiral-Dive"), gettext("Random"),
            gettext("Sinusoidal"), gettext("Linear")], "|");

    uicontrol(frame_target_model,...
        "tag", "target_model",...
        "string", target_models,...
        "style", "popupmenu",...
        "backgroundcolor", [1 1 1],...
        "fontSize", 11,...
        "value", 1,...
        "callback", "update_target_model"...
    );

    //Listbox alternative
    //"style", "listbox",...
    //List alternative
    //"string", gettext("item1|item2|item3|item4"),...
////////////////////////////////////////////////////

////////////////////////////////////////////////////
//Run simulation
////////////////////////////////////////////////////
    frame_buttons = uicontrol(frame_left,...
        "style", "frame",...
        "backgroundcolor", [1 1 1],...
        "constraints", createConstraints("gridbag", [1, 9, 1, 1], [0, 0], "both", "center"),...
        "border", createBorder(...
```

```
            "titled",...
            createBorder("line", "lightGray", 1),...
            _(""),...
            "center",...
            "top",...
            createBorderFont("", 11, "normal"),...
            "black"...
        ),...
        "layout_options", createLayoutOptions("grid", [1, 1])...
    );

    frame_buttons.layout = "grid";

    run_simulation_button = uicontrol(frame_buttons,...
        "string", "Run Simulation",...
        "style", "pushbutton",...
        "fontsize", 16,...
        "callback", "run_simulation()"...
    );
////////////////////////////////////////////////

////////////////////////////////////////////////
//UI draw
////////////////////////////////////////////////
    filter_ui.immediate_drawing = "off";

    update_plot();

    filter_ui.immediate_drawing = "on";
    filter_ui.visible           = "on";
endfunction
////////////////////////////////////////////////

////////////////////////////////////////////////
```

```
//About menu function
////////////////////////////////////////////////
function filter_about()
   msg = msprintf(...
      gettext(...
         "Radar Target Tracking Filter Simulator V" + string(VERSION) + "."
                  + string(REVISION) +...
         "\nAuthor: E.F. Bauermeister" +...
         "\nDate: " + DATE...
      )...
   );
   messagebox(msg, gettext("About"), "info", "modal");
endfunction
////////////////////////////////////////////////

////////////////////////////////////////////////
//Update main plot function
////////////////////////////////////////////////
function update_plot();
   global t;
   global x_true;
   global x_detections;
   global x_filtered;

   drawlater();

   set(gca(), "auto_clear", "on");

   plot2d(0, 0);

   plot_legend = " ;

   if(TRUE_TRACK == TRUE) then
      plot2d(t, x_true);
```

```
    c = gca();
    p = c.children.children(1);
    p.line_mode = "on";
    p.mark_mode = "off";
    p.thickness = 2;
    p.foreground = 2;   //blue
    plot_legend = cat(1, plot_legend, 'True Track');
    set(gca(), "auto_clear", "off");
end;

if(FILTERED_TRACK == TRUE) then
    plot2d(t, x_filtered);
    c = gca();
    p = c.children.children(1);
    p.line_mode = "on";
    p.mark_mode = "off";
    p.thickness = 2;
    p.foreground = 5;   //red

    plot_legend = cat(1, plot_legend, 'Filtered Track');
    set(gca(), "auto_clear", "off");
end;

if(DETECTIONS == TRUE) then
    plot2d(t, x_detections);
    c = gca();
    p = c.children.children(1);
    p.line_mode = "off";
    p.mark_mode = "on";
    p.thickness = 2;
    p.mark_style = 2;
    p.mark_size = 0;
    plot_legend = cat(1, plot_legend, 'Detections');
    set(gca(), "auto_clear", "off");
```

```
    end;

    xgrid(1);
    p = c.x_label;
    p.font_size = 3;
    xlabel('Time');
    p = c.y_label;
    p.font_size = 3;
    ylabel('Track (norm.)');
    p = c.title;
    p.font_size = 4;
    title('Normalized Target Track versus Time');

    drawnow();

    set(gca(), "auto_clear", "off");

    if(plot_legend <> '') then
        legend(plot_legend);
        e = gce();
        e.font_size = 2;
    end;
endfunction
////////////////////////////////////////////////

////////////////////////////////////////////////
//Simulation function
////////////////////////////////////////////////
function run_simulation()
    global FILTER_TYPE;
    global PARTICLES_PF;
    global ALPHA_ABF;
    global BETA_ABF;
    global NOISE_MODEL;
```

```
    global SIMULATION_TYPE;
    global BATCH_SIZE;
    global DISPLAY_LOCK;
    global TRUE_TRACK;
    global DETECTIONS;
    global FILTERED_TRACK;
    global PLOT_TYPE;
    global ERROR_PLOT;
    global TARGET_MODEL;
    global DETECT_MISSED;
    global DETECT_MISSED_SIZE;
    global DETECT_TRANSIENT;
    global DETECT_TRANSIENT_SIZE;

    global xd_errors;
    global xf_errors;
    global pcrlb;
    global time;
    global rmse_fil;
    global rmse_det;
    global batch_rmse_filtered;
    global batch_rmse_detections;
    global batch_simulation_time;

    if(SIMULATION_TYPE == SINGLE_SHOT) then
        generate_true();
        generate_detections();

        //Required to reset stopwatch timer?
        tic();
        toc();

        tic(); //start stopwatch timer
        generate_filtered();
```

```
    time = toc(); //stop stopwatch timer

    pcrlb = zeros(1, m);

    compute_error();

    for(i = 1:m)
       pcrlb(i) = sqrt(pcrlb(i));
    end;

    update_plot();
    display_aux_plots();
    save_results();
    display_results();
  elseif(SIMULATION_TYPE == BATCH) then
    time = 0;
    temp_xd_errors = 0;
    temp_xf_errors = 0;
    pcrlb = zeros(1, m);
    for(i = 1:BATCH_SIZE)
       generate_true();
       generate_detections();

       //Required to reset stopwatch timer?
       tic();
       toc();

       tic(); //start stopwatch timer
       generate_filtered();
       time = time + toc(); //stop stopwatch timer
       compute_error();

       batch_simulation_time(i) = time;
       batch_rmse_detections(i) = rmse_det;
```

```
        batch_rmse_filtered(i) = rmse_fil;

        temp_xd_errors = temp_xd_errors + xd_errors;
        temp_xf_errors = temp_xf_errors + xf_errors;
        if(DISPLAY_LOCK == FALSE) then
            update_plot();
            display_aux_plots();
        end;
        clc();
        mprintf("Running batch simulation %d of %d...", i, BATCH_SIZE);
    end;
    xd_errors = temp_xd_errors / BATCH_SIZE;
    xf_errors = temp_xf_errors / BATCH_SIZE;
    for(i = 1:m)
        pcrlb(i) = sqrt(pcrlb(i) / BATCH_SIZE);
    end;
    display_aux_plots();
    save_results();
    display_results();
end;

if(DEBUG == TRUE) then
    disp("run_simulation()...");
    mprintf("FILTER_TYPE = %d\n", FILTER_TYPE);
    mprintf("PARTICLES_PF = %d\n", PARTICLES_PF);
    mprintf("ALPHA_ABF = %1.2f\n", ALPHA_ABF);
    mprintf("BETA_ABF = %1.2f\n", BETA_ABF);
    mprintf("NOISE_MODEL = %d\n", NOISE_MODEL);
    mprintf("SIMULATION_TYPE = %d\n", SIMULATION_TYPE);
    mprintf("BATCH_SIZE = %d\n", BATCH_SIZE);
    mprintf("DISPLAY_LOCK = %d\n", DISPLAY_LOCK);
    mprintf("TRUE_TRACK = %d\n", TRUE_TRACK);
    mprintf("DETECTIONS = %d\n", DETECTIONS);
    mprintf("FILTERED_TRACK = %d\n", FILTERED_TRACK);
```

```
        mprintf("PLOT_TYPE = %d\n", PLOT_TYPE);
        mprintf("ERROR_PLOT = %d\n", ERROR_PLOT);
        mprintf("TARGET_MODEL = %d\n", TARGET_MODEL);
        mprintf("DETECT_MISSED = %d\n", DETECT_MISSED);
        mprintf("DETECT_MISSED_SIZE = %d\n", DETECT_MISSED_SIZE);
        mprintf("DETECT_TRANSIENT = %d\n", DETECT_TRANSIENT);
        mprintf("DETECT_TRANSIENT_SIZE = %d\n", DETECT_TRANSIENT_SIZE);
    end;
endfunction
////////////////////////////////////////////////

////////////////////////////////////////////////
//Control panel update functions
////////////////////////////////////////////////
function update_filter_type()
    global FILTER_TYPE;
    global PARTICLES_PF;
    global ALPHA_ABF;
    global BETA_ABF;

    if(get(gcbo, "tag") == "particle") then
        FILTER_TYPE = PARTICLE;
        set("filter_ui/frame_left/frame_auxiliary_plots/pcrlb_plot", "enable", "on");
    elseif(get(gcbo, "tag") == "alpha_beta") then
        FILTER_TYPE = ALPHA_BETA;
        set("filter_ui/frame_left/frame_auxiliary_plots/pcrlb_plot", "enable", "off");
    elseif(get(gcbo, "tag") == "kalman") then
        FILTER_TYPE = KALMAN;
        set("filter_ui/frame_left/frame_auxiliary_plots/pcrlb_plot", "enable", "off");
    elseif(get(gcbo, "tag") == "particles_pf") then
        PARTICLES_PF = get(gcbo, "value");
    elseif(get(gcbo, "tag") == "alpha_abf") then
        ALPHA_ABF = get(gcbo, "value");
    elseif(get(gcbo, "tag") == "beta_abf") then
```

```
        BETA_ABF = get(gcbo, "value");
    end;

    if(DEBUG == TRUE) then
        disp("update_noise_model()...");
        mprintf("FILTER_TYPE = %d\n", FILTER_TYPE);
        mprintf("PARTICLES_PF = %d\n", PARTICLES_PF);
        mprintf("ALPHA_ABF = %1.2f\n", ALPHA_ABF);
        mprintf("BETA_ABF = %1.2f\n", BETA_ABF);
    end;
endfunction

function update_noise_model()
    global NOISE_MODEL;

    if(get(gcbo, "tag") == "none") then
        NOISE_MODEL = NONE;
    elseif(get(gcbo, "tag") == "gaussian") then
        NOISE_MODEL = GAUSSIAN;
    elseif(get(gcbo, "tag") == "laplacian") then
        NOISE_MODEL = LAPLACIAN;
    end;

    if(DEBUG == TRUE) then
        disp("update_noise_model()...");
        mprintf("NOISE_MODEL = %d\n", NOISE_MODEL);
    end;
endfunction

function update_simulation_type()
    global SIMULATION_TYPE;
    global BATCH_SIZE;
    global DISPLAY_LOCK;
```

```
    if(get(gcbo, "tag") == "single_shot") then
        SIMULATION_TYPE = SINGLE_SHOT;
    elseif(get(gcbo, "tag") == "batch") then
        SIMULATION_TYPE = BATCH;
    elseif(get(gcbo, "tag") == "batch_size") then
        BATCH_SIZE = get(gcbo, "value");
    elseif(get(gcbo, "tag") == "display_lock") then
        DISPLAY_LOCK = DISPLAY_LOCK * FALSE;
    end;

    if(DEBUG == TRUE) then
        disp("update_simulation_type()...");
        mprintf("SIMULATION_TYPE = %d\n", SIMULATION_TYPE);
        mprintf("BATCH_SIZE = %d\n", BATCH_SIZE);
        mprintf("DISPLAY_LOCK = %d\n", DISPLAY_LOCK);
    end;
endfunction

function update_visible_plots()
    global TRUE_TRACK;
    global DETECTIONS;
    global FILTERED_TRACK;

    if get(gcbo, "tag") == "true_track" then
        TRUE_TRACK = TRUE_TRACK * FALSE;
    elseif get(gcbo, "tag") == "detections" then
        DETECTIONS = DETECTIONS * FALSE;
    elseif get(gcbo, "tag") == "filtered_track" then
        FILTERED_TRACK = FILTERED_TRACK * FALSE;
    end;

    update_plot();

    if(DEBUG == TRUE) then
```

```
        disp("update_visible_plots()...");
        mprintf("TRUE_TRACK = %d\n", TRUE_TRACK);
        mprintf("DETECTIONS = %d\n", DETECTIONS);
        mprintf("FILTERED_TRACK = %d\n", FILTERED_TRACK);
    end;
endfunction

function update_auxiliary_plots()
    global ERROR_PLOT;
    global PCRLB_PLOT;

    if get(gcbo, "tag") == "error_plot" then
        ERROR_PLOT = ERROR_PLOT * FALSE;
    elseif get(gcbo, "tag") == "pcrlb_plot" then
        PCRLB_PLOT = PCRLB_PLOT * FALSE;
    end;

    if(DEBUG == TRUE) then
        disp("update_auxiliary_plots()...");
        mprintf("ERROR_PLOT = %d\n", ERROR_PLOT);
        mprintf("PCRLB_PLOT = %d\n", PCRLB_PLOT);
    end;
endfunction

function update_plot_type()
    global PLOT_TYPE;

    if get(gcbo, "tag") == "plot_2d_horz" then
        PLOT_TYPE = PLOT_2D_HORZ;
    elseif get(gcbo, "tag") == "plot_2d_vert" then
        PLOT_TYPE = PLOT_2D_VERT;
    elseif get(gcbo, "tag") == "plot_3d" then
        PLOT_TYPE = PLOT_3D;
    end;
```

```
    if(DEBUG == TRUE) then
        disp("update_plot_type()...");
        mprintf("PLOT_TYPE = %d\n", PLOT_TYPE);
    end;
endfunction

function update_detect_parameters()
    global DETECT_MISSED;
    global DETECT_MISSED_SIZE;
    global DETECT_TRANSIENT;
    global DETECT_TRANSIENT_SIZE;

    if get(gcbo, "tag") == "detect_missed" then
        DETECT_MISSED = DETECT_MISSED * FALSE;
    elseif(get(gcbo, "tag") == "detect_missed_size") then
        DETECT_MISSED_SIZE = get(gcbo, "value");
    elseif get(gcbo, "tag") == "detect_transient" then
        DETECT_TRANSIENT = DETECT_TRANSIENT * FALSE;
    elseif(get(gcbo, "tag") == "detect_transient_size") then
        DETECT_TRANSIENT_SIZE = get(gcbo, "value");
    end;

    if(DEBUG == TRUE) then
        disp("update_detect_parameters()...");
        mprintf("DETECT_MISSED = %d\n", DETECT_MISSED);
        mprintf("DETECT_MISSED_SIZE = %d\n", DETECT_MISSED_SIZE);
        mprintf("DETECT_TRANSIENT = %d\n", DETECT_TRANSIENT);
        mprintf("DETECT_TRANSIENT_SIZE = %d\n", DETECT_TRANSIENT_SIZE);
    end;
endfunction

function update_target_model()
    global TARGET_MODEL;
```

```
    TARGET_MODEL = get(gcbo, "value");

    generate_true();

    if(DEBUG == TRUE) then
        disp("update_target_model()...");
        mprintf("TARGET_MODEL = %d\n", TARGET_MODEL);
    end;
endfunction
////////////////////////////////////////////////

////////////////////////////////////////////////
//Generate true target track
////////////////////////////////////////////////
function generate_true()
    global TARGET_MODEL;

    global t;
    global x_true;

    if(TARGET_MODEL == SPIRAL_DIVE) then
        x1 = cos(2 * %pi * t / m); //aircraft dive
        x2 = SPIRAL_AMPLITUDE * sin(SPIRAL_RATE * 2 * %pi * t / m); //aircraft spiral
        x_true = x1 + x2;
    elseif(TARGET_MODEL == RANDOM) then
        x_true = rand(1, m); //true target position - "random"
    elseif(TARGET_MODEL == SINUSOIDAL) then
        x_true = sin(2 * %pi * t / 50); //true target position - sinusoidal
    elseif(TARGET_MODEL == LINEAR) then
        x_true = ones(1, m); //true target position - linear
    end;
endfunction
////////////////////////////////////////////////
```

```
////////////////////////////////////////////////
//Generate target detections
////////////////////////////////////////////////
function generate_detections()
   global NOISE_MODEL;
   global DETECT_MISSED;
   global DETECT_MISSED_SIZE;
   global DETECT_TRANSIENT;
   global DETECT_TRANSIENT_SIZE;

   global x_true;
   global x_detections;

   noisegen(1, m, Gaussian_noise_std_dev); //Gaussian noise generator
   seed = getdate("s"); //get current date in seconds for rng seed
   rand("seed", seed); //seed random number generator

   if(NOISE_MODEL == NONE) then
      x_detections = x_true;
   elseif(NOISE_MODEL == GAUSSIAN) then
      x_detections = x_true + feval(t, Noise); //Gaussian noise
   elseif(NOISE_MODEL == LAPLACIAN) then
      noise = rand(1, m, "uniform"); //default
      for(i = 1:m)
         if(noise(i) > 0.5) then
            temp1 = log(2 * (1 - noise(i)));
            temp2 = (-1 * (sqrt(Laplacian_noise_variance) * (1 / sqrt(2))) * real(temp1));
            x_detections(i) = x_true(i) + (noise_scale * temp2);
            laplacian_noise(i) = temp2;
         else
            temp1 = log(2 * noise(i));
            temp2 = (sqrt(Laplacian_noise_variance) * (1 / sqrt(2)) * real(temp1));
            x_detections(i) = x_true(i) + (noise_scale * temp2);
```

```
            laplacian_noise(i) = temp2;
        end;
    end;
end;

if(DEBUG_NOISE == TRUE)
    if(NOISE_MODEL == NONE) then
        if (get_figure_handle(100003) <> []) then
            xdel(100003);
        end;
    elseif(NOISE_MODEL == GAUSSIAN) then
        f = gca();
        scf(100003);
        set(gca(), "auto_clear", "on");
        subplot(121);
        plot(feval(t, Noise));
        xgrid(1);
        title('Noise Data Plot');
        subplot(122);
        histplot(n, feval(t, Noise));
        a = gca();
        xgrid(1);
        title('Noise Histogram Plot');
        set(gca(), "auto_clear", "off");
        sca(f);
    elseif(NOISE_MODEL == LAPLACIAN) then
        f = gca();
        scf(100003);
        set(gca(), "auto_clear", "on");
        subplot(121);
        plot(laplacian_noise);
        xgrid(1);
        title('Noise Data Plot');
        subplot(122);
```

```
        histplot(n, laplacian_noise);
        a = gca();
        xgrid(1);
        title('Noise Histogram Plot');
        set(gca(), "auto_clear", "off");
        sca(f);
    end;
else
    if (get_figure_handle(100003) <> []) then
        xdel(100003);
    end;
end;

if(DETECT_MISSED == TRUE) then
    missed = round(m * (DETECT_MISSED_SIZE / 100));
    for(i = 1:missed)
        missed_index = round(rand(1) * m);
        while((missed_index == 0) | isnan(x_detections(missed_index)))
            missed_index = round(rand(1) * m);
        end;
        x_detections(missed_index) = %nan;
    end;
end;

if(DETECT_TRANSIENT == TRUE) then
    transient_index = round(rand(1) * m);
    while(transient_index == 0)
        transient_index = round(rand(1) * m);
    end;

    if(x_detections(transient_index) >= 0) then
        transient_polarity = 1;
    else
        transient_polarity = -1;
```

```
        end;

        x_detections(transient_index) = x_detections(transient_index) + (transient_polarity
            * max(x_true) * DETECT_TRANSIENT_SIZE / 100);
        if(DEBUG == TRUE) then
            disp("generate_detections()...");
            mprintf("transient_index = %d\n", transient_index);
            mprintf("transient_polarity = %d\n", transient_polarity);
        end;
    end;
endfunction
////////////////////////////////////////////////

////////////////////////////////////////////////
//Generate filtered
////////////////////////////////////////////////
function generate_filtered()
    global FILTER_TYPE;
    global PARTICLES_PF;
    global ALPHA_ABF;
    global BETA_ABF;

    global t;
    global x_detections;
    global x_filtered;
    global x_true;

    if(FILTER_TYPE == PARTICLE) then
        y_initial = x_detections(1); //initial filter state
        var_measure = 0.1; //measurement noise covariance
        var_init_est = 0.1; //variance of the initial estimate
        y_particles = []; //array of particles

        //initialise randomly generated particles
```

```
for(i = 1:PARTICLES_PF)
   y_particles(i) = (2.4 * rand(1)) - 1.2;
end;

y_estimate = y_initial; //current particle filter estimate

for(i = 1:m) //observation interval
   if(i <> 1)
      if(~isnan(x_detections(i)) & ~isnan(x_detections(i - 1))) then
         y = x_detections(i); //current detection
         for(j = 1:PARTICLES_PF)
            particle_update(j) = y_particles(j) + (x_detections(i) - x_detections(i - 1));
            particle_weights(j) = (1 / sqrt(2 * %pi * var_measure)) * exp(-(y
                  - particle_update(j))^2 / (2 * var_measure));
         end;
         particle_weights = particle_weights ./ sum(particle_weights);
            //normalize PDF (i.e. sum to 1)

         //resample
         for(j = 1:PARTICLES_PF)
            y_particles(j) = particle_update(find(rand(1) <= cumsum(particle_weights), 1));
         end;
         y_estimate = mean(y_particles);
      end;
   else
      if(~isnan(x_detections(i))) then
         y = x_detections(i); //current detection

         for(j = 1:PARTICLES_PF)
            particle_update(j) = y_particles(j);
            particle_weights(j) = (1 / sqrt(2 * %pi * var_measure)) * exp(-(y
                  - particle_update(j))^2 / (2 * var_measure));
         end;
         particle_weights = particle_weights ./ sum(particle_weights);
```

```
                    //normalize PDF (i.e. sum to 1)

            //resample
            for(j = 1:PARTICLES_PF)
               y_particles(j) = particle_update(find(rand(1) <= cumsum(particle_weights), 1));
            end;
            y_estimate = mean(y_particles);
         end;
      end;

      x_filtered(i) = y_estimate;
   end;
elseif(FILTER_TYPE == ALPHA_BETA) then
   xk_past = 0; //initial filtered position
   vk_past = 0; //initial filtered velocity
   for(i = 1:m)
      xk_now = xk_past + (vk_past * di_abf);
         //current predicted position = past filtered position +
               //(past filtered velocity * detection interval)
      vk_now = vk_past; //current filtered velocity = past filtered velocity

      if(~isnan(x_detections(i))) then
         rk = x_detections(i) - xk_now;
               //deviation = measured target position - current predicted position
      else
         rk = 0;
      end;

      xk_now = xk_now + (ALPHA_ABF * rk);
               //current predicted position = current predicted position + (alpha * deviation)
      vk_now = vk_now + ((BETA_ABF * rk) / di_abf);
         //current filtered velocity = current filtered velocity +
               // (beta * deviation / detection interval)
      xk_past = xk_now; //past predicted position = current predicted position
```

```
        vk_past = vk_now; //past filtered velocity = current filtered velocity
        x_filtered(i) = xk_now; //filtered position
    end;
  elseif(FILTER_TYPE == KALMAN) then
if(~isnan(x_detections(1))) then
    X0 = [x_detections(1) ; 0 ; 0]; //fixed velocity, zero acceleration
    else
        X0 = [0 ; 0 ; 0]; //fixed velocity, zero acceleration
        error = 0;
    end;

    tau = 1; //target position sampling time interval
    measure_noise_variance = 0.1; //noise variance
    state_noise_variance = 0.1; //state noise variance
    X = X0; //initial estimate for the state vector
    S_matrix = [1 1 1 ; 1 1 1 ; 1 1 1]; //initial value of the predication covariance matrix
    PHI = [1 tau ((tau^2) / 2) ; 0 1 tau ; 0 0 1]; //initial value of the transition matrix PHI

    //covariance matrix for the state noise
    Q_matrix(1, 1) = (state_noise_variance * (tau^5)) / 20;
    Q_matrix(1, 2) = (state_noise_variance * (tau^4)) / 8;
    Q_matrix(1, 3) = (state_noise_variance * (tau^3)) / 6;
    Q_matrix(2, 1) = Q_matrix(1, 2);
    Q_matrix(2, 2) = (state_noise_variance * (tau^3)) / 3;
    Q_matrix(2, 3) = (state_noise_variance * (tau^2)) / 2;
    Q_matrix(3, 1) = Q_matrix(1, 3);
    Q_matrix(3, 2) = Q_matrix(2, 3);
    Q_matrix(3, 3) = state_noise_variance * tau;

    for(i = 1:m)
        XN = PHI * X;
        S_matrix = PHI * S_matrix * PHI' + Q_matrix;

        //Kalman filter gain computation
```

```
        Kalman_gain(1) = S_matrix(1, 1) / (S_matrix(1, 1) + measure_noise_variance);
        Kalman_gain(2) = S_matrix(1, 2) / (S_matrix(1, 1) + measure_noise_variance);
        Kalman_gain(3) = S_matrix(1, 3) / (S_matrix(1, 1) + measure_noise_variance);

        //update state estimate
        if(~isnan(x_detections(i))) then
           error = x_detections(i) + (rand(1, "normal") / 10) - XN(1);
        end;
        X(1) = XN(1) + (Kalman_gain(1) * error);
        X(2) = XN(2) + (Kalman_gain(2) * error);
        X(3) = XN(3) + (Kalman_gain(3) * error);
        x_filtered(i) = X(1);

        //update the error covariance matrix
        S_matrix(1, 1) = S_matrix(1, 1) * (1 - Kalman_gain(1));
        S_matrix(1, 2) = S_matrix(1, 2) * (1 - Kalman_gain(1));
        S_matrix(1, 3) = S_matrix(1, 3) * (1 - Kalman_gain(1));
        S_matrix(2, 1) = S_matrix(1, 2);
        S_matrix(2, 2) = -Kalman_gain(2) * S_matrix(1, 2) + S_matrix(2, 2);
        S_matrix(2, 3) = -Kalman_gain(2) * S_matrix(1, 3) + S_matrix(2, 3);
        S_matrix(3, 1) = S_matrix(1, 3);
        S_matrix(3, 3) = -Kalman_gain(3) * S_matrix(1, 3) + S_matrix(3, 3);
      end;
   end;
endfunction
////////////////////////////////////////////////

////////////////////////////////////////////////
//Compute error metrics
////////////////////////////////////////////////
function compute_error()
   global x_true;
   global x_detections;
   global x_filtered;
```

```
    global xd_errors;
    global xf_errors;
    global rmse_fil;
    global rmse_det;
    global pcrlb;

    rmse_fil = 0;
    rmse_det = 0;
    for(i = 1:m)
        if(~isnan(x_detections(i))) then
            xd_errors(i) = x_true(i) - x_detections(i);
            xf_errors(i) = x_true(i) - x_filtered(i);
            pcrlb(i) = pcrlb(i) + (xf_errors(i))^2;
            rmse_fil = rmse_fil + (x_true(i) - x_filtered(i))^2;
            rmse_det = rmse_det + (x_true(i) - x_detections(i))^2;
        else
            xd_errors(i) = 0;
            xf_errors(i) = 0;
        end;
    end;
    if(DETECT_MISSED == TRUE) then
        missed = round(m * (DETECT_MISSED_SIZE / 100));
    else
        missed = 0 ;
    end;
    rmse_fil = sqrt(rmse_fil / (m - missed));
    rmse_det = sqrt(rmse_det / (m - missed));
endfunction
////////////////////////////////////////////////

////////////////////////////////////////////////
//Save results
////////////////////////////////////////////////
function save_results()
```

```
global FILTER_TYPE;
global PARTICLES_PF;
global ALPHA_ABF;
global BETA_ABF;
global NOISE_MODEL;
global SIMULATION_TYPE;
global BATCH_SIZE;
global DISPLAY_LOCK;
global TRUE_TRACK;
global DETECTIONS;
global FILTERED_TRACK;
global PLOT_TYPE;
global ERROR_PLOT;
global TARGET_MODEL;
global DETECT_MISSED;
global DETECT_MISSED_SIZE;
global DETECT_TRANSIENT;
global DETECT_TRANSIENT_SIZE;

global rmse_fil;
global rmse_det;
global time;
global filename_results;
global filename_data;
global t;
global x_true;
global x_detections;
global x_filtered;
global xd_errors;
global xf_errors;
global pcrlb;
global batch_rmse_filtered;
global batch_rmse_detections;
global batch_simulation_time;
```

```
c = clock();
filename_base = string(c(1)) + '-';
if(c(2) > 9) then
   filename_base = filename_base + string(c(2)) + '-';
else
   filename_base = filename_base + '0' + string(c(2)) + '-';
end
if(c(3) > 9) then
   filename_base = filename_base + string(c(3)) + '_';
else
   filename_base = filename_base + '0' + string(c(3)) + '_';
end
if(c(4) > 9) then
   filename_base = filename_base + string(c(4)) + '-';
else
   filename_base = filename_base + '0' + string(c(4)) + '-';
end
if(c(5) > 9) then
   filename_base = filename_base + string(c(5)) + '-';
else
   filename_base = filename_base + '0' + string(c(5)) + '-';
end
if(int(c(6)) > 9) then
   filename_base = filename_base + string(int(c(6)));
else
   filename_base = filename_base + '0' + string(int(c(6)));
end
filename_results = pwd() + '\' + filename_base + '.txt';

//save simulation results
fid = mopen(filename_results, 'wt');
if(fid == -1) then
   error("Cannot open file for writing.");
```

```
else
    mputl("RMSE Filtered: " + string(rmse_fil * 100) + " %", fid);
    mputl("RMSE  Detections: " + string(rmse_det * 100) + " %", fid);
    mputl("Simulation Time: " + string(time) + " s", fid);
    mputl("", fid);
    mputl("", fid);
    mputl("Configuration:", fid);
    mputl("FILTER_TYPE = " + string(FILTER_TYPE), fid);
    mputl("PARTICLES_PF = " + string(PARTICLES_PF), fid);
    mputl("ALPHA_ABF = " + string(ALPHA_ABF), fid);
    mputl("BETA_ABF = " + string(BETA_ABF), fid);
    mputl("NOISE_MODEL = " + string(NOISE_MODEL), fid);
    mputl("SIMULATION_TYPE = " + string(SIMULATION_TYPE), fid);
    mputl("BATCH_SIZE = " + string(BATCH_SIZE), fid);
    mputl("DISPLAY_LOCK = " + string(DISPLAY_LOCK), fid);
    mputl("TRUE_TRACK = " + string(TRUE_TRACK), fid);
    mputl("DETECTIONS = " + string(DETECTIONS), fid);
    mputl("FILTERED_TRACK = " + string(FILTERED_TRACK), fid);
    mputl("PLOT_TYPE = " + string(PLOT_TYPE), fid);
    mputl("ERROR_PLOT = " + string(ERROR_PLOT), fid);
    mputl("TARGET_MODEL = " + string(TARGET_MODEL), fid);
    mputl("DETECT_MISSED = " + string(DETECT_MISSED), fid);
    mputl("DETECT_MISSED_SIZE = " + string(DETECT_MISSED_SIZE), fid);
    mputl("DETECT_TRANSIENT = " + string(DETECT_TRANSIENT), fid);
    mputl("DETECT_TRANSIENT_SIZE = " + string(DETECT_TRANSIENT_SIZE), fid);
    mclose(fid);
end;

//save simulation data
filename_data = pwd() + '\' + filename_base + '_data.txt';
fid = mopen(filename_data, 'wt');
if(fid == -1) then
    error("Cannot open file for writing.");
else
```

```
        if(SIMULATION_TYPE == SINGLE_SHOT) then
            mputl("TIME TRUE DETECTIONS FILTERED DETECTIONS-ERRORS
                    FILTERED-ERRORS", fid);
            for(i = 1:m)
                temp = string(t(i)) + ' ' + string(x_true(i)) + ' ' + string(x_detections(i))
                    + ' ' + string(x_filtered(i)) + ' ' + string(xd_errors(i)) + ' ' + string(xf_errors(i));
                mputl(temp, fid);
            end;
        elseif(SIMULATION_TYPE == BATCH) then
            mputl("SIMULATION DETECTIONS-RMSE FILTERED-RMSE
                    SIMULATION-TIME", fid);
            for(i = 1:BATCH_SIZE)
                temp = string(i) + ' ' + string(batch_rmse_detections(i))
                    + ' ' + string(batch_rmse_filtered(i)) + ' ' + string(batch_simulation_time(i));
                mputl(temp, fid);
            end;
        end;
        mputl('', fid);
        mputl('PCRLB', fid);
        for(i = 1:m)
            temp = string(i) + ' ' + string(pcrlb(i));
            mputl(temp, fid);
        end;
        mclose(fid);
    end;
endfunction
////////////////////////////////////////////////

////////////////////////////////////////////////
//Display auxiliary plots
////////////////////////////////////////////////
function display_aux_plots()
    global ERROR_PLOT;
    global PCRLB_PLOT;
```

```
global xd_errors;
global xf_errors;
global pcrlb;
global time;

plot_legend = " ;

if(ERROR_PLOT == TRUE) then
    f = gca();
    scf(100002);

    drawlater();

    set(gca(), "auto_clear", "on");

    plot2d3(t, xd_errors);
    c = gca();
    p = c.children.children(1);
    p.thickness = 10;
    p.foreground = 9; //dark blue
    plot_legend = cat(1, plot_legend, 'Detection Error');
    set(gca(), "auto_clear", "off");

    plot2d(t, xf_errors);
    c = gca();
    p = c.children.children(1);
    p.line_mode = "on";
    p.mark_mode = "off";
    p.thickness = 2;
    p.foreground = 3; //green
    plot_legend = cat(1, plot_legend, 'Filter Error');

    xgrid(1);
```

```
    p = c.x_label;
    p.font_size = 3;
    xlabel('Time');
    p = c.y_label;
    p.font_size = 3;
    ylabel('Error');
    p = c.title;
    p.font_size = 4;
    title('Detection & Filter Errors versus Time');
    legend(plot_legend);
    e = gce();
    e.font_size = 2;
    c.data_bounds = [0, -1; m, 1];

    drawnow();

    sca(f);
  else
    if (get_figure_handle(100002) <> []) then
      xdel(100002);
    end;
  end;

  if(PCRLB_PLOT == TRUE & (get("filter_ui/frame_left/frame_auxiliary_plots
      /pcrlb_plot", "enable") == "on")) then
    f = gca();
    scf(100003);

    drawlater();

    set(gca(), "auto_clear", "on");

    plot2d(t, pcrlb);
    c = gca();
```

```
    p = c.children.children(1);
    p.line_mode = "on";
    p.mark_mode = "off";
    p.thickness = 2;
    p.foreground = 6; //magenta
    if(SIMULATION_TYPE == BATCH) then
       plot_legend = cat(1, plot_legend, 'NP = ' + string(PARTICLES_PF) + ', BS = '
                + string(BATCH_SIZE));
    else
       plot_legend = cat(1, plot_legend, 'NP = ' + string(PARTICLES_PF) + ', BS = 1');
    end;

    xgrid(1);
    p = c.x_label;
    p.font_size = 3;
    xlabel('Time');
    p = c.y_label;
    p.font_size = 3;
    ylabel('PCRLB');
    p = c.title;
    p.font_size = 4;
    title('PCRLB versus Time');
    legend(plot_legend);
    e = gce();
    e.font_size = 2;
    c.data_bounds = [0, 0; m, max(pcrlb)];

    drawnow();

    sca(f);
  else
    if (get_figure_handle(100003) <> []) then
       xdel(100003);
    end;
```

```
    end;
endfunction
////////////////////////////////////////////////

////////////////////////////////////////////////
//Display simulation results
////////////////////////////////////////////////
function display_results()
    global rmse_fil;
    global rmse_det;
    global time;
    global filename_results;

    msg = msprintf(...
        gettext(...
            "\nRMSE Filtered: " + string(rmse_fil * 100) + " %%" +...
            "\n" +...
            "\nRMSE Detections: " + string(rmse_det * 100) + " %%" +...
            "\n" +...
            "\nSimulation Time: " + string(time) + " s" +...
            "\n" +...
            "\nFilename: " + filename_results ...
        )...
    );
    messagebox(msg, gettext("Simulation Results"), "info", "modal");
endfunction
////////////////////////////////////////////////

////////////////////////////////////////////////
//Initialize
////////////////////////////////////////////////
function initialize()
    global t;
    global x_detections;
```

```
global x_filtered;
global xd_errors;
global xf_errors;
global pcrlb;
global rmse_fil;
global rmse_det;
global time;

t = linspace(1, m, n);
      //(start of simulation, simulation length, number of target detections in simulation length)
generate_true();
x_detections = zeros(1, m);
x_filtered = zeros(1, m);
xd_errors = zeros(1, m);
xf_errors = zeros(1, m);
pcrlb = zeros(1, m);
rmse_fil = 0;
rmse_det = 0;
time = 0;

if(DEBUG_SPIRAL_DIVE == TRUE) then
   plot2d(t, x_true);
   c = gca();
   p = c.children.children(1);
   p.line_mode = "on";
   p.mark_mode = "off";
   p.thickness = 2;
   p.foreground = 2;   //blue
   set(gca(), "auto_clear", "off");
   xgrid(1);
   p = c.x_label;
   p.font_size = 3;
   xlabel('X');
   p = c.y_label;
```

```
        p.font_size = 3;
        ylabel('Z');
        p = c.title;
        p.font_size = 4;
        title('Target Spiral-Dive External Model');
        c.data_bounds = [0, -1.5; m, 1.5];
    end;
endfunction
////////////////////////////////////////////////

initialize();
uicontrol_filter();
```

Appendix B : EBE Faculty – Assessment of Ethics in Research Projects

EBE Faculty: Assessment of Ethics in Research Projects

Any person planning to undertake research in the Faculty of Engineering and the Built Environment at the University of Cape Town is required to complete this form before collecting or analysing data. When completed it should be submitted to the supervisor (where applicable) and from there to the Head of Department. If any of the questions below have been answered YES, and the applicant is NOT a fourth year student, the Head should forward this form for approval by the Faculty EIR committee. Submit to Ms Zakiya Chikte (Zakiya.chikte@uct.ac.za), New EBE Building, Ph 021 650 5739). Students must include a copy of the completed form with the dissertation/thesis when it is submitted for examination.

Name of Principal Researcher/Student: Etienne F. Bauermeister **Department:** Electrical Engineering

If a Student: **Degree:** M.Eng (Radar) **Supervisor:** Assoc. Prof. Daniel O'Hagan

If a Research Contract indicate source of funding/sponsorship: n/a

Research Project Title: On Particle Filters in Radar Target Tracking

Overview of ethics issues in your research project:

Question 1: Is there a possibility that your research could cause harm to a third party (i.e. a person not involved in your project)?	YES	NO X
Question 2: Is your research making use of human subjects as sources of data? If your answer is YES, please complete Addendum 2.	YES	NO X
Question 3: Does your research involve the participation of or provision of services to communities? If your answer is YES, please complete Addendum 3.	YES	NO X
Question 4: If your research is sponsored, is there any potential for conflicts of interest? If your answer is YES, please complete Addendum 4.	YES	NO X

If you have answered YES to any of the above questions, please append a copy of your research proposal, as well as any interview schedules or questionnaires (Addendum 1) and please complete further addenda as appropriate.

I hereby undertake to carry out my research in such a way that

- there is no apparent legal objection to the nature or the method of research; and
- the research will not compromise staff or students or the other responsibilities of the University;
- the stated objective will be achieved, and the findings will have a high degree of validity;
- limitations and alternative interpretations will be considered;
- the findings could be subject to peer review and publicly available; and
- I will comply with the conventions of copyright and avoid any practice that would constitute plagiarism.

Signed by:

	Full name and signature	Date
Principal Researcher/Student:	Etienne F. Bauermeister	3/2/2016

This application is approved by:

Supervisor (if applicable):	Assoc. Prof Daniel W. O'Hagan	
HOD (or delegated nominee): Final authority for all assessments with NO to all questions and for all undergraduate research.		17/2/16
Chair: Faculty EIR Committee: For applicants other than undergraduate students who have answered YES to any of the above questions.		

ADDENDUM 1:
Please append a copy of the research proposal here, as well as any interview schedules or questionnaires:

ADDENDUM 2: To be completed if you answered YES to Question 2:

It is assumed that you have read the UCT Code for Research involving Human Subjects (available at http://web.uct.ac.za/depts/educate/download/uctcodeforresearchinvolvinghumansubjects.pdf) in order to be able to answer the questions in this addendum.

2.1 Does the research discriminate against participation by individuals, or differentiate between participants, on the grounds of gender, race or ethnic group, age range, religion, income, handicap, illness or any similar classification?	YES	NO
2.2 Does the research require the participation of socially or physically vulnerable people (children, aged, disabled, etc) or legally restricted groups?	YES	NO
2.3 Will you not be able to secure the informed consent of all participants in the research? (In the case of children, will you not be able to obtain the consent of their guardians or parents?)	YES	NO
2.4 Will any confidential data be collected or will identifiable records of individuals be kept?	YES	NO
2.5 In reporting on this research is there any possibility that you will not be able to keep the identities of the individuals involved anonymous?	YES	NO
2.6 Are there any foreseeable risks of physical, psychological or social harm to participants that might occur in the course of the research?	YES	NO
2.7 Does the research include making payments or giving gifts to any participants?	YES	NO

If you have answered YES to any of these questions, please describe how you plan to address these issues (append to form):

ADDENDUM 3: To be completed if you answered YES to Question 3:

3.1 Is the community expected to make decisions for, during or based on the research?	YES	NO
3.2 At the end of the research will any economic or social process be terminated or left unsupported, or equipment or facilities used in the research be recovered from the participants or community?	YES	NO
3.3 Will any service be provided at a level below the generally accepted standards?	YES	NO

If you have answered YES to any of these questions, please describe how you plan to address these issues (append to form)

ADDENDUM 4: To be completed if you answered YES to Question 4

4.1 Is there any existing or potential conflict of interest between a research sponsor, academic supervisor, other researchers or participants?	YES	NO
4.2 Will information that reveals the identity of participants be supplied to a research sponsor, other than with the permission of the individuals?	YES	NO
4.3 Does the proposed research potentially conflict with the research of any other individual or group within the University?	YES	NO

If you have answered YES to any of these questions, please describe how you plan to address these issues(append to form)