UNIVERSITY OF LIVERPOOL

Efficient Kalman Filtering and Smoothing

by

Siu Lun Yeung

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the
Faculty of Science and Engineering
Department of Electrical Engineering and Electronics

August 2021

Declaration of Authorship

I, SIU LUN YEUNG, declare that this thesis titled, 'Efficient Kalman Filtering and Smoothing' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Acknowledgements

It has been challenging and diffcult for the past four years of my pursuit of the PhD degree in University of Liverpool. However, I cannot achieve it alone without the help from many others. To name a few, I would like to thank the following people.

First, I want to thank my mentor, Dr Jeyan Thiyagalingam. He is the sole reason why I can finish this journey. Thank you for choosing me to be one of your students. I cannot be more grateful for that. Without his help, I am sure I would not have achieived a lot during this hard times. His passion and care for students has been invaluable to me. I consider myself very lucky to have him as my advisor, to be on my side when I was confused and puzzled on the road of research. I particularly admire his work ethic and dedication to his students and his work. Whenever I needed his help or guidance, he would be there for me no matter the time and place. Without his inspiration and encouragment, I really couldn't have reached such proud achievement.

Moreover, I want to thank my supervisor, Dr Angel Garcia-Fernandez, who always give me guidance and advise despite the pandemic. Not only he is dedicated and responsible to research, he is also friendly and thoughtful to students. I have learned a lot from him in terms of technical knowledge and how to get through the PhD. I appreciate his patient and gave me a lot of freedom to acheive my dream. Also, I would like to thank Dr Alexei Lisitsa and Prof Frans Coenen for the advice and help from the Independent Progress Assessment Panel (IPAP) reviews.

Finally, I would like to thank my parents, Kit-Hing Sin and Yu-Pui Yeung, their love have been one of the main fuels to help me overcome every difficulties. Their selfless and unlimited support on my education and life make me the person I am today. For that, I cannot be more thankful.

UNIVERSITY OF LIVERPOOL

Abstract

Faculty of Science and Engineering
Department of Electrical Engineering and Electronics

Doctor of Philosophy

by Siu Lun Yeung

The Kalman filter and Kalman smoother are important components in modern multi-target tracking systems. Their application are vast which include guidance, navigation and control of vehicles. On top of that, when the motion model is uncertain, Multiple-Model approach can be combined with the filtering and smoothing method. However, with large amount of retrodiction window size, number of motion models and large number of targets, this process can become very computationally intensive and thus time consuming. Very often, real-time processing is needed in the world of tracking and therefore, this computational bottleneck become a problem. This is the motivation behind this thesis, to reduce the computational complexity when multi-target, multi-window or multi-model applications are used. This thesis presents several approaches to tackle this multi-dimensional problems in terms of complexity while maintaining satisfactory precision.

A natural step forward will be in leveraging the modern multi-core architectures. However, in order to parallelize such process, these algorithms have to be reformulated to be fitted into the parallel processors. In order to parallelise multi-target and multi-window scenario, this thesis introduce nested parallelism and prefix-sum algorithm to tackle the problem and realised this on Intel Knights Landing (KNL) Processor and OpenMP memory model.

On the other hand, in the case of limited parallel resources, this thesis also develop alternatives called Fast Kalman smoother (FRTS) to lower the computation complexity due to multi-window problem. Specifically the smoother algorithm is reformulated such that it is computationally independent of number of window size in the fixed-lag configuration. Although the underlying mathematics is the same as the conventional approach, FRTS introduced numerical stability issue which makes the smoother unstable. Therefore, this thesis introduce the idea of condition number to monitor the deterioration rate in order to correct the numerical error once the pre-set threshold is breached.

In addition to the large number of targets and retrodiction window size mentioned earlier, the number of models running simultaneously make the problem even more challenging in the perspective of real-time performance. Since such algorithms are the fundamental backbone of a large amount of multi-frame tracking algorithms, it would be beneficial to have a multi-model algorithm that is computationally independent to number of model utilised. Consequently, this thesis extend the FRTS concept to fixed-lag Multiple-Model smoothing method to achieve this goal.

The proposed algorithms are compared and tested through an extensive and exhaustive set of evaluations against the literature, and discuss the relative merits. These evaluations show that these contributions pave a way to secure substantial performance gains for multi-dimensional tracking algorithms over conventional approaches.

Contents

List of Figures

List of Tables

Abbreviations

AMMF	Autonomous Multiple Model Filter
AMMS	Autonomous Multiple Model Smoother
ASIPDA	Augmented State Integrated Probabilistic Data Association
BMFLS	Biswas-Mahalanabis Fixed-Lag Smoother
BLAS	Basic Linear Algebra Subroutines
C2C	Computation-to-communication
CV	Cosntant Velocity
CT	Cosntant Turn
CNG-FRTS	Condition Number-Guided Fast Rauch-Tung-Striebel
cRTS	Conventional fixed-lag RTS
DR	Data Reuse
fRTS$^-$	Fast RTS without stability renewal module
FLOPs	Floating Point Operations
FMMF	Fast Multiple Model Filtering
FMMS	Fast MM Fixed-Lag Smoother
FRTS	Fast Rauch-Tung-Striebel
GPU	Graphical Processing Unit
GPBn	Generalized Pseudo-Bayesian n-order
GNN	Global Nearest Neighbour
IMM	Interactive Multiple Model
IMMS	Interactive Multiple Model Smoothing
IMMF	Interactive Multiple Model Filter
IPDA	Integrated Probabilitic Data Association
JPDA	Joint Probabilitic Data Association
KF	Kalman Filtering

K-FRTS	Constant Gain Fast Rauch-Tung-Striebel
KFG-FRTS	Kalman Filter-Guided Fast Rauch-Tung-Striebel
KNL	Knight Landing processor
MHT	Multiple Hypothesis Tracking
MM	Multiple Model
MI	Matrix Inverse
MVM	Matrix-Vector Multiplication
MMM	Matrix-Matrix Multiplication
MMA	Matrix-Matrix Addition
MTT	Multi Target Tracking
MTTR	Multi Target Tracking with Retrodiction
MI	Matrix Inverse
NDR	No Data Reuse
NN	Nearest Neighbour
PRTS	Parallel Rauch-Tung-Striebel
PPMHT	Point Probabilistic Multiple Hypothesis Tracking
PT,SRTS	Parallel Tracking Sequential RTS
PT,PRTS	Parallel Tracking Parallel RTS
PRTS-DR	Parallel RTS with Data Reuse
PRTS-NDR	Parallel RTS with no Data Reuse
PT	Parallel Tracking
RTS	Rauch-Tung-Striebel
RMSE	Root-mean-squared error
sIPDA	Smoothing with IPDA
SRTS-DR	Sequential RTS with Data Reuse
SRTS-NDR	Sequential RTS with No Data Reuse
SIMD	Single instruction, multiple data
SRTS	Sequential Rauch-Tung-Striebel
ST	Single-core Tracking
TPM	Transition Probability Matrix
VVA	Vector-Vector Addition
VSMM	Variable Structure Multiple Model

Dedicated to my Father

Chapter 1

Introduction

This chapter gives a brief overview of the work developed in this thesis, beginning by describing the motivations behind it in Section 1.1. Followed by Section 1.2 which discuss specific goals this thesis aims to achieve. Then, a short summary to outline the thesis structure and contributions of this PhD are presented in Section 1.3.

1.1 Motivations

The Kalman Filter is well recognised in the field of signal processing. The applications of it encompasses multiple discipline of industrial usage since the 1970s including trajectory estimation, state prediction for control or diagnosis, denoising and so on [1–10]. The basic filtering algorithm is often combined with Kalman Smoothing algorithm to offer optimal filtering performance. Although the computational complexity of the combined filtering and smoothing algorithms is often acceptable, it can become an issue if they were to be scaled across a very large number of targets. This is particularly the case with current demands in aerospace applications (such as drone surveillance) [11, 12] where there is a compelling need to perform both filtering and smoothing at real-time across a very large number of targets simultaneously. The total number of computations or the number of floating point operations carried out by these algorithms per second (known as FLOPs) can increase dramatically with the number of targets within the field of view. This multi-target tracking scenario is illustrated in Figure 1.1 with lines showing the trajectories of the targets. To guarantee the real-time utility of the filtering and smoothing algorithms, it is essential to ensure that the overall FLOPs performance or computational complexity is managed well.

The tracking procedure begins with the filtering process to estimate the state based on the prediction model and measurements. A smoother can be used to re-calculate the

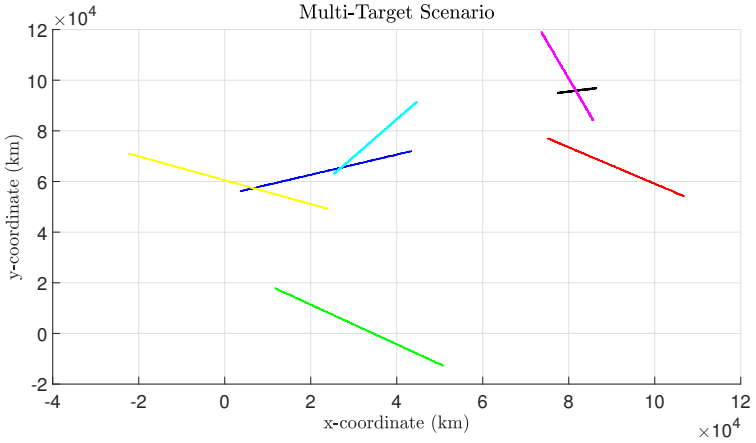

FIGURE 1.1: Multi-Target Tracking

trajectories of the targets when a small amount of delay is acceptable. Smoothing is often carried out over a set of points with past measurements, known as smoothing window size. Rauch-Tung-Striebel (RTS) smoother is one of the most widely used techniques for improving state estimation. However, the computational performance of smoothing algorithms may begin to deteriorate with a large number targets, particularly when combined with large smoother window sizes. This is rather crucial as most of the tracking systems rely on real-time or near-real-time performance. Therefore, in order to bring significant gain comparing with other smoother, the underlying mathematics of the algorithm have to be reformulated and reassembled to meet the challenge.

Furthermore, the real utility of smoothing algorithms do not end with state estimation. Instead, they have a much deeper influence on overall performance tracking systems, and influences different tracking scenarios. Among others, maneuvering targets requires a significant attention here. Nearly all practically significant tracking scenarios often involve maneuvering targets, where simple single-model based tracking method fail. In this context, Multiple Model (MM) tracking allows multiple prediction models to be executed simultaneously to better capture the appropriate, resulting motion model of targets. The usage of smoothing algorithms in conjunction with multiple-model tracking further improve the tracking accuracy of the algorithm. However, multiple model methods require a bank of filters and smoothers operating together to improve the overall accuracy which makes real-time performance even more challenging. The number of filters and smoothers utilised are proportional to the number of models used. Therefore, not only multi-target and multi-window become a problem in complexity, multi-model is also another hurdle to overcome in order to render practical tracking algorithm for real-time usage. This multi-dimensional problem is illustrated in Figure 1.2. In terms of multi-model smoothing method, in general, each of the filters need to carry out smoothing on the estimates from multiple model. On top of that, smoothing usually needs a

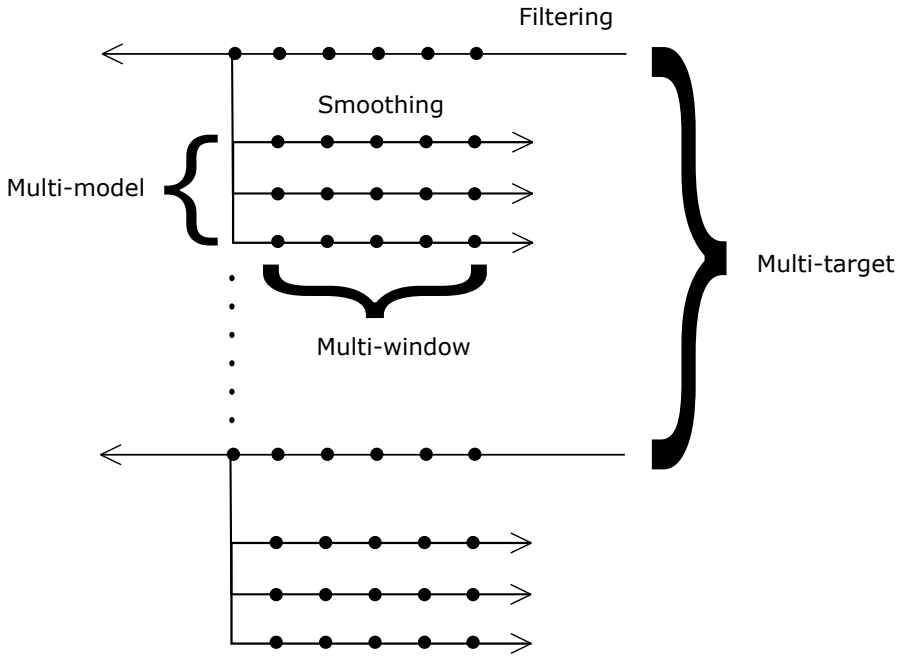

FIGURE 1.2: Problem Illustration

couple of smoothing points to lower estimation error. Furthermore, this become even more complicated when tracking multiple models are needed to be tracked. This multi-dimensional complexity problem is the main focus in this thesis.

1.2 Aims and Objectives

As outlined in Section 1.1, the multi-dimensional problem which incules multi-target, multi-window and multi-model can impede tracking algorithm to reach real-time performance. A natural and trivial way to address these issues would be to exploit parallel architecture and parallelise these algorithms. Although there have been several bodies of work addressing the parallelisation aspects of filtering or smoothing, they are often focused at a coarse-grain level, such as parallelisation across multiple sensors or across targets. This approach, opposed to fine-grained parallelism, where the focus is on parallelising algorithms at the inner level (across window size), cannot scale optimally particularly when the fine-grained parallelism aspects are fully ignored [13]. However, fine-grained parallelisation is a rather challenging endeavour for several reasons. First, it may often require substantial reformulation of the underlying algorithm to fit the architecture and memory model, particularly the current multi-core, shared memory model. Secondly, it requires careful implementations of the algorithm on modern architectures. One of the aims of this thesis is to reformulate the smoothing algorithm, in terms of target tracking, to allow easy implementation on a range of parallel architectures. Given

the non-trivial nature of this endeavour, the literature around fine-grained parallelisation on filtering and smoothing algorithms, particularly for the latter, is considerably limited. For instance, numerous approaches have been proposed to improve the performance of the filtering through various designs (which are in fact instances of parallelisation), such as multi-stage estimators [14–16], their performance have not been transformational, as discussed in Section 2.2.

For maneuvering targets, Multiple Model (MM) tracking allows multiple prediction models to be executed simultaneously to better capture the motion of them. The difference between exiting algorithms is that they fuse the estimates from individual filters differently. Moreover, the computational complexity of this tracking algorithm is largely governed by the number of targets, number of delayed time points and number of models. The most popular method for MM is Interacting Multiple Model filtering (IMM). According to the MM survey conducted in [17, 18], IMM is the most efficient and accurate MM algorithm. Therefore, several papers have tried different approaches to find a faster solution for IMM filtering, for example in [17–22]. In order to accommodate more number of models and window size in the MM algorithms without increasing the computational burden, novel approach has to be investigated by combining the existing method with new idea in order to breakthrough the barrier.

In this thesis, the following concerns are addressed, by making following three contributions:

1. First, to address the multi-target, multi-window problem, a fully parallelised filtering and smoothing algorithm targeting contemporary multi-core architectures is formulated. More specifically, the underlying mathematics are reformulated to fit the notion of multi-core parallelism and shared-memory model that underpin modern systems. The Rauch-Tung-Striebel (RTS) smoother is chosen to be the parallelised smoother, as RTS is one of the most widely used smoothing technique for improving the state estimation.

2. Secondly, without the usage of multi-core architecture, a novel algorithm, namely, the Fast RTS Smoother (FRTS), is investigated to address the multi-window problem, that is, the complexity of the algorithm is independent to number of window size. However, after the formulation, numerical instability arise as small error accumulate over number of time steps. Therefore, additional work and research is carried out to make FRTS feasible.

3. Thirdly, by using Interacting Multiple Model smoothing (IMM) algorithm as a baseline algorithm, this thesis develops a novel, fast, computationally cost effective algorithm for multiple model smoothing to address the multi-model and

multi-window complexity problem. Although there are a numerous variations of MM algorithms, IMM is the most wide used, most efficient and accurate MM algorithm [17, 18, 22]. More specifically, the proposed algorithm could accommodates a large number of models and a large range of window sizes without increasing the computational complexity.

1.3 Outline of the Thesis

As stated before, the aim of this thesis is to improve the existing state-of-the-art tracking algorithms in the aspect of filtering, smoothing and multiple model tracking. Tracking is a complex procedure which involves multiple stages in order to obtain reliable estimations. Due to the increasing complexity of tracking algorithms, it is natural to seek more efficient techniques not only can reduce the complexity while keeping RMSE within acceptable range. Therefore, in this thesis, novel tracking techniques are presented and compared with the state-of-the-art approach in order to address these problems.

Chapter 2 introduce the basic building blocks of tracking algorithm and notations for the following chapters. The main contributions of the thesis are outlined in the following Chapters.

1. The first contribution: A Parallel Retrodiction Algorithm for Large-Scale Multi-target Tracking is presented in Chapter 3. This chapter addresses the issue brought by number of targets and number of window size of smoother by introducing parallelism in both the filtering and smoothing algorithms. By reformulating the underlying algorithms as well as applying them on modern multi-core architectures, significant performance improvement can be achieved. This piece of work is also published in the following journal:

    - S. Yeung and J. Thiyagalingam, A Parallel Retrodiction Algorithm for Large-Scale Multi-target Tracking, Accepted in IEEE Transactions on Aerospace and Electronic Systems [23].

2. In Chapter 4, it presents the idea of Fast Fixed-Lag Smoothing Algorithm (FRTS). The key novelty here is the computational complexity of the algorithm being fully independent of the smoothing window size. Most existing smoothing algorithms in the literature have their computational complexity linearly coupled to the smoothing window size. This chapter also includes techniques to address the numerical instabilities that may arise due to such decoupling, and a number of guided-smoothing algorithms are proposed to address these concerns.

3. In Chapter 5, it includes different generations of Multiple Model Target tracking algorithms: the autonomous, the cooperative and the variable structure multiple models. By combining the strength of each of these techniques, a novel MM algorithm is suggested which include filtering and smoothing stage.

Chapter 6 summarizes the work presented in this thesis and outlines future research based on the work developed.

Chapter 2

Background

2.1 Introduction

Since the aim of the thesis is to develop efficient Kalman Filtering and Smoothing in the sense of multi-model usage, in this section, the basic formulation of the Kalman Filter, Kalman Smoother and the Interactive Multiple Model Smoother are covered. Reader can refer to the notations here for the following chapters. The current state-of-the-art techniques are presented in Section 2.1. Furthermore, in 2.2, this thesis include some of the work using different approaches to reduce the computation cost and reach real-time performance.

2.1.1 Problem Formulation

The purpose of tracking is to estimate the unknown state of the target, denoted as $\mathbf{x_t}$, by incorporating the information from the measurement generated from the sensor. As measurement comes in discrete moment in time, the tracking process is also performed in discrete time step. At any time step t, this thesis seek the estimate with higher accuracy than measurements which has lower estimation error. In Bayesian point of view, the posterior probability density function (pdf) is computed

$$p(x_t|Z^t)$$

$\mathbf{Z^t}$ is the measurements sequence from $t = 1$ to T. It can be a scalar value or a vector of measurements such as positions and velocities in different dimensions. Therefore, $\mathbf{Z^t}$ is a measurement set with vectors z_t written as

$$\mathbf{Z^t} = \{z_1, ..., z_t\} \tag{2.1}$$

The Kalman filter computes the minimum mean-squared error (MMSE) estimate by using the posterior pdf which is given by

$$\hat{x}_t^{MMSE} = \mathbb{E}\{x_t|Z^t\} = \int x_t p(x_t|Z^t)dx_t \tag{2.2}$$

where $\mathbb{E}$ is the expectation.

2.1.2 Linear State Space Models

A linear state space model consists of a sequence of state space process which can be given by x_t where $t = 0, 1, \ldots, T$ and $x_t \in \mathbb{R}^n$ is the state of the system at time step t with vector dimension n. The state can be updated at every time step by the following dynamic model and measurements model:

$$\begin{aligned} x_t &= A_t x_{t-1} + w_t \\ z_t &= H_t x_t + v_t \end{aligned} \tag{2.3}$$

at which A_t is the transition matrix of the dynamic model and w_t is the process noise with known covariance Q_t. Whereas, z_t is the measurement with dimension m at time t, v_t is the measurement error assumed to be a white noise process with known covariance R_t and has zero cross-correlation with the process noise. H_t is the measurement model matrix.

2.1.3 Kalman Filter

The original paper for Kalman filter can be found in [24]. Kalman filters are based on linear dynamical systems discretized in the time domain. It is the closed form solution to the Bayesian filtering equations. The dynamic and measurement models are assumed as linear Gaussian. The aim of the Kalman filter is to minimise the estimation error which is given by

$$e_t = x_t - \hat{x}_t \tag{2.4}$$

where, x_t is the true value of the state and $\hat{x}_t$ is the estimate calculated by the filter. Therefore, e_t is the error from the estimation. The associated error covariance matrix is

$$P_t = E[e_t e_t^T] = E[(x_t - \hat{x}_t)(x_t - \hat{x}_t)^T] \tag{2.5}$$

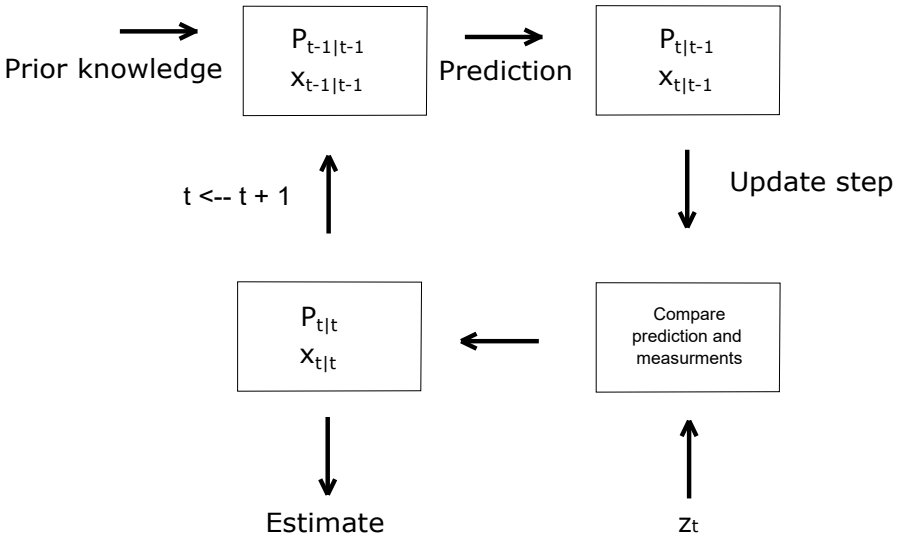

FIGURE 2.1: Kalman filter loop

With the assumption of prior estimate $x_{t|t-1}$, the Kalman filter can be divided into two steps: prediction and update, in order to calculate the distribution of x_k. The prediction step is

$$\begin{aligned} x_{t|t-1} &= A_t x_{t-1} + w_t \\ P_{t|t-1} &= A_t P_{t-1} A_t^T + Q_t \end{aligned} \tag{2.6}$$

Followed by the update step,

$$\begin{aligned} S_t &= H_t P_{t|t-1} H_t^T + R_t \\ K_t &= P_{t|t-1} H_t^T S_t^{-1} \\ x_{t|t} &= x_{t|t-1} + K_t(z_t - H_t x_{t|t-1}) \\ P_{t|t} &= P_{t|t-1} - K_t S_t K_t^T \end{aligned} \tag{2.7}$$

where S_t is the innovation matrix, K_t is the Kalman Gain and z_t is the measurement vector which can be calculated by using (2.3). The full recursive process is described in Figure 2.1

2.1.4 Kalman Smoother

The original paper for Kalman smoother can be found in [25]. In smoothing, it considers the estimation of the past rather than in the future as in the case of filtering, which can be described as $p(x_t|z_{1:T})$ (probability of x_t conditioned on $z_{1:T}$), where T is the current time step and t is the time step of interest where $T > t$. Smoothing can further reduce

the root-mean-squared error (RMSE) of the estimates combined with filtering. There are three common classifications of smoothing [26, 27], which are: Fixed-Interval Smoothing, Fixed-Lag Smoothing and Fixed-Point Smoothing. Fixed-Interval Smoothing seeks the optimal estimates at all the data points within a fixed interval, whereas Fixed-Point Smoothing repeatedly estimates the same data point in time based on future data. When a small delay can be tolerated, fixed-lag smoothing can be used to seek the estimate at a particular time lag with respect to the present estimate. This configuration of smoothing is common in target tracking as it allows real-time performance while reducing the RMSE [25]. The Rauch-Tung-Striebel Smoother, which is called the Kalman Smoother, is used to compute the closed form solution. The smoothed solution is conditioned on the whole measurement space $z_{1:T}$, whereas the filtering only considers measurements $z_{1:t}$ which is up to time t.

$$\hat{x}_{t|N} = \hat{x}_{t|t} + C_t(\hat{x}_{t+1|N} - \hat{x}_{t+1|t}) \tag{2.8}$$

with C_t being defined as the smoother gain as follows:

$$C_t = P_{t|t}A_t^T P_{t+1|t}^{-1} \tag{2.9}$$

However, $P_{t|t}$, $\hat{x}_{t|t}$ and $x_{t+1|t}$ and $P_{t+1|t}^{-1}$ are computed as part of the forward pass for the filter before the smoother process begins. Hence, the results of these computations can be saved in memory and can be reused during the backward pass, whose initial values will be $P_{t+1|N}$ and $\hat{x}_{t+1|N}$. The exact distance of smoothing from the last time step is determined by the *window size* N. In each step of the backward sweep, the old filter estimate is renewed to improve the accuracy. The smoothed error covariance matrix for the smoothed estimates is given by:

$$P_{t|N} = P_{t|t} + C_t(P_{t+1|N} - P_{t+1|t})C_t^T \tag{2.10}$$

The full recursive process is described in Figure 2.1. The number of cycles to loop depends on the window size N.

2.1.5 Motion Model

To predict target motion, a moton model can be used. Two of the most common models [17, 28] to describe target dynamics are Constant Velocity (CV) Model and Constant Turn (CT) Model. Constant Velocity Model describes non-maneuver and uniform motion with constant velocity v. The model A_{CV} can be represented by the

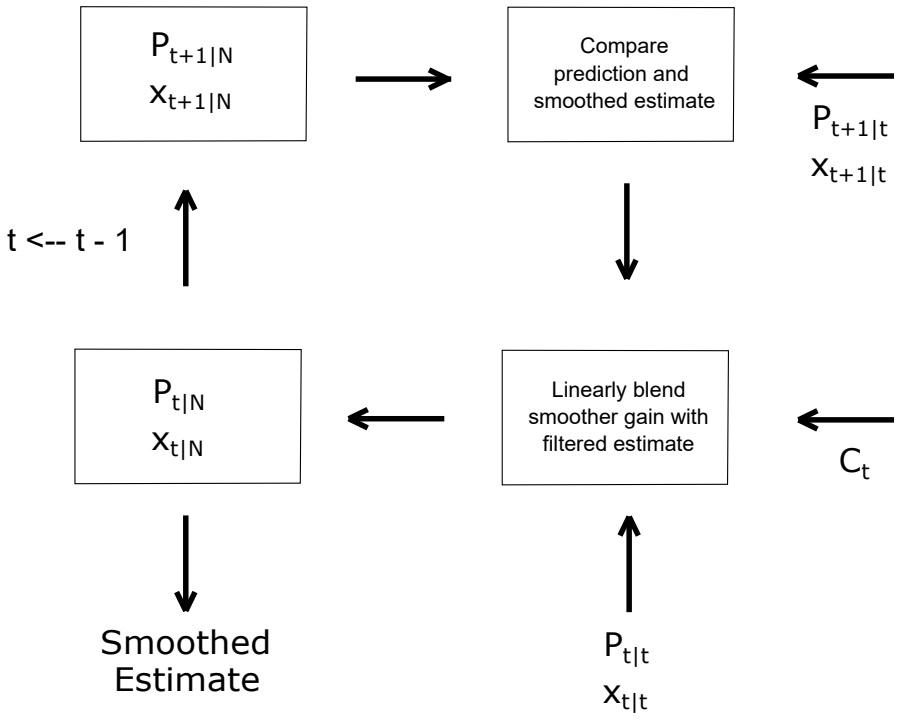

FIGURE 2.2: Kalman smoother loop

following matrix with sampling time §.

$$A_{cv} = \begin{pmatrix} 1 & 0 & § & 0 \\ 0 & 1 & 0 & § \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

On the other hand, if the target exhibits turning behaviour with constant speed v and constant turn rate ω. The following matrix A_{turn} (in 2D) can be used.

$$A_{turn} = \begin{pmatrix} 1 & 0 & \frac{sin(w§)}{w} & \frac{cos(w§)-1}{w} \\ 0 & 1 & \frac{1-cos(w§)}{w} & \frac{sin(w§)}{w} \\ 0 & 0 & cos(w§) & -sin(w§) \\ 0 & 0 & sin(w§) & cos(w§) \end{pmatrix}$$

A simple solution to track manoeuvring targets is to incorporate multiple motion models to predict target motion which leads to Multiple Model (MM) tracking. Multiple Model methods are the conventional approach in single target tracking under motion

uncertainty and in the absence of measurement origin uncertainty. Basically, these methods resolve the target motion uncertainty by using multiple models at a time for a maneuvering target. Numerous techniques have been developed to resolve the target motion uncertainty, amongst the more popular, there are the Autonomous Multiple Model [29, 30] and the Interacting Multiple Model algorithm [31].

Furthermore, all the MM approaches can be categorized into two groups: the Fixed Structure and the Variable Structure [18]. The first has received a lot of attention from the scientific community, and there is little to be improved, on the other hand, the latter is relatively new ad appeared to have an alternative to upgrade the Fixed Structure Multiple Model algorithms. In the survey conducted by Li and Jilkov [18], a comprehensive study of all the main Variable Structure Multiple Model (VSMM) algorithms is presented.

2.1.6 Interactive Multiple Model Filter

Target tracking is more difficult when the target in motion is also maneuvering which makes the motion seemingly stochastic and unpredictable. By tuning the process noise of the Kalman filter, this shortcoming can be mitigated. However, in practice, that may not be very usable since the that involves trial and error to tune the process noise. Interactive Multiple Model comprises of different motion models which can anticipate a lot more circumstances than Kalman filter. In the theis, one of the aims is to make such an algorithm efficient such that the computation is independent to number of model used. Therefore, the details of the Interactive Multiple Model Filter and its notations are provided in this section. The details can be found in [32].

The main difference IMMF compared to a conventional Kalman filter is IMMF use several possible models for the target's motion and a probabilistic switching mechanism govern the usage of these models. In this case, a bank of Kalman filters are operated in parallel (not in the sense of parallel computing), each of the filter correspond to one of the models. The information of each of the model would be exchanged or mixed in each time cycle. As a result, the overall state estimate is a combination of the state estimates from the individual filters. Consider that there are M models which is given as $m_1, ..., m_r$ such that each of the filter would have individual Markov jump-linear system

$$\begin{aligned} x_t &= A_t^{m_r} x_{t-1} + w_t^{m_r} \\ z_t &= H_t^{m_r} x_t + v_t^{m_r} \end{aligned} \tag{2.11}$$

$A_t^{m_r}$ and $H_t^{m_r}$ are the state transition matrix and observation matrix at time t with model m_r. Process noise $w_t^{m_r}$ and measurement noise $v_t^{m_r}$ are assumed to be white

Gaussian noise as well. It is assumed that the target is switched between the models according to the Markov chain with known transition matrix. This matrix consist of the probabilities of the transition from one to another where p_{ij} is the probability at which M_i model at time step t is switching over to $t+1$ and $i, j = 1, ..., M$. Each of the model has a mode probability $\mu^i_{t|t}$ which is updated in every recursion. The IMM recursion at time t are as follows

1. Mixing probabilities:

   The mixing probabilities $\mu^{i|j}_{t-1|t-1}$ are given by

   $$\begin{aligned} \mu^{i|j}_{t-1|t-1} &= \frac{1}{\bar{c}_j} p_{ij} \mu^i_{t-1|t-1} \\ \bar{c}_j &= \sum_{i=1}^{M} p_{ij} \mu^i_{t-1|t-1} \end{aligned} \tag{2.12}$$

2. Mixing:

   For j^{th} model, the mixed estimate $\hat{x}^{0j}$ and covariance matrix P^{0j} are given by

   $$\begin{aligned} \hat{x}^{0j}_{t-1|t-1} &= \sum_{i=1}^{M} \mu^{i|j}_{t-1|t-1} \hat{x}^i_{t-1|t-1} \\ P^{0j}_{t-1|t-1} &= \sum_{i=1}^{M} \mu^{i|j}_{t-1|t-1} [P^i_{t-1|t-1} + e^{oj} e^{oj^T}] \end{aligned} \tag{2.13}$$

   where $e^{0j} = \hat{x}^i_{t-1|t-1} - \hat{x}^{0j}_{t-1|t-1}$ is the mixed estimate error with each of the i^{th} filtered estimate

3. Mode-matched filtering:

   The usual Kalman filter equation from (2.6) to (2.7) can be used to update the mixed state estimates and covariance with the received measurement and corresponding motion model

4. Mode probability update:

   After the filtering step, the mode probability can also be updated by the following,

   $$\begin{aligned} \mu^j_{t|t} &= \frac{1}{c} \Lambda^j_t \bar{c}_j \\ c &= \sum_{j=1}^{M} \Lambda^j_t \bar{c}_j \end{aligned} \tag{2.14}$$

   where Λ^j_t is the j^{th} mode likelihood

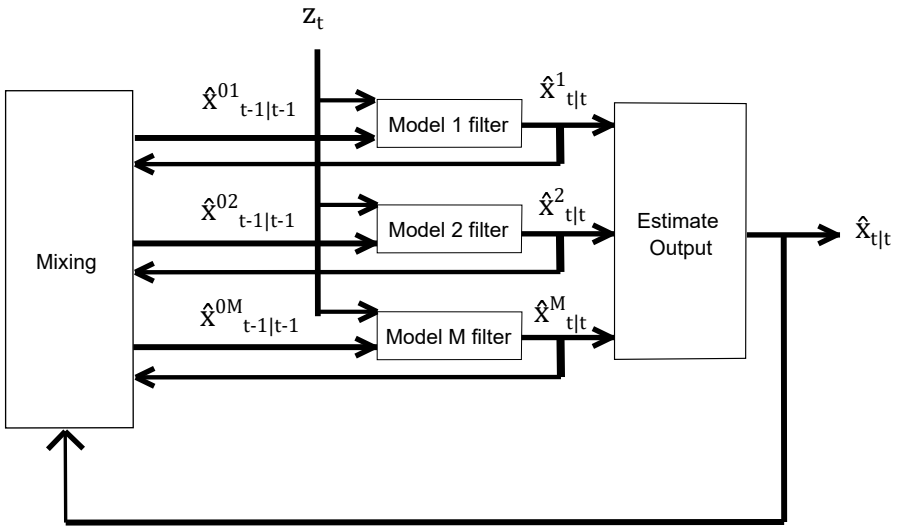

FIGURE 2.3: IMM Filter loop

5. Estimate:

   After the filtering step, the mode probability can also be updated by the following,

$$\begin{aligned} \hat{x}_{t|t} &= \sum_{i=1}^{M} \mu^{j}_{t|t} \hat{x}_{t|t} \\ P_{t|t} &= \sum_{i=1}^{M} \mu^{j}_{t|t} [P^{j}_{t|t} + e^{j} e^{j^T}] \end{aligned} \tag{2.15}$$

   where $e^j = \hat{x}^j_{t|t} - \hat{x}_{t|t}$ is the estimate error with each of the j^{th} filtered estimate and final estimate

The full recursive process is described in Figure 2.3

2.1.7 Multi-Target Tracking

Multiple Target Tracking (MTT) problem extends the single target tracking scenario to a situation where the number of targets may not be known and varies with time, leading to the presence of measurement origin uncertainty in addition to the target motion uncertainty. Ideally, the number of measurements at each time step matches with the number of target presented in the field of view. However, this rarely occur in practice. This is due to the fact that sensors have their limitations or obstruction occur and not all targets can be detected. Moreover, false measurements is possible as unwanted object can be detected because of noise or random signal reflections which this thesis refer as clutter. Besides, the resolution of the detector may not be ideal and

so multiple detection can be found on single object. The multi-target tracking problem can be formulated as follows. Assume at any time index t, there are U number of targets and V number of measurements. Each of the targets has its own state denoted as x_t and form a set X_t and Z_t such that

$$\begin{aligned} X_t &= [x_t^{(1)}, ..., x_t^{(U)}] \\ Z_t &= [z_t^{(1)}, ..., z_t^{(V)}] \end{aligned} \tag{2.16}$$

The problem of multi-target tracking is to assign each of the measurements to the existing target (target-oriented) or assign each of the target to measurement (measurement-oriented).

2.1.7.1 Gating

One of the approaches to solve the MTT problem is to assign label to each of the detected target and remain a target list ad assign new detections to the existing targets throughout the tracking process. Targets can be added or removed based on the probability model. Track initialisation is the first issue when it comes to multi-target tracking. Correct track initialisation can lower the computation due to data association. The process of Gating helps deciding if an detection is a probable candidate to be assigned to existing tracks. In other words, it screens out the false signal such as clutter which can also lower the computation burden. The region enclosed by the gate, with the predicted value as center, is called the validation region as shown in Figure 2.4. The following scenarios might occur when gating is used

1. More than one observation satisfy the gate of a track
2. One observation satisfy more than one gate of the tracks
3. Observation might be used to initiate a track even if it falls inside the gate
4. Observation fall outside of gates and form tentative tracks

To see whether a measurement satisfy the gate limit, the residual vector is defined and given by

$$v_t = z_t - H\hat{x}_{t|t-1} \tag{2.17}$$

The innovation matrix S is the same as (2.7) and given by,

$$S = HPH^T + R \tag{2.18}$$

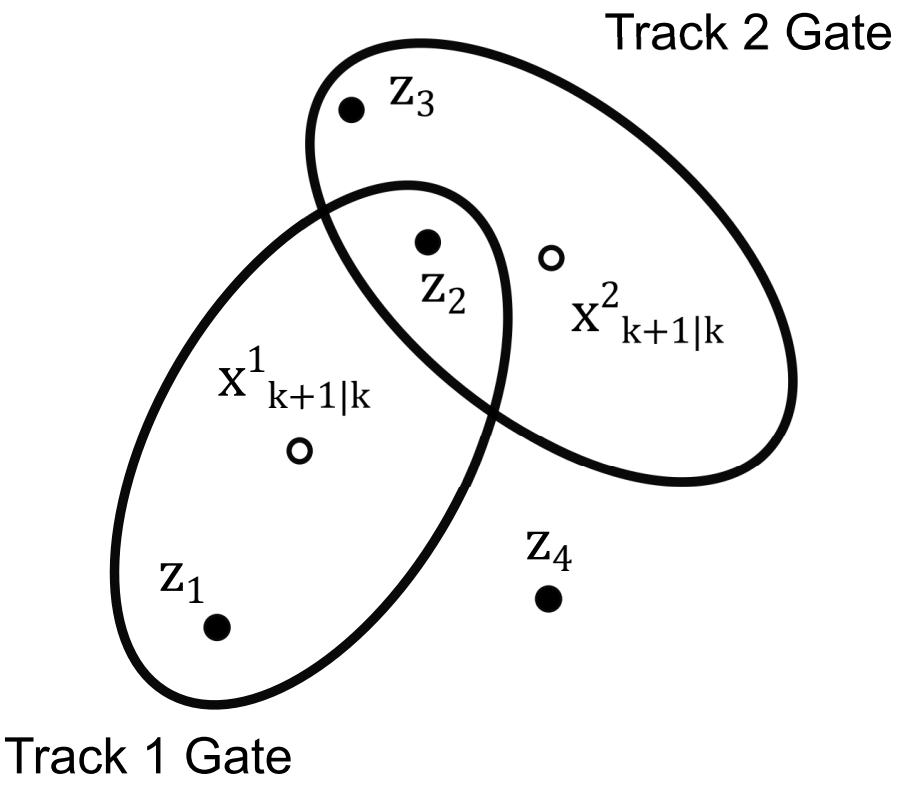

FIGURE 2.4: Gating

There is a correlation between the observation and track if the norm of the residual vector d^2 is less than a certain gate threshold G such that,

$$d^2 = {v_t}^T S^{-1} v_t \leq G \tag{2.19}$$

2.1.7.2 Nearest Neighbour Data association

The simplest way to handle data association is called nearest neighbour method (NN) which assign the closest measurement to a target after gating is performed. This operations function sequentially and so the solution usually is not globally optimal and leads to poor tracking result. Consequently, track loss occur and important information is loss. A global version of NN is the Global Nearest Neighbour method. Instead of consider tracks and measurements one by one, it computes all the possible distance combination and try to find a solution to lower the global distance. In other words, it consider targets and measurements simultaneously. This GNN approach become a optimization method which can be handled by algorithm like Auction algorithm and Hungarian methods. There are other techniques developed to resolve the measurement

origin uncertainty, such as the Joint Probabilistic Data Association (JPDA) Filter [33], and the Multiple Hypothesis Tracking (MHT) Filter [28].

2.2 Related Work and Applications

As outlined in earlier part of this chapter, the main objectives of this thesis are to develop a suite of efficient filtering and smoothing algorithms particularly targeting the real-time performance. Furthermore, as outlined before, the proposed algorithms are built on existing developments around Kalman filtering and smoothing algorithms. As both filtering and smoothing algorithms utilise matrix algebra, such as matrix multiplications and inversions, the computational complexity of filtering and smoothing algorithms are dominated by the complexity of relevant matrix operations. More specifically, the order of computational complexity is often around $O(n^3)$ [34] when multiplying two $n \times n$ matrices. Numerous efforts can be found in the literature towards reducing the computational complexity of filtering and smoothing algorithms. These can be considered in three different strands, as follows:

- Matrices properties
- Algorithm approximations
- Software and Hardware architecture

The literature around improving computational complexity of matrix algebra is very rich [35]. As such, exploring the literature around efficient matrix operations is beyond the focus of this thesis. However, it is worth highlighting the key outcomes of the survey highlighted here. One of the best performance around matrix-matrix multiplication can be achieved using efficient matrix operation algorithm such as Strassen's algorithm [36] which has complexity $O(n^{2.807})$. Also one can exploit the sparsity and symmetries property of matrices to avoid excessive calculations such as Sparse Basic Linear Algebra Subprograms [37] or Matlab.

Moreover, there have been several efforts for improving the computational cost of Kalman filtering using algorithm approximation. Friedland introduced a two-stage estimator [14]. The main idea is to separate the computations into two parallel filters, a full-order filter and the other one handles the augmented state. The computation can be reduced by using these interconnected Kalman flter. However the performance is not as good as the regular Kalman filter. There are other approaches that extend this idea such as in [15]

and in [16], Hsieh and Chen proposed a new version of this two-stage Kalman filter and is mathematically equivalent as the regular Kalman filter.

Finally, improving the performance of matrix operations through software-hardware co-design and software optimisation have been the subject of computer and computational science for several decades [35]. In terms of software and hardware architecture, there are software packages targeting matrix operations such as [38, 39]. Moreover, with the recent development of parallel architecture, the multi-sensor multi-target tracking algorithm can be parallelised and reduce computation time. There are different ways in parallelising the algorithm. For instance, the operations within the matrices can be parallelised known as fine-grained parallelism [40]. On the other hand, multiple Kalman filters can be parallelised as well known as coarse-grained parallelism [41].

Chapter 3

A Parallel Retrodiction Algorithm for Large-Scale Multitarget Tracking

3.1 Introduction

Multitarget tracking (MTT) problems arise in a number of different applications such as surveillance, control, navigation, failure detection and medicine [1–10]. In the context of surveillance and navigation, the key aim of MTT is to estimate the states of multiple targets using the measurements returned by sensors, which are inherently noisy. The fundamental cornerstone of MTT algorithms are state estimation filters [42–45], which provide a means for obtaining the best possible estimates for the target states, given a dynamical model to represent the target motion.

In general, the state estimation process works in two stages: prediction and update. In the first or the prediction stage, the state at the current time step is used to predict the state at the next time step. This predicted state is then updated (or corrected) using the measurement obtained at the subsequent time step during the second stage. Among different filters that can be used for this predict-update process, the Kalman Filter [24] is a popular basis for handling linear models and non-linear models through necessary extensions [27, 46, 47]. The key aspect here is that the update is not possible until future measurements arrive. In cases where a delay in estimation can be tolerated, or where the estimation is performed for off-line use (i.e. the whole set of measurements are already available), the estimation accuracy can be improved rather significantly by incorporating future measurements. This process, known as smoothing or retrodiction [48], produces much better and cleaner estimates. In fact, the accuracy improves with delay [49, 50],

which is often referred to as the retrodiction window size. As such, the MTT with retrodiction (MTTR) is an attractive option for producing better state estimates. By linking MTT with a smoothing algorithm of choice, the overall quality of state estimates can be improved. Among different smoothing algorithms, such as augmented fixed-lag smoothing [51, 52], Fraser-Potter smoothing [53], and Rauch-Tung-Striebel (RTS) smoothing [25], the RTS smoother is one of the most widely used, for being the fastest fixed-interval smoother [54, 55] and for its simplicity [56]. For example, in [57], the RTS algorithm is used in conjunction with Multiple Hypothesis Tracking (MHT) and the Interacting Multiple Model (IMM) algorithm for tracking maneuvering targets. In [58], fixed interval smoothing for integrated probabilistic data association (IPDA) [59] is proposed using the RTS smoother. In [60], and [61], the RTS smoother is used for backward Point Probabilistic Hypothesis Tracking (PPMHT). Given this widespread adoption of the RTS smoother, it is very common to see the RTS smoother as a backbone for MTTR algorithms.

As the RTS smoother tends to provide cleaner estimates and better tracking capability, it is natural to extend the MTT algorithms with the RTS smoother. However, the downside of this approach is that the overall process is likely to become computationally expensive. This is particularly true with increasing number of targets and smoothing window size. This can potentially limit the approach from being deployed in applications where real-time performance is required. An obvious approach would be to parallelize the MTTR algorithms, which essentially boils down to parallelizing both the MTT and the retrodiction components. The latter task is focused on a well-contained, single component such as the RTS smoother or the Fraser-Potter smoother. However, the parallelization of the MTT component requires parallelizing a number of sub-components, including the Kalman filtering and data association. With the fact that Kalman filter operations are very matrix-matrix or matrix-vector focused, parallelization at the linear algebraic-level operations is very common. For example, for parallel Kalman filters, some hardware-specific approaches are outlined in [62, 63]. More relevant work includes [64] which reformulates the Kalman filter operations in order to dismantle the data dependencies. This approach enables the necessary data to fit into a cache memory achieving a linear speed-up. In [65], a matrix operations library *matriplex* is created targeting small-sized matrices so that the operations with single instructions operating on multiple streams of data (SIMD operations) which can be optimized on vector processing units. In [66], all matrices in the Kalman filtering algorithm are transformed into a banded form achieving a linear speed-up with an increasing number of cores. In addition to these, a number of parallel MTT approaches have been proposed. For example, a parallelization scheme for IMM tracking is suggested in [67]. These are often integrated with the assignment component. For example, a coarse-grained parallel

m-best S-dimensional assignment algorithm is proposed in [68]. A similar scheduling algorithm for the IMM-assignment problem is proposed in [69] utilizing the task-level parallelization. Moreover, a comparison of sequential and parallel implementations of a multi-sensor multi-target JPDA tracking algorithm has been studied in [70]. It is reported that the complexity of the parallel approach has an exponential relationship with the number of sensors, whilst the serial (non-parallelized) method has a linear relationship. Hence, simply parallelizing the association algorithms alone does not always yield performance gain [69].

With a number of approaches to parallelize different components, as outlined above, it is possible to integrate these different parallel solutions to formulate a fully parallelized MTTR solution. However, such an approach can only offer sub-optimal benefits. There are several reasons for this. More specifically, the parallelization of the RTS smoother is a challenging problem, predominantly due to the dependencies between computations within the RTS smoother. These dependencies limit the scope for parallelization, despite having sufficient parallel resources. As such, despite the fact that MTT aspects are parallelizable, the sequential nature of the RTS smoother can severely limit the overall performance due to Amdahl's Law [71]. Secondly, the parallelization must be aimed at all levels of granularity [72], as opposed to aiming for only the fine-grained parallelism. Such an approach for parallelization is likely to bring more tangible benefits than either coarse or fine-grained parallelism alone.

In this chapter, this thesis aims to address this problem of offering a truly parallel MTTR solution, particularly that leverages RTS smoothing. By refining and reformulating the underlying mathematical framework for RTS smoothing, this thesis derive a parallel retrodiction algorithm. This thesis then apply this algorithm, along with the parallelized components of MTT, to offer a unified solution for handling large-scale multitarget tracking with retrodiction. A thorough evaluation of the proposed algorithms under various conditions show that they offers substantial performance improvements over various previously proposed methods, and can truly handle the issues associated with large-scale target tracking. The key contributions of this chapter are:

- **Reformulation of the Parallelized RTS Smoother**: By carefully refining and reformulating the underlying mathematical framework for the RTS smoother, this thesis derive an easily parallelizable RTS smoother. In particular, the proposed smoother algorithm offers a $\mathcal{O}(\log N)$ complexity (as opposed to $\mathcal{O}(N)$) upon parallelization. Furthermore, the proposed algorithm offers significant reduction on the overall number of Floating Point Operations (FLOPs) performed during the smoothing process;

- **Data Reuse**: Redundant and repeated computations often waste computational resources and directly reflect on the overall performance.This thesis propose an approach for reducing the redundant computations through extensive re-use of previously computed results for the RTS smoothing process. The benefit of this approach is directly evidenced by the significantly reduced number of FLOPs, and better runtime performance;
- **Nested Parallelism**: With a finite set of computational resources, such as parallel threads, it is always a challenge to understand the best practices for the best utilization of parallel resources. Using OpenMP as the underlying model, this thesis show how the overall MTTR problem can be treated as a hierarchical parallelization problem. To this end, this thesis introduce the notion of nested parallelism into the MTTR problem. With this in place, this thesis show how the parallel resources can be mapped between the tracking and smoothing aspects so as to maximize the overall performance;

3.2 Approach to Parallelized Multitarget Tracking with Retrodiction

In order to render an MTTR system based on RTS smoothing, the Kalman filter, algorithms for gating and measurement-to-track association and the RTS algorithm have to be parallelized. These will be covered in the section that follows this. However, any effort in parallelizing these combination of algorithms will have to account for the hardware platform on which the parallelization will be based upon. The exact choice of a hardware platform for parallelizing these algorithms may vary depending on the volume of the data being processed and on the amount of computation-to-communication ratio. Potential options are graphical processing units (GPUs), modern multi-core processors, multi-node clusters and / or a combination of these. In all these platforms, the physical unit that offers the resources for parallelism is a set of computational cores. For instance, the current generation of GPUs (Volta generation) can offer up to 2560 lightweight cores while the current multi-core processors may offer up to 64 heavy-weight physical cores. Lightweight and heavyweight cores refer to the mechanics of a multi-processing system. Heavyweight cores mean it needs more processing power to undergo parallel computing while lightweight cores can carry out such a process more easily. While the GPUs offer very lightweight cores with tightly-coupled lock-step-fashioned processing capability, the latter offers substantial processing capability with very good autonomy between cores. The cluster system, can integrate a number of GPUs or CPUs or both to provide a network of computational resources so as to provide scalability up to hundreds of thousands

of cores. The important aspect here is that the core count does not directly translate to automatic performance benefits. The exact performance improvement resulting from these platforms depends on several factors, such as:

- capability of the computational cores;
- autonomy between computational cores;
- method of interconnection between cores and the main processing unit of the system; and
- the underlying programming model and the associated complexity of the model for using these cores.

With these in place, a number of aspects have to be weighed in to decide the exact cost-benefit aspects of any platform. For instance, despite offering a very large core count, GPU cores are very lightweight. As such, the basic executing unit (known as thread or process) on GPUs can only handle a limited set of operations. On the other hand, cores from multi-core CPUs are capable of performing more complex operations. Furthermore, GPUs are an add-on device to a system (also known as accelerators) and hence, they are not part of the main system. As such, computations on GPUs have to be off-loaded along with the data on which the computation has to be performed. In contrast, CPUs are a core unit of any system. As such, no off-loading process takes place when running applications on multi-core CPUs. This directly translates to a prerequisite of computation-to-communication (C2C) ratio [73]. The C2C ratio mandates that the time savings from the computations should offset the time spent on data movements. Although different algorithms may exhibit different C2C ratios, with GPUs and a cluster of nodes, the communication time of algorithms becomes significant compared to the multi-core algorithms. To this end, the C2C ratio for MTTR algorithms is often low, given that measurements and state-vectors are small in size. In order to eliminate the extra communication, this thesis is primarily focused on the multi-core parallelization, which this thesis refer to as shared-memory parallelism, where a number of cores (processors) share a single memory and the data movements are between the processor and the main memory.

The exact programming model this thesis use to exemplify the proposed approach for parallelization is multi-threaded programing, namely the OpenMP model [74]. The basic mechanism of the OpenMP approach is that the application starts off with a single thread, running on a single core (known as the master thread), and this master thread can fork out a number of threads as required. The exact number of threads that the master thread decides to fork out may be controlled as required, by the algorithm

developer. The ratio between the number of threads and the number of physical cores is known as the subscription ratio. A subscription ratio of one indicates that a core runs a single thread. Modern hardware platforms, such as Intel Knights Landing or Skylake Processors [75] directly support the subscription radio of four, meaning that each core can easily handle four threads at any given time, without noticing any delays. Although higher values can be used, such an effort will lead to sub-optimal outcomes. In this chapter, by using one of the latest, state-of-the-art multi-core processors, Intel processor Knights Landing (KNL) [75] as the target architecture, this thesis demonstrate the performance benefits of the proposed approach. The KNL processor, has 64 heavy-weight cores, each capable of supporting a subscription ratio of four, leading to 256 threads at a time.

The next stage of parallelization is to decide where to dedicate these parallel resources (threads). This thesis proposes the following different design possibilities:

- **Parallelism across Targets**: In this case, each thread can be assigned to take care of a target, and a number of threads are assigned to a number of targets. In fact, owing to the latencies in measurements between targets, a single thread can be assigned to handle multiple targets at the same time. However, the downside of this approach is that a single thread has to handle the Kalman filtering, gating and association, and the retrodiction process. However, as will be seen later, both the filtering and the retrodiction processes are fully matrix-based, which is highly parallelizable using threads. As such, although the parallelism exists at the target tracking level, the filtering and retrodiction are highly sequential. This thesis refer to this approach as parallel tracking, sequential RTS (PT, SRTS);
- **Parallelism across Targets and Operations**: As the filtering and retrodiction processes are parallelizable themselves, another approach would be allocate the threads between two aspects: between targets and between filtering / retrodiction processes. However, the exact allocation of threads between the number of targets and matrix operations is very subjective. This thesis refer to this scheme as Parallel Tracking, Parallel RTS (PT, PRTS).

3.3 Parallelization of Components for MTTR

In this section, this thesis cover the computational and parallelization aspects of the Kalman filter, gating and measurement-to-track association and retrodiction. The parallelization of the Kalman filter is a well explored area, and more details can be found

in [63, 66, 67, 76, 77]. In discussing this aspect here, this thesis give a special emphasis on the computational aspects of the Kalman filter.

3.3.1 Parallelization of the Kalman Filter

The parallelization efforts for the Kalman filter are primarily focused on the underpinning matrix-matrix, and matrix-vector operations (collectively referred to as Basic Linear Algebra Subroutines or BLAS) [38] within the Kalman filter. Let n_s be the number of states and n_m be the size of the measurement vector. Using the standard notation for matrices and vectors in the Kalman filter [24], this thesis show all the BLAS operations of the Kalman filter, required within a single iteration (time step k), in Table 3.1, where $k = 0, ..., K$. This thesis also indicate the exact number of FLOPs required for performing each of these BLAS operations, with the acronyms MVM, MMM, MMA, VVA, and MI denoting matrix-vector multiplication, matrix-matrix multiplication, matrix-matrix addition (or subtraction), vector-vector addition (subtraction) and matrix inverse, respectively. In summary, a single iteration of the Kalman filter will require the following:

- eight matrix-matrix multiplications;
- three matrix-vector multiplications;
- three matrix additions (subtractions);
- two vector additions (subtractions);
- three matrix transpose operations; and
- one matrix inversion.

The matrix-matrix multiplication of $H_{k+1}P_{k+1}$ is repeated across two stages of the Kalman filter (covariance estimation and computing the innovation matrix). It is worth noting that these matrices remain invariant between these computations. As such, the result for the $H_{k+1}P_{k+1}$ computation can be re-used without recomputing and hence this reduces the total number of matrix-matrix multiplications to seven. Although there are three matrix transpositions, they can easily be avoided by the right indexing of the matrix (row-order as opposed to column-order or vice versa). As the state matrices are significantly smaller than cache-line capacities, this does not cause any performance issues. Hence, there are a total of 18 operations for a single iteration of KF. In general, matrix-matrix multiplication is an operation with the cubic complexity. Although it is possible to seek sub-cubic algorithms, such as Strassen [36] or Coppersmith–Winograd

TABLE 3.1: Operational Complexity of the Kalman Filter.

Operation	FLOP count	Operation
State Prediction:		
1. $x_{k+1\|k} = A_k x_{k\|k}$	$2{n_s}^2 - n_s$	MVM
Covariance Prediction:		
2. $A_k P_{k\|k}$	$2{n_s}^3 - {n_s}^2$	MMM
3. $A_k P_{k\|k} A_k^T$	$2{n_s}^3 - {n_s}^2$	MMM
4. $P_{k+1\|k} = A_k P_{k\|k} A_k^T + Q_k$	${n_s}^2$	MMA
Measurement Prediction:		
5. $z_{k+1\|k} = H_{k+1} x_{k+1\|k}$	$2n_s n_m - n_m$	MVM
6. $\hat{z}_{k+1\|k} = z_{k+1} - z_{k+1\|k}$	n_m	VVA
Innovation Covariance:		
7. $H_{k+1} P_{k+1\|k}$	$2n_m {n_s}^2 - n_s n_m$	MMM
8. $H_{k+1} P_{k+1\|k} H_{k+1}^T$	$2n_s {n_m}^2 - {n_m}^2$	MMM
9. $S_{k+1} = H_{k+1} P_{k+1\|k} H_{k+1}^T + R_{k+1}$	${n_m}^2$	MMA
Kalman Gain:		
10. $P_{k+1\|k} H_{k+1}^T$	$2{n_s}^2 n_m - n_s n_m$	MMM
11. S_{k+1}^{-1}	$\approx {n_m}^3$	MI
12. $W_{k+1} = P_{k+1\|k} H_{k+1}^T S_{k+1}^{-1}$	$2n_s {n_m}^2 - n_s n_m$	MMM
State Update:		
13. $W_{k+1} \hat{z}_{k+1\|k}$	$2n_s n_m - n_s$	MVM
14. $\hat{x}_{k+1\|k+1} = x_{k+1\|k} + W_{k+1} \hat{z}_{k+1\|k}$	n_s	VVA
Covariance Update:		
15. $W_{k+1} H_{k+1}$	$2{n_s}^2 n_m - {n_s}^2$	MMM
16. $W_{k+1} H_{k+1} P_{k+1}$	$2{n_s}^3 - {n_s}^2$	MMM
17. $P_{k+1\|k+1} = P_{k+1\|k} - W_{k+1} H_{k+1} P_{k+1}$	n_s^2	MMA

algorithms [78], not only are they very complex to implement, but maintaining their numerical stabilities is also a separate task in itself [79]. More details can be found in Section 2.2. For these reasons, they are seldom used, and hence this thesis seek only the standard (yet optimized) implementation here. As such, the overall runtime complexity of the Kalman filter is the sum of all operations shown in Table 3.1, which in turn influences the overall runtime performance [80]. However, the total runtime complexity can be approximated by the dominating runtime complexity. The total number of FLOPs in the Kalman filter is:

$$6{n_s}^3 + 6n_m {n_s}^2 + 4n_s n_m^2 + n_m^3 + n_s n_m - n_s \tag{3.1}$$

where the dominant term, namely $6{n_s}^3$, decides the overall runtime complexity to be approximated to $\mathcal{O}({n_s}^3)$. With this, the computational complexity of the Kalman filter will increase in a cubic manner as the dimensions and / or the states increase.

TABLE 3.2: Operational Complexity of the RTS smoother

Operation	FLOPs	BLAS Operation
Smoother gain:		
$P_{i\|i}A_i^T$	$2{n_s}^3 - {n_s}^2$	MMM
$P_{i+1\|i}^{-1}$	$\approx {n_s}^3$	MI
$C_i = P_{i\|i}A_i^T P_{i+1\|i}^{-1}$	$2{n_s}^3 - {n_s}^2$	MMM
Smoothed state:		
$\hat{x}_{i+1\|N} - \hat{x}_{i+1\|i}$	n_s	VVA
$C_i(\hat{x}_{i+1\|N} - \hat{x}_{i+1\|i})$	$2{n_s}^2 - n_s$	MVM
$x_{i\|N} = \hat{x}_{i+1\|i} + C_i(\hat{x}_{i+1\|N} - \hat{x}_{i+1\|i})$	n_s	VVA
Smoothed covariance:		
$P_{i+1\|N} - P_{i+1\|i}$	${n_s}^2$	MMA
$C_i(P_{i+1\|N} - P_{i+1\|i})$	$2{n_s}^3 - {n_s}^2$	MMM
$C_i(P_{i+1\|N} - P_{i+1\|i})C_i^T$	$2{n_s}^3 - {n_s}^2$	MMM
$P_{i\|i} + C_i(P_{i+1\|N} - P_{i+1\|i})C_i^T$	${n_s}^2$	MMA

3.3.2 Retrodiction

As mentioned in Section 3.1, there are a number of smoothing algorithms that can fit within the scope of MTTR. This chapter focus on the most widely adopted RTS smoother [25], also known as a two-pass smoother. In the first (or forward) pass, the estimate of the states and corresponding covariance are recursively computed. In the second (or backward) pass, the smoothed states are computed. The details of the RTS smoother has been presented in Section 2.1.4. It is often the case that the smoothing process is incorporated as part of the Kalman filtering. In other words, in every iteration of the KF, the smoother algorithm will iterate through N data points for deriving smoothed estimates. This increases the computational intensity of the overall MTTR algorithm. Table 3.2 shows the the BLAS operations and the number FLOPs associated with the RTS retrodiction algorithm.

The total number of FLOPs for the RTS smoothing algorithm with the window size of N, is thus, $N(9{n_s}^3 + n_s)$. The computational complexity of the retrodiction algorithm can be approximated by the dominating term $\mathcal{O}(n_s^3)$. However, noting that the size of n_s is usually smaller than N implies that the dominating complexity term can be affected linearly with N so that the overall complexity is $\mathcal{O}(Nn_s^3)$.

3.3.3 Measurement-to-Track Association

An MTT system tracks multiple targets using a set of measurements generated by one or more sensors. This is performed by carefully maintaining the state of tracks assigned

to each target. In this context, during a single scan, a number of measurements are returned from the sensor which must be associated to either existing tracks or new tracks to be created. There are a large number of algorithms for measurement-to-track association, each offering a different capability and performance. As such, the exact data association algorithm to be used depends on a number of factors [33, 81, 82], such as accuracy and speed. In this chapter, in order to demonstrate the approach while attaining real-time performance, this thesis choose the Nearest Neighbor (NN) [28] algorithm as the underlying algorithm for measurement-to-track association. However, this can be replaced with another association algorithm as needed, subject to the parallelization efforts towards the chosen algorithm. With a diverse set of association algorithms, it is challenging to present a generic parallelized approach for association without going beyond the scope of this chapter — parallel retrodiction. The approximated computational complexity would be $O(n^3)$ as matrix operations are involved.

In terms of nearest neighbor data assocation, let there be M_t measurements and B_t targets at the t-th time-step. Table 3.3 shows the BLAS operations, and the number of FLOPs for each measurement j returned during the t-th time-step. In other words, these operations are repeated M_t times such that $j = 0, \ldots, M_t - 1$ to find the observation that has the closest distance to the predicted state. For simplicity, the measurement covariance R_{t+1} is considered the same for all measurements.

3.3.4 Computational Complexity of MTTR

The overall computational complexity of the MTTR pipeline is the sum of individual complexities. However, asymptotically, only the dominating components of the complexity matters. If assumed that the M_t and B_t remains almost constant, i.e. $M = M_t$ and $B = B_t$, the total number of FLOPs for the MTTR pipeline (covering KF, RTS smoothing and association) can be approximated to:

$$\begin{aligned} B[(9N+6){n_s}^3 + 6n_m{n_s}^2 + (4n_m^2 + N)n_s + n_m^3 \\ + n_s n_m - n_s + M(2{n_m}^2 + 2n_m - 1)] \end{aligned} \tag{3.2}$$

The dominating component of the computational complexity for measurement-to-track association is $\mathcal{O}(n_m^2)$. Hence, the overall computational complexity for the MTTR is still $\mathcal{O}(BN{n_s}^3)$.

TABLE 3.3: Operational Complexity of the NN

Operation	FLOPs	BLAS Operation
Nearest Neighbor:		
1. $v_{t+1,j} = z_{t+1,j} - z_{t+1\|t}$	$M(n_m)$	VVA
2. $v_{t+1,j}^T S_{t+1}^{-1}$	$M({2n_m}^2 - n_m)$	MVM
3. $d_j^2 = v_{t+1,j}^T S_{t+1}^{-1} v_{t+1,j}$	$M(2n_m - 1)$	MVM

3.4 Parallel RTS Retrodiction

There have been attempts in the past to parallelize the retrodiction process, particularly the RTS smoother [83, 84]. In [83], the RTS smoother is combined with the Mayne-Fraser two-filter smoother [85] to form an approach for parallel smoothing. Their approach is to partition the target smoothing window into sub-intervals, and perform parallel smoothing on each of those sub-intervals before combining them. More specifically, their algorithm consists of three steps. In the first step, the interval to be smoothed is partitioned into a number of sub-intervals, and each sub-interval is further divided into two halves so that right half of the interval will be forward filtered while the left half of the interval is backward smoothed. This is done in parallel, and across all intervals with little or no communication. Then at the point where a forward filter and a backward smoother from two adjacent sub-intervals meet, they employ the Mayne-Fraser smoother [85] to obtain the smoothed estimate of the common point. In the third, and final stage, two separate RTS smoothers will be used (one forward and backward), in parallel, to calculate the smoothed estimate of each point within sub-intervals. Although this approach is different to earlier approaches, the smoothing is done in two stages: local smoothing and global smoothing.

Algorithm 1 RTS Fixed-Interval Smoother - Naive Version (Sequential).

for `k=0; k<K; k++` **do**
  ▷ Forward Pass: Kalman Filter and NN
  $[P_{k+1|k+1}, P_{k+1|k}, \hat{x}_{k+1|k+1}, \hat{x}_{k+1|k}] =$
    $KF[\hat{x}_{k|k}, P_{k|k}]$
  ▷ Backward Pass: Sequential RTS Smoothing
  for `i=N-1; i>=0; i--` **do**
    $C_i \leftarrow P_{i|i} F^T P_{i+1|i}^{-1}$
    $\hat{x}_{i|N} \leftarrow \hat{x}_{i|i} + C_i(\hat{x}_{i+1|N} - \hat{x}_{i+1|i})$
    $P_{i|N} \leftarrow P_{i|i} + C_i(P_{i+1|N} - P_{i+1|i})C_i^T$
    $\hat{x}_{i+1|N} \leftarrow \hat{x}_{i|N}$
    $P_{i+1|N} \leftarrow P_{i|N}$
  end for
end for

In contrast, the algorithm proposed in this chapter is significantly different in many aspects. First, the proposed parallel algorithm does not require local and global smoothing. Secondly, this thesis exploit the operational characteristics of prefix sum [86] to build a reduction tree, which can be realized efficiently on a parallel system, to find the final smoothed estimate. As will be demonstrated below, this approach is more straightforward to implement and efficient enough to achieve a much reduced computational complexity. Let N be the window size of the smoother, such that $i = 0, 1, \ldots, N-1$. With this notion, the basic, sequential, naive version of the RTS fixed-interval smoother, for a single target, is presented in Algorithm 1. This thesis refer to this version as the sequential RTS smoother (SRTS) in the following sections.

3.4.1 Enabling the Parallelization of the RTS Smoother

The basic requirement for parallelizing two statements or computations in an algorithm is that both of them are independent of each other. In other words, one of these computations should not rely on the result of the other, which this thesis refer to as dependencies. The original version of the RTS smoother presented in Algorithm 1 has a number of dependencies between computations, that inhibit straightforward parallelization. These are:

- **Loop-Carried Dependencies**: There are dependencies that are carried by the inner loop (`for i=`). More specifically, computations towards $\hat{x}_{i+1|N}$ and $P_{i+1|N}$ depend on the result from the previous iteration, for any value of i.
- **Intra-Loop Dependencies**: One or more computations within the inner-loop are coupled to the results from preceding statements (from the same iteration of the loop). For instance, when considering the computations in statements 7–11, there are a number of dependencies (Statements 8 and 9 depend on 7; 10 on 8; 11 on 9 and alike).

These two dependencies prevent both the loop (`for k`) and the statements within the loop from being parallelized. There is no direct approach to handle this problem. To reveal the potential opportunity for parallelization, this thesis *unroll* the inner loop. To reveal the parallelism, consider Equation 2.8, where the state estimates, $\hat{x}_{i+1|N}$, are performed recursively. Let b_i be:

$$b_i = \hat{x}_{i|i} - C_i \hat{x}_{i+1|i} \tag{3.3}$$

For a window size of N, the first iteration of the RTS smoother should provide:

$$\begin{aligned} \hat{x}_{N-1|N} &= \hat{x}_{N-1|N-1} + C_{N-1}(\hat{x}_{N|N} - \hat{x}_{N|N-1}) \\ &= b_{N-1} + C_{N-1}\hat{x}_{N|N} \end{aligned} \tag{3.4}$$

The second iteration should lead to:

$$\begin{aligned} \hat{x}_{N-2|N} &= \hat{x}_{N-2|N-2} + C_{N-2}(\hat{x}_{N-1|N} - \hat{x}_{N-1|N-2}) \\ &= \hat{x}_{N-2|N-2} \\ &+ C_{N-2}\Big((b_{N-1} + C_{N-1}\hat{x}_{N|N}) - \hat{x}_{N-1|N-2}\Big) \\ &= \hat{x}_{N-2|N-2} - C_{N-2}\hat{x}_{N-1|N-2} + C_{N-2}b_{N-1} \\ &+ C_{N-2}C_{N-1}\hat{x}_{N|N} \\ &= b_{N-2} + C_{N-2}b_{N-1} + C_{N-2}C_{N-1}\hat{x}_{N|N} \end{aligned} \tag{3.5}$$

Going through the last (N-th) iteration should render the following expression:

$$\begin{aligned} \hat{x}_{0|N} &= (b_0 + C_0 b_1) + (C_0 C_1)(b_2 + C_2 b_3) + \\ &\quad [C_0 C_1 C_2 C_3] \\ &\qquad [(b_4 + C_4 b_5) + (C_4 C_5)(b_6 + C_6 b_7)] + \\ &\dots \\ &\quad (C_0 C_1 \dots C_{N-2} C_{N-1})\hat{x}_{N|N} \end{aligned} \tag{3.6}$$

Equation 3.6 shows no obvious recursive nature or dependencies between the terms of the equation. Therefore, each of them can be processed independently. A closer look at the same equation may appear to have an implicit dependency. For instance, it may be misinterpreted that the computation of $C_0 C_1 \dots C_i$ has a loop-carried dependency. However, although it is true that to compute C_i (for some i), all terms up to C_{i-1} are needed, these terms can be computed during Kalman filtering (Statement 3), and hence these terms can be precomputed. A similar argument can be put forward towards computing the terms b_i. More specifically, although the individual terms are available as part of the Kalman filter iteration, the overall terms of the form

$$\alpha_q = \alpha_{q-1} \oplus \beta x_q \tag{3.7}$$

for $q = 0, 1, \dots, n \in \mathbf{N}$ and for some operator $\oplus$ (multiply, divide, plus or minus), tend to indicate that there are dependencies. However, Equation 3.7 is in fact a known pattern of parallelism, which can be parallelized across a number of communicating processors

by divide-and-conquer [87–89], which will be discussed in the section that follows this. With this pattern being exploited, by eliminating the dependency issues contained in the inner loop by unrolling, and by back-substituting the smoother gains in each iteration of the retrodiction, the smoothing operation can be parallelized.

The smoothed covariance, $P_{i|N}$, is useful for assessing the performance of a smoother. However, it is an auxiliary data that does not provide any state-specific information. As such, it is generally ignored by smoothers. However, there are association algorithms, for example [58], that require the smoothed covariance for better data association. In such circumstances, parallelization of the smoother without having an approach for parallelizing the smoothed covariance is less appealing. To overcome this issue, this thesis also present the parallelization of smoothed covariance. Let D_i be:

$$D_i = P_{i|i} - C_i P_{i+1|i} C_i^T \tag{3.8}$$

For the smoother of window size N, the smoothed covariance at the first iteration will be:

$$\begin{aligned} P_{N-1|N} &= P_{N-1|N-1} + C_{N-1}(P_{N|N} - P_{N|N-1})C_{N-1}^T \\ &= D_{N-1} + C_{N-1} P_{N|N} C_{N-1}^T \end{aligned} \tag{3.9}$$

By successively back-substituting corresponding values of D_k, at the last iteration,

$$\begin{aligned} P_{0|N} &= (D_0 + C_0 D_0 C_0^T) \\ &+ (C_0 C_1)(D_2 + C_2 D_3 C_2^T)(C_0 C_1)^T \\ &+ [C_0 C_1 C_2 C_3][(D_4 + C_4 D_5 C_4^T) + (C_4 C_5)(D_6 \\ &+ C_6 D_7 C_6^T)(C_5^T C_4^T)][C_3 C_2 C_1 C_0]^T \\ \dots & \\ &+ [C_0 ... C_{N-1}] P_{N|N} [C_{N-1} ... C_0]^T \end{aligned} \tag{3.10}$$

The smoothed covariance can also be computed in parallel, similar to that of Equation 3.7, given that the values of D_i can be precomputed and the compositions of the smoothed covariance $P_{0|N}$ can be processed independently. And hence, any dependencies within the RTS smoother towards computing the smoothed states and the smoothed covariance matrices have fully been eliminated. As such, the RTS smoother is ready to be parallelized. This thesis outline this approach in the next section.

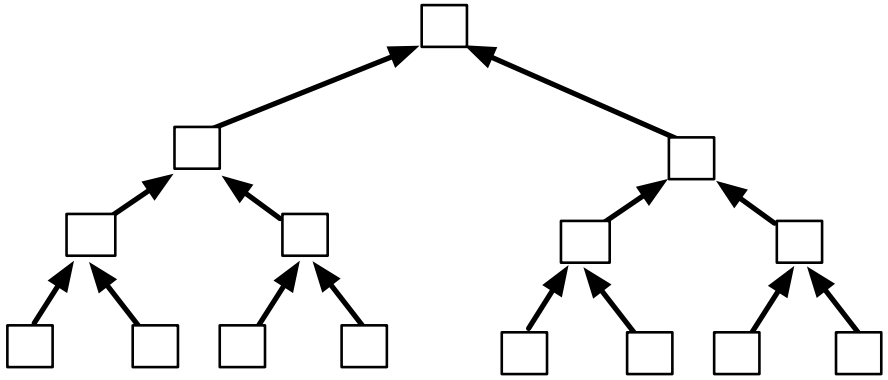

FIGURE 3.1: Reduction on n elements using p processors using the Binary Tree.

3.4.2 Parallelizing the RTS Smoothing with the Prefix-Sum Algorithm

The aim of Equation 3.7 is to sum a numbers of vectors together to obtain a final vector sum which can be represented as in Figure 3.1. Suppose there are 8 elements or vectors in the beginning as shown in the Figure. 4 processors are used to add elements according to the tree and produce 4 vectors. Then, 2 processors will be used to produce 2 vectors which subsequently produce the final vector sum without explicit communication between cores in each stage. In general, this vector sum problem can be represented in the following summation sequence,

$$\mathbf{y} = \sum_{i=0}^{n} y_i \tag{3.11}$$

for $i = 0, \ldots, e$ where e is the number of vectors and p is the number of processors. With $p(< e)$ processors in the system, each will have $\frac{e}{p}$ partition of the vector $\mathbf{y}$. Thus, each processor can sum the $\frac{e}{p}$ section of the vector $\mathbf{y}$ without any communication with any of the other processors. Once this is done, each processor can pair-up with another, to sum up their local partitions, and this can be repeated. In other words, the overall reduction operation can be performed using a binary tree as shown in Figure 3.1. Hence, the reduction operation can be performed in parallel simply by using $p = n/2$ processors. More specifically, there will be $\log n$ operations. This relies on the fact that the reduction operation is associative. This is referred to as *up-sweep*.

In this case, the operands of the prefix operations are not primitive types. Instead, they themselves are vectors and matrices. This renders the operation non-associative as for primitive types. Hence, retaining the order of operations is essential for delivering numerically correct results. As discussed in the previous section, the values for b_i, C_i and D_i can be computed during the KF process for all values of $i = 0, \ldots, N-1$. With this in

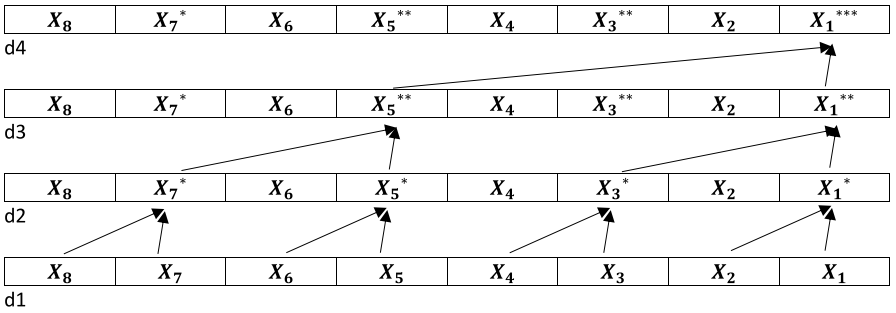

FIGURE 3.2: Prefix-Sum on b_i, C_i, D_i.

place, Figure 3.2 shows the prefix operation for this context, where $x \in X = \{b, C, D\}$. The exact operation (whether product or sum) varies depending on the operand. More specifically, this thesis modify values *in-place*, implying that when any of x is updated, this thesis update the current value with the new value. The values of x can be updated in any of the levels in the binary tree. To distinguish the updated value from the original value, this thesis denote the updated value as x^{d*}, where d represents the level at which the value was last updated. For instance, C_i^{2*} or C_i^{**} indicates that C_i has been last updated in the second level of the reduction process. More specifically, in the example shown in Figure 3.2,

$$C_i^* = C_i C_{i+1} \tag{3.12}$$
$$C_i^{**} = C_i^* C_{i+2}^* \tag{3.13}$$
$$C_i^{***} = C_i^{**} C_{i+4}^{**} \tag{3.14}$$

for updates at levels $d1$, $d2$ and $d3$. Similarly, the updates for b_i at the same levels are:

$$b_i^* = b_i + C_i b_{i+1} \tag{3.15}$$
$$b_i^{**} = b_i^* + C_i^* b_{i+2}^* \tag{3.16}$$
$$b_i^{***} = b_i^{**} + C_i^{**} b_{i+4}^{**} \tag{3.17}$$

With these, Equation 3.6 can now be re-written as

$$\begin{aligned} \hat{x}_{0|N} &= b_0^* + C_0^* b_2^* + C_0^{**}(b_4^* + C_4^* b_6^*) + C_0^{***} \hat{x}_{8|N} \\ &= b_0^{**} + C_0^{**} b_4^{**} + C_0^{***} \hat{x}_{8|N} \\ &= b_0^{***} + C_0^{***} \hat{x}_{8|N} \end{aligned} \tag{3.18}$$

Along the same argument, let $C_i'^{d*}$ represent the *in-place* updated smoothed covariance $P_{i|N}$. That is, $C_i'^{*} = C_{i+1} \oplus C_i$. Note that the order of operands is different to that of the above to preserve the correctness of the operation. With this, updates of the $P_{i|N}$ and D_i at levels $d1$, $d2$ and $d3$ become,

$$C_i'^{*} = C_{i+1}C_i \tag{3.19}$$
$$C_i'^{**} = C_{i+2}'^{*}C_i'^{*} \tag{3.20}$$
$$C_i'^{***} = C_{i+4}'^{**}C_i'^{**} \tag{3.21}$$

and

$$D_i^{*} = D_i + C_i D_{i+1} C_i^T \tag{3.22}$$
$$D_i^{**} = D_i^{*} + C_i^{*} D_{i+2}^{*} C_i'^{*T} \tag{3.23}$$
$$D_i^{***} = D_i^{**} + C_i^{**} D_{i+4}^{**} C_i'^{**T} \tag{3.24}$$

With these, Equation 3.10 can now be re-written as:

$$\begin{aligned} P_{0|N} &= D_0^{*} + C_0^{*} D_2^{*} C_0'^{*T} + C_0^{**}(D_4^{*} + C_4^{*} D_6^{*}) C_0'^{**T} \\ &+ C_0^{***} P_{8|N} C_0'^{***T} \\ &= D_0^{**} + C_0^{**} D_4^{**} C_0'^{**T} + C_0^{***} P_{8|N} C_0'^{***T} \\ &= D_0^{***} + C_0^{***} P_{8|N} C_0'^{***T} \end{aligned} \tag{3.25}$$

In summary, Equations 3.18 and 3.25 can be evaluated in parallel using p processors provided that the necessary values have been precomputed during the Kalman filtering process. In the next section, this thesis outline the aspect of data-reuse, which aims to eliminate unnecessary pre-computations during Kalman filtering.

3.4.3 Data Reuse and Performance

The parallelization of the RTS smoothing algorithm relies on the fact that certain values have been pre-computed during the Kalman filtering and are available prior to parallelization. Although pre-computing the values of $P_{i|i}$, $\hat{x}_{i|i}$, $x_{i+1|i}$ and $P_{i+1|i}^{-1}$ is trivial, the availability of these values can only be guaranteed by storing them in a data structure for later consumption. This leads to re-calculation of C_i from Equation 2.9 in every iteration of the smoother. In [26], it has been reported that the calculation of C_i can be reused. However, the idea of data reuse is extended in this chapter to the terms b_i and D_i. A much better approach is derived here to store the computed results of b_i, C_i and

D_i which can drastically reduce the number of redundant operations. This approach is, however, valid when no individual values are required for any other purposes or no reconstructed values are needed at latter stage. In the case of a smoother, this is very true. The sequential RTS smoothing algorithm with data reuse is shown in Algorithm 2. With this, the proposed algorithm for time index $k = 0, ..., K$ is presented in Algorithm 3. The algorithm uses an auxiliary function `genericRTS` for handling non-power-of-two window sizes. A detailed analysis of this algorithm is performed in the following sub-sections.

Algorithm 2 RTS Fixed Interval Smoother - With Data Reuse (Sequential)

for `k=0; k<K; k++` **do**
  ▷ Forward Pass: Kalman Filter and NN
  $[P_{k+1|k+1}, P_{k+1|k}, \hat{x}_{k+1|k+1}, \hat{x}_{k+1|k}] =$
    $KF[\hat{x}_{k|k}, P_{k|k}]$
  $C_k \leftarrow P_{k|k} A^T P_{k+1|k}^{-1}$
  $b_k \leftarrow \hat{x}_{k|k} - C_k \hat{x}_{k+1|N}$
  $D_k \leftarrow P_{k|k} - C_k P_{k+1|k} C_k^T$
  ▷ Backward Pass: Sequential RTS Smoothing
  for `i=N-1; i>=0; i--` **do**
    $\hat{x}_{i|N} \leftarrow b_i + C_i \hat{x}_{i+1|N}$
    $P_{i|N} \leftarrow D_i + C_i P_{i+1|N} C_i^T$
    $\hat{x}_{i+1|N} \leftarrow \hat{x}_{i|N}$
    $P_{i+1|N} \leftarrow P_{i|N}$
  end for
end for

3.4.4 Complexity of the Proposed Parallel RTS Algorithm

Since the performance gains from the proposed algorithm stems from both the parallelization and from extensive data re-use, this thesis compare their FLOP counts. For a single target tracking scenario with a window size of N, the number of FLOPs of the sequential RTS smoother with data re-use embedded is shown in Table 3.4. The approximate total number of FLOPs is:

$$(4N + 9){n_s}^3 - 2{n_s}^2 \tag{3.26}$$

It can be seen that the overall computational complexity is still approximately $\mathcal{O}(N{n_s}^3)$. After leveraging the reuse of previous results, the RTS smoother can be parallelized to yield additional performance gain. The number of FLOPs for the parallelized RTS smoothing algorithm (including the data reuse) is shown in Table 3.5. From this, the approximate number of total FLOPs is:

$$(13 + 6 \log N){n_s}^3 + {n_s}^2 - n_s \tag{3.27}$$

Algorithm 3 RTS Fixed Interval Smoother -
Proposed Parallelized Algorithm.

for `k=0; k<K; k++` **do**
  Each index h will be executed in parallel
  for `h=0; h<T; h++` **do**
    ▷ Forward Pass: Kalman Filter and NN
    ▷ Parallelized Backward Pass: RTS Smoothing
    if N is power of 2 **then**
      for `e=0; e<` $\log N$`; e++` **do**
        $q \leftarrow 2^{\log N - e}$
        $s \leftarrow N/q$
        Each index p will be executed in parallel
        for `p=0; p <` $q/2$`; p++` **do**
          $u_{p,h} \leftarrow p \times s \times 2$
          $w_{p,h} \leftarrow s + u_{p,h}$
          $D_{k,u_{p,h}} \leftarrow D_{k,u_{p,h}} +$
            $C_{k,u_{p,h}} D_{k,w_{p,h}} C'^{T}_{k,u_{p,h}}$
          $C'_{k,u_{p,h}} \leftarrow C_{k,w_{p,h}} C_{k,u_{p,h}}$
          $b_{k,u_{p,h}} \leftarrow b_{k,u_{p,h}} + C_{k,u_{p,h}} b_{k,w_{p,h}}$
          $C_{k,u_{p,h}} \leftarrow C_{k,u_{p,h}} C_{k,w_{p,h}}$
        end for
      end for
      $\hat{x}_{0|N,h} \leftarrow b_{k,h,0} + C_{k,h,0} \hat{x}_{k+1|N,h}$
      $P_{0|N,h} \leftarrow D_{k,h,0} + C_{k,h,0} P_{k+1|N,h} C'^{T}_{k,h,0}$
    else
      $[\hat{x}_{0|N,h}, P_{0|N,h}] \leftarrow$
        `genericRTS`$(b_{k,h}, C_{k,h}, D_{k,h})$
    end if
  end for
end for

The overall computational complexity is approximately $\mathcal{O}(\log N {n_s}^3)$. Here, the $\log N$ term signifies the reduction operation. By comparing the sequential RTS smoother (SRTS), sequential RTS smoother with data reuse (SRTS-DR) and parallel RTS smoother with data reuse (PRTS-DR), a number of observations can be drawn. Most importantly,

1. A large number of computational aspects have been parallelized. More specifically, computations at line 6 of Algorithm 1 (the calculations of C_i) are hoisted within the Kalman filtering and saved up sequentially during the forward sweep for re-use. The same is applicable to b_i and D_i.

2. The new algorithm offers a substantial reduction in the number of equivalent FLOPs, and this reduction increases with the window size N. Let F_N^S and F_N^P be the overall number of FLOPS in the sequential and parallel version of the RTS smoother. The theoretical gain g in terms of FLOPs by parallelization is given by

TABLE 3.4: Computational Complexity of the Proposed RTS Smoothing Algorithm with Data Reuse.

Operation	Equivalent FLOP	BLAS Operation
Smoother gain C_k:		
$P_{k\|k}A_k^T$	$2{n_s}^3 - {n_s}^2$	MMM
$P_{k+1\|k}^{-1}$	$\approx {n_s}^3$	MI
$C_k = P_{k\|k}A_k^T P_{k+1\|k}^{-1}$	$2{n_s}^3 - {n_s}^2$	MMM
Calculation of b_k:		
$C_k\hat{x}_{k+1\|k}$	$2{n_s}^2 - n_s$	MVM
$b_k = \hat{x}_{k\|k} - C_k\hat{x}_{k+1\|k}$	n_s	VVA
Calculation of D_k:		
$C_k P_{k+1\|k}$	$2{n_s}^3 - {n_s}^2$	MMM
$C_k P_{k+1\|k}C_k^T$	$2{n_s}^3 - {n_s}^2$	MMM
$D_k = P_{k\|k} + C_k P_{k+1\|k}C_k^T$	${n_s}^2$	MMA
Sequential RTS Smoothing with Data Reuse Repeat N times		
$C_i\hat{x}_{i+1\|N}$	$2{n_s}^2 - n_s$	MVM
$\hat{x}_{i\|N} = b_i + C_i\hat{x}_{i+1\|N}$	n_s	VVA
$C_i P_{i+1\|N}$	$2{n_s}^3 - {n_s}^2$	MMM
$C_i P_{i+1\|N}C_i^T$	$2{n_s}^3 - {n_s}^2$	MMM
$P_{i\|N} = D_i + C_i P_{i+1\|N}C_i^T$	${n_s}^2$	MMA

$$g_F = \frac{F_N^S}{F_N^P}$$

3. In order to parallelize and exploit the data reuse within the algorithm, additional steps are needed. Therefore, for low window sizes, SRTS, SRTS-DR and PRTS-DR may not show any relative benefits. For instance, for $N = 1$, the FLOP count for SRTS, SRTS-DR and PRTS-DR are 1950, 2736 and 2838, respectively. However, as N increases, the benefit of SRTS-DR and PRTS-DR can be visualized, particularly for problems with a large number of states, as shown in Figure 3.3. The speed-ups of different RTS smoother versions are shown in Figure 3.4. It can be observed that both parallelization and extensive data reuse result in substantial performance gains. As the window size, N, increases, the gains become much more significant, which will also be demonstrated in Section 3.5. Also, a graph of window size of 8 and 64 is plotted in Figure 3.5 to show the speed-up of algorithm when the number of states change. It shows that number of states is independent to speed-up of algorithm.

4. The overall FLOP count can further reduced by combining the filtering and measurement-to-track association, if desired. If this is performed, the overall number of FLOPs

TABLE 3.5: Complexity of the Proposed RTS Smoothing Algorithm.

Operation	Equivalent FLOP	BLAS Operation
Smoother gain C_k:		
$P_{k\|k}A_k^T$	$2n_s^3 - n_s^2$	MMM
$P_{k+1\|k}^{-1}$	$\approx n_s^3$	MI
$C_k = P_{k\|k}A_k^T P_{k+1\|k}^{-1}$	$2n_s^3 - n_s^2$	MMM
Calculation of b_k:		
$C_k\hat{x}_{k+1\|k}$	$2n_s^2 - n_s$	MVM
$b_k = \hat{x}_{k\|k} - C_k\hat{x}_{k+1\|k}$	n_s	VVA
Calculation of D_k:		
$C_k P_{k+1\|k}$	$2n_s^3 - n_s^2$	MMM
$C_k P_{k+1\|k}C_k^T$	$2n_s^3 - n_s^2$	MMM
$D_k = P_{k\|k} + C_k P_{k+1\|k}C_k^T$	n_s^2	MMA
Smoother Parallel region start		
Repeat $\log N$ **times**		
$C_u = C_u C_w$	$2n_s^3 - n_s^2$	MMM
$C_u b_w$	$2n_s^2 - n_s$	MMM
$b_u = b_u + C_u b_w$	n_s	VVA
$C_u^{'} = C_w C_u$	$2n_s^3 - n_s^2$	MMM
$C_u D_w$	$2n_s n_d^3 - {n_s}^2$	MMM
$C_u D_w C_u^{'T}$	$2{n_s}^3 - {n_s}^2$	MMM
$D_u = D_u + C_u D_w C_u^{'T}$	${n_s}^2$	MMA
Smoother Parallel region end		
Smoothed state		
$C_0\hat{x}_{k+1\|N}$	$2{n_s}^2 - n_s$	MVM
$\hat{x}_{k\|N} = b_0 + C_0\hat{x}_{k+1\|N}$	n_s	VVA
Smoothed covariance		
$C_0 P_{k+1\|N}$	$2{n_s}^3 - {n_s}^2$	MMM
$C_0 P_{k+1\|N}C_0^{'T}$	$2{n_s}^3 - {n_s}^2$	MMM
$P_{k\|N} = D_0 + C_0 P_{k+1\|N}C_0^{'T}$	${n_s}^2$	MMA

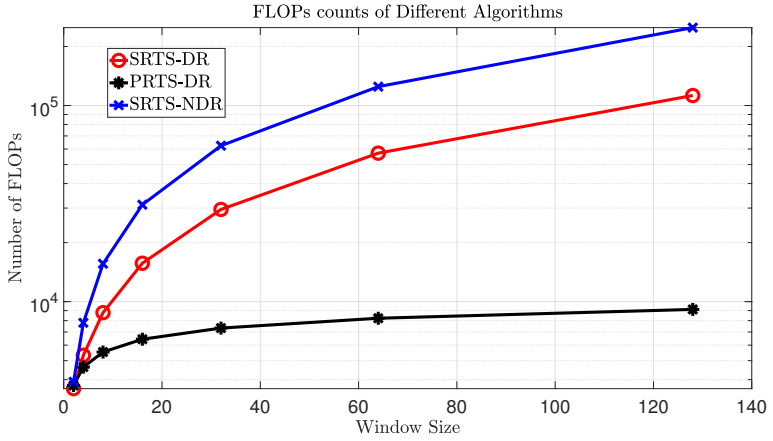

FIGURE 3.3: Theoretical Performance Comparison in terms of FLOP Counts (log scale)

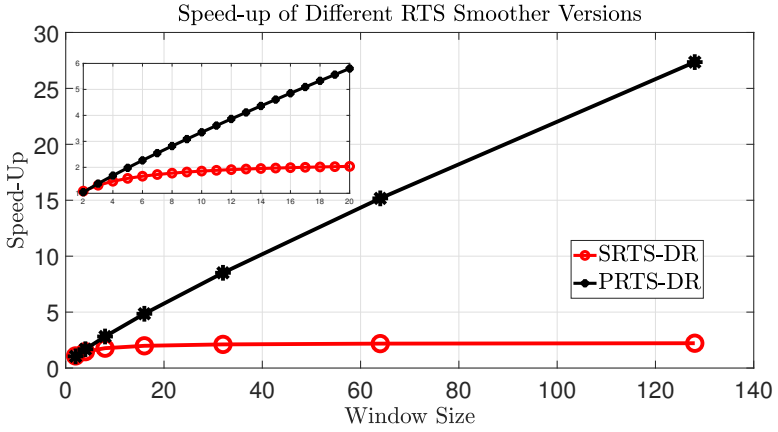

FIGURE 3.4: Speed-up for a range of window size

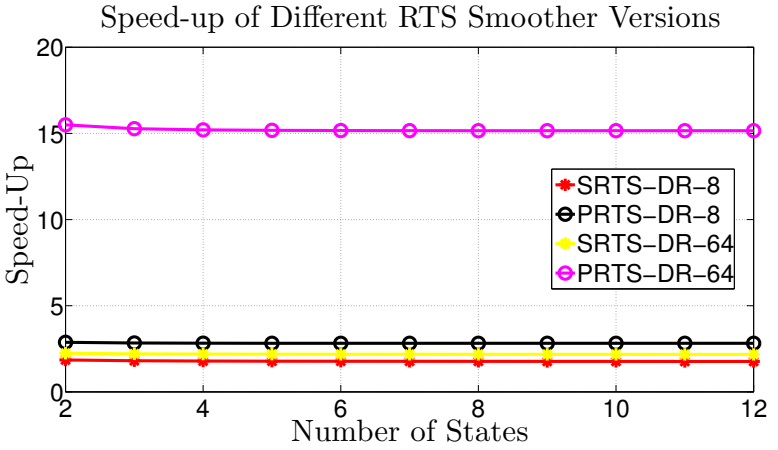

FIGURE 3.5: Speed-up for a range of number of states

becomes

$$\mathcal{F} = (19 + 6\log N)(n_s)^3 + (6n_m + 1)(n_s)^2 + 4n_s n_m^2 + n_s n_m - n_s + M(2{n_m}^2 + 2n_m - 1) \tag{3.28}$$

With multitarget tracking, the FLOPs count becomes

$$\mathcal{F}_{\mathcal{B}} = B[(19 + 6\log N)(n_s)^3 + (6n_m + 1)(n_s)^2 + 4n_s n_m^2 + n_s n_m - n_s + M(2{n_m}^2 + 2n_m - 1)] \tag{3.29}$$

5. The overall computational complexity for B targets becomes $\mathcal{O}(B\log N{n_s}^3)$.

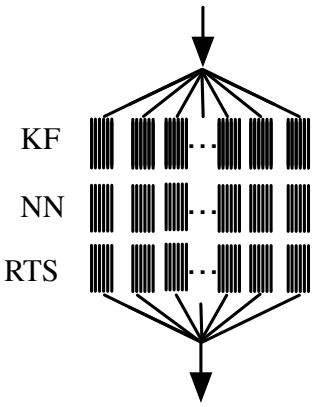

FIGURE 3.6: OpenMP Fork-Join Model.

3.4.5 Realizing MTTR on Parallel Systems

The approach this thesis have outlined hitherto is independent of any target platform or architecture. As such, Algorithm 3 can be realized on any target platform of choice. This can be a shared-memory system, a GPU system or a cluster-based parallel platform. The exact mechanism of how the parallelization is exploited in the target architecture, however, varies depending on the choice of platform.

In this thesis, this thesis solely focus on a parallel model known as shared-memory parallelism — common in contemporary systems having multiple cores or multiple processors. One of the shared-memory parallelism programming model is OpenMP (Open Multi-Processing) [74] and is used in this thesis. Furthermore, a framework that encapsulates the processes concerning multi-target tracking, gating and association (using the NN) and smoothing (using the RTS smoother) is created.

This thesis show this overall flow of processes within the proposed parallel model in Figure 3.6. The ideal fork-join model this thesis seek here is that the master thread will launch a number of independent threads, each handling a single target. Each of these threads will then in turn, launch a number of threads themselves to perform matrix-vector operations within the Kalman filter, NN and the RTS smoothing operations in parallel. For instance, if there are B targets, and B threads are created with one thread handling one target, it is possible to create a configuration so that each thread can launch a threads themselves to handle the necessary matrix operations in a parallel manner. This will result in a total of aB threads. However, in reality, due to the overheads associated with spawning and managing threads and limits of the subscription ratio, the actual number of threads, B_R, used in the parallelization s will be much less than aT, such that $B_R << aB$. As such, the choice of a often becomes task-specific. In this case, it depends on the number of targets T. This phenomena will be demonstrated in

Section 3.5. Regardless of the platform, depending on the overall number of threads, the overall complexity will be reduced when the NN and RTS smoothing algorithms are parallelized. The overall complexity of the parallelized algorithm is:

$$\begin{aligned}\mathcal{F}_{\mathcal{P}} = (19 + 6\log N)(n_s)^3 + (6n_m + 1)(n_s)^2 + 4n_s n_m^2 + n_s n_m \\ -n_s + M({2n_m}^2 + 2n_m - 1)\end{aligned} \tag{3.30}$$

which further reduces the computational complexity to $\mathcal{O}((\log N){n_s}^3)$.

3.4.6 Algorithm that is Independent to the Window Size

Although the parallelization approach presented in the previous section offers a substantial reduction in FLOPs and offers reduced computational complexity, the algorithm has a subtle limitation: it only works for window sizes that are of a power-of-two, owing to the limitations stemming from the original prefix-sum algorithm. As such, an approach is required to handle non-power-of-two window sizes. This is achieved by decomposing the non-power-of-two window size N' into a $m+1$ separate power-of-two window sizes $[N'_a, \ldots, N'_0]$ such that

$$\begin{aligned} N' &= N'_a + \ldots + N'_0 \\ &= q_a.2^a + \ldots + q_1.2^1 + q_0.2^0 \end{aligned} \tag{3.31}$$

where $q_i \in \{0, 1\}$ and $a = \lfloor \log_2(N') \rfloor$. This is akin to expressing a number in a binary format. There will be at most $2^a < \log_2(N')$ power-of-two-window sizes. These 2^a separate power-of-two-window retrodictions can be performed using the proposed algorithm, in parallel. In other words, it can be considered as reducing 2^a independent reduction trees. This process can be repeated recursively until no more decomposition can be performed, at which stage the standard (non-parallel) RTS smoother will be used. This thesis illustrate this process in Figure 3.7 and show the corresponding steps in Algorithm 4 with c and β_l denoting the remaining window sizes to be processed and the maximum value for the next power-of-two tree-size.

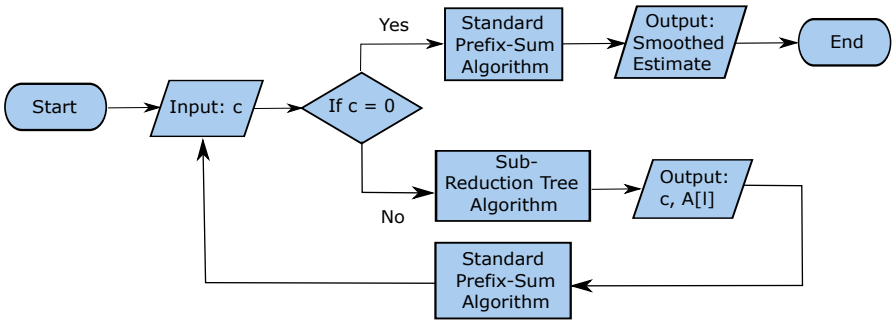

FIGURE 3.7: Block Diagram Illustrating the Window Size-Oblivious RTS Smoothing Algorithm.

Algorithm 4 Sub-reduction Tree Algorithm.

$c \leftarrow 1$
$l \leftarrow 0$
while $c \neq 0$ **do**
    $\beta_l \leftarrow \lfloor \ln N \rfloor$
    $c \leftarrow N - 2^{\beta_l}$
    $N \leftarrow c$
    $l \leftarrow l + 1$
end while

3.5 Simulation and Evaluation

3.5.1 Simulated Scenario

To demonstrate the efficacy of the proposed approach, consider a scenario with multiple targets flying at constant altitude and constant velocity over a field of view. Assume that the number of targets is known and fixed. As the exact motion model is not directly relevant to the direct contributions of the chapter, the simulation will rely on a two-dimensional constant acceleration model. More specifically, this thesis set the state transition matrix A, measurement matrix H, state vector $\mathbf{x}$, process and measurement noise covariance matrices Q and R as follows:

$$A = \begin{pmatrix} 1 & \S & \frac{\S^2}{2} & 0 & 0 & 0 \\ 0 & 1 & \S & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \S & \frac{\S^2}{2} \\ 0 & 0 & 0 & 0 & 1 & \S \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad H = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

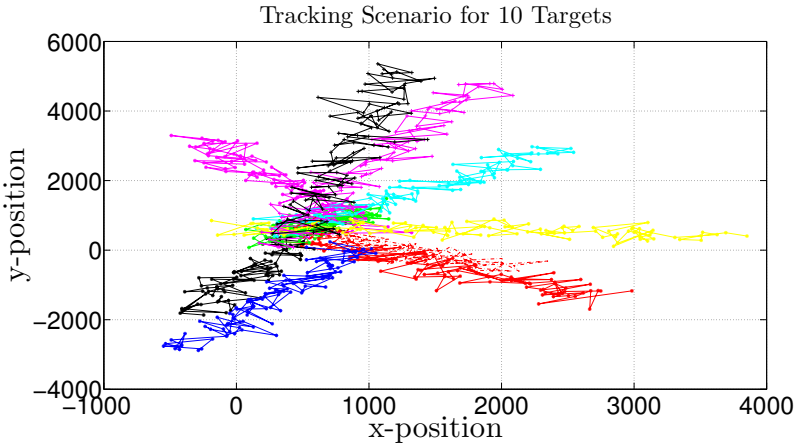

FIGURE 3.8: Simulation for Multi-target Tracking Scenario (in meters)

$$\mathbf{x} = \begin{pmatrix} x \\ \dot{x} \\ \ddot{x} \\ y \\ \dot{y} \\ \ddot{y} \end{pmatrix} \qquad \mathbf{z} = \begin{pmatrix} x \\ y \end{pmatrix}$$

$$Q = E[v_k {v_k}^T]$$

$$R = E[w_k {w_k}^T]$$

Furthermore, the initial x- and y-position of targets prior to tracking are uniformly and randomly generated with the range from 0 to 1000m. The initial velocities in x- and y- directions of targets are also uniformly generated ranging from -100 to $100 \mathrm{m} s^{-1}$. A time-flattened version of the simulated scenarios is shown in Figure 3.8 for tracking ten targets over a period of 100 time steps. In each time step, a normally distributed noise is added as process and measurement noise with standard deviation of 100m with $R = 100^2 m^2$. This thesis also report up to 1000 targets when discussing the performance results. In the evaluation of the smoother, 100 Monte Carlo runs were used for the trajectory and the total performance measure is the mean value of the runtime and speed-up.

3.5.2 Evaluation Framework

The thesis tests the proposed algorithms on a multi-core shared memory system, with an Intel KNL (7210) processor containing 256 cores and a memory of 96GB. Furthermore, the thesis relied on vendor-supplied or equivalent BLAS libraries that have already been parallelized and performance tuned for the architectures this thesis have used. As such, they provide the standard approach for a number of BLAS operations such as matrix-matrix multiplication or matrix inverse. In particular, this thesis used the EIGEN

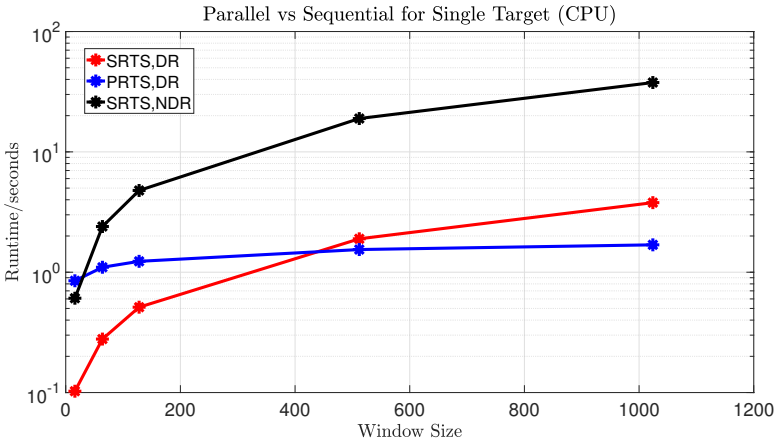

FIGURE 3.9: Runtime of Single Target Tracking Against Different Window Sizes (log scale)

Library (v3.3.3) for the evaluation. Finally, this thesis repeated the experiments a 100 times and the mean of the runtime is selected.

3.5.3 Parallelism and Data Reuse

As mentioned before, the performance gains for the proposed parallel RTS smoothing algorithm stem from both parallelization and from data reuse. To quantify the benefits of each of these components, this thesis first use a single target tracking scenario. By varying the window size N from 2 through 1024, the sequential RTS smoother with data reuse SRTS-DR and parallelized RTS smoother with data reuse PRTS-DR is compared against the naive version. The runtime results are shown in Figure 3.9 and the associated speed-up against naive version is shown in Figure 3.10. As shown in Figure 3.9, the SRTS-DR variant shows better performance than the PRTS-DR variant, for smoothing with windows sizes smaller than 250. This indicates that overheads arising out of parallelization, negatively impact the overall performance gain. In other words, for smaller window sizes, computational intensity is not sufficient to warrant excessive parallelization. However, for smoothing with window sizes larger than 250, the performance gain of PRTS-DR increases significantly. Furthermore, from Figure 3.10, it can be observed that for window sizes smaller than the threshold window size, $\eta_N \approx 250$, the SRTS-DR version performs better than the PRTS-DR version. Therefore, this thesis proposed a thread allocation approach as outlined in Section 3.5.4 as part of the overall MTTR system, that adaptively selects the number of threads and the appropriate algorithm depending on the number of targets, available number of threads on a given platform, and window size.

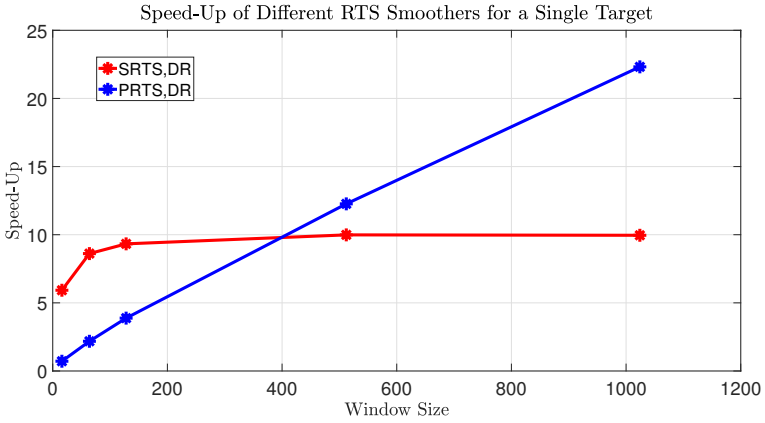

FIGURE 3.10: Speedup of Single Target Tracking Against Different Window Sizes

3.5.4 Thread Allocation and Choice of Smoothing Algorithm

The MTTR algorithm is fully customizable in terms of number of targets (fixed and known), window size, number of threads to be allocated at the target-level and smoother process. This thesis consider four different variants in evaluating the performance of the proposed algorithm:

1. Multitarget tracking is performed in parallel across all targets using the PRTS with data reuse embedded. In other words, a number of threads are run in parallel, each taking care of a target, and each of these threads can fork a number of threads in turn to perform the RTS smoothing in parallel. This thesis refer to this version as (PT, PRTS-DR);

2. Multitarget tracking is performed in parallel using the sequential RTS smoother with data reuse embedded (PT, SRTS-DR);

3. Multitarget tracking is performed in parallel using the sequential RTS smoother without data reuse embedded (PT, SRTS-NDR) which is alternatively referred to as (PT, SRTS) at times; and

4. Multitarget tracking is performed sequentially using the sequential RTS smoother without data reuse embedded (ST, SRTS).

By default, this thesis use the window size of $N = 64$, but wherever applicable, this thesis vary the window size from $N = 2$ through to $N = 4096$. To avoid a large number of combinations, this thesis vary the number of targets $8, 64, 128$ and 1024. In order to visualize the effect of the number of threads allocated at the target level, the (PT, SRTS-NDR) is going to be tested for a range of threads. The runtime for processing 1000

measurements are recorded in each of simulation (100 times and mean is calculated). As the overall number of threads in a system is finite, this thesis tested the implication of number of threads on the overall performance by varying the total number of threads from 1 to 256. This thesis show the resulting speedup against the single threaded (PT, SRTS-NDR) version in Figure 3.11, for number of targets set at $8, 128$ and 1024. There are a number of observations that can be made here. First and foremost, the multi-threaded (PT, SRTS-NDR) version offers remarkable performance improvement over the single threaded version (ST, SRTS). This speedup increases with the number of targets. Furthermore, increasing the number of threads does not increase the speedup in a linear manner. Instead, for a given number of targets, there is a maximum number of threads from which the MTTR can benefit from, beyond which, allocating more threads only results in diminishing returns. In fact, to maximize the performance, the number of targets must be accounted for when allocating threads. For instance, when tracking 128 targets, allocating 128 threads leads to the best overall speed-up of 25 over the single-threaded version. This is true when tracking $n_B = 8$ targets. However, for $n_B = 1,024$ targets, the overall performance peaks at 256 beyond which there are no performance gains. This figure matches the exact number of threads that can be spawned with hardware support, indicating that over-subscribing the system with additional threads does not improve the performance. To conclude, number of threads chosen should match number of targets to yield maximum performance gain.

Since the proposed model can be tuned to use the notion of nested parallelism, if there are any leftover threads after allocating threads for multitarget tracking, they can be allocated to parallelize the smoother operations. For instance, this is possible where the number of targets is less than the maximum number of hardware threads. To test the efficacy of the nested parallelism model, this thesis evaluated the framework as follows: this thesis set the number of targets to 8, and this thesis then allocate the remaining number of threads to parallelize the RTS smoother. For instance, if two threads are allocated at the second level (RTS smoother level), a total of 16 threads will be spawned in the RTS smoother stage for tracking 8 targets. This thesis then compare this against the version where each RTS smoother uses only one thread. This thesis show the overall results in Figure 3.12 for different smoother window sizes. This thesis observe that for window sizes less than $1,024$, nested parallelism does not offer any benefits. In fact, there is a slowdown. However, for window sizes larger than (or equal to) 2048, there are clear performance benefits. Therefore, the nested threshold window size η_n for performance benefit in nested parallelism in tracking $n_B = 8$ targets is approximately 2048.

With all these observations, this thesis select the number of threads and/or the smoother algorithms as follows:

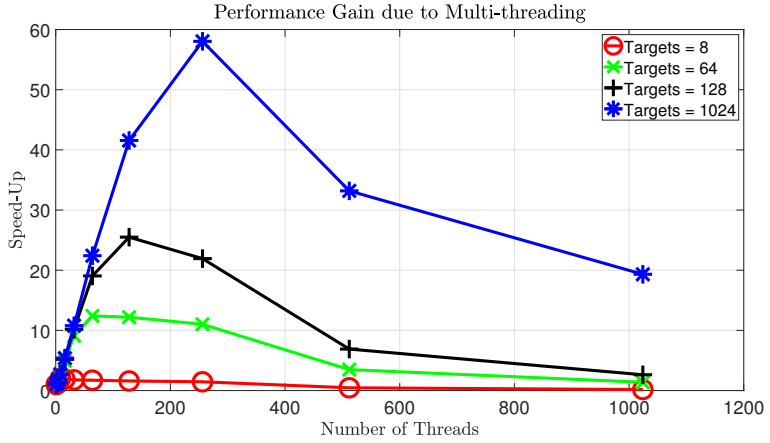

FIGURE 3.11: Performance of Multitarget Tracking against the Number of Threads

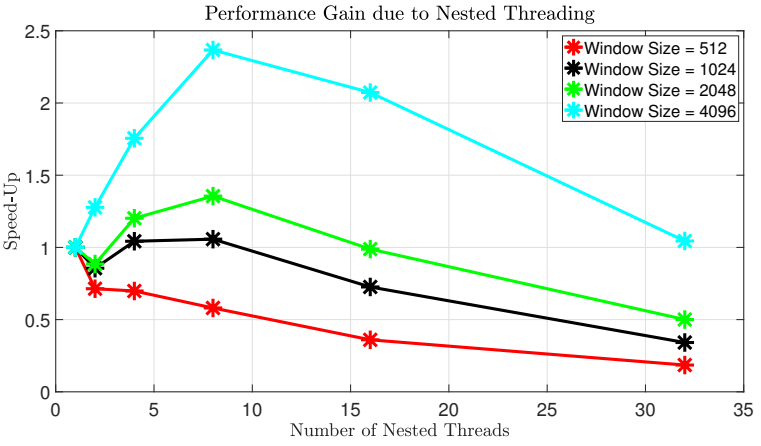

FIGURE 3.12: The Effect of Nested Parallelism on the Overall Performance of Multi-Target Tracking

1. For single target tracking, when the tracking algorithm has a window size smaller than η_N, the smoothing algorithm of choice should be SRTS-DR, and PRTS-DR otherwise;

2. Threads are primarily allocated at the target level. This is justified as the overall tracking process is much more computationally intensive than RTS smoother. When the number of targets is larger or equal to number of hardware threads, all hardware threads should be dedicated to the outer target level to obtain maximum performance and so SRTS-DR should be used for RTS smoother.

3. When allocating threads at the target level, the actual number of threads allocated at this level is, $T_B = \min(n_C S_r, n_B)$, where n_C is the number of cores, S_r is the subscription rate, and n_B is the number of targets;

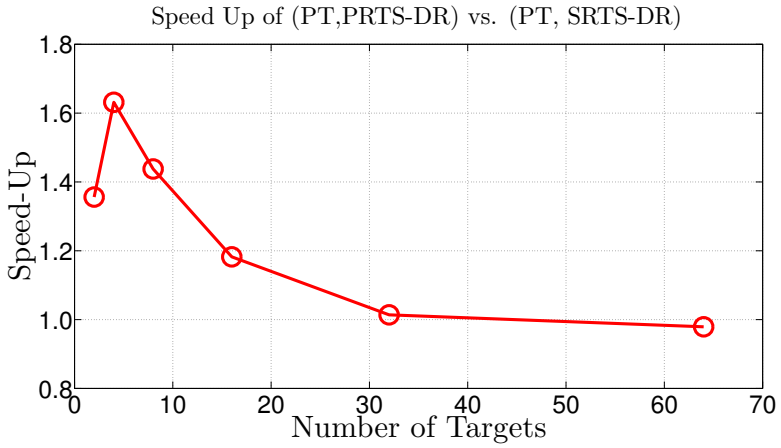

FIGURE 3.13: Performance of Multitarget Tracking against the Number of Targets

4. If the number of targets is smaller than number of hardware threads, also, when the tracking algorithm has a window size bigger than η_n, the smoothing algorithm of choice should be PRTS-DR;

The exact values for η_N, and the number of nested threads allocated at the RTS smoother level will vary depending on the platform of choice, number of cores and the subscription ratio S_r. Although the number of cores and the subscription ratio are deterministic, the overheads of spawning threads across different platforms are never the same.

With this methodology in place, the impact of number of targets on the performance of the SRTS-DR and the PRTS-DR variants are of interest. To assess this aspect, the window size is set to $4,096$ and smoothing is performed with $20,000$ time steps. The speed-up of the PRTS-DR variant is shown in Figure 3.13. The speed up is quantified by using the SRTS-DR version as the baseline to observe the benefit of introducing parallelism. As can be observed here, the performance benefits of the PRTS-DR (over SRTS-DR) diminishes with the increasing number of targets. With the increasing number of targets, the framework will start to allocate more threads towards target tracking and the actual number of threads for the RTS smoothing process will begin to decrease. With this, the performance of the RTS smoother will begin to deteriorate. Thus, if the number of targets exceeds that of the number of cores, it will be prudent to select the (PT, SRTS-DR) variant over others for better performance.

This thesis then compare all four different versions of the algorithms, namely (ST, SRTS-NDR), (PT, SRTS-NDR), (PT, SRTS-DR) and (PT, PRTS-DR), for different window sizes, while fixing the number of targets. To ensure that this thesis have a sufficient number of threads allocated at the RTS smoother level, this thesis set the number of targets to $n_B = 8$ and $n_B = 64$ On a system with $n_C * S_r = 256$, this leaves up to 32

FIGURE 3.14: Runtime for Tracking Multiple Targets for a Range of Window Sizes ($n_B = 8$) (log scale)

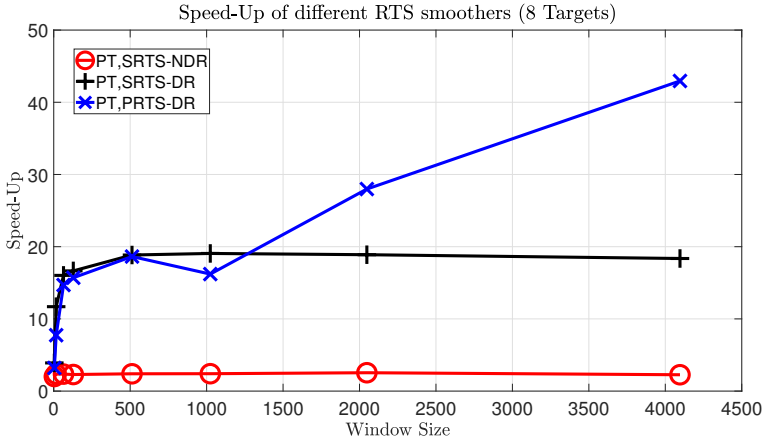

FIGURE 3.15: Speed-Up of Tracking Multiple Targets over a range of Window Sizes ($n_B = 8$)

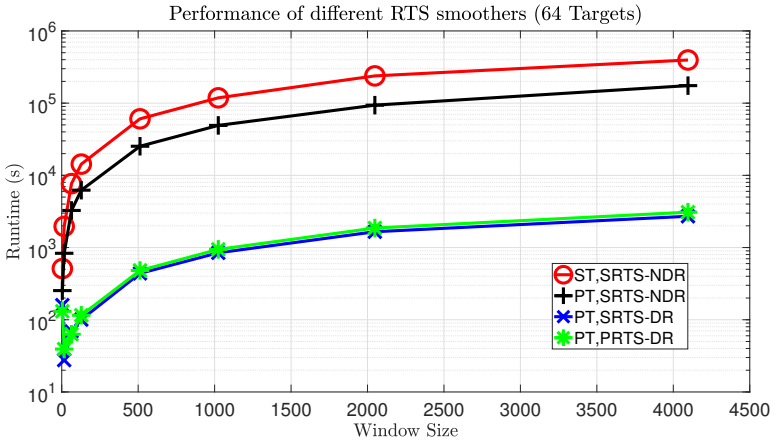

FIGURE 3.16: Runtime of Tracking Multiple Targets for a range of Window Sizes ($n_B = 64$) (log scale)

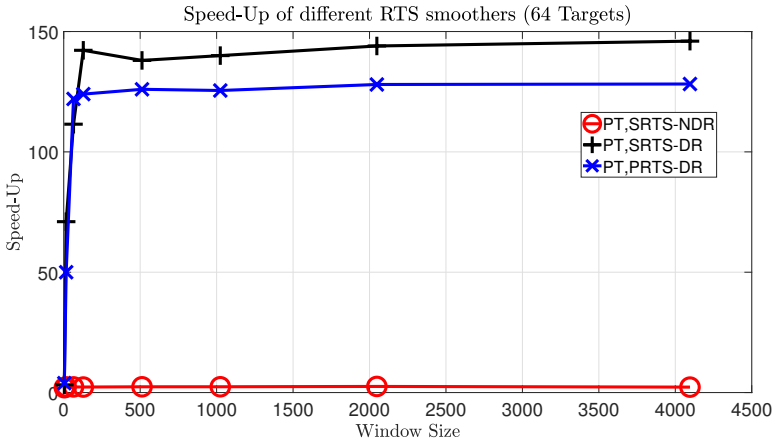

FIGURE 3.17: Speed-Up of Tracking Multiple Targets for a range of Window Sizes ($n_B = 64$)

and 4 threads to be allocated at the RTS smoother level, respectively. For each target-level setting, this thesis vary the window size from 4 through 4,096. The speedups are measured by using the (ST, SRTS-NDR) version as the baseline.

For $n_B = 8$, as observed before, for window sizes smaller than 2,048, the (PT, SRTS-DR) variant will deliver the best tracking performance. However, for window sizes larger than 2,048, the (PT, PRTS-DR) version is the best choice. For the case of $n_B = 64$, 25% of the available threads will be allocated to the target tracking level. The runtime and speed-up graphs for these cases are shown in Figures 3.14, 3.15, 3.16 and 3.17. A number of observations can be drawn from these figures. For instance, the (PT, SRTS-DR) version has the best performance among all four algorithms for window sizes smaller than 2,048. However, for cases with window sizes larger than (or equal to) 2,048, nested threading (PT, PRTS-DR) can offer attractive performance over the (PT, SRTS-DR) version. It is also evident that naively parallelizing tracking algorithms can only lead to sub-optimal performance gains. By utilizing and balancing the parallelization and data reuse, it is possible to attain a 40-fold speedup against the baseline. It can also be observed that when using the (PT, PRTS-DR) version, the performance for tracking $n_B = 8$ and $n_B = 64$ are almost the same (approximately around 9.862 seconds). Finally, for $n_B = 64$, the (PT, SRTS-DR) diagram shows better performance than the (PT, SRTS-NDR) variant. These results indicate that by carefully utilizing the framework, a nearly 150-fold speedup can be achieved.

3.5.5 Evaluation of the Window Size-Independent Algorithm

As the last aspect of the evaluation, this thesis has considered the algorithm that is designed to be independent to the inherent limitation of the basic algorithm that is

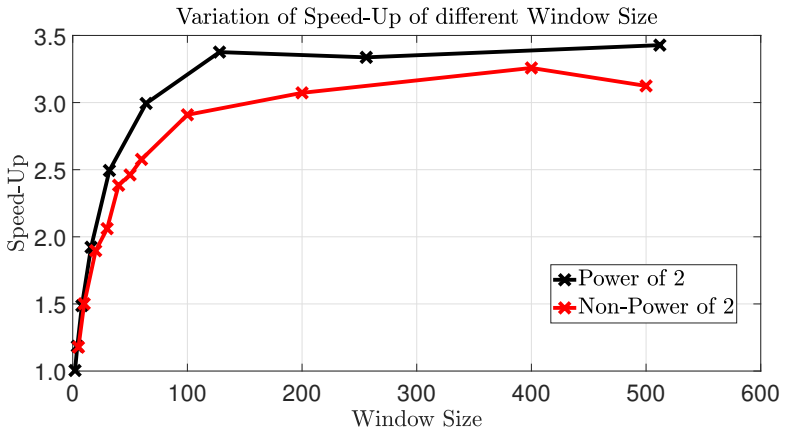

FIGURE 3.18: Performance Gain of the Overall PRTS for a Range of Window Sizes

reliant on power-of-two window sizes. For this case, this thesis varied the window size N, within the range of 4 through 512, for 100 targets over $10,000$ time steps. This thesis varied the window sizes so as to include non-power-of-two window sizes. This thesis show the resulting speed-up of the proposed algorithm in Figure 3.18. From these results, it is evident that the proposed algorithm offers notable performance gains over the SRTS variant.

3.6 Summary

In this chapter, this thesis created a fully functional parallel framework for multitarget tracking with retrodiction. The key contribution, which underpinned this overall framework here, is a novel parallel retrodiction algorithm based on the RTS smoother. To overcome the limitations within the original RTS smoother, more specifically the ones that impede parallelization, this thesis have reformulated and refined the underlying mathematical constructions. These reformulations, which are based on the prefix sum operation, have minimized the interdependencies between the operations and eliminated most of the redundant computations, improving reuse. With these in place, the proposed approach leads to a very highly performing parallelized filter for retrodiction, enabling large-scale multi-target tracking.

This approach, compared to the approaches that have been hitherto based on complex partitioning mechanisms, is easier to implement. The parallel algorithm, in addition to reducing the number of FLOPs, has a reduced computational complexity. More specifically, owing to the formulations stemming from the prefix-sum, the original algorithm was only capable of handling power-of-two window sizes. This chapter addressed this

issue by formulating a window size-oblivious version of the proposed algorithm. Addressing this subtle issue enables the proposed algorithm to be used for any window size. The resulting algorithm (PRTS) offers rather significant performance gains compared to the sequential version of the RTS smoother (SRTS).

In addition to parallelization of the RTS smoother, parallelizing the tracking and measurement to track association offers additional performance improvements. In essence, the proposed framework offers very notable performance gains over the conventional approaches, which rely on vanilla version of the RTS smoother. This exhaustive simulations have demonstrated the potential benefits of the proposed approach. Although the performance gains are likely to vary between different computing platforms (and tracking scenarios), the approach is readily adoptable across different computing platforms. More specifically, the exploitation of data reuse in the proposed algorithm is very notable.

Going beyond the performance on a single target, the proposed algorithm can also bring notable performance gain in multi-target tracking scenarios. Besides, the overheads caused by thread launching and the number of cores on the CPU is very likely to improve in the future. With these, this thesis firmly believe that the proposed approach is both novel and offers an avenue for tracking a large number of targets, if sufficient parallel resources can be found. Given the rise and the norm of modern parallel platforms, this is a very promising approach for tracking a very large number of targets in real-time.

Chapter 4

Fast Fixed-Lag Smoother

4.1 Introduction

Smoothing is one of the basic techniques that is fundamental to a number of target tracking scenarios [44, 45], particularly where delays can be tolerated. Given a sequence of observations $z_{1:t}$, and corresponding states $x_{1:t}$, the basic idea of smoothing is to compute the posterior density $p(x_t|z_{1:t+l})$ where $l \geq 0$. In other words, future observations are used to *smooth-out* the current states so as to minimize the estimation errors. In general, there are three variants of smoothing: fixed-interval, fixed-lag and fixed-point smoothing. All of these techniques vary depending on what is being smoothed out. Fixed-interval smoothing [55] uses all measurements obtained over a fixed interval to compute smoothed estimates for each required time point. As such, the computational complexity is a function of the length of the interval. Fixed-lag smoothing [90, 91] is performed by delaying the estimation process to incorporate future measurements in order to improve the estimation accuracy, particularly where a certain time delay L can be tolerated.The fixed-point approach is similar to that of the fixed-lag. However, the point on which the estimate to be refined is fixed instead of varying with time which has the complexity. By caching the intermediate history of the states, a recursive fixed-point smoother can be achieved without stepping back through each of the intermediate estimates [26]. For instance, Biswas proposed an augmented fixed-point smoother by using the augmented matrix structure [91].

The application of smoothing is rich which includes radar tracking [92], economics [93], signal processing [94], traffic modelling [95] and so on. In particular, for real-time application, aircraft radar guidance system rely on filtering and smoothing heavily to provide accurate position data [92]. In economics, filtering and smoothing are used in predicting real-time market price to reduce the investment risk and play an important role in the

economic development and financial building [96]. In traffic systems modelling, a short-term traffic volume model is developed based on filtering and smoothing to monitor and predict real-time traffic volume [97]. Therefore, filtering and smoothing are important tools in modern signal processing and system modelling.

The Rauch-Tung-Striebel (RTS) smoother [25], also called the Kalman smoother, is used in conjunction with the Kalman filter [24]. Due to its simplicity, it is widely used in the tracking community. For example, in [57], the authors employ the RTS algorithm for MHT applications in conjunction with the Interacting Multiple Model (IMM) algorithm for maneuvering target tracking. In [19], the IMM smoothing is further improved by only using M filters/smoothers for a bank of M models. In [58], fixed interval smoothing IPDA (sIPDA) was proposed, which also use RTS smoothing formulae in order to obtain better target estimate and target probability existence. Furthermore, in [61], RTS is used for backward Point Probabilistic Hypothesis Tracking (PPMHT) pass. It is also regarded as the fastest implementation among a number of fixed-interval smoothing variants [54] with the computational complexity of $O(N(n^3))$, where N is the window size and n is the number of states. When fixed-lag smoothing is used, they are mostly augmented fixed-lag smoothers. For instance, in [51], a fixed-lag state augmented system is proposed to utilize the IMM and Joint Probabilistic Data Association (JPDA) to improve state estimations. In [98], a method called Augmented State Integrated Probabilistic Data Association (ASIPDA) is proposed to fuse IPDA with a retrodiction approach. In [53], the authors extend the idea for multi-scan target existence information.

However, one of the major drawbacks of smoothing, regardless of the variant, is the computational complexity, and thus the number of computations, which increase with the time-interval or delay, N. These computations can easily begin to dominate the overall number of computations performed during state estimation, particularly when numerous targets need to be tracked. When the number of tracks to be smoothed increases, the overall number of computations can become a bottleneck, even where small delays are tolerable. There is a body of work around improving the computational performance of smoothers, for instance [26, 99]. The central idea behind the body of work is to render a fast fixed-lag smoother since it is mostly used for real time processing and so computation is expected to be minimised. Although the proposed approach aims to overcome the performance issues, as will be discussed in latter sections, this approach is numerically unstable, and becomes practically unusable. The motivation for this chapter is to fix this numerical instability.

In theory, it is possible to have a conventional fixed-lag RTS smoother (cRTS) that is computationally independent of the window size N as will be shown in this chapter. However, in practice, such an algorithm will lead to erroneous results. Therefore, this

thesis aim to derive the fastest, numerically stable, fixed-lag smoother so that the computational complexity is no longer strongly coupled to the time-delay or window-size. In doing so, this thesis make the following key contributions:

1. This thesis revisit the existing body of work that aimed to offer the computationally efficient version of the RTS, and show how the cRTS can be reformulated to yield the computationally independent to window size (which refer to as fRTS$^-$) in Theorem 1;
2. This thesis carry out a detailed analysis on the stability of the fRTS$^-$, and show that it can exhibit divergence and thus may become unusable over time; and
3. This thesis offer a solution to address the stability issue, which this thesis refer to as FRTS, that offers both performance and numerical stability.

This evaluations, based on a number of simulations, show that all the proposed approaches offer superior performance and numerical stability when compared against the fRTS$^-$ version. With these results, this chapter shows that RTS smoother can be implemented in a way that it is indepednent of the window size with acceptable accuracy. The rest of the chapter is organized as follows: In Section 4.2, this thesis concentrate on presenting the linear state space model and FRTS to serve as a background for the section that follows this. Also, this thesis prove that FRTS can be computationally independent to window size after reformulation. In Section 4.3.1, this thesis show the stability issue of fRTS$^-$ and the source of the problem. Section 4.3.2 introduces the notion of using condition number to solve and monitor the algorithm to assure feasibility and usability. Then, Section 4.4 displays the complexity of the proposed algorithm and compares FLOPs counts with cRTS. After that, Section 4.5 shows the accuracy and speed-up of the proposed algorithms from a series of simulations. Finally, Section 4.6 concludes the chapter.

4.2 Background

4.2.1 Fast RTS Smoother

For window size N, the smoothed estimate will be calculated as

$$\begin{aligned} x_k^s &= x_k - C_k x_{k+1}^- + C_k x_{k+1}^s \\ x_{k-1}^s &= x_{k-1} - C_{k-1} x_k^- + C_{k-1} x_k^s \\ &\vdots \\ x_{k-N}^s &= x_{k-N} - C_{k-N} x_{k-N+1}^- + C_{k-N} x_{k-N+1}^s \end{aligned} \tag{4.1}$$

The dependency issue of the RTS algorithm can be clearly seen in (4.1). In order to calculate the time-lag smoothed estimate at $k-N$, denoted as x_{k-N}^s, the smoothed estimate of the next time step at $k-N+1$ has to be known. This dependency propagate to the smoothed estimate at the beginning of the window x_k^s. Moreover, the computations increase with N.

Theorem 1. The conventional RTS, which has complexity of $\mathcal{O}(N(n^3))$, can be reformulated to be computationally independent to window size N which gives $\mathcal{O}(n^3)$.

Proof. To simplify the operations in (4.1), the following can be defined

$$\beta_k = x_k - C_k x_{k+1}^- \tag{4.2}$$

Therefore, (4.1) can be written as

$$\begin{aligned} x_k^s &= \beta_k + C_k x_{k+1}^s \\ x_{k-1}^s &= \beta_{k-1} + C_{k-1} x_k^s \\ &\vdots \\ x_{k-N}^s &= \beta_{k-N} + C_{k-N} x_{k-N+1}^s \end{aligned} \tag{4.3}$$

Therefore, x_p^s can be simplified as, where $p = k - N$

$$x_p^s = \beta_p + \sum_{i=p}^{p+N-1} [\prod_{j=p}^{i} C_j]\beta_{i+1} + [\prod_{j=p}^{p+N} C_j] x_{k+1}^s \tag{4.4}$$

The following are defined to further simplify the expression,

$$\begin{aligned}\Psi_p &= \beta_p + \sum_{i=p}^{p+N-2} [\prod_{j=p}^{i} C_j] b_{i+1} \\ \Omega_p &= \prod_{j=p}^{p+N-1} C_j \end{aligned} \tag{4.5}$$

which makes

$$x_p^s = \Psi_p + \Omega_p \beta_k + \Omega_p C_k x_{k+1}^s \tag{4.6}$$

As the value of k changes in every time step and so does $k - N$, however, the whole summation sequence Ψ and Ω do not need to be re-calculated if the following operations are performed in each time step instead

$$\begin{aligned}\Omega_{p+1} &= C_p^{-1} \Omega_p C_k \\ \Psi_{p+1} &= C_p^{-1} [\Psi_p - \beta_p] + \Omega_{p+1} \beta_{k+1} \end{aligned} \tag{4.7}$$

Therefore, the smoothed estimate in the next time step will be

$$x_{p+1}^s = \Psi_{p+1} + \Omega_{p+1} C_{k+1} x_{k+2}^s \tag{4.8}$$

The calculations involved in (4.7) and (4.8) are independent of window size N and so this algorithm has the complexity of $\mathcal{O}(n^3)$. Same principle can be applied to compute the smoothed covariance. The smoothed covariance for window size of N can be calculated as

$$\begin{aligned} P_{k-N}^s = P_{k-N} &- C_{k-N} P_{k-N+1}^- C_{k-N}^T \\ &+ C_{k-N} P_{k-N+1}^s C_{k-N}^T \end{aligned} \tag{4.9}$$

To simplify the operations, the following can be defined,

$$D_k = P_k - C_k P_{k+1}^- C_k^T \tag{4.10}$$

Therefore,

$$P_p^s = D_p + C_p P_{p+1}^s C_p^T \tag{4.11}$$

When propagating P^s_{k-N+1} back to P^s_{k+1}, it can be written as

$$\begin{aligned} P^s_p = D_p + \sum_{i=p}^{p+N-1} [\prod_{j=p}^{i} C_j] D_{i+1} [\prod_{j=-i}^{-p} C^T_{-j}] \\ + [\prod_{j=p}^{p+N} C_j] P^s_{k+1} [\prod_{j=-p-N}^{-p} C^T_{-j}] \end{aligned} \tag{4.12}$$

The following are defined to further simplify the expression,

$$\begin{aligned} \Gamma_p &= D_p + \sum_{i=p}^{p+N-2} [\prod_{j=p}^{i} C_j] D_{i+1} [\prod_{j=-i}^{-p} C^T_{-j}] \\ \Delta_p &= \prod_{j=-(p+N-1)}^{-p} C^T_{-j} \end{aligned} \tag{4.13}$$

Therefore,

$$P^s_p = \Gamma_p + \Omega_p D_k \Delta_p + \Omega_p C_k P^s_{k+1} C^T_k \Delta_p \tag{4.14}$$

To obtain computational independence, the following operations can be performed on the summation sequence Γ and Δ,

$$\begin{aligned} \Delta_{p+1} &= C^T_k C^T_p \Delta_p C^{-T}_p C^T_k \\ \Gamma_{p+1} &= C^{-1}_p [\Gamma_p - D_p] C^{-T}_p + \Omega_{p+1} D_{k+1} \Delta_{p+1} \end{aligned} \tag{4.15}$$

Therefore, the smoothed covariance in the next time step will be

$$P^s_{p+1} = \Gamma_{p+1} + \Omega_{p+1} C_{k+1} P^s_{k+2} C^T_{k+1} \Delta_{p+1} \tag{4.16}$$

As a result, the computation of the mean and covariance from the smoother can be independent to the window size N. The fRTS$^-$ algorithm is shown in Algorithm 5.

Algorithm 5 fRTS$^-$ Fixed-Lag Smoother

▷ Initialize Ψ as a vector of zero
▷ Initialize Ω as an identity matrix
▷ Initialize Γ as a matrix of zero
▷ Initialize Δ as an identity matrix
for `t=0; t<T; i++` **do**
    ▷ Forward Pass: KF
    $[P_{t+1}, P^-_{t+1}, x_{t+1}, x^-_{t+1}] = KF[x_t, P_t]$
    ▷ Backward Sweep: RTS
    $C_t \leftarrow P_t F^T (P^-_{t+1})^{-1}$
    $\beta_t \leftarrow \hat{x}_t - C_t \hat{x}^{-1}_{t+1}$
    $D_t \leftarrow P_t - C_t P^{-1}_{t+1} C^T_t$
    if $(0 < t < N)$ **then**
        $\Psi \leftarrow \Psi + \Omega \beta_t$
        $\Gamma \leftarrow \Gamma + \Omega D_t \Delta$
        $\Omega \leftarrow \Omega C_t$
        $\Delta \leftarrow C^T_t \Delta$
    else
        $\Psi \leftarrow \Psi + \Omega \beta_t$
        $\Gamma \leftarrow \Gamma + \Omega D_t \Delta$
        $\Omega \leftarrow \Omega C_t$
        $\Delta \leftarrow C^T_t \Delta$
        $x^s_p \leftarrow \Psi + \Omega x_t$
        $P^s_p \leftarrow \Gamma + \Omega P_t \Delta$
        $\Omega \leftarrow C^{-1}_p \Omega$
        $\Delta \leftarrow \Delta C^{-T}_p$
        $\Psi \leftarrow C^{-1}_p (\Psi - \beta_p)$
        $\Gamma \leftarrow C^{-1}_p (\Gamma - D_p) C^{-T}_p$
    end if
end for

4.3 Numerical Stability Analysis

4.3.1 Numerical Stability of fRTS$^-$

Although the fRTS$^-$ version of the RTS minimizes redundant computations through re-use, it is practically unusable due to issues relating to its numerical stability. In the original fRTS$^-$ algorithm, with the majority of the computations being chained via

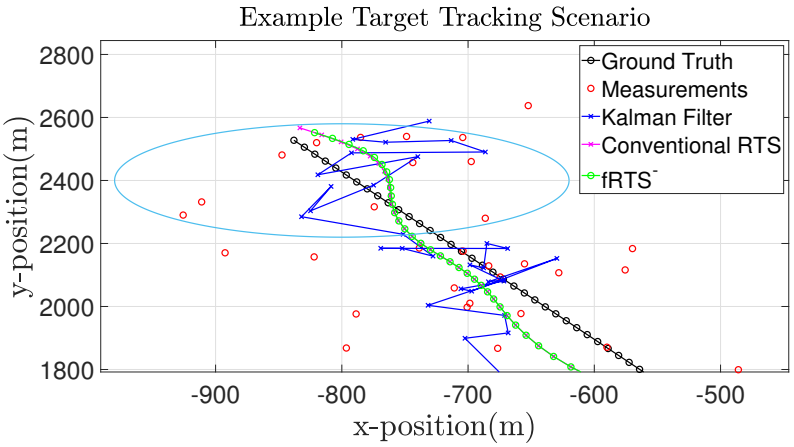

FIGURE 4.1: Divergence issues of the fRTS$^-$ for an example scenario

matrix-matrix multiplications, errors, albeit small, from each of the smoother steps is likely to accumulate over iterations. Over a number of steps, the accumulated error will be large enough so that the estimated state will begin to deviate from the ground truth as shown in circled region in Figure 4.1. This divergence issue is also reported in [26].

This chapter simulated the movement of a single target moving diagonally along a North-Westerly direction along a straight-line on a single plane, denoted by $y = -3x + 100$, for $x = 0$ through $x = -800$. In the result, the cRTS and fRTS$^-$ methods begin with producing similar estimates, over time, the estimates from the fRTS$^-$ start to deteriorate, and this can be observed in Figure 4.2. This thesis also show how the error develops for the covariance matrix P over time, in Figure 4.3.

Despite simulating a simple example, it was sufficient enough to show the divergence issues of the fRTS$^-$ method. Given that fRTS$^-$ is derived to match the cRTS, one would expect the estimates produced by both methods to be the same. However, the non-deterministic nature of the execution of instructions inside the CPUs, and associative properties of floating point addition and multiplication operations, lead to different results.

4.3.2 Addressing the Numerical Instability

The key computations that lead to accumulated errors over time are through cached or reused values of Ψ, Ω, Γ and Δ from (4.5) and (4.13). At every iteration of the sweep, their original values are overwritten by the multiplication of the inverse of the smoother gain and cached gain Ω. Ideally, having stable values at every iteration would not lead to divergence, which is impossible in the fRTS$^-$ version. However, if these expressions can be computed at regular intervals instead of at every time step as in cRTS does, it can help restoring stability while preserving the computational benefit.

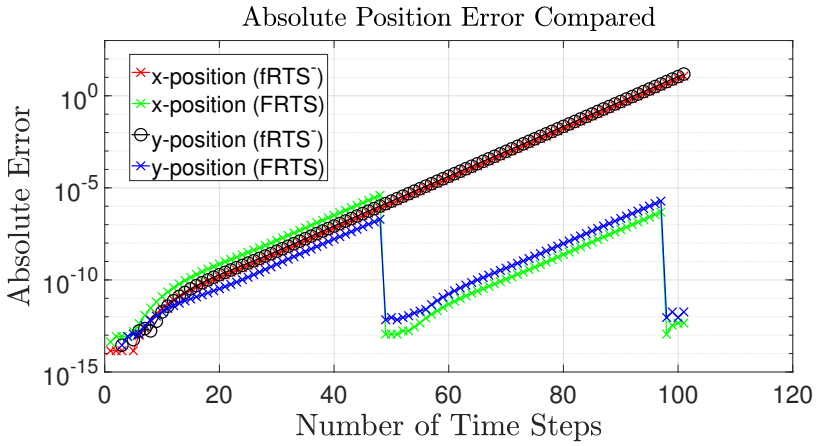

FIGURE 4.2: Accumulated position error of the cRTS and FRTS smoothers compared (in log scale)

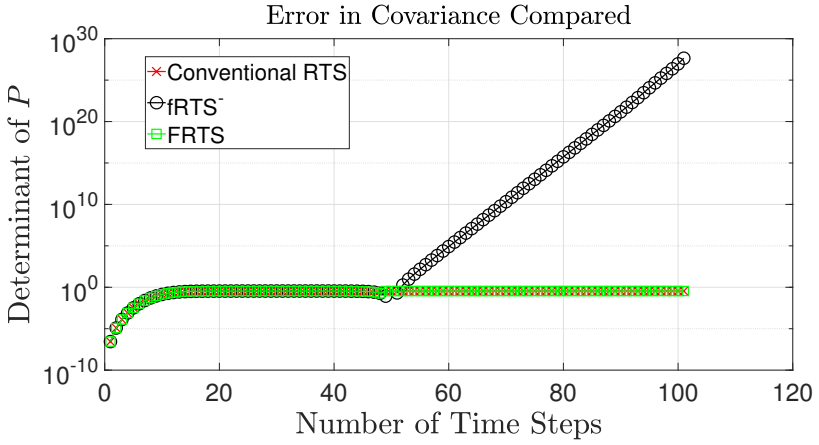

FIGURE 4.3: Accumulated error in the covariance of the cRTS and FRTS smoothers compared (in log scale)

This chapter demonstrate the possibility of restoring the stability of the fRTS$^-$ in Figure 4.2, where the relevant values are restored using the cRTS at regular intervals. With the help of cRTS to compute the error-free values of Ψ, Ω, Γ and Δ, the error is bounded. This thesis also show how the error of covariance matrix varies over time in Figure 4.3. Here, this thesis renew the values every $w_r = 50$ measurements. However, the choice of renewal window w_r may vary with different initializations, and in fact between problems. It is purely about trading off computational efficiency with numerical stability. If computed at every time step, fRTS$^-$ may converge to the cRTS, but this thesis would lose the computational efficiency. On the other hand, if restored sparsely, it may become unstable.

4.3.3 Condition Number

To understand the rate of divergence, a monitoring metric is needed to efficiently renew the error accumulated in fRTS$^-$. During computation, particularly with repeated matrix-vector multiplications, numerical errors can be encountered for a number of reasons, including, but not limited to, rounding, truncation, propagation, and accumulation [100]. The condition number of a matrix can be an effective method to signify how sensitive the computation is to perturbations [101], and this has a wide variety of applications in numerical analysis and linear algebra [102–104]. In this case, the main form of computation where errors are encountered is when attempting to solve the equations in the form of $x = \rho^{-1}b$ as in (4.7). More details can be found in [105]. Since computer algorithms can only provide an approximation of the vector b and x, namely, x^* and b^*. To understand the relative error induced in x caused by b, the condition number can be used.

In general, a matrix with high condition number is described as "ill-conditioned", and low condition number as "well-conditioned". The condition number is defined as [100]

$$\kappa(\rho) = \|\rho\| \left\|\rho^{-1}\right\| \tag{4.17}$$

$\|\cdot\|$ is the norm of the matrix. In this analysis, which is presented in a later section of this chapter, the L^2-norm is used. If ρ is singular, then $\kappa(\rho) \to \infty$, and the computation of its inverse is likely to have large numerical errors. On the other hand, if ρ is isotropic, then $\kappa(\rho) = 1$, and a stable algorithm is guaranteed. However, in iterative methods, the consideration of $\kappa(\rho)$ might not be enough. In particular, for problems addressed in this chapter, where one iteratively computes $x_{t+1} = \rho_t^{-1}x_t$, where $t = 0, 1, \ldots, T$ denote the time step, the errors become accumulated over time, regardless of how well-conditioned the matrix ρ is. As such, the algorithm may diverge over time even if $\kappa(\rho)$ is small enough to guarantee $x = \rho^{-1}b$ to be non-divergent. Here, the numerical accuracy of x_t depends on the product of $\kappa(\rho_t)$, namely, $\kappa(\rho_0)\kappa(\rho_1)...\kappa(\rho_T)$ instead of $\kappa(\rho)$ at only any particular time instant. This can also be seen through considering the solution of a linear system $\rho x = b$, and suppose the vector b is perturbed by a small amount of vector and give out the value $\hat{b}$. This error will induce perturbation to x and give out $\hat{x}$. The relative error of vector x can be represented as

$$\frac{\|x - \hat{x}\|}{\|x\|} \leq \kappa(\rho)\frac{\left\|b - \hat{b}\right\|}{\|b\|} \tag{4.18}$$

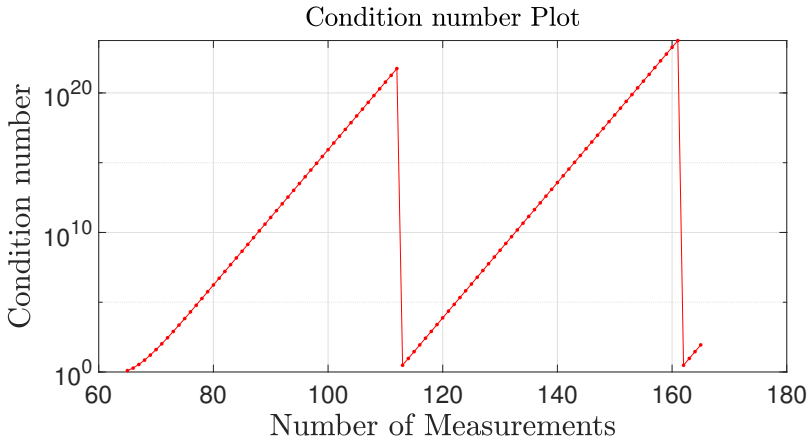

FIGURE 4.4: Variation of condition number of the smoother gain (proposed FRTS) with the renewal process

Suppose (4.18) is iterative and start from $t = 0$ to T

$$\frac{\|x_T - \hat{x}_T\|}{\|x_T\|} \leq [\prod_{j=0}^{T-1} \kappa(\rho_j)] \frac{\|x_0 - \hat{x}_0\|}{\|x_0\|} \tag{4.19}$$

From (4.19), it can be shown that the relative error in vector x_T is bound by the relative error of the vector x_0 at $t = 0$ and the product of condition number $\kappa(\rho_t)$. Therefore, product of condition number $\kappa(\rho_t)$ has a magnifying effect on the relative error on the vector x_T. In other words, monitoring the product of the condition number can reflect the perturbation range of the algorithm fRTS$^-$.

In the algorithm fRTS$^-$, it is found that there are a number of operations that can lead to divergence. Among these, the divergence is dominated by C_p^{-1}. For instance, consider the expression (4.7),

$$\Psi_{p+1} \leftarrow C_p^{-1}\Psi_p - C_p^{-1}\beta_p + \Omega_{p+1}\beta_{k+1}$$

which after the h^{th} time step, can be simplified as

$$\Psi_{p+h} \leftarrow (C_p C_{p+1} \dots C_{p+h-1})^{-1}\Psi_p + \mathbf{C}$$

ignoring the constant $\mathbf{C}$, the expression can be seen in the form of

$$x = \rho^{-1}b$$

where $x = \Psi_{t+h}$, $\rho = C_p C_{p+1} \dots C_{p+h-1}$ and $b = \Psi_t$ on the expression above. As a result, the accuracy of $\mathbf{\Psi}$ deteriorates over time as it is affected by the condition number

of C_t at every time step. Therefore, by monitoring the condition number of matrix multiplication train $X = \kappa(C_j)$, the amount of error can be estimated.

Apart from using the condition number train, the initial relative error of vector x_0 in Equation (4.19) also needed to be considered to predict the error propagation. It is shown that the intermediate error is irrelevant in the relative error of interest at $t = T$ as the error will be bounded by the previous time step. Therefore, considering the initial relative error will be sufficient. In order to calculate the initial relative error, this thesis need to know the divergence of Ψ after using the fRTS$^-$ for one time step comparing with cRTS. It implies that cRTS is needed to be implemented along side with fRTS$^-$ to witness the degree of divergence. However, as only the vector Ψ is needed to be calculated, the full calculation of cRTS is not needed as shown in Accuracy Renewal Module Part 2 in Table 4.2. Therefore, by using the following as a monitoring metric,

$$J = [\prod_{j=0}^{T-1} \kappa(\rho_j)] \frac{\|x_0 - \hat{x}_0\|}{\|x_0\|} \tag{4.20}$$

where J represents the magnification of relative error over time and is always higher than the relative error of the vector x_T. Therefore, whenever J exceeds a preset threshold θ, cRTS can be used to cease the error propagation. If user only can tolerate 10 percent of percentage error, θ is needed to set to 0.1. An example case is illustrated in Figure 4.4, where the growth of the product of condition number is presented and reset by using the cRTS approach which forms the accuracy renewal algorithm. The algorithm is shown in Algorithm 6. With the help of the accuracy renewal module on fRTS$^-$, a stable algorithm can be achieved, which this thesis refer as the Fast RTS smoother (FRTS).

Algorithm 6 Accuracy Renewal Module Part 1

if $J > \theta$ **then**
  $\Psi \leftarrow 0$
  $\Omega \leftarrow I$
  $\Gamma \leftarrow 0$
  $\Delta \leftarrow I$
  $X \leftarrow 1$
  for `k=0; k<N-1; k++` **do**
    $\Psi \leftarrow \Psi + \Omega\beta_k$
    $\Gamma \leftarrow \Gamma + \Omega D_k \Delta$
    $\Omega \leftarrow \Omega C_k$
    $\Delta \leftarrow C_k^T \Delta$
  end for
end if
$X \leftarrow cond(C_p)X$

4.4 Computational Complexity

The smoothing process is often incorporated as part of the Kalman filtering process. In other words, at every iteration of the filtering, the smoother algorithm will iterate through N data points for deriving smoothed estimates. Let n_s be the number of states. In summary, a single iteration of the RTS smoother with a window size of N will require the following:

- $6N$ matrix-matrix multiplications;
- N matrix-vector multiplications;
- $4N$ matrix additions (subtractions);
- $2N$ vector additions (subtractions);
- N matrix transpose operations; and
- N matrix inversions.

TABLE 4.1: Operational Complexity of the RTS smoother

Operation	**FLOPs**	**Type**
Smoother gain:		
$P_{k\|k}A_k^T$	${n_s}^3 - {n_s}^2$	MMM
$P_{k+1\|k}^{-1}$	$\approx {n_s}^3$	MI
$C_k = P_{k\|k}A_k^T P_{k+1\|k}^{-1}$	${n_s}^3 - {n_s}^2$	MMM
Smoothed state:		
$\hat{x}_{k+1\|N} - \hat{x}_{k+1\|k}$	n_s	VVA
$C_i(\hat{x}_{k+1\|N} - \hat{x}_{k+1\|k})$	${n_s}^2 - n_s$	MVM
$x_{k\|N} = \hat{x}_{k+1\|k} + C_i(\hat{x}_{k+1\|N} - \hat{x}_{k+1\|k})$	n_s	VVA
Smoothed covariance:		
$P_{k+1\|N} - P_{k+1\|k}$	${n_s}^2$	MMA
$C_k(P_{k+1\|N} - P_{k+1\|k})$	${n_s}^3 - {n_s}^2$	MMM
$C_k(P_{k+1\|N} - P_{k+1\|k})C_i^T$	${n_s}^3 - {n_s}^2$	MMM
$P_{k\|k} + C_i(P_{k+1\|N} - P_{k+1\|k})C_k^T$	${n_s}^2$	MMA

The matrix transposition of A will be performed in the KF step, which can be cached to avoid repeated computations. This can be easily avoided by choosing an appropriate

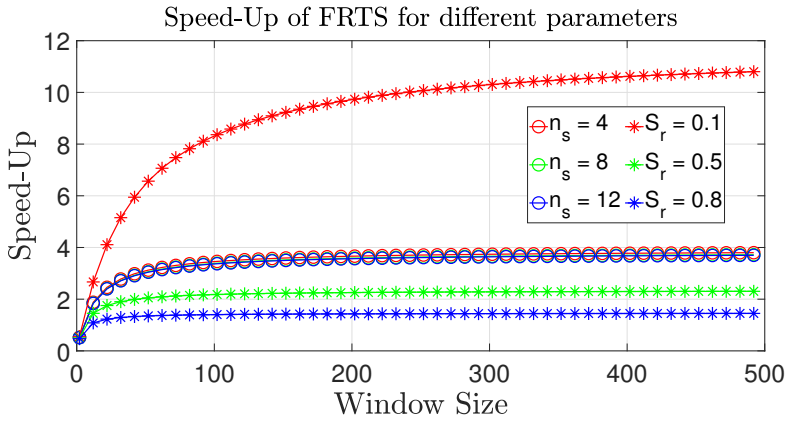

FIGURE 4.5: Speed-up of different FRTS algorithms for different n_s and S_r

indexing order of the matrix, (row-order as opposed to column-order or vice versa). As the state matrices are significantly smaller than cache-line capacities, this does not cause any performance issues. Hence, there are a total of $15N$ operations for a single iteration of RTS smoother for window size N. In general, matrix-matrix multiplication is an operation with the cubic complexity. The overall complexity of the cRTS is the sum of all operations shown in Table 4.1, which in turn influences the overall runtime performance [80]. However, the total runtime complexity can be approximated by the dominating runtime complexity. The total number of FLOPs Ł$_1$ in the RTS smoother is:

$$Ł_1 = N(9{n_s}^3 + n_s) \tag{4.21}$$

where the dominating term, namely $9N{n_s}^3$, decides the overall runtime complexity to be approximated to $\mathcal{O}(N{n_s}^3)$. With this, the computational intensity of the RTS smoother will increase in a cubic manner as the dimensions of the state vector increase.

4.4.1 Complexity of FRTS

The performance gain of this algorithm relies highly on the frequency of the accuracy renewal. In other words, lower performance gain will be obtained if a higher accuracy is needed. The FLOPs count of the FRTS is listed in Table 4.2. Apart from the accuracy renewal module, all operations are only needed to be calculated once for every time step as opposed to N times as in cRTS. The total FLOPs of the FRTS Algorithm Ł$_2$ without accuracy renewal is given as

$$Ł_2 = 33{n_s}^3 - 3{n_s}^2 \tag{4.22}$$

TABLE 4.2: Computational Complexity of the FRTS

Operation	**FLOPs**	**Type**
Calculation of C_k:		
$P_{k\mid k}A_k^T$	$2{n_s}^3 - {n_s}^2$	MMM
$P_{k+1\mid k}^{-1}$	$\approx {n_s}^3$	MI
$C_k = P_{k\mid k}A_k^T P_{k+1\mid k}^{-1}$	$2{n_s}^3 - {n_s}^2$	MMM
Calculation of b_k:		
$C_k\hat{x}_{k+1\mid k}$	$2{n_s}^2 - n_s$	MVM
$b_k = \hat{x}_{k\mid k} - C_k\hat{x}_{k+1\mid k}$	n_s	VVA
Calculation of D_k:		
$C_k P_{k+1\mid k}$	$2{n_s}^3 - {n_s}^2$	MMM
$C_k P_{k+1\mid k}C_k^T$	$2{n_s}^3 - {n_s}^2$	MMM
$D_k = P_{k\mid k} - C_k P_{k+1\mid k}C_k^T$	${n_s}^2$	MMA
Calculation of Ψ, Ω, Γ and Δ:		
Ωb_k	$2{n_s}^2 - n_s$	MVM
$\Psi = \Psi + \Omega b_k$	n_s	VVA
ΩD_k	$2{n_s}^3 - {n_s}^2$	MMM
$\Omega D_k \Delta$	$2{n_s}^3 - {n_s}^2$	MMM
$\Gamma = \Gamma + \Omega D_k \Delta$	n_s^2	MMA
$\Omega = \Omega C_k$	$2{n_s}^3 - {n_s}^2$	MMM
$\Delta = C_k^T \Delta$	$2{n_s}^3 - {n_s}^2$	MMM
Calculation of $\hat{x}_{k-N\mid N}$ and $\hat{P}_{k-N\mid N}$:		
$\Omega\hat{x}_{k\mid k}$	$2{n_s}^2 - n_s$	MVM
$\hat{x}_{k-N\mid N} = \Psi + \Omega\hat{x}_{k\mid k}$	n_s	VVA
$\Omega P_{k\mid k}$	$2{n_s}^3 - {n_s}^2$	MMM
$\Omega P_{k\mid k}\Delta$	$2{n_s}^3 - {n_s}^2$	MMM
$P_{k-N\mid N} = \Gamma + \Omega P_{k\mid k}\Delta$	n_s^2	MMA
Calculation of Ψ, Ω, Γ and Δ:		
C_{k-N}^{-1}	$\approx {n_s}^3$	MI
$\Omega = C_{k-N}^{-1}\Omega$	$2{n_s}^3 - {n_s}^2$	MMM
$\Delta = \Delta C_{k-N}^{-T}$	$2{n_s}^3 - {n_s}^2$	MMM
$\Psi - b_{k-N}$	n_s	VVA
$\Psi = C_{k-N}^{-1}\Psi$	$2{n_s}^2 - n_s$	MVM
$\Gamma - D_{k-N}$	n_s^2	MMA
$C_{k-N}^{-1}\Gamma$	$2{n_s}^3 - {n_s}^2$	MMM
$\Gamma = C_{k-N}^{-1}\Gamma C_{k-N}^{-T}$	$2{n_s}^3 - {n_s}^2$	MMM
Accuracy Renewal Module Part 1 (Repeat $N-1$ times)		
Ωb_k	$2{n_s}^2 - n_s$	MVM
$\Psi = \Psi + \Omega b_k$	n_s	VVA
ΩD_k	$2{n_s}^3 - {n_s}^2$	MMM
$\Omega D_k \Delta$	$2{n_s}^3 - {n_s}^2$	MMM
$\Gamma = \Gamma + \Omega D_k \Delta$	n_s^2	MMA
$\Omega = \Omega C_k$	$2{n_s}^3 - {n_s}^2$	MMM
$\Delta = C_k^T \Delta$	$2{n_s}^3 - {n_s}^2$	MMM
Accuracy Renewal Module Part 2 (Repeat N times)		
Ωb_k	$2{n_s}^2 - n_s$	MVM
$\Psi = \Psi + \Omega b_k$	n_s	VVA

The FLOP counts for the accuracy renewal part 1 and 2 of the algorithm Ł$_2^{'}$ is given as

$$Ł_2^{'} = (N-1)(8{n_s}^3 - {n_s}^2) + 2Nn_s^2 \tag{4.23}$$

The speed-ups of FRTS algorithm, based on the FLOP counts, are compared to cRTS and is shown against a range of window sizes in Figure 4.5. The metric, Renewal Fraction S_r, is used to to better describe the Figure. S_r is computed as

$$S_r = \frac{T_r}{T} \tag{4.24}$$

where T is the total number of time steps and T_r is the number of times the expression for Ψ, Ω, Γ and Δ are re-computed. As shown in the graph, $S_r = 0.1$ can lead to speed-ups of over 10 times for FRTS. To demonstrate the effect of S_r on FRTS, different values of S_r are used. For $S_r = 0.8, 0.5$ and 0.1, a speed up of over $1.5, 2$ and 10 can be achieved, respectively. However, the speed up for different values of n_s does not have a big difference and all have speed up of 4 for $S_r = 0.2$.

4.5 Simulation and Evaluation

4.5.1 Simulated Scenario

Given that the techniques outlined in this chapter are not restricted to or based on any specific motion model, this thesis use a simple, yet effective scenario to demonstrate the efficacy of the proposed approach. Consider a scenario with a single target flying at constant altitude and constant velocity over. More specifically, this thesis set the state transition matrix A, measurement matrix H, state vector $\mathbf{x}$, process and measurement noise covariance matrices Q and R as follows:

$$A = \begin{pmatrix} 1 & 0 & § & 0 \\ 0 & 1 & 0 & § \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Furthermore, the initial velocities in the x- and y- directions of a target is randomly generated. In each time step, a normally distributed noise $\mathcal{N}(\mu, \sigma^2)$ with $\mu = 0$ and $\sigma = 100$m is added with $R = 100^2 m^2$. The noise level is independent from the accuracy of the FRTS algorithm as compared to the cRTS. The RMSE will increase if higher noise level is used for cRTS and hence so does FRTS. Therefore, a scenario with a single level of noise is sufficient.

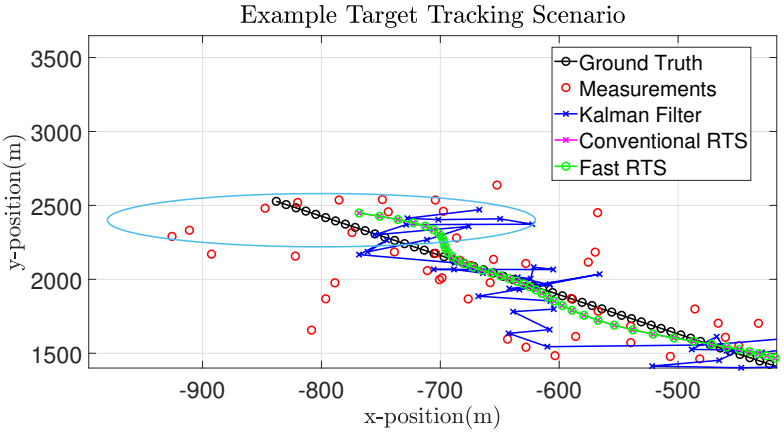

FIGURE 4.6: Example target tracking scenario

4.5.2 Evaluation Method

This thesis have tested the proposed algorithm on a state-of-the-art multi-core shared memory system, with an Intel KNL (7210) processor containing 256 cores and a memory of 96GB. Furthermore, this thesis relied on vendor-supplied or equivalent BLAS libraries that have already been parallelized and performance-tuned for the architecture this thesis have used. As such, they provide the best possible performance for a number of BLAS operations such as, MMM or MI. In particular, this thesis used the EIGEN Library (v3.3.3) for the evaluation. All experiments are repeated 100 times and the mean of the results are recorded.

4.5.3 Accuracy of FRTS

To evaluate the accuracy of the FRTS approach, a window size of 8 is used for 1000 time steps. The relative error threshold θ is set to 0.1 which represents degree of error tolerance as explained in (4.20). The tracking scenario is shown in Figure 4.6. The relative error of the FRTS compared to the cRTS is shown in Figure 4.7, in log scale. As can be observed from the Figure, the relative error is minimal here, and this error is bound within 0.1, comparing to cRTS smoother estimate. Moreover, the relative error of the norm of the covariance of cRTS and FRTS is compared and plotted in Figure 4.14 which also shows the error is bound. In this simulation, accuracy renewal step is activated 24 times to restore the accuracy of the algorithm ($S_r = 24/992 \approx 0.024$) as can be visualised in Figure 4.13 as condition number is being renewed. In the evaluation of the smoother, 100 Monte Carlo runs were used for the trajectory and the total performance measure is the mean value of the average position and velocity error. The mean x-position, y-position, x-velocity and y-velocity error are $2.25 \times 10^{-14}, 2.15 \times 10^{-14}, 4.03 \times 10^{-12}$ and 2.89×10^{-12} respectively. Same experiments are conducted with

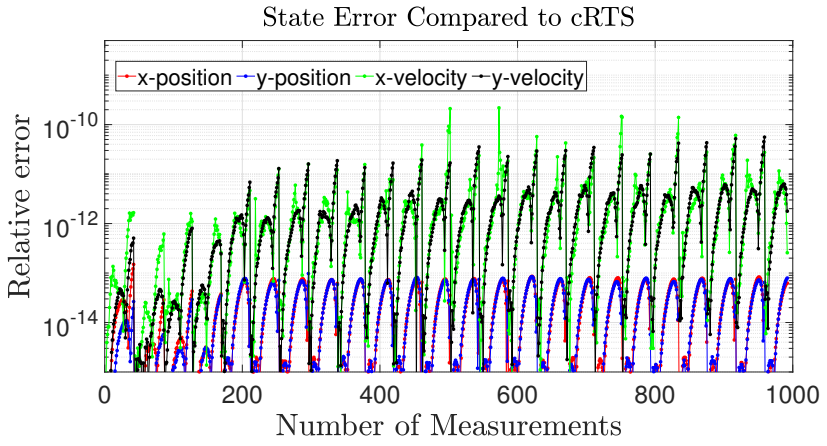

FIGURE 4.7: Relative error of state for FRTS ($\theta = 0.1$) (log scale)

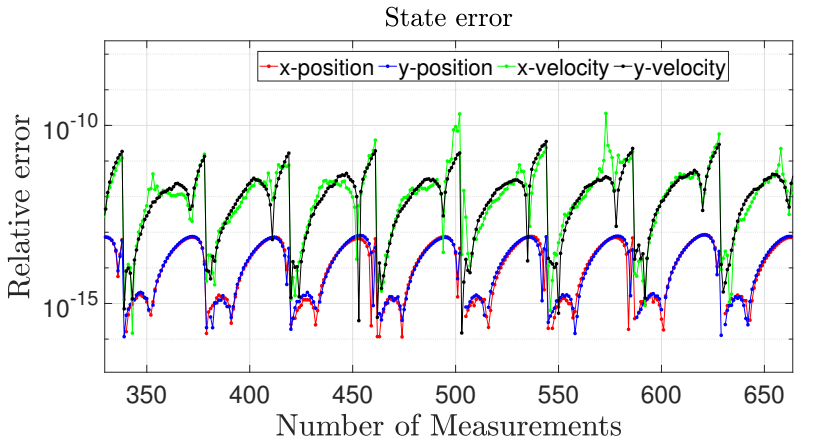

FIGURE 4.8: Relative error of state for FRTS (Magnified) ($\theta = 0.1$) (log scale)

relative error threshold θ set to 1 to compare the difference. As shown in Figure 4.16 to 4.20, the frequency of accuracy renewal is lower as the error tolerance is higher. Comparing with $\theta = 0.1$, the accuracy renewal step is activated 4 times only as shown in the Figures. Also note that in Figure 4.16, the relative error is bound to 1 as compared to Figure 4.7, the relative error is bound to 0.1 as different values of θ is used. This experiment also shows that there is a trade off between accuracy and complexity of the proposed algorithm as less accuracy renewal step (less calculations) is activated if higher error tolerance is allowed.

4.5.4 Speed-Up of FRTS Algorithm

In this section, this thesis compare the runtime performance of the proposed algorithms FRTS to cRTS. This thesis recorded the runtimes of these algorithms for processing the simulated scenario across 10000 time steps for a range of window sizes from 2 to 512. Then, they are compared to the corresponding runtimes of the conventional RTS. This

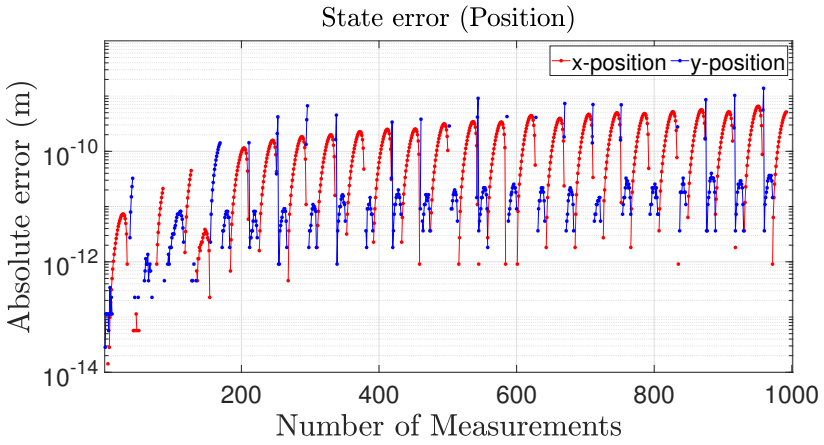

FIGURE 4.9: Absolute error of state (position) for FRTS ($\theta = 0.1$) (log scale)

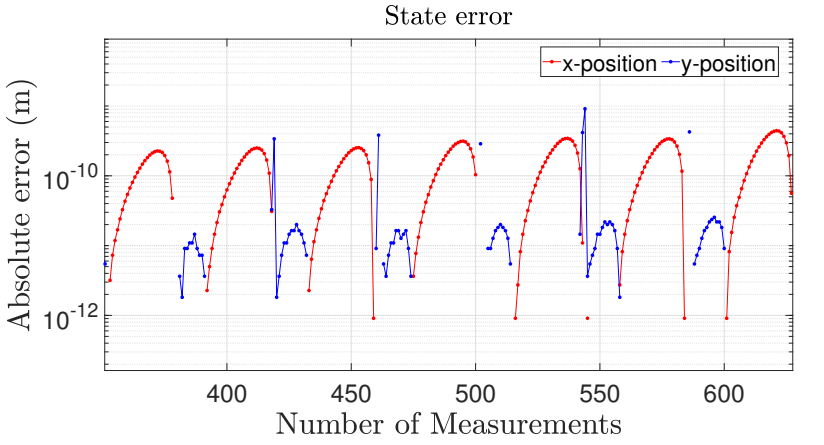

FIGURE 4.10: Absolute error of state (position) for FRTS (Magnified) ($\theta = 0.1$) (log scale)

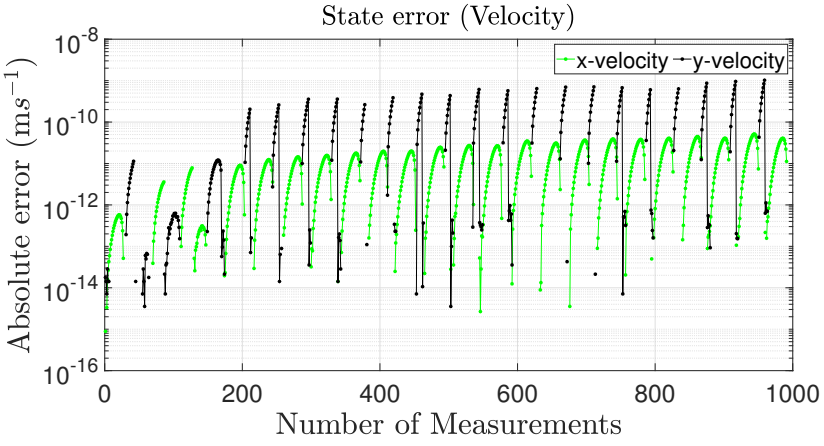

FIGURE 4.11: Absolute error of state (velocity) for FRTS ($\theta = 0.1$) (log scale)

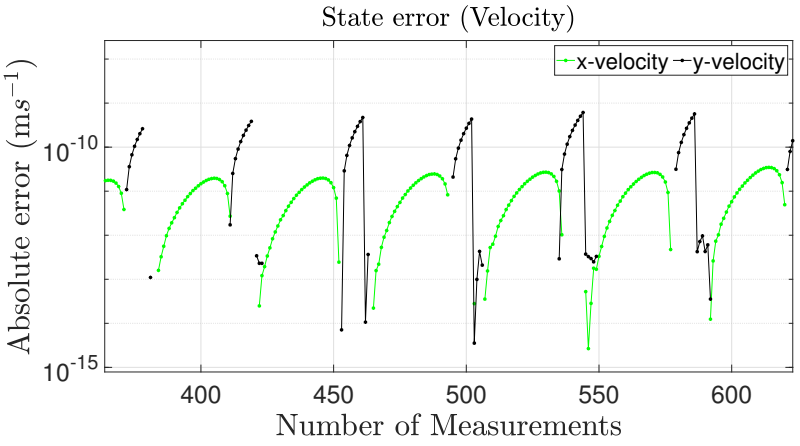

FIGURE 4.12: Absolute error of state (velocity) for FRTS (Magnified) ($\theta = 0.1$) (log scale)

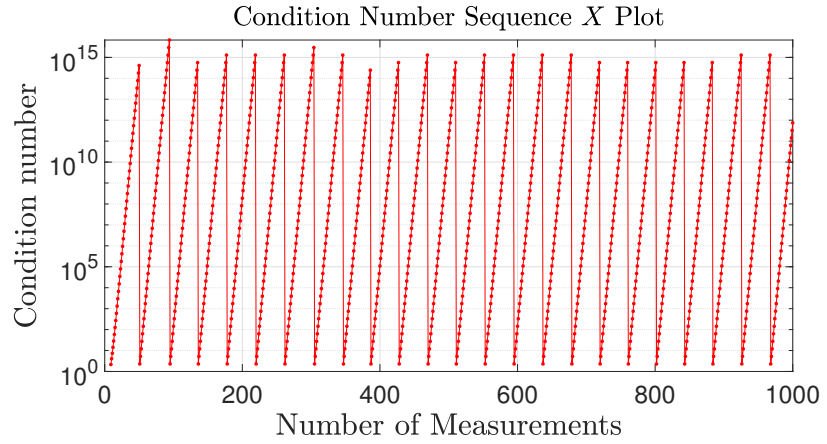

FIGURE 4.13: Variation of condition number for FRTS ($\theta = 0.1$) (log scale)

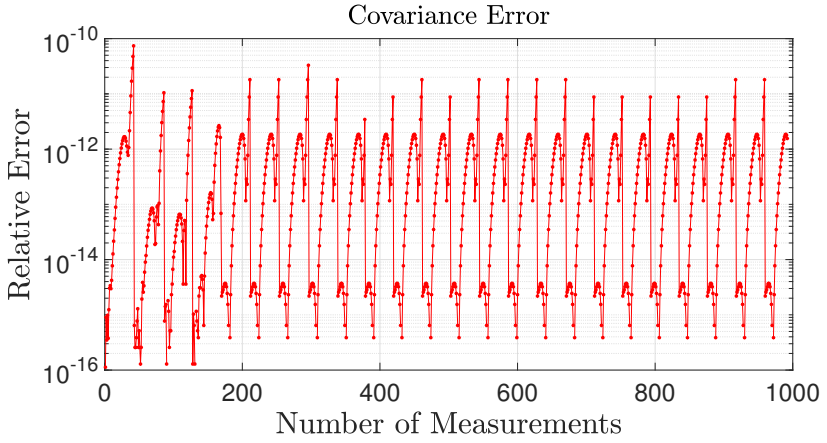

FIGURE 4.14: Relative error of covariance between cRTS and FRTS ($\theta = 0.1$) (log scale)

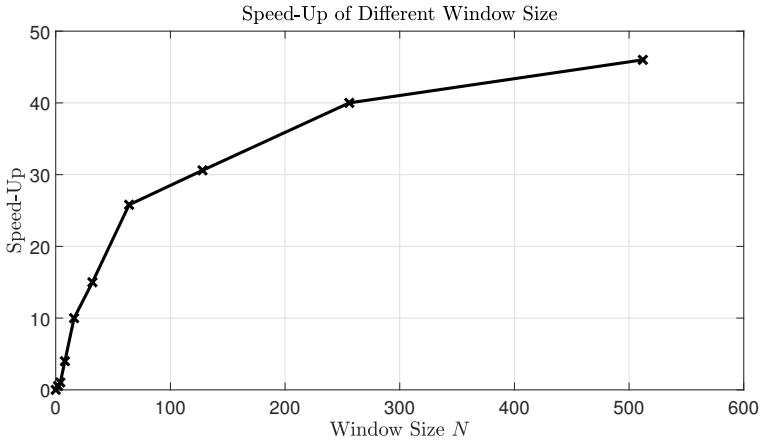

FIGURE 4.15: Performance gain of proposed algorithms

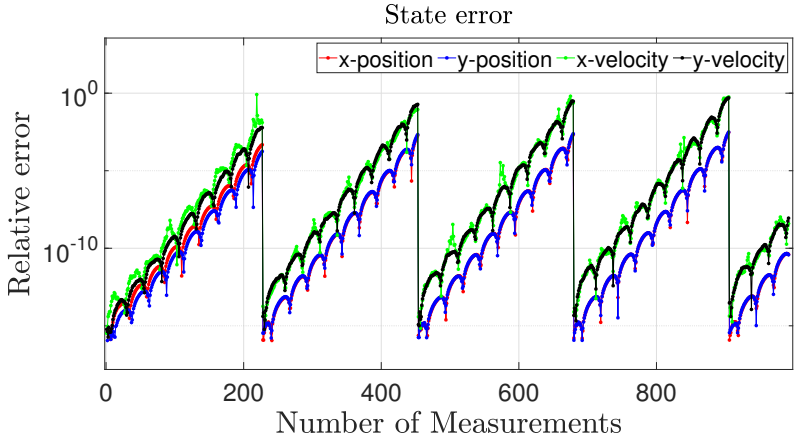

FIGURE 4.16: Relative error of state for FRTS ($\theta = 1$) (log scale)

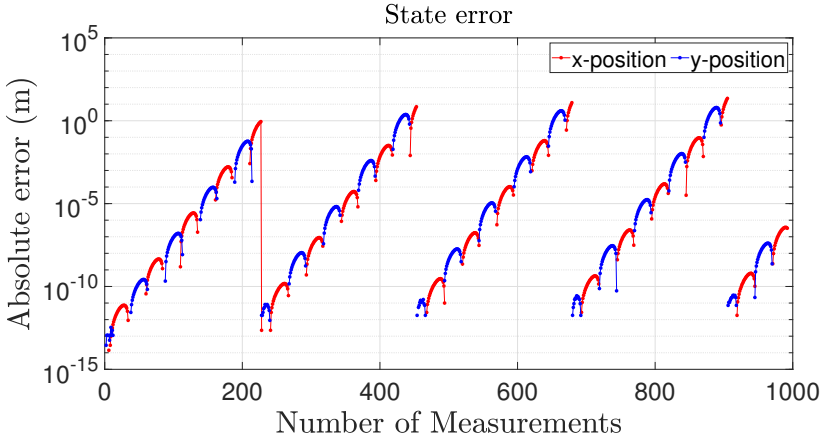

FIGURE 4.17: Absolute error of state (position) for FRTS ($\theta = 1$) (log scale)

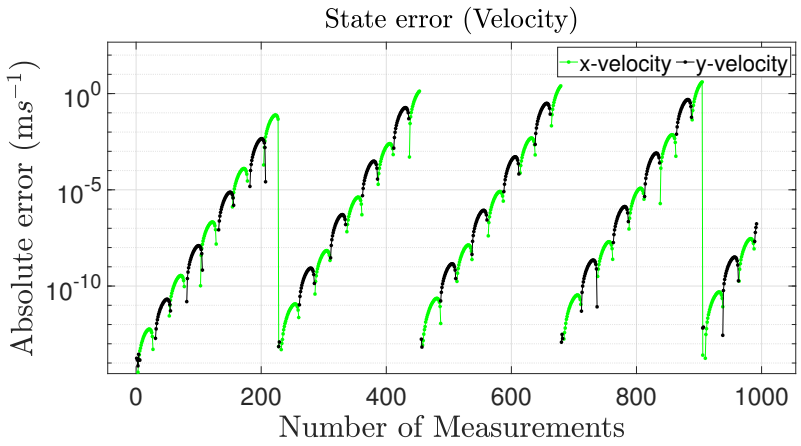

FIGURE 4.18: Absolute error of state (velocity) for FRTS ($\theta = 1$) (log scale)

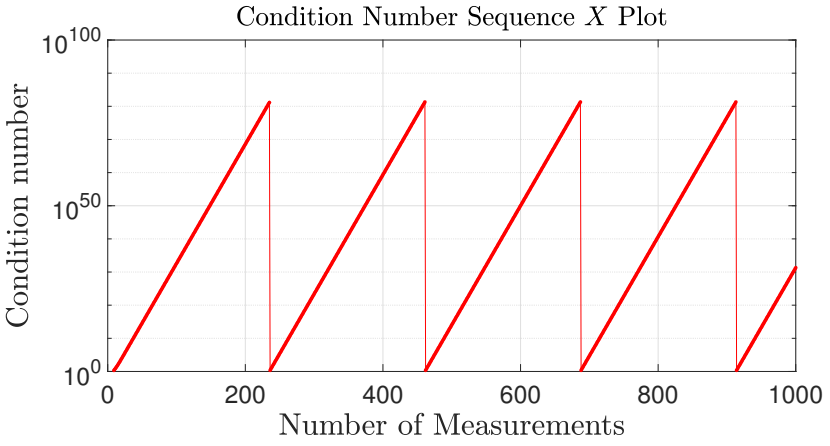

FIGURE 4.19: Variation of condition number for FRTS ($\theta = 1$) (log scale)

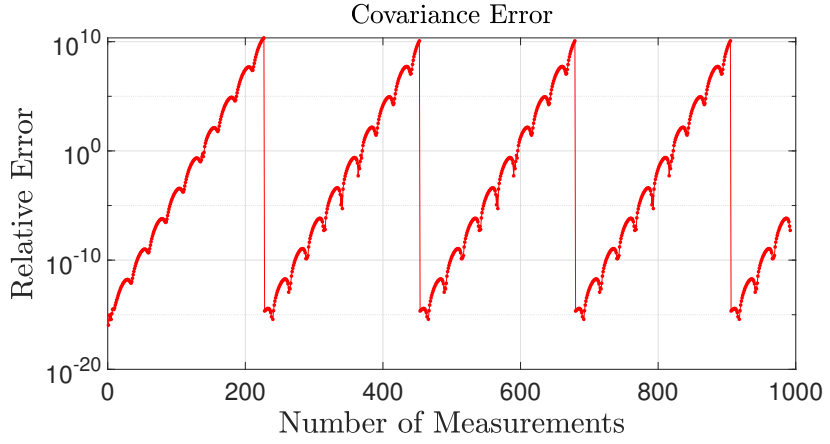

FIGURE 4.20: Relative error of covariance between cRTS and FRTS ($\theta = 1$) (log scale)

TABLE 4.3: Runtimes (in seconds)

	Window Size				
	2	8	32	128	512
FRTS	4.877	4.928	5.229	5.387	7.419
cRTS	3.411	8.095	26.33	98.45	372.1

thesis then compute the runtime speedup S_t, as

$$S_t = \frac{t_f}{t_c}$$

where t_f denotes the runtime of the algorithm of FRTS and t_c being the runtime of the cRTS algorithm. This thesis show the overall speedup in Figure 4.15. To provide a detailed information, this thesis also show the runtimes in Table 4.3. As can be observed from the graph, FRTS shows superior performance gain when compared against the cRTS. Moreover, it can be observed that the runtimes of the FRTS are almost the same across range of window sizes, with significant speedups when compared to the cRTS. The speedup increase with window size and reaches over 50 times speed-up for the window size of 512. The value of S_r is approximately equal to 0.02 and so the proposed algorithm FRTS has an edge in complexity over cRTS.

4.5.5 Multi-Target Scenario

The proposed algorithm will also work on multi-target tracking. Since the RTS smoother is irrelevant to data association which is a step to assign observed measurements to existing tracks, the multi-target scenario considered here will be independent. The starting position and velocity of the targets are randomly generated from an uniform distribution. 5 targets are moving in constant velocities for 1000 time steps and are shown in Figure 4.21 which shows FRTS can track the target without huge error as in Figure 4.1. As before, addictive Gaussian noise of variance of $100m$ is added to each time step. Window size of 8 for the fixed-lag FRTS smoother. The tracking result is plotted in Figure 4.21. The accuracy of the FRTS and cRTS are compared. To clearly observe the change of propgated error. The x-positions and y-positions estimated by FRTS for all 5 targets and 300 time steps are plotted in Figure 4.22. It can be observed that the error is lower than θ which is set to 0.1. For x-, y- velocities, the mean of the estimated value is provided in Table 4.4 to compare with Ground Truth values. Moreover, the absolute error of x-, y- positions are plotted in Figure 4.24 and 4.25.

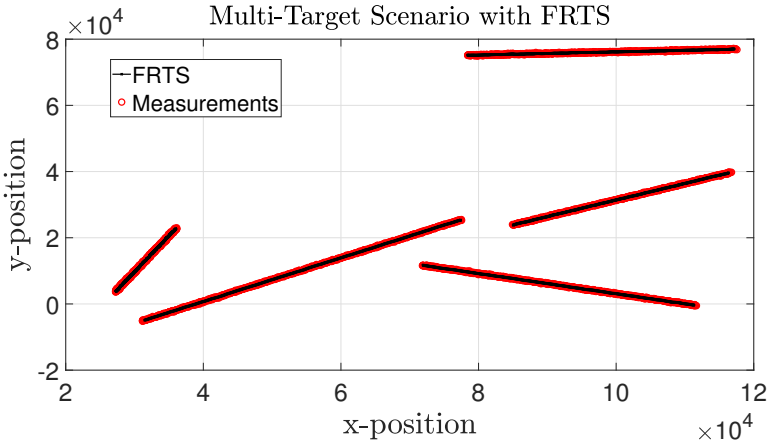

FIGURE 4.21: Multi-Target Tracking Scenario with FRTS

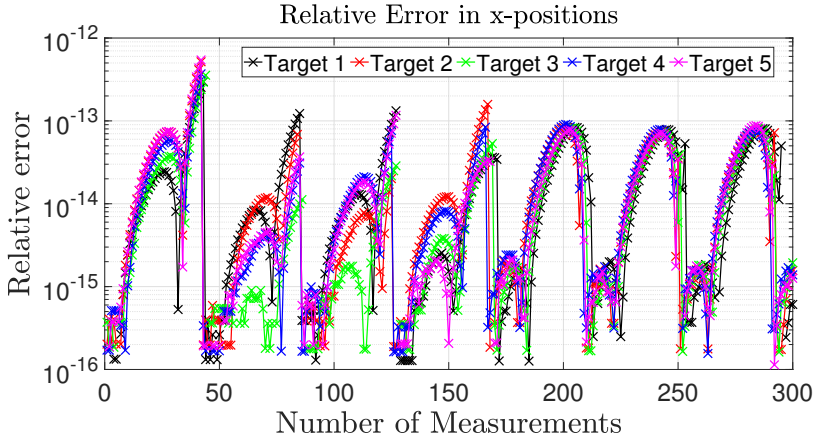

FIGURE 4.22: Relative Error of x-positions (log scale)

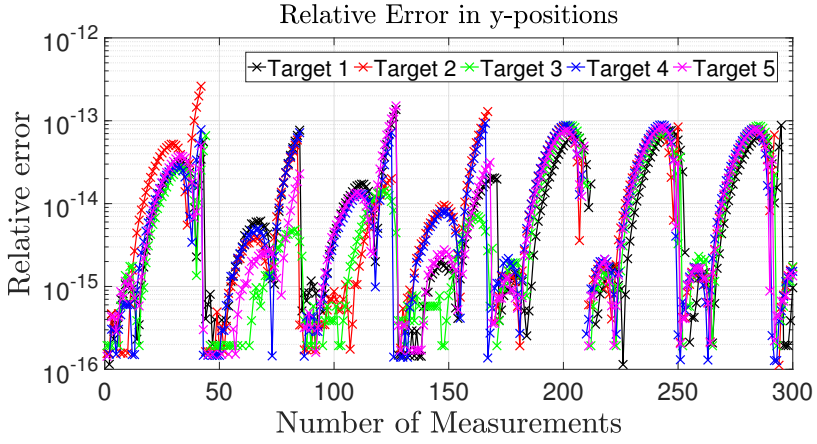

FIGURE 4.23: Relative Error of y-positions (log scale)

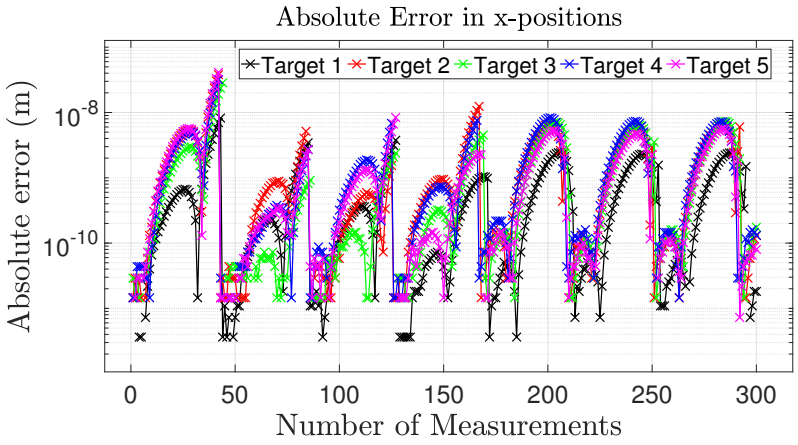

FIGURE 4.24: Relative Error of x-positions (log scale)

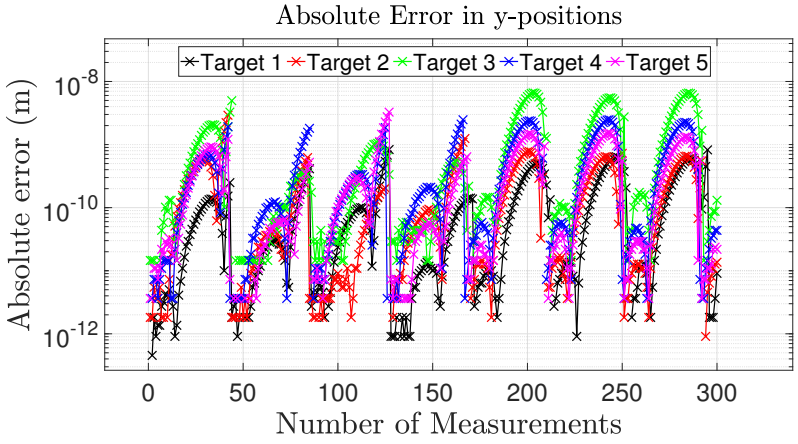

FIGURE 4.25: Relative Error of y-positions (log scale)

TABLE 4.4: Velocities of Targets

	Target				
	1	2	3	4	5
x-vel (GT)	8.81	39.8	39.2	31.6	-46.4
y-vel (GT)	19.1	-12.1	1.85	15.8	-30.6
x-vel (FRTS)	8.74	39.7	39.1	31.6	-46.3
y-vel (FRTS)	18.8	-12.2	1.86	15.9	-30.4

4.6 Conclusions

Retrodiction or smoothing is an essential process in the context of tracking and often becomes embedded as part of the of the Kalman filter estimation process. The RTS smoother is one of the most widely used smoothing algorithm in the tracking community. However, a naïve acceleration of the RTS algorithm, $fRTS^{-}$, can become numerically unstable across window sizes, or across time steps. In this chapter, this thesis developed an algorithm, FRTS, to address the shortcomings of $fRTS^{-}$. This thesis showed that stability issue can be identified by the condition number, and how this can be used to monitor the progression of accuracy errors across time steps.

This thesis presented a detailed analysis on their computational complexities, showing how FRTS can offer a significant performance advantage over the cRTS. This thesis also verified these claims using a synthetic, yet realistic, evaluation, showing how the accuracy errors can be contained within 10^{-9} percentage while offering over 50 times speedup in runtime performance. This thesis show that these algorithms offer superior performance both in terms of accuracy and computational complexity. This thesis shows that FRTS is more capable than cRTS to handle a large number of targets within multi-target tracking systems in real time.

Chapter 5

Cost-effective Multiple Model Tracking

5.1 Introduction

Tracking algorithms require an accurate approximation of the motion of the targets. The Multiple Model estimator has proven to be useful in tracking targets with different modes of motion and has been widely researched [31, 32, 106–108]. MM includes all the possible motion models in the algorithm and a bank of filters are operated on each of the models. The motion estimation process works in two stages: prediction and update. In the prediction stage, the state at the current time step is used to predict the state at the next time step. This predicted state is then updated (or corrected) using the measurement obtained at the subsequent time step during the second stage. Among different filters that can be used for this predict-update process, the Kalman Filter [24] is a popular basis for handling linear, non-linear and non-Gaussian models through necessary extensions [26]. When the dynamics of the targets can switch among different modes, it is more robust to have more than one motion model in the tracking algorithm and hence the existence of many multiple-model (MM) algorithms.

In general, MM algorithms are formed by running several filters with different motion model in each of them independently that match different maneuvering movement. The output of these filters are fused to create an overall estimate. Each of the MM algorithms have different configuration of fusing the estimates. The underlying structure of MM algorithms can be explained as follows [18]

1. Model-set determination: This includes the choice of the model set incorporated in the MM algorithm to achieve better estimates. Depends on the circumstances,

it can be fixed-structure of variable-structure. Fixed-structure denotes that the number of models is fixed in the MM algorithm whereas variable-structure varies with time from a fix pool of model depends on which suits the algorithm the best at each give time.

2. Cooperation strategy: This represents the strategy to deal with the uncertainties in the model used which includes pruning of unlikely model, merging of similar model and selection of the most likely model sequence. Also, iterative method can be used such as expectation-maximization (EM) algorithm.

3. Output process: This generates overall esitmates based on all filters from number of models and consider the strategy of fusing the best ones from them.

With different strategies, MM algorithms can be catogorized into three generations [109]. Different generations have different limitations and characteristics. The first generation MM method was proposed by Magill and others [29, 30]. It mainly relies on the fact that each elemental filter operates individually and independently. Then the output of each individual filter are fused to produce the final estimate. The most well known of all is autonomous MM (AMM) algorithm. The second generation extends the idea of first generation MM method by introducing reconditioning of each filter at each time step. For example, in the Interacting Multiple Model (IMM) algorithm, there is a reinitialization step prior to the filtering step. The third generation introduces the notion of variable structure, which allows varying number of models in each time step. It eliminates the models that are not matched with the true modes of motions. This generation is known as Variable Structure Multiple Model (VSMM). Amongst them, IMM is the most well known and the most cost-effective due to its simplicity and effectiveness [110–114]. All of the MM filtering stages involve using L number of filters and hence the complexity of MM filtering algorithm is in general $O(Ln^3)$. However, some of the algorithms, for example, GPB2 algorithm needs L^2 operations for filtering operations [18].

In cases where a delay in estimation can be tolerated, the Multiple Model estimation accuracy can be also improved rather significantly by incorporating a future measurement. This process is known as Multiple Model smoothing or retrodiction. Similar to single model tracking, the accuracy improves with the retrodiction window size. Several methods which incorporated smoothing in the MM regime is presented in [57, 115–117]. In [115, 117], the authors proposed a two-filter methods which combine the forward and backward time filters to calculate the smoothed estimates. The backward time filter often become the bottleneck of this approach as the inverse of target dynamics are needed to be defined. In [116], augmented filters as mentioned in [90, 91] were used to combine with the MM approach. However, this method is not widely used as there is

no mode jump between models. Rauch-Tung-Striebel (RTS) smoothing [25] based MM smoothing is presented in [57]. This approach use the conventional IMM forward filter and then fuse with the RTS backward smoothing by merging the smoothed estimates with different hypothesis. In general, the smoothed estimate is obtained by merging estimates from a L smoothers with L being the number of models [118].

In many cases, a large number of models are required to cover the entire range of the possible modes of motion which can become computationally infeasible. To reduce this complexity, several papers have presented different methods in tackling this problem. Most of the work relies on the assumption that the system would approach to steady-state and so a constant Kalman Gain can be used for each of the filter bank. For instance, in [20], a comparative study is presented to show the computation superiority of the constant gain approach. Furthermore, a constant gain approach of the IMM (fastIMM) based on α-β and α-β-γ filters are described in [21]. Besides, another fixed gain approach at which the Kalman gain is calculated off-line prior to the tracking [22]. Since the IMM is regarded as the most cost-effective MM filter in the tracking community, a lot of work has been done to improve the complexity of such an algorithm. However, not much has been explored in the smoother space to find a more cost-effective tracking algorithm as a whole alongside smoothing. Especially, for smoothing, at each time step, a multiple model algorithm with L models has to be executed N times and so smoothing in MM settings can be a computation of bottleneck when L, N or the number of targets increases. This chapter try to seek a cost-effective real-time approach of MM tracking algorithms including the fixed-lag smoothing stage. Not only do the proposed algorithms have unprecedented complexity advantages, they also do not rely on constant gain to minimise computations. The key contributions of this Chapter are:

- Autonomous MM Fixed-Lag Smoother (AMMS):
  This algorithm is similar to the idea of AMM filter but in smoother form. It utilises the RTS smoother as the backbone and L number of smoothers to produce the smoothed estimate. It also eliminates the reconditioning step of IMMS as the Markov process has already been included in the filtering step if second generation filter is used. Adding another reconditioning step in smoother would be redundant in terms of cost-effectiveness as shown in the evaluation section.

- Fast MM Fixed-Lag Smoother 2 (FMMS):
  This algorithm makes the computation complexity independent of number of models L which result in $O(Nn^3)$ instead of $O(LNn^3)$ as in the conventional method.

- Complexity Analysis of IMM and proposed algorithms:
  A detailed complexity analysis is provided to show the computational advantage of the proposed algorithm over the conventional approach.

- Tracking Scenarios Simulations:
  Two scenarios with complex manoeuvring patterns are simulated to see the runtime and accuracy performance of the proposed approach. Also, a performance index to calculate cost-effectiveness is suggested to quantify and to find a more cost-effective MM tracking algorithm alongside with smoothing procedures.

This evaluations, based on a number of simulations, show that all the proposed approaches offer superior performance in terms of runtime and accuracy when compared against the conventional approach. These results show that the proposed approach not only offers good complexity performance, but can also provide more time of tolerance for smoothing, leading to better state estimates as shown in Section 5.6. The rest of the Chapter is organized as follows: Section 5.2 concentrate on introducing the AMM Filter which serve as a background for the sections that follows this. Section 5.3 and 5.4 focus on deriving the proposed smoother algorithms: AMMS and FMMS. After that, a detailed complexity analysis is presented in Section 5.5 to show the benefits on complexity of the proposed algorithms. Furthermore, two complex tracking scenarios are used in Section 5.6 to compare the runtime and accuracy of the algorithms and finally, Section 5.7 conclude the findings.

5.2 Background

5.2.1 Autonomous Multiple Model Filter

The Autonomous Multiple Model Filter (AMMF) [109] gives the MMSE estimate $\hat{x}_{k|k} = E[x_k|z^k, m_j^{k-1}]$ where m_j^{k-1} is the true model at time $k-1$ and $j = 1, ..., L$. It is assumed that the true system mode is time invariant and identical to one of the finite model set used. This assumption separates first and second generations like IMM since IMM assume the true mode sequence is Markov or random which allow a time-varying mode sequence. The AMM algorithm runs a Kalman Filter (KF) on each of the model and so the posterior probability of each model is calculated. Then, the model probabilities are used to weight each of the models to produce the final estimate. In other words, each of the model is operated independently and they only interact with each other in the output stage. The AMMF recursion is executed as below

1. Mode-conditioned filtering stage:
   KF will be used on each of the mode-model m^j with initial condition $x^j_{k-1|k-1}$ and $P^j_{k-1|k-1}$

2. Mode probability update stage:
   The mode probability can be calculated as

$$\mu_k^j = \frac{\mu_{k-1}^j L_k^j}{\sum_{i=1}^N \mu_{k-1}^i L_k^i} \tag{5.1}$$

   where $L_k^j = \mathcal{N}(\hat{z}_{k|k-1}^j,\, S_k^j)$

3. Estimate fusion stage:
   The estimate of the final output are mixed from each of the filter with corresponding mode probability

$$\begin{aligned} \hat{x}_{k|k} &= \sum_{j=1}^N \mu_k^j \hat{x}_{k|k}^j \\ P_{k|k} &= \sum_{j=1}^N \mu_k^j \{P_{k|k}^j + [\hat{x}_{k|k}^j - \hat{x}_{k|k}][\hat{x}_{k|k}^j - \hat{x}_{k|k}]^T\} \end{aligned} \tag{5.2}$$

Since AMMF only has one fusion stage where each bank of filter interact with each other and so it is very computationally attractive being the simplest form of MM algorithm. However, as it assumes true mode sequence is time invariant and so practical application can be limited.

5.3 Formulation of AMMS

The use of a fixed-lag smoother is essential in real time tracking. Even with a small lag, the smoother can outperform MM filtering significantly [119, 120]. The purpose of the MM smoother is to calculate a new sets of estimates and re-weight the output from each of the mode-filter in order to create an estimate with lower error in the case of model switching. However, with large number of models and window size, the MM algorithm can become a bottleneck in the smoothing step. In order to obtain a smoothing algorithm which has lower computation time than the IMMS but comparable accuracy, a smoothing algorithm is investigated based on AMM smoothing, which is referred as AMMS. The AMM state estimator is given by the total expectation theorem,

$$\begin{aligned} \hat{x}_k &= E[x_k|z^k] \\ &= \sum_{j=1}^N E[x_k|z^k, m_j^k] P\{m_j^k|z_k\} \end{aligned} \tag{5.3}$$

Also, the covariance $P_{k|k}$ will be given as

$$P_{k|k} = MSE(\hat{x}_{k|k}|z^k) \tag{5.4}$$

Similarly, in backward smoothing for AMMS, the smoothed density will be given as $p(x_t|z^k)$ where $t < k$. The expectation of this density will be

$$\begin{aligned}\hat{x}_{t|k} &= E[x_t|z^k] \\ &= \sum_{i=1}^{N} E[x_t|z^k, m_i^k]P\{m_i^k|z_k\} \\ &= \sum_{i=1}^{N} \hat{x}_{t|k}^i \mu_k^i \qquad (5.5)\end{aligned}$$

$$\tag{5.6}$$

Also, the covariance $P_{t|k}$ can be given as

$$\begin{aligned}P_{t|k} &= MSE(\hat{x}_{t|k}|z^k) \\ &= \sum_{i=1}^{N} [P_{t|k}^i + (\hat{x}_{t|k}^i - \hat{x}_{t|k})(\hat{x}_{t|k}^i - \hat{x}_{t|k})^T]\mu_k^i \qquad (5.7)\end{aligned}$$

RTS smoother can be used to calculate $\hat{x}_{t|k}^i$ and $P_{t|k}^i$ for each model. The smoothing algorithm recursion for one step is executed as below

1. Smoothing stage:
   RTS smoother will be used on each of the mode-model m^i, where $i = 1, ..., L$ with initial condition $x_{k+1|k+1}^i$ and $P_{k+1|k+1}^i$

2. Mode probability update stage:
   The mode probability calculation is similar to (5.1) which is given as

   $$\mu_{t|k}^i = \frac{\mu_{t+1|k}^i L_t^i}{\sum_{j=1}^{N} \mu_{t+1|k}^j L_t^j} \tag{5.8}$$

   where $L_t^i = \mathcal{N}(\hat{x}_{t+1|k}^i; \hat{x}_{t+1|t}^i, P_{t+1|t}^i)$

3. Estimate fusion stage:
   The smoothed of the final output are mixed from each of the jth smoother with

corresponding mode probability

$$\begin{aligned} \hat{x}_{t|k} &= \sum_{j=1}^{N} \mu_{t|k}^{j} \hat{x}_{t|k}^{j} \\ P_{t|k} &= \sum_{j=1}^{N} [P_{t|k}^{j} + (\hat{x}_{t|k}^{j} - \hat{x}_{t|k})(\hat{x}_{t|k}^{j} - \hat{x}_{t|k})^{T}] \mu_{t|k}^{i} \end{aligned} \tag{5.9}$$

The smoothing regime of AMMS is shown in Figure 5.1. Three models with three different initial estimates, denoted as dots, are smoothed on the right of the Figure by the RTS smoother individually. Then, the smoothed estimates of each smoother, denoted as squares, are fused by Equation (5.9). As shown in Figure 5.2, IMMS reinitialize each of the smoother estimates before smoothing based on the smoothed estimate from other smoother models. Therefore, AMMS is more computationally attractive than IMMS since AMMS has omitted the reinitialization stage. This omission is due to the fact that the Markov process has already been assumed in the filtering stage if a second generation filter is used. Therefore, in terms of cost-effectiveness, the reconditioning step in smoothing stage might be redundant to achieve real-time performance. The smoother scheme by using the GBP1, which refer as GBP1S is also displayed in Figure 5.3 for comparison. The main difference is that GBP1S use the fused estimate $\hat{x}_{k-1|k-1}$ for the next time step. The algorithm of AMMS for time lag L is displayed in Algorithm 7.

Algorithm 7 AMMS Algorithm

for `k=0; k<T; k++` **do**
  ▷ Forward Pass: MM Filtering
  $[P_{k+1|k+1}^{i}, P_{k+1|k}^{i}, \hat{x}_{k+1|k+1}^{i}, \hat{x}_{k+1|k}^{i}] =$
    $Filter[\hat{x}_{k|k}^{i}, P_{k|k}^{i}]$
  ▷ Backward Pass: AMMS
  if $k > L$ **then**
    for `t=k-1; t>=k-L; t--` **do**
      ▷ Smoothing stage
      $[P_{t|k}^{i}, \hat{x}_{t|k}^{i}] =$
        $RTS[\hat{x}_{t+1|t}^{i}, P_{t+1|t}^{i}, \hat{x}_{t|t}^{i}, P_{t|t}^{i}]$
      ▷ Mode probability update stage
      Equation (5.8)
      ▷ Estimate fusion stage
      Equation (5.9)
    end for
  end if
end for

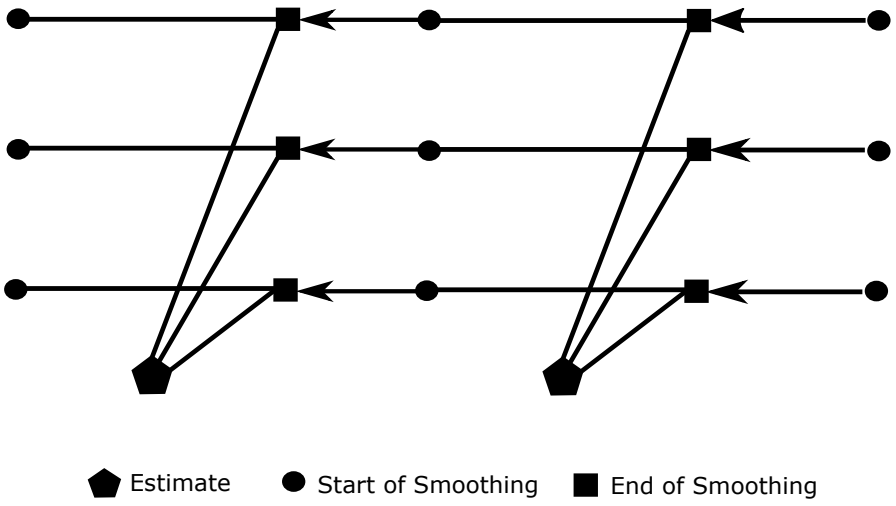

FIGURE 5.1: Operations of the AMMS

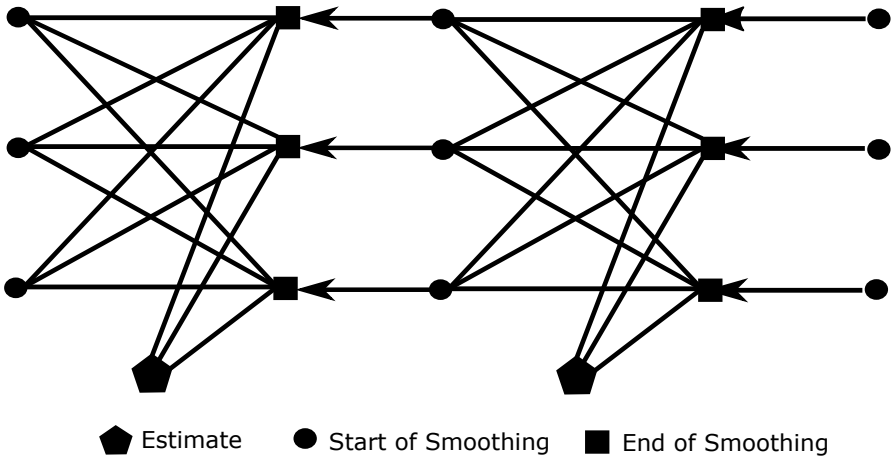

FIGURE 5.2: Operations of the IMMS

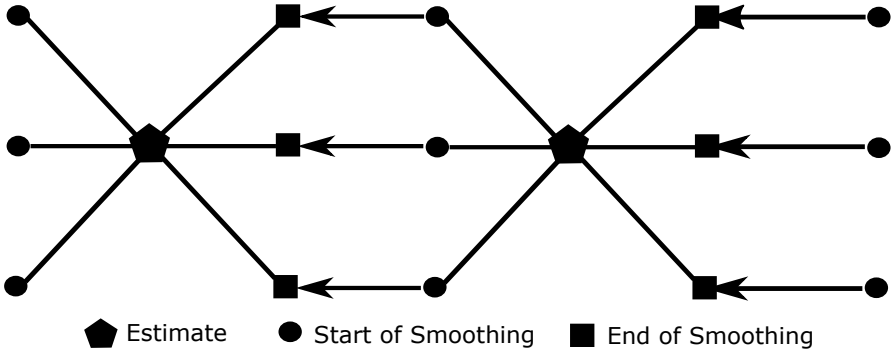

FIGURE 5.3: Operations of the GBP1S

5.4 Formulation of the FMMS

Extending the idea from the AMM and the AMMS, this section derives a new method of smoothing which is part of the contribution of the thesis. Apart from smoothing each of the models and fusing the estimate at each step, it is possible to fuse the statistics prior to the smoothing stage in order to remove the computational burden of smoothing N models at each step. The MMSE estimate from the estimate fusion stage as shown in (5.2) pre-fuses the estimate in order to remove the computational burden of smoothing N models at each step. By looking at the RTS smoother equations in (**??**) to (**??**), the statistics that is needed to implement the RTS smoother are $\hat{x}_{k|k}$, $\hat{x}_{k+1|k}$, $P_{k|k}$ and $P_{k+1|k}$. For $\hat{x}_{k|k}$ and $P_{k|k}$, the MMSE estimate from the estimate fusion stage as shown in (5.2) can be used. Same principle can be applied to the predicted statistics, $\hat{x}_{k+1|k}$ and $P_{k+1|k}$. By using the total probability theorem,

$$\hat{x}_{k+1|k} = \sum_{j=1}^{N} \mu_{k|k}^{j} \hat{x}_{k+1|k}^{j} \tag{5.10}$$

$$\begin{aligned} P_{k+1|k} = \sum_{j=1}^{N} \mu_{k|k}^{j} \{ & P_{k+1|k}^{j} + [\hat{x}_{k+1|k}^{j} - \hat{x}_{k+1|k}] \\ & [\hat{x}_{k+1|k}^{j} - \hat{x}_{k+1|k}]^{T} \} \end{aligned} \tag{5.11}$$

However, a new smoother gain needs to be derived. This is because for the usual MM algorithm, each of the mode model A^i is used to calculate the smoother gain individually to smooth each of the mode estimate. In order to obtain an MM smoother that has the complexity independent to the number of models N, the smoother gain needed to be fused before the implementation of the smoother step. The smoother gain from each mode filter can be calculated by

$$C_k^j = P_{k|k}^j A_k^{j\,T} \left(P_{k+1|k}^j\right)^{-1} \tag{5.12}$$

Since smoother gain is just a weight and therefore can be re-weighted based on the model probability $\mu_{k|k}^j$. To fuse the smoother gain C_k^j, total probability theorem can be used which gives the mixed gain Φ

$$\Phi_k = \sum_{j=1}^{N} \mu_{k|k}^{j} C_k^j \tag{5.13}$$

Therefore, the estimates obtained from filtering can be smoothed by a simple RTS smoother with dynamic model according to the mode probability. Due to the fact that the estimates and predictions are mixed together prior smoothing step, the complexity

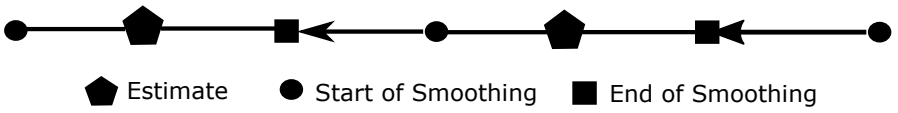

FIGURE 5.4: Operations of the FMMS

of number of models L can also be taken out which leads to complexity of $O(Nn^3)$. Note that, the calculations of (5.10) to (5.13) can be performed prior to the smoother stage. The FMMS algorithm is shown in Algorithm 8 and the smoothing regime is shown as Figure 5.4.

Algorithm 8 FMMS Fixed-Lag Smoother

for `k=0; k<T; k++` **do**
  ▷ Forward Pass: MM Filtering
  $[P^i_{k+1|k+1}, P^i_{k+1|k}, \hat{x}^i_{k+1|k+1}, \hat{x}^i_{k+1|k}] =$
    $Filter[\hat{x}^i_{k|k}, P^i_{k|k}]$
  Equation (5.10) *to* (5.13)
  ▷ Backward Pass: FMMS
  if $k > L$ **then**
    for `t=k-1; t>=k-L; t--` **do**
      $[P_{t|k}, \hat{x}_{t|k}] =$
        $FMMS[\hat{x}_{t+1|t}, P_{t+1|t}, \hat{x}_{t|t}, P_{t|t}]$
    end for
  end if
end for

5.5 Computational Complexity Analysis

In this section, this thesis provide a detailed analysis on the computational complexity of the Interactive Multiple Smoothing as well as the proposed algorithms, namely AMMS and FMMS. Both smoother and filtering operations often involve manipulation of matrices, and thus basic linear algebra operations such as matrix-matrix multiplication (MMM), matrix-vector multiplication (MVM), matrix inverse (MI), matrix-matrix addition (or subtraction) (MMA), element-wise operation (EWO) and vector-vector addition (or subtraction) (VVA). These are often referred to as Basic Linear Algebra Subroutine operations or BLAS operations [38]. For each of the algorithm, this thesis discuss the relevant BLAS operations. In addition to this, this thesis provide the number of floating point operations (FLOPs). A floating-point operation can be either a summation or multiplication. However, in practice, the real number of FLOP differs between them. In here, for simplicity, any operation is counted as a single FLOP.

5.5.1 Interactive Multiple Model Filtering and Smoothing

IMMF can be regarded as running several KF smoothers in parallel with some mixing procedures. The number of operations of KF have already been presented in [23] and Table 3.1 and so will not be analysed in this Chapter. In summary, a single iteration of KF needs the following operations.

- Eight matrix-matrix multiplications;
- Three matrix-vector multiplications;
- Three matrix additions (subtractions);
- Four vector additions (subtractions);
- Three matrix transpose operations; and
- One matrix inversions.

The FLOPs of IMMF are presented in Table 5.1. Prior to the stage of filtering for each model, there are two stages: calculation of mixing probabilities and the mixing stage. In the calculation of the mixing probabilities stage, the total predicted model probability $\bar{c}$ with size $L \times 1$ is calculated through MVM. When calculating mixing probabilities $\mu^{i|j}_{k-1|k-1}$ with size $L \times L$, each of the element in the matrix is calculated with 1 element wise multiplication and 1 division which leads to $2L^2$ FLOP. In the mixing stage, the estimate and its covariance are mixed with the corresponding probabilities. The mixing of estimate $\bar{x}^j_{k-1|k-1}$ with size $n_s \times L$ can be calculated as MMM. For the mixing of covariance, it calculates L numbers of covariances, each of them with size $n_s \times n_s$. At first, the difference of the mixed estimate and the previous estimate are calculated which leads to n_s FLOPs. There will be L^2 vector subtractions involved and so the total FLOP will be $L^2 n_s$. Then, L^2 vector and vector multiplications which is counted as $(Ln_s)^2$. Since vector transpose can be omitted with correct permutations, the FLOP involved can be omitted. For the addition of $P^i_{k-1|k-1}$ and vv^T, there will be L^2 number of additions of n_s^2 FLOPs. Finally each of the covariance component will be weighted by the probabilities which result in $(Ln_s)^2$. Then, there are $L \times (L-1)$ number of summations to calculate each of the model covariance. After that, the mixed estimates and covariance are processed by the filtering stage. The total number of FLOPs of KF is [23]

$$\mathbb{F}_{kf} = 6{n_s}^3 + 6n_m{n_s}^2 + 4n_s n_m^2 + n_m^3 + n_s n_m - n_s \tag{5.14}$$

where n_m represents number of dimensions. After the filtering stage, the likelihood of each of the model is used to update the mode probability. The normalising constant c

involves L element wise multiplication and $L-1$ summations and so $2L-1$ FLOPs. After that, the mode probability $\mu^j_{k|k}$ needs 1 multiplication and 1 division for each element in $\mu^j_{k|k}$ with total of L elements and so $2L$ FLOPs are needed. Finally, for the overall estimate, there are Ln_s element wise multiplication and $L-1$ additions of vectors with size n_s which leads to $2Ln_s - n_s$. For overall covariance, the procedure is similar to covariance mixing. However, instead of calculating covariance for each of the model, only one covariance is needed to calculate. Therefore, there will be L number of operations instead of L^2. As a result, the total number of FLOPs of IMMF including the filtering stage,

$$\begin{aligned}\mathbb{F}_{immf} = 4(Ln_s)^2 + 2L^2n_s + 3L(n_s)^2 + L(\mathbb{F}_{kf}) \\ 4L^2 + 3L + 2Ln_s - n_s^2 - n_s - 1\end{aligned} \tag{5.15}$$

For IMMS, the backbone of the algorithm is RTS smoother. Similarly, the details of its FLOPs will not be included here. In summary, it needs

- Four matrix-matrix multiplications;
- One matrix-vector multiplications;
- Two matrix additions (subtractions);
- Two vector additions (subtractions);
- One matrix transpose operations; and
- One matrix inversion.

which results in

$$\mathbb{F}_{rts} = 9n_s^3 + n_s \tag{5.16}$$

The operations of IMMS is similar to IMMF but instead of filtering step, a smoothing procedure is used. Also, a backward transition probability is needed to be calculated. Therefore, the overall number of FLOPs including the RTS stage is

$$\begin{aligned}\mathbb{F}_{imms} = 4(Ln_s)^2 + 2L^2n_s + 3Ln_s^2 + L\mathbb{F}_{rts} \\ 8L^2 + 2L + 2Ln_s - n_s^2 - n_s - 1\end{aligned} \tag{5.17}$$

TABLE 5.1: Operational Complexity of the IMMF

Operation	**FLOPs**	**Type**
Mixing probabilities:		
$\bar{c}_j = \sum_{i=1}^{L} p_{ij}\mu^i_{k-1\|k-1}$	$2L^2 - L$	MVM
$\mu^{i\|j}_{k-1\|k-1} = \frac{1}{\bar{c}_j} p_{ij}\mu^i_{k-1\|k-1}$	$2L^2$	EWO
Estimate Mixing:		
$\bar{x}^{0j}_{k-1\|k-1} = \sum_{i=1}^{L} \mu^{i\|j}_{k-1\|k-1}\hat{x}^i_{k-1\|k-1}$	$2L^2 n_s - Ln_s$	MMM
Covariance Mixing:		
$v = \hat{x}^i_{k-1\|k-1} - \bar{x}^{0j}_{k-1\|k-1}$	$L^2 n_s$	VVA
vv^T	$(Ln_s)^2$	VVM
$P^i_{k-1\|k-1} + vv^T$	$(Ln_s)^2$	MMA
$\mu^{i\|j}_{k-1\|k-1}[P^i_{k-1\|k-1} + vv^T]$	$(Ln_s)^2$	EWO
$P^{0j}_{k-1\|k-1} = sum(P)$	$L(L-1)n_s^2$	MMA
Mode-matched filtering:		
$[P^j_{k+1\|k+1}, P^j_{k+1\|k}, \hat{x}^j_{k+1\|k+1}, \hat{x}^j_{k+1\|k}] = KF[\hat{x}^j_{k\|k}, P^j_{k\|k}]$	$L\mathbb{F}_{kf}$	KF
Mode probability update:		
$c = \sum_{i=1}^{L} \Lambda^j_k \bar{c}_j$	$2L - 1$	EWO
$\mu^j_{k\|k} = \frac{1}{c}\Lambda^j_k \bar{c}_j$	$2L$	EWO
Overall Estimate:		
$\hat{x}_{k\|k} = \sum_{j=1}^{L} \mu^j_{k\|k}\hat{x}^j_{k\|k}$	$2Ln_s - n_s$	MVM
Overall Covariance:		
$v = \hat{x}^j_{k\|k} - \hat{x}_{k\|k}$	Ln_s	VVA
vv^T	Ln_s^2	VVM
$P^j_{k\|k} + vv^T$	Ln_s^2	MMA
$\mu^j_{k\|k}[P^j_{k\|k} + vv^T]$	Ln_s^2	EWO
$P_{k\|k} = sum(P)$	$Ln_s^2 - n_s^2$	MMA

TABLE 5.2: Operational Complexity of the IMMS

Operation	FLOPs	Type
Backward transition probability:		
$e_j = \sum_{l=1}^{L} p_{li}\mu_{t\|t}^{l}$	$2L^2 - L$	MVM
$b_{ij} = \frac{1}{\bar{e}_i} p_{ji}\mu_{t\|t}^{j}$	$2L^2$	EWO
Backward mixing probability:		
$d_j = \sum_{l=1}^{L} b_{ij}\mu_{t+1\|k}^{l}$	$2L^2 - L$	MVM
$\mu_{t+1\|k}^{i\|j} = \frac{1}{d_j} b_{ij}\mu_{t+1\|k}^{i}$	$2L^2$	EWO
Estimate and Covariance Mixing:		
$\hat{x}_{t+1\|k}^{0j} = \sum_{i=1}^{L} \mu_{t+1\|k}^{i\|j}\hat{x}_{t+1\|k}^{i}$	$2L^2 n_s - Ln_s$	MMM
$v = \hat{x}_{t+1\|k}^{i} - \bar{x}_{t+1\|k}^{0j}$	$L^2 n_s$	VVA
vv^T	$(Ln_s)^2$	VVM
$P_{t+1\|k}^{i} + vv^T$	$(Ln_s)^2$	MMA
$\mu_{t+1\|k}^{i\|j}[P_{t+1\|k}^{i} + vv^T]$	$(Ln_s)^2$	EWO
$P_{t+1\|k}^{0j} = sum(P)$	$L(L-1)n_s^2$	MMA
Mode-matched Smoothing:		
$[P_{t\|k}^{j}, \hat{x}_{t\|k}^{j}]$ $= RTS[\hat{x}_{t\|t}^{j}, P_{t\|t}^{j}, \hat{x}_{t+1\|t}^{j}, P_{t+1\|t}^{j}]$	$L\mathbb{F}_{rts}$	RTS
Smoothed mode probability:		
$\mu_{t\|k}^{j} = \frac{1}{f}\Lambda_{t\|k}^{j}\mu_{t\|t}^{j}$	$2L$	EWO
$f = \sum_{i=1}^{L} \Lambda_{t\|k}^{j}\mu_{t\|t}^{j}$	$2L - 1$	EWO
Overall Smoothed Estimate and Covariance:		
$\hat{x}_{t\|k} = \sum_{j=1}^{L} \mu_{t\|k}^{j}\hat{x}_{t\|k}^{j}$	$2Ln_s - n_s$	MVM
$v = \hat{x}_{t\|k}^{j} - \hat{x}_{t\|k}$	Ln_s	VVA
vv^T	Ln_s^2	VVM
$P_{t\|k}^{j} + vv^T$	Ln_s^2	MMA
$\mu_{t\|k}^{j}[P_{t\|k}^{j} + vv^T]$	Ln_s^2	EWO
$P_{t\|k} = sum(P)$	$Ln_s^2 - n_s^2$	MMA

5.5.2 Autonomous Multiple Model Filtering and Smoothing

In order to analyse the improvement of complexity of AMMS, the following FLOPs count analysis is presented. The FLOPs count of AMMS is presented in Table 5.3 for window size N. The total number of FLOPs will be

$$\mathbb{F}_{amms} = N(4Ln_s^2 + 3Ln_s + 4L + L\mathbb{F}_{rts} - n_s^2 - n_s - 1) \tag{5.18}$$

Since AMMF is the same algorithm but with the smoothing stage instead of filtering stage. The overall FLOPs count of the algorithms are the same as AMMS and so the overall FLOPs for AMMF is

$$\mathbb{F}_{ammf} = 4Ln_s^2 + 3Ln_s + 4L + L\mathbb{F}_{kf} - n_s^2 - n_s - 1 \tag{5.19}$$

5.5.3 Fast Multiple Model Smoothing

For FMMS, the complexity burden of L model is released in the smoothing stage. There is only one set of RTS smoother calculation needed in the smoothing stage. Also, the mixed smoother gain Φ can be cached and so only needed to be calculated once in every time step instead of every window step. Moreover, the mixed estimate and covariance predictions are needed to be calculated for FMMS as in (5.10) and (5.11). The FLOPs count of FMMS is presented in Table 5.4. Therefore, the total FLOPs for window size N is

$$\begin{aligned}\mathbb{F}_{fmms} = 5Ln_s^3 + 3Ln_s^2 + 3Ln_s - n_s^2 - 2n_s \\ + N(4n_s^3 + 2n_s^2 - n_s)\end{aligned} \tag{5.20}$$

5.5.4 Comparison

To compare the FLOPs count of the proposed algorithms and the conventional IMM, the following gain metric g_{su} is defined.

$$g_{su} = \frac{F_{base}}{F_{algo}} \tag{5.21}$$

where F_{base} represents the FLOPs of the conventional algorithm and F_{algo} denotes the FLOPs of algorithm being compared. The speed-up of AMMS and FMMS over IMMS are shown in Figure 5.5. For a window size of 4, the advantage of FMMS and AMMS over IMMS increases linearly with the number of models. In practice, $4 - 10$ models are used in MM algorithms which can bring over 8 times and nearly 2 times speed up comparing with IMMS when 10 models are used. As expected, FMMS has the highest speed-up as it is computationally independent of the number of models L. To observe the speed-up of algorithms across different window size $1 - 32$, the number of model is set to four and the result is shown in Figure 5.6. The performance gain of AMMS is restricted to 1.3× while FMMS can reach up to over 10× faster than the conventional IMMS approach.

5.6 Simulation and Evaluation

5.6.1 Simulated Scenario

To demonstrate the efficacy of the proposed approach, 2 tracking scenarios are simulated. Consider a scenario with single target flying over a surveillance region of for 500 time steps. all experiments are implemented 100 times and average values are calculated. At

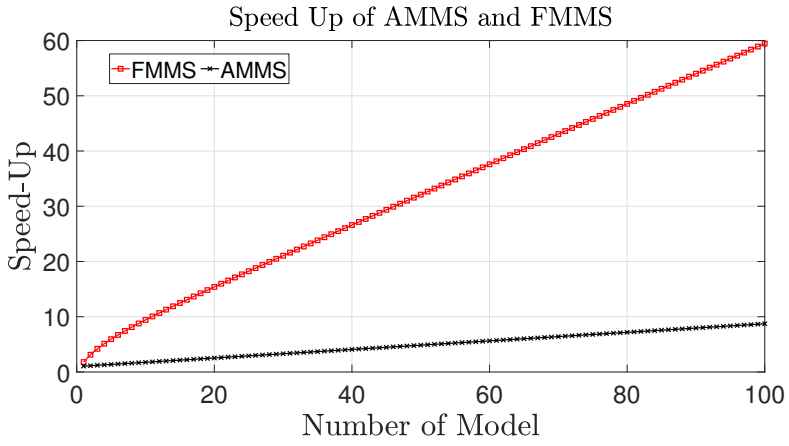

FIGURE 5.5: Speed-up of AMMS and FMMS in comparison with IMM when $N = 4$

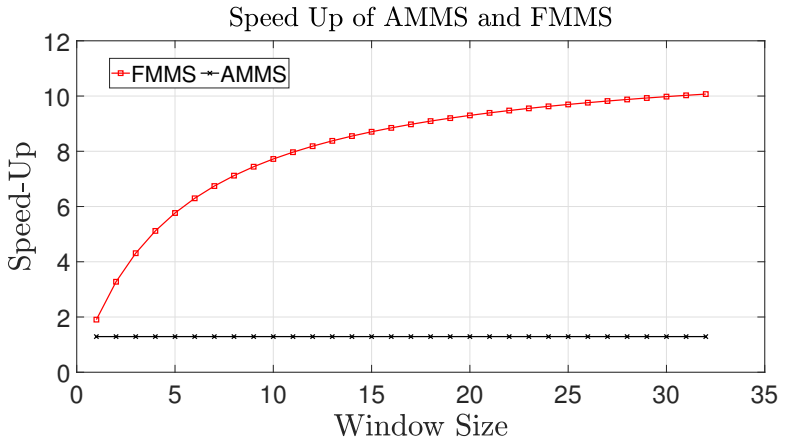

FIGURE 5.6: Speed-up of AMMS and FMMS in comparison with IMM when $L = 4$

t equals to $100, 200$ and 300, it undergoes constant turn for 50 time steps at turn rate of $0.17, 0.03$ and $-0.17 rad/s$. It remains to be constant velocity dynamics ($23.5 m/s$) for the rest of the motions. A 4-model MM algorithms is used. The tracking scenario is shown in Figure 5.7. The second scenario, as shown in Figure 5.8, has 700 time steps and at t equals to 50, 200, 300, 400, 500 and 600, it undergoes constant turn for 50 time steps at turn rate of -0.34, 0.17, -0.16, 0.04, 0.34 and $-0.03 rad/s$. The simulation will rely on a 7-model MM algorithm comprised of a constant velocity model and 6 constant turn model with the respective known turn rate. More specifically, this thesis set the state transition matrix A, measurement matrix H, turn rate w, state vector $\mathbf{x}$, process and measurement noise covariance matrices Q and R as follows:

TABLE 5.3: Operational Complexity of the AMMS

Operation	**FLOPs**	**Type**
Mode-matched Smoothing: $[P^j_{t\|k}, \hat{x}^j_{t\|k}] = RTS[\hat{x}^j_{t\|t}, P^j_{t\|t}, \hat{x}^j_{t+1\|t}, P^j_{t+1\|t}]$	$L\mathbb{F}_{rts}$	RTS
Mode probability: $\mu^i_{t\|k} = \frac{\mu^i_{t+1\|k} L^i_l}{\sum_{j=1}^{L} \mu^j_{t+1\|k} L^j_l}$	$4L-1$	EWO
Overall Smoothed Estimate: $\hat{x}_{t\|k} = \sum_{j=1}^{L} \mu^j_{t\|k} \hat{x}^j_{t\|k}$	$2Ln_s - n_s$	MVM
Overall Covariance:		
$v = \hat{x}^j_{t\|k} - \hat{x}_{t\|k}$	Ln_s	VVA
vv^T	Ln_s^2	VVM
$P^j_{t\|k} + vv^T$	Ln_s^2	MMA
$\mu^j_{t\|k}[P^j_{t\|k} + vv^T]$	Ln_s^2	EWO
$P_{t\|k} = sum(P)$	$Ln_s^2 - n_s^2$	MMA

$$A_{cv} = \begin{pmatrix} 1 & 0 & \S & 0 \\ 0 & 1 & 0 & \S \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$A_{turn} = \begin{pmatrix} 1 & 0 & \frac{\sin(w\S)}{w} & \frac{\cos(w\S)-1}{w} \\ 0 & 1 & \frac{1-\cos(w\S)}{w} & \frac{\sin(w\S)}{w} \\ 0 & 0 & \cos(w\S) & -\sin(w\S) \\ 0 & 0 & \sin(w\S) & \cos(w\S) \end{pmatrix}$$

Furthermore, the initial velocities in x- and y- directions of the target are randomly generated. At each time step, a Gaussian distributed noise $\mathcal{N}(\mu, \sigma^2)$ with $\mu = 0$ and $\sigma = 50m$ with $R = 50^2m^2$ is added for scenario 1 and $\mu = 0$ and $\sigma = 100$m with $R = 100^2m^2$ for scenario 2. For the IMM and the IMMS, the following Markov transition matrix is defined for scenario 1 and 2.

TABLE 5.4: Operational Complexity of the FMMS

Operation	FLOPs	Type
Mixed Prediction Estimate:		
$\hat{x}_{k+1\|k} = \sum_{j=1}^{L} \mu^j_{k\|k} \hat{x}^j_{k+1\|k}$	$2Nn_s - n_s$	MVM
Mixed Prediction Covariance:		
$v = \hat{x}^j_{k+1\|k} - \hat{x}_{k+1\|k}$	Ln_s	VVA
vv^T	$L(n_s)^2$	VVM
$P^j_{k+1\|k} + vv^T$	$L(n_s)^2$	MMA
$\mu^j_{k\|k}[P^j_{k+1\|k} + vv^T]$	$L(n_s)^2$	EWO
$P_{k+1\|k} = sum(P)$	$(L-1)n_s^2$	MMA
Mixed Gain:		
$P^j_{k\|k} {F^j_k}^T$	$L(2{n_s}^3 - {n_s}^2)$	MMM
${P^j_{k+1\|k}}^{-1}$	$\approx L{n_s}^3$	MI
$C^j_k = P^j_{k\|k} {F^j_k}^T {P^j_{k+1\|k}}^{-1}$	$L(2{n_s}^3 - {n_s}^2)$	MMM
$\Phi_k = \sum_{j=1}^{L} \mu^j_{k\|k} C^j_k$	$L{n_s}^2 + n_s(N-1)$	EWO
Smoothed state:		
$\hat{x}_{t+1\|k} - \hat{x}_{t+1\|t}$	n_s	VVA
$\Phi_k(\hat{x}_{t+1\|k} - \hat{x}_{t+1\|t})$	$2{n_s}^2 - n_s$	MVM
$\hat{x}_{t\|k} = \hat{x}_{t\|t} + \Phi_k(\hat{x}_{t+1\|k} - \hat{x}_{t+1\|t})$	n_s	VVA
Smoothed covariance:		
$P_{t+1\|k} - P_{t+1\|t}$	${n_s}^2$	MMA
$\Phi_k(P_{t+1\|k} - P_{t+1\|t})$	$2{n_s}^3 - {n_s}^2$	MMM
$\Phi_k(P_{t+1\|k} - P_{t+1\|t}){\Phi_k}^T$	$2{n_s}^3 - {n_s}^2$	MMM
$P_{t\|k} = P_{t\|t} - \Phi_k(P_{t+1\|k} - P_{t+1\|t}){\Phi_k}^T$	${n_s}^2$	MMA

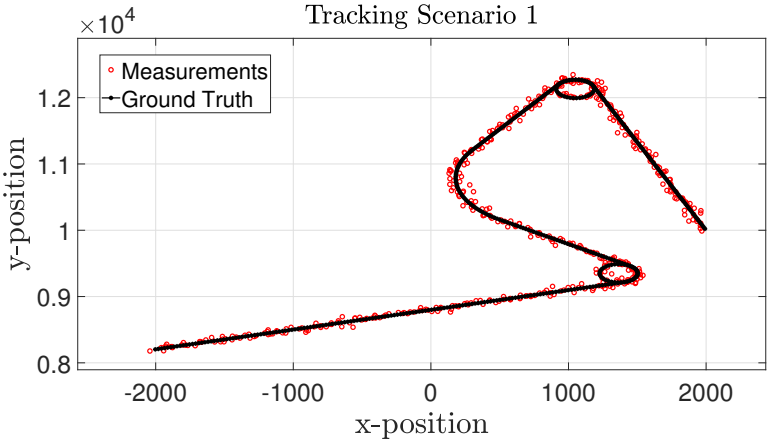

FIGURE 5.7: Scenario 1 for Target Tracking

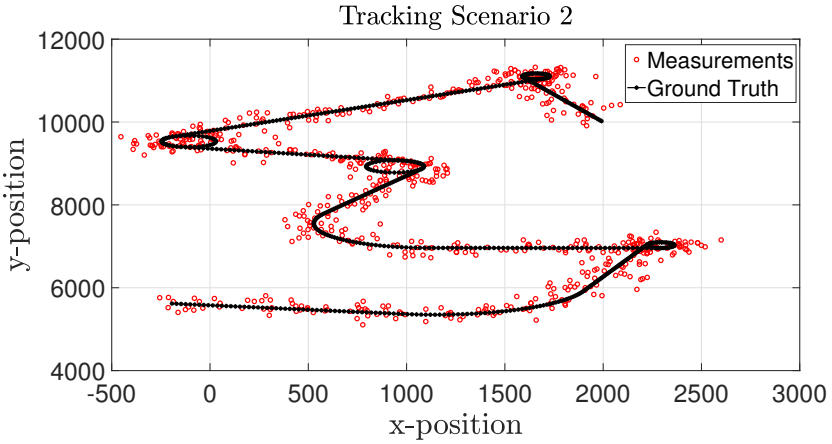

FIGURE 5.8: Scenario 2 for Target Tracking

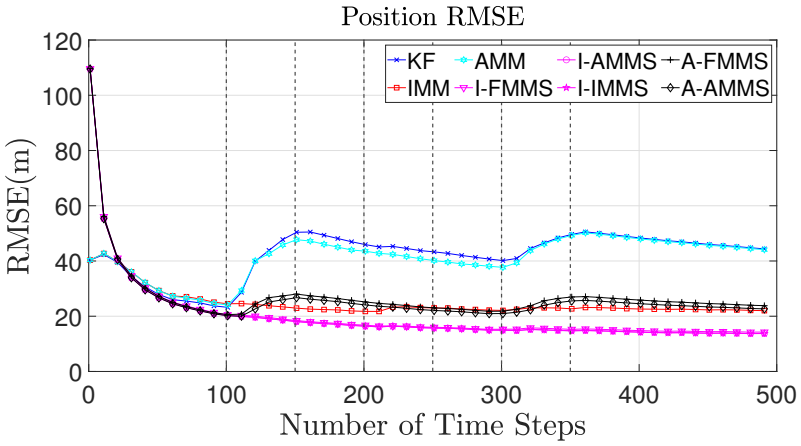

FIGURE 5.9: Position Error Comparison for Tracking Scenario 1

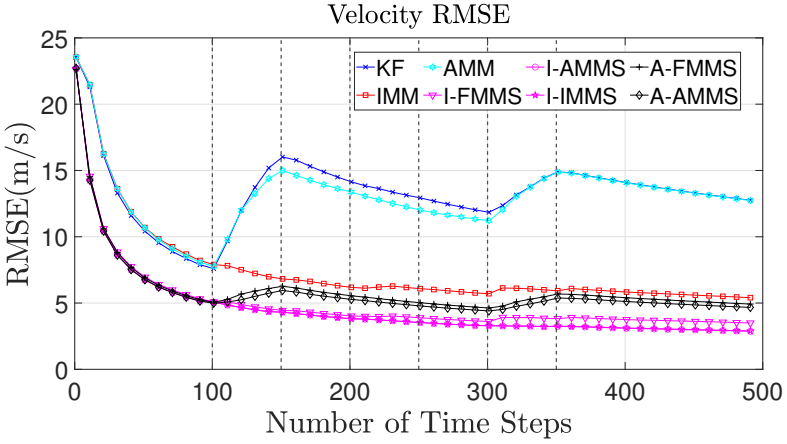

FIGURE 5.10: Velocity Error Comparison for Tracking Scenario 1

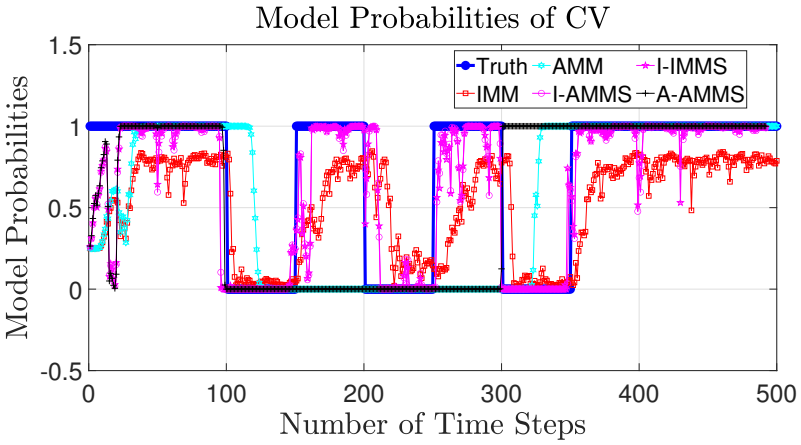

FIGURE 5.11: Comparison of Mode Probability for CV for Scenario 1

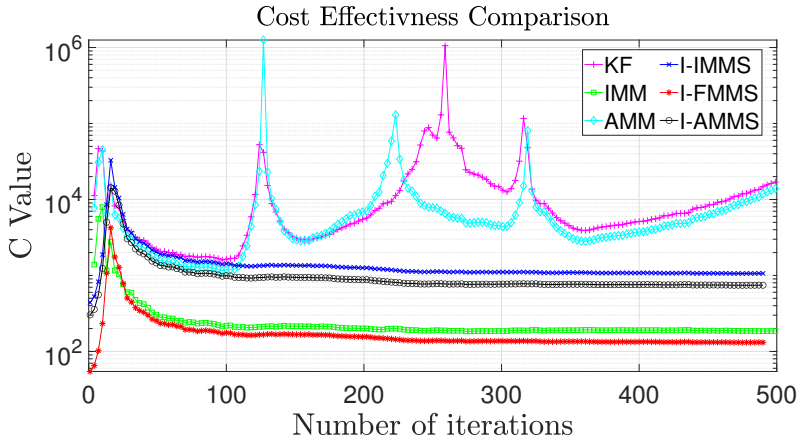

FIGURE 5.12: Cost effectiveness plot for Tracking Scenario 1

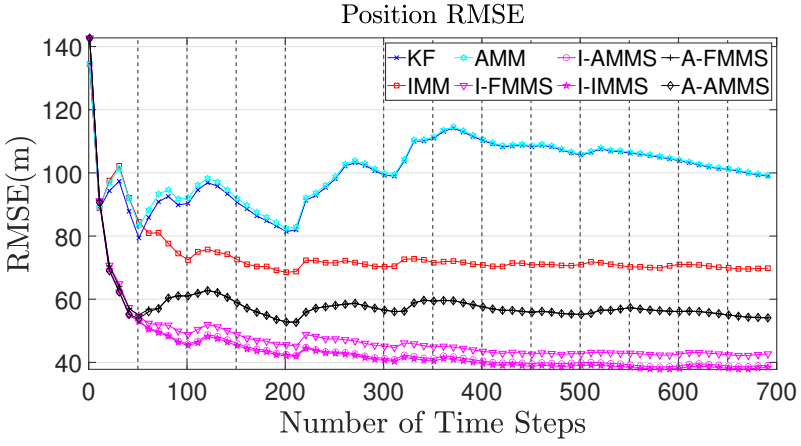

FIGURE 5.13: Error Comparison for Tracking Scenario 2

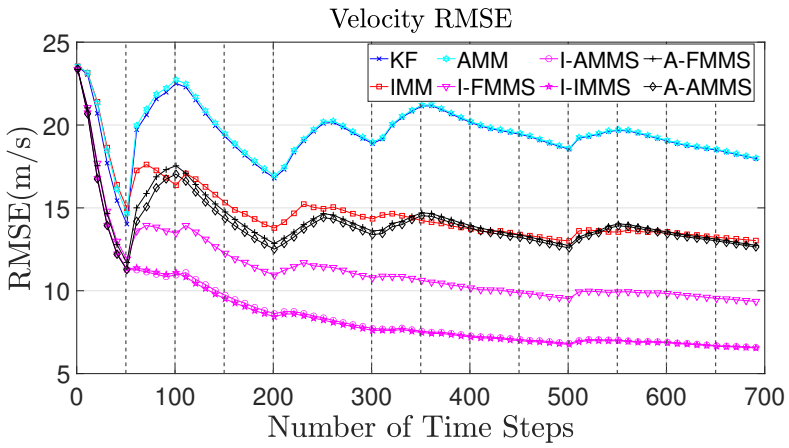

FIGURE 5.14: Velocity Error Comparison for Tracking Scenario 2

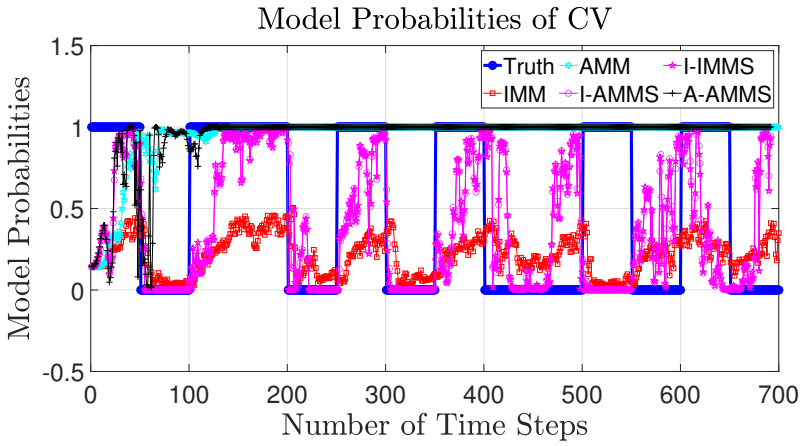

FIGURE 5.15: Comparison of Mode Probability for CV for Scenario 2

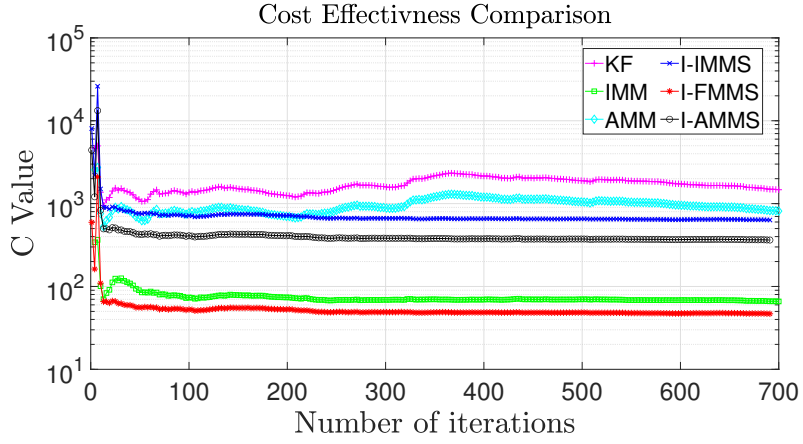

FIGURE 5.16: Cost effectiveness plot for Tracking Scenario 2

$$p_{ij}^{1} = \begin{pmatrix} 0.97 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.97 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.97 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.97 \end{pmatrix}$$

$$p_{ij} = \begin{pmatrix} 0.94 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.94 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.94 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.94 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.94 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.94 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.94 \end{pmatrix}$$

where I_n is an identity matrix of dimension n

5.6.2 Error-to-Complexity Performance Index

In order to quantify the cost effectiveness of different algorithms, the Error-to-Complexity performance index γ is proposed.

$$\gamma = \frac{N_{flops}}{E_{rmse}^{data} - E_{rmse}} \tag{5.22}$$

where E_{rmse} is the RMSE of the algorithm, whereas E_{rmse}^{data} is the RMSE of the measurements to ground truth and N_{flops} is the number of FLOPs of algorithm. γ can be interpreted as the cost of FLOPs needed to account for the accuracy increase from taking measurements as estimates. Therefore, the algorithm which has the lower value of γ will be more cost-effective.

5.6.3 Analysis and Discussion

To investigate the accuracy and efficiency of the proposed algorithms, they are used to compare with other MM algorithms on both tracking scenarios. The tracking algorithm

comprised of filtering and smoothing stage. The relationship between different filtering and smoothing combinations is also studied. The filtering algorithms are IMM and AMM. For smoothing, AMMS, FMMS and IMMS are compared. If the MM algorithm use IMM as filter and IMMS as smoothing, the algorithm will be termed as I-IMMS. If AMM is used, it will be named as A-AMMS. Moreover, the result of KF which use CV model is compared as well as a baseline. The algorithms: KF, AMM, IMM, I-AMMS, I-IMMS, A-FMMS and A-AMMS, are used to perform state estimations on tracking scenario 1 and 2. The results are plotted in Figure 5.9 to 5.16.

For tracking scenario 1, it is shown that from time step 0 to 100 in Figure 5.9, the object is moving in constant velocity and all models have similar RMSE, however, when the object undergoes constant turn at $0.17rad/s$ for 50 time steps, the KF, AMM has a sudden increase in RMSE. A-FMMS and A-AMMS also has the same effect but due to the use of smoother, the RMSE is lowered. While the RMSE for IMM, I-FMMS, I-AMMS and I-IMMS remain steady which indicates the change of model do not have an effect of the RMSE. The ones that have the IMM as filter and equipped with a smoother have the lowest RMSE, regardless of the choice of smoother. When the time step is at 200, another turn is taken at $0.03rad/s$ for 50 time steps. No significant RMSE error change is observed due to the fact that the turn rate is not too high as so a CV will be suffice for algorithms that does not have good switching capability. When at time step 300, a turn rate of $-0.17rad/s$ is taken and similar RMSE can be observed as in time step 100. The RMSE of AMM, KF, A-AMMS, and A-FMMS increase while IMM, I-FMMS, I-IMMS and I-AMMS are independent to model change. The velocities estimates have a similar pattern to position as shown in Figure 5.10. The switching efficiency of the algorithms can be explained more through looking at the mode probability change of CV as shown in Figure 5.11. In the beginning from time step 0 to 100, all algorithms can identify CV as the correct model. However, at time step 100, AMM has a late switching response and at $t = 150$, AMM and A-AMMS determine that the probability of CV is close to 0 while indeed CV is the correct model, whereas IMM, I-AMMS and I-IMMS can correctly keep up with the switching. The same happen at time step 300 at which AMM and A-AMMS incorrectly determine the model choice. IMM, I-AMMS and I-IMMS have the correct model switching with I-AMMS and I-IMMS perform better with the help of smoother. Therefore, they perform better as shown in Figure 5.9 and 5.10. Moreover, to compare the effectiveness of algorithms, the γ value of them are plotted in Figure 5.12 which shows that I-FMMS has the lowest γ value which indicates the most effective out of the algorithms shown. The IMM is the second most cost effective, meaning although I-IMMS perform better in estimation, the decrease in RMSE does not account for the increase in FLOPs. Therefore, in real time tracking scenarios, users may take this into consideration when designing tracking algorithms. However, the proposed algorithm has

an even lower γ value than IMM which shows the decrease in RMSE brought by the smoother does account for the increase in FLOPs.

For scenario 2, targets are more manoeuvrable and a higher noise level is simulated which increase the difficulty of tracking. However, similar result are shown in Figures 5.13 to 5.14, with I-FMMS, I-AMMS and I-IMMS show the lowest RMSE and I-FMMS being the most cost-effective as shown in Figure 5.16. It is interesting to note that the Markov transition and reinitialisation step as in the IMM is not important in the smoothing stage as justified y Figures 5.9 to 5.11 for scenario 1 and Figures 5.13 to 5.15 for scenario 2. Similar performance can be achieved, comparing to the I-IMMS, without using the Markov transition matrix and shows that the I-AMMS and I-FMMS has better complexity than te I-IMMS.

5.7 Conclusions

The Multiple Model (MM) estimation approach offers outstanding performance for tracking maneuvering targets. The Interacting Multiple Model (IMM) estimator has been regarded as the state-of-the art in the domain of multiple model tracking due to its simplicity and accuracy. Furthermore, when a delay can be tolerated, retrodiction or smoothing can be used in the context of tracking and often becomes part of the estimation process. The Rauch-Tung-Striebel (RTS) Smoother is one of the most widely used smoothing algorithm in the tracking community. Interacting Multiple Model (IMM) smoother is a smoother regime that utilise RTS and IMM together. However, with a large number of window size and number of models, such a MM tracking algorithm is difficult to achieve real-time performance. This chapter has proposed an efficient smoother algorithm by using Autonomous Multiple Model as the backbone as well as derived a new Fast Multiple Model smoother (FMMS) to look for a better compromise between floating point operations and accuracy in terms of RMSE.

This chapter presented a detailed analysis on their computational complexities, showing how these algorithms can offer a significant performance advantage over the conventional IMM and IMMS algorithms. Among these, the FMMS offers the best computational complexity, which release the computational burden of number of models and achieve the unprecedented complexity of $O(Nn^3)$. Besides, it is shown that, in this study, it is the more cost-effective over the conventional approach this chapter have compared. It also show that these algorithms offer comparable performance in terms of accuracy in two tracking scenarios.

Chapter 6

Conclusions

The aim of this thesis is to present novel algorithms that can improve the performance of the tracking procedures in the aspect of filtering, smoothing and multiple model tracking. Under this thesis scope, a collection of novel algorithms for target tracking was developed. This chapter will thus emphasize the main results drawn from the algorithms compared and presented in various tracking scenarios.

In Chapter 3, a novel parallelisation algorithm the RTS Smoother was developed which brings notable performance gain along with the parallelisation in handling the number of targets. This leads to nested parallelism and studies suggested that in order to maximise performance gain, a careful understanding of the computing architecture as well as the window size and the number of targets are needed. After re-formulating the RTS smoother algorithm, it was discovered that a lot of data can be reused and so a massive amount of FLOPs can be omitted. The simulation results show that nearly a 150-fold speed-up can be achieved by compared to the naive approach.

Apart from introducing parallelism in RTS smoother, it is also possible to omit the intermediate calculations and obtain the fixed-lag estimate directly in every time step. By studying the nature of fixed-point smoother, the fixed-lag RTS Smoother has been re-formulated in order to accommodate the caching mechanism in the algorithm. However, after reformulation, the algorithm becomes unstable and so a thorough stability analysis has been presented in Chapter 4. After identifying the source of the stability problem, condition number guided-smoother is proposed to monitor the deterioration rate of the smoother. As a result, a 50 times speedup can be achieved in runtime performance with comparable RMSE to the conventional RTS smoother.

Apart from filtering and smoothing, when target motion is uncertain, multiple model algorithms are used to improve tracking maneuvering targets. The first generation

of MM algorithms is the most limited, since it assumes the target is in a constant maneuver, such that the target has one (unknown) constant mode. The other two generations have more complex concepts, even though they are not computationally more complex. Their tracking capability is much better, since they assume the targets can have multiple maneuvering motions. The CMM generation has little more room for improvement, leading to the general belief in the tracking community, that the VSMM algorithms will outperform fixed-structure MM algorithms significantly, since they allow lower computational complexity with larger model-sets. However, the VSMM algorithms are relatively new and need more research to be well established as a robust alternative algorithm to CMM. Therefore, IMM remains the state-of-the-art approach for MM algorithms. In order to seek a more cost-effective approach, a compromise between the first and second generation MM algorithms is investigated. A Fast Multiple Model Filter along with smoothing are introduced and presented in Chapter 5. Not only does it possess the light weight nature of first generation algorithm, it also has comparable accuracy comparing with IMM and IMMS. From the complexity analysis, a 10 times speedup can be achieved.

The advantage of the suggested algorithms are having the computational edge. However, the algorithms become more complex and more difficult to implement than the conventional RTS. For example, the programming of parallel RTS algorithm mentioned in Chapter 3 needs to consider threads spawning and synchronization which needs more time to tune and more complicated to code. Moreover, parallel computing might leads to different result due to truncation and calculation approximations. The FRTS and FMMS algorithms suggested in Chapter 4 and 5 also suffer similar problem. It is a trade-off between accuracy and complexity. Although 50 times speed-up can be achieved, certain amount of accuracy is loss during the process. The amount of loss is controlled by the users and user can find a optimal point between accuracy and speed.

6.1 Future Work

The additional research that can be realized in this area is vast. There are several additional topics to extend this thesis work, they are: i) Non-linear tracking algorithms, ii) alternative algorithms and iii) measurement origin uncertainty algorithm.

All algorithms presented in the thesis are applied to linear Kalman filtering. However, there are other filters in the tracking community such as the extended Kalman filter, particle filtering and PHD filtering that use the same principles as the linear Kalman filter. Moreover, since these filters need to handle more complicated problems in non-linear scenarios, they tend to have higher complexity and are more computationally

intensive. The proposed techniques can possibly be transferred to these filtering and smoothing algorithms to improve the existing methods.

In Chapter 3, the proposed parallelized retrodiction algorithm can be further improved by considering thread spawning times and can be fully customized when number of window size is fixed before deploying the use fo the algorithm. Since thread spawning can be a small bottleneck for parallel algorithm, threads can be pre-allocated to designated operations to achieve better performance. For the proposed algorithm FRTS, the trade-off between accuracy and complexity solely replies on the accuracy renewal step. The complexity benefit of the FRTS would be minimised if the renewal step is needed to be used too frequently when high accuracy is required. Therefore, more work can be done on this area to further improve the accuracy of the proposed algorithm FRTS and so it can run more smoothly without monitoring the deterioration of the algorithm. In terms of the MM approach, there can be two directions to exploit in order to achieve better accuracy for tracking manoeuvring targets in real-time. The first one is to design a better set of models to better capture different possibility of target movement and incorporating them into the MM algorithms. The other approach is to develop and design better algorithms which this thesis is proposing. However, the proposed method does not show any improvement in tracking accuracy but only in complexity. On this note, more research can be carried out to develop novel methods for achieving higher accuracy while maintaining complexity independent to number of models used and smoothing window size.

Also, measurement origin uncertainty algorithm has not been considered on in this thesis. However, it is a very important stage in tracking which can impact the tracking performance. The process of filtering and smoothing can help these algorithms to obtain more accurate results and so false detections can be reduced.

Bibliography

[1] Z. Ge, F. Chang, and H. Liu. Multi-target tracking based on Kalman filtering and optical flow histogram. In *2017 Chinese Automation Congress (CAC)*, pages 2540–2545, Oct 2017.

[2] X. R. Li. The PDF of nearest neighbor measurement and a probabilistic nearest neighbor filter for tracking in clutter. In *Proceedings of 32nd IEEE Conference on Decision and Control*, pages 918–923 vol.1, Dec 1993.

[3] Xuezhi Wang, S. Challa, and R. Evans. Gating techniques for maneuvering target tracking in clutter. *IEEE Transactions on Aerospace and Electronic Systems*, 38 (3):1087–1097, Jul 2002.

[4] Z. Bekhtaoui, A. Meche, M. Dahmani, and K. A. Meraim. Maneuvering target tracking using q-learning based Kalman filter. In *2017 5th International Conference on Electrical Engineering - Boumerdes (ICEE-B)*, pages 1–5, Oct 2017.

[5] Wolfgang Koch. GMTI-tracking and information fusion for ground surveillance. volume 4473, page 12, 2001.

[6] D. Zhang, X. Qian, and Y. Zhang. Research on abnormal behavior target tracking algorithm in airport intelligent video surveillance. In *2017 International Conference on Progress in Informatics and Computing (PIC)*, pages 154–158, Dec 2017.

[7] D. S. R. Kondru and M. Celenk. Predictive airborne target tracking using all-terrain fusion based mobile surveillance system. In *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, pages 1–6, March 2018.

[8] N. T. B. Bui, D. C. Pham, B. Q. Nguyen, and S. T. Le. Tracking a 3D target with fusion of 2D radar and bearing-only sensor. In *2018 IEEE International Conference on Industrial Technology (ICIT)*, pages 1532–1537, Feb 2018.

[9] G. Kumar, D. Prasad, and R. P. Singh. Target tracking using adaptive Kalman Filter. In *2017 International Conference on Smart grids, Power and Advanced Control Engineering (ICSPACE)*, pages 376–380, Aug 2017.

[10] Y. Kong, X. Zhang, and W. Bai. Extended target tracking algorithm based on improved Bernoulli filter. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pages 2522–2527, Dec 2017.

[11] Huan H. Zhou and B. Ni. Tracking of drone flight by neural network siamese-rpn. In *2020 6th International Conference on Engineering, Applied Sciences and Technology (ICEAST)*, pages 1–3, 2020.

[12] J. Park et al. A comparison of convolutional object detectors for real-time drone tracking using a ptz camera. In *2017 17th International Conference on Control, Automation and Systems (ICCAS)*, pages 696–699, 2017.

[13] P. Pacheco. *An Introduction to Parallel Programming*. Morgan Kaufmann, 2011.

[14] B. Friedland. Treatment of bias in recursive filtering. *IEEE Transactions on Automatic Control*, 14(4):359–367, 1969.

[15] J.Y. Keller and M. Darouach. Two-stage Kalman estimator with unknown exogenous inputs. *Automatica*, 35(2):339–342, 1999.

[16] C.S. Hsieh. General two-stage extended Kalman filters. *IEEE Transactions on Automatic Control*, 48(2):289–293, 2003.

[17] X. Rong Li and V. P. Jilkov. Survey of maneuvering target tracking. part i. dynamic models. *IEEE Transactions on Aerospace and Electronic Systems*, 39(4): 1333–1364, 2003.

[18] X. Rong Li and V. P. Jilkov. Survey of maneuvering target tracking. part v. multiple-model methods. *IEEE Transactions on Aerospace and Electronic Systems*, 41(4):1255–1321, Oct 2005.

[19] R. Lopez and P. Danès. Low-complexity IMM smoothing for jump markov nonlinear systems. *IEEE Transactions on Aerospace and Electronic Systems*, 53:1261–1272, June 2017.

[20] E. Derbez, B. Remillard, and A. Jouan. A comparison of fixed gain IMM against two other filters. In *Proceedings of the Third International Conference on Information Fusion*, volume 2, pages THB2/3–THB2/9 vol.2, 2000. doi: 10.1109/IFIC.2000.859872.

[21] D. Mohammed, K. Mokhtar, Q. Abdelaziz, and M. Abdelkrim. A new IMM algorithm using fixed coefficients filters (fastIMM). *AEU - International Journal of Electronics and Communications*, 64(12):1123 – 1127, 2010.

[22] S.S. Khalid and S. Abrar. A low-complexity interacting multiple model filter for maneuvering target tracking. *AEU - International Journal of Electronics and Communications*, 73:157 – 164, 2017.

[23] S. L. Yeung, S. Tager, P. Wilson, R. Tharmarasa, W. Armour, and J. Thiyagalingam. A parallel retrodiction algorithm for large-scale multitarget tracking. *IEEE Transactions on Aerospace and Electronic Systems*, pages 1–1, 2020.

[24] R.E. Kalman. A new approach to linear filtering and prediction problems. *ASME. J. Basic Eng*, pages 35–45, 1960.

[25] H. E. Rauch, C. T. Striebel, and F. Tung. Maximum likelihood estimates of linear dynamic systems. *AIAA Journal*, 3(8):1445–1450, 1965.

[26] Simo Särkkä. *Bayesian Filtering and Smoothing*. Cambridge University Press, New York, NY, USA, 2013.

[27] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2), Feb 2002.

[28] Jitendra R. Raol. *Multi-Sensor Data Fusion with MATLAB*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2009.

[29] D. G. Lainiotis. Optimal adaptive estimation: Structure and parameter adaptation. In *1970 IEEE Symposium on Adaptive Processes (9th) Decision and Control*, pages 143–143, Dec 1970.

[30] D. G. Lainiotis. Partitioning: A unifying framework for adaptive systems, i: Estimation. *Proceedings of the IEEE*, 64(8):1126–1143, Aug 1976.

[31] X. R. Li and Y. Bar-Shalom. Estimation and tracking: Principles, techniques, and software [reviews and abstracts]. *IEEE Antennas and Propagation Magazine*, 38 (1):62–, Feb 1996.

[32] H. A. P. Blom and Y. Bar-Shalom. The interacting multiple model algorithm for systems with Markovian switching coefficients. *IEEE Transactions on Automatic Control*, 33(8):780–783, Aug 1988.

[33] Kuo-Chu Chang and Y. Bar-Shalom. Joint probabilistic data association for multitarget tracking with possibly unresolved measurements and maneuvers. *IEEE Transactions on Automatic Control*, 29(7):585–594, Jul 1984.

[34] J. Mendel. Computational requirements for a discrete Kalman filter. *IEEE Transactions on Automatic Control*, 16(6):748–758, 1971.

[35] G.H. Golub and C. van Loan. *Matrix Computations*. JHU Press, fourth edition, 2013.

[36] M. Paprzycki and C. Cyphers. Using Strassen's matrix multiplication in high performance solution of linear systems. *Computers and Mathematics with Applications*, 31(4):55 – 61, 1996. Selected Topics in Numerical Methods.

[37] Iain S. Duff, Michele Marrone, Giuseppe Radicati, and Carlo Vittoli. Level 3 Basic Linear Algebra Subprograms for Sparse Matrices: A User-Level Interface. *ACM Trans. Math. Softw.*, 23(3):379–401, September 1997.

[38] B. Gravelle and B. Norris. Performance analysis of compressed batch matrix operations on small matrices. In *2019 International Conference on High Performance Computing Simulation (HPCS)*, pages 424–427, 2019.

[39] Craig C. Douglas, Michael Heroux, Gordon Slishman, and Roger M. Smith. GEMMW: A Portable Level 3 BLAS Winograd Variant of Strassen's Matrix-Matrix Multiply Algorithm. *Journal of Computational Physics*, 110(1):1 – 10, 1994.

[40] D. Lawrie and P. Fleming. Fine-grain parallel processing implementations of Kalman filter algorithms. In *International Conference on Control 1991. Control '91*, pages 867–870 vol.2, 1991.

[41] Blattner Timothy and Yang Shiming. Performance study on CUDA GPUs for parallelizing the local ensemble transformed Kalman filter algorithm. *Concurrency and Computation: Practice and Experience*, 24(2):167–177, 2012.

[42] L. Huo and Z. Wang. A Target Tracking Algorithm Using Grey Model Predicting Kalman Filter in Wireless Sensor Networks. In *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 604–610, June 2017.

[43] L. ChongYi, L. Cheng, F. LinYu, and Y. JingTing. Target tracking based on extended Kalman particle filter. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pages 1715–1719, Dec 2017.

[44] Yaakov Bar-Shalom, Thiagalingam K, and Li X-Rong. *Estimation with Applications to Tracking and Navigation*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[45] Yaakov Bar-Shalom, Thiagalingam K, and Li X-Rong. *Multitarget-Multisensor Tracking: Principles and Techniques*. Bertrams, YBS Publishing, Storrs, USA, 1995.

[46] S.J. Julier and J.K. Uhlmann. A new extension of the kalman filter to nonlinear systems. In *Proc. of AeroSense: The 11th Int. Symp. on Aerospace/Defense Sensing, Simulations and Controls*, 1997.

[47] E. A. Wan and R. Van Der Merwe. The unscented Kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, pages 153–158, 2000.

[48] A. Harvey. *Forecasting, Structural Time Series Models and the Kalman Filter.* Cambridge University Press, Cambridge, 1990.

[49] J.B Moore B, Anderson. *Optimal filtering.* Pretice-Hall, New Jersey, USA, 1979.

[50] H. Rauch. Solutions to the linear smoothing problem. *IEEE Transactions on Automatic Control*, 8(4):371–372, Oct 1963.

[51] Chen B and Tugnait J. K. Tracking of multiple maneuvering targets in clutter using IMM/JPDA filtering and fixed-lag smoothing. *Automatica*, 37(2):239 – 249, 2001.

[52] U. Khan, D. Musicki, and T. L. Song. A fixed lag smoothing IPDA tracking in clutter. In *17th International Conference on Information Fusion (FUSION)*, pages 1–7, July 2014.

[53] D. Mušicki, T. L. Song, and T. H. Kim. Smoothing Multi-Scan Target Tracking in Clutter. *IEEE Transactions on Signal Processing*, 61(19):4740–4752, Oct 2013.

[54] P. G. Park and T. Kailath. New square-root smoothing algorithms. *IEEE Transactions on Automatic Control*, 41:727–732, 1996.

[55] Grewal. M. S and Andrews A. P. *Kalman Filtering: Theory and Practice with MATLAB.* Wiley-IEEE Press, 4th edition, 2014.

[56] Robert Grover Brown and Patrick Y C Hwang. *Introduction to random signals and applied Kalman Filtering: with MATLAB exercises and solutions; 3rd ed.* Wiley, New York, NY, 1997.

[57] W. Koch. Fixed-interval retrodiction approach to bayesian IMM-MHT for maneuvering multiple targets. *IEEE Transactions on Aerospace and Electronic Systems*, 36(1):2–14, Jan 2000.

[58] Song T. L and D. Mušicki. Smoothing innovations and data association with IPDA. *Automatica*, 48(7):1324 – 1329, 2012.

[59] D. Musicki, R. Evans, and S. Stankovic. Integrated probabilistic data association. *IEEE Transactions on Automatic Control*, 39(6):1237–1241, Jun 1994.

[60] T E. Luginbuhl R L. Streit. Maximum likelihood method for probabilistic multi-hypothesis tracking. volume 2235, page 12, 1994.

[61] T. L. Song, D. Musicki, H. H. Lee, and X. Wang. Point target probabilistic multiple hypothesis tracking. *IET Radar, Sonar Navigation*, 5(6):632–637, July 2011. ISSN 1751-8784. doi: 10.1049/iet-rsn.2010.0260.

[62] M. P. Lyster, K. Ekers, J. Guo, M. Harber, D. Lamich, J. W. Larson, R. Lucchesi, R. Rood, S. Schubert, W. Sawyer, M. Sienkiewicz, A. d. Silva, J. Stobie, L. L. Takacs, R. Todling, J. Zero, C. H. Q. Ding, and R. Ferraro. Parallel Computing at the NASA Data Assimilation Office (DAO). In *Supercomputing, ACM/IEEE 1997 Conference*, pages 26–26, Nov 1997.

[63] M. A. Palis and D. K. Krecker. Parallel Kalman filtering on the Connection Machine. In *The Third Symposium on the Frontiers of Massively Parallel Computation*, pages 55–58, Oct 1990.

[64] O. Rosen and A. Medvedev. Efficient Parallel Implementation of State Estimation Algorithms on Multicore Platforms. *IEEE Transactions on Control Systems Technology*, 21(1):107–120, Jan 2013.

[65] G. Cerati et.al. Kalman-Filter-Based Particle Tracking on Parallel Architectures at Hadron Colliders. In *Proceedings, 2015 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC 2015): San Diego, California, United States*, page 7581932, 2016.

[66] O. Rosen and A. Medvedev. Parallelization of the Kalman filter for banded systems on multicore computational platforms. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 2022–2027, Dec 2012.

[67] R. L. Popp, K. R. Pattipati, Y. Bar-Shalom, and M. Yeddanapudi. Parallelization of a multiple model multitarget tracking algorithm with superlinear speedups. *IEEE Transactions on Aerospace and Electronic Systems*, 33(1):281–290, Jan 1997.

[68] R. L. Popp, K. R. Pattipati, and Y. Bar-Shalom. m-best s-d assignment algorithm with application to multitarget tracking. *IEEE Transactions on Aerospace and Electronic Systems*, 37(1):22–39, Jan 2001.

[69] K. R. Pattipati, T. Kurien, R. T. Lee, and P. B. Luh. On mapping a tracking algorithm onto parallel processors. *IEEE Transactions on Aerospace and Electronic Systems*, 26(5):774–791, Sep 1990.

[70] L. Y. Pao and C. W. Frei. A comparison of parallel and sequential implementations of a multisensor multitarget tracking algorithm. In *American Control Conference, Proceedings of the 1995*, volume 3, pages 1683–1687 vol.3, Jun 1995.

[71] M. D. Hill and M. R. Marty. Amdahl's Law in the Multicore Era. *Computer*, 41 (7):33–38, July 2008.

[72] J. Kwiatkowski. Evaluation of Parallel Programs by Measurement of Its Granularity. In *Parallel Processing and Applied Mathematics*, pages 145–153. Springer Berlin Heidelberg, 2002.

[73] M. Crovella, R. Bianchini, T. LeBlanc, E. Markatos, and R. Wisniewski. Using communication-to-computation ratio in parallel program design and performance prediction. In *[1992] Proceedings of the Fourth IEEE Symposium on Parallel and Distributed Processing*, pages 238–245, 1992.

[74] OpenMP Architecture Review Board. OpenMP Application Program Interface.

[75] James J. Jeffers, J. Reinders and A. Sodani. *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition 2nd Edition.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2016.

[76] J.M. Jover and T. Kailath. A parallel architecture for kalman filter measurement update and parameter estimation. *Automatica*, 22(1):43–57, 1986.

[77] D.J. Potter and M.P. Cline. Parallel algorithms for 2d kalman filtering. In *[1990 Proceedings] The Third Symposium on the Frontiers of Massively Parallel Computation*, pages 47–50, 1990.

[78] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251 – 280, 1990.

[79] V. Y. Pan. Strassen's algorithm is not optimal trilinear technique of aggregating, uniting and canceling for constructing fast algorithms for matrix operations. In *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, pages 166–176, Oct 1978.

[80] T.H. Cormen et.al. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

[81] Y. Bar-Shalom, F. Daum, and J. Huang. The probabilistic data association filter. *IEEE Control Systems*, 29(6):82–100, Dec 2009.

[82] S. S. Blackman. Multiple hypothesis tracking for multiple target tracking. *IEEE Aerospace and Electronic Systems Magazine*, 19(1):5–18, Jan 2004.

[83] A.H. Tewfik, A.S. Willsky, and B.C. Levy. Parallel smoothing. *Systems and asas Control Letters*, 14(3):253 – 259, 1990.

[84] A.S. Willsky, A.H. Tewfik, and B.C. Levy. A New Parallel Smoothing Algorithm. *IEEE Conference on Descicion and Control*, 25, 1986.

[85] F. Badawi, A. Lindquist, and M. Pavon. On the mayne-fraser smoothing formula and stochastic realization theory for nonstationary linear stochastic systems. In *1979 18th IEEE Conference on Decision and Control including the Symposium on Adaptive Processes*, volume 2, pages 505–510, Dec 1979.

[86] G. E. Blelloch. Prefix Sums and Their Applications. *Synthesis of Parallel Algorithms*, 1990.

[87] R. Rabenseifner and J.L. Träff. More Efficient Reduction Algorithms for Non-Power-of-Two Number of Processors in Message-Passing Parallel Systems. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 36–46, 2004.

[88] P. J. Slater, E. J. Cockayne, and S. T. Hedetniemi. Information Dissemination in Trees. *SIAM Journal on Computing*, 10(4):692–701, 1981.

[89] E. E. Santos. Optimal and Efficient Algorithms for Summing and Prefix Summing on Parallel Machines. *Journal of Parallel and Distributed Computing*, 62(4):517 – 543, 2002.

[90] J.B. Moore. Discrete-time Fixed-lag Smoothing Algorithms. *Automatica*, pages 163–173, 1973.

[91] K. Biswas and A. Mahalanabis. Optimal fixed-lag smoothing for time delayed system with colored noise. *IEEE Transactions on Automatic Control*, 17(3):387–388, June 1972.

[92] C. K. Chui and G. Chen. *Kalman Filtering with Real-Time Applications.* SpringerVerlag, New York, 1987.

[93] P. S. Maybeck. *Stochastic Models, Estimation, and Control.* Academic Press, New York, 1982.

[94] H. W. Sorenson. *Kalman Filtering: Theory and Application.* IEEE Press, New York, 1985.

[95] M. S. Grewal and H. J. Payne. Identification of Parameters in a Freeway Traffic Model. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(3):176–185, 1976.

[96] Z. Fang, G. Luo, F. Fei, and S. Li. Stock Forecast Method Based on Wavelet Modulus Maxima and Kalman Filter. In *2010 International Conference on Management of e-Commerce and e-Government*, pages 50–53, 2010.

[97] Y. Gong and Y. Zhang. Research of Short-Term Traffic Volume Prediction Based on Kalman Filtering. In *2013 6th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, pages 99–102, 2013.

[98] R. Chakravorty and S. Challa. Augmented state integrated probabilistic data association smoothing for automatic track initiation in clutter. *Journal of Advances in Information Fusion*, 1(1):63–74, July 2006.

[99] C N. Kelly and B D.O. Anderson. On the stability of fixed-lag smoothing algorithms. *Journal of the Franklin Institute*, 291(4):271 – 281, 1971.

[100] E. W. Cheney and D.R. Kincaid. *Numerical Mathematics and Computing*. Brooks/Cole Publishing Co., Pacific Grove, CA, USA, 6th edition, 2007.

[101] A. K. Cline, C. B. Moler, G. W. Stewart, and J. H. Wilkinson. An Estimate for the Condition Number of a Matrix. *SIAM Journal on Numerical Analysis*, 16(2): 368–375, 1979.

[102] J.P. Merlet. Jacobian, Manipulability, Condition Number and Accuracy of Parallel Robots. *ASME J. of Mechanical Design*, 128:199–206, 01 2006.

[103] R. Kress. Minimizing the condition number of boundary integral operators in acoustic and electromagnetic scattering. *The Quarterly Journal of Mechanics and Applied Mathematics*, 38(2):323–341, 05 1985.

[104] S. Skare et.al. Condition Number as a Measure of Noise Performance of Diffusion Tensor Data Acquisition Schemes with MRI. *Journal of Magnetic Resonance*, 147 (2):340 – 352, 2000.

[105] N.K. Tsao and F.F. Kuo. On machine precision, computation error and condition number in solving linear algebraic systems. *Computers and Electrical Engineering*, 1(3):459–464, 1973.

[106] X. R. Li and Y. Bar-Shakm. Mode-set adaptation in multiple-model estimators for hybrid systems. In *1992 American Control Conference*, pages 1794–1799, June 1992.

[107] X. R. Li and Y. Bar-Shalom. Multiple-model estimation with variable structure. *IEEE Transactions on Automatic Control*, 41(4):478–493, April 1996.

[108] H. A. P. Blom. An efficient filter for abruptly changing systems. In *The 23rd IEEE Conference on Decision and Control*, pages 656–658, Dec 1984.

[109] D. Magill. Optimal adaptive estimation of sampled stochastic processes. *IEEE Transactions on Automatic Control*, 10(4):434–439, October 1965.

[110] A. Averbuch, S. Itzikowitz, and T. Kapon. Radar target tracking-Viterbi versus IMM. *IEEE Transactions on Aerospace and Electronic Systems*, 27(3):550–563, May 1991.

[111] Y. Bar-Shalom, K. C. Chang, and H. A. P. Blom. Automatic track formation in clutter with a recursive algorithm. In *Proceedings of the 28th IEEE Conference on Decision and Control,*, pages 1402–1408 vol.2, Dec 1989.

[112] Y. Bar-Shalom and W. Blair. *Multitarget-multisensor Tracking: Applications and Advances*. Number v. 3 in Artech House radar library. Artech House, 1990.

[113] J.A Guu and C.H. Wei. Maneuvering target tracking using IMM method at high measurement frequency. *IEEE Transactions on Aerospace and Electronic Systems*, 27(3):514–519, May 1991.

[114] R. Kenefic. Active sonar application of a U-D square root PDAF. *IEEE Transactions on Aerospace and Electronic Systems*, 26(5):850–857, Sep. 1990.

[115] H.A.P. Blom and Y. Bar-Shalom. Time-reversion of a hybrid state stochastic difference system with a jump-linear smoothing application. *IEEE Transactions on Information Theory*, 36(4):836–847, 1990.

[116] B. Chen and J.K. Tugnait. Interacting multiple model fixed-lag smoothing algorithm for Markovian switching systems. *IEEE Transactions on Aerospace and Electronic Systems*, 36(1):243–250, 2000.

[117] R.E. Helmick, W.D. Blair, and S.A. Hoffman. Fixed-interval smoothing for markovian switching systems. *IEEE Transactions on Information Theory*, 41(6):1845–1855, 1995.

[118] N. Nadarajah et.al. Imm forward filtering and backward smoothing for maneuvering target tracking. *IEEE Transactions on Aerospace and Electronic Systems*, 48: 2673–2678, 07 2012.

[119] O. E. Drummond. Multiple target tracking with multiple frame, probabilistic data association. In *Signal and Data Processing of Small Targets 1993*, volume 1954, pages 394 – 408. International Society for Optics and Photonics, SPIE, 1993.

[120] O E. Drummond. Target tracking with retrodicted discrete probabilities. In *Signal and Data Processing of Small Targets 1997*, volume 3163, pages 249 – 268. International Society for Optics and Photonics, SPIE, 1997.