UNIVERSITY OF LIVERPOOL

Thesis submitted in accordance with the requirements of the University of Liverpool for the degree of
Doctor in Philosophy

Mathematical Models and Monte-Carlo Algorithms for Improved Detection of Targets in the Commercial Maritime Domain

Written by
Lyudmil Zhivkov Vladimirov

Supervised by
Prof. Simon Maskell
Prof. Jason Ralph
Department of Electrical Engineering and Electronics,
University of Liverpool

In Industrial Collaboration with
Dr. Simon Lee
Denbridge Marine Ltd.

DENBRIDGE MARINE
innovation in maritime surveillance

June 30, 2020

Abstract

Commercial Vessel Traffic Monitoring Services (VTMSs) are widely used by port authorities and the military to improve the safety and efficiency of navigation, as well as to ensure the security of ports and marine life as a whole. Technology based on the Kalman Filtering framework is in widespread use in modern operational VTMS systems. At a research level, there has also been a significant interest in Particle Filters, which are widely researched but far less widely applied to deliver an operational advantage. The Monte-Carlo nature of Particle Filters places them as the ideal candidate for solving the highly non-linear, non-Gaussian problems encountered by modern VTMS systems. However, somewhat counter-intuitively, while Particle Filters are best suited to exploit such non-linear, non-Gaussian problems, they are most frequently used within a context that is mostly linear and Gaussian. The engineering challenge tackled by the PhD project reported in this thesis was to study and experiment with models that are well placed to capitalise on the abilities of Particle Filters and to develop solutions that make use of such models to deliver a direct operational advantage in real applications within the commercial maritime domain.

Acknowledgments

The success and final outcome of this project required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my thesis.

Firstly, I want to thank my close friends and family for their endless love and support during this long journey. I am especially grateful to my parents, Nadezhda and Zhivko, who supported me both emotionally and financially in times of need, while encouraging me in all of my pursuits, inspiring me to follow my dreams and teaching me that my job in life was to learn. I would also like to take this opportunity to commemorate my recently departed grandfather Metodi, who never failed to lift me up in tough moments with his jokes and kind words.

Continuing, I would like to express my sincere gratitude to both my academic supervisor Prof. Simon Maskell from University of Liverpool, and my industrial supervisor Dr. Simon Lee from Denbridge Marine Ltd., for their continuous guidance and support over the past 5 years, starting even before the start and continuing even beyond the end of my PhD. I would also like to thank them both for their patience, empathy, encouragement, motivation, friendship and great sense of humor over all these years. I have been extremely lucky to have had not one, but two supervisors who cared so much both on a professional, as well as on a personal level.

I would also like to thank Mr James Wright, Mr Lykourgos Kekempanos, Dr Paul Horridge, Dr Yifan Zhou, Dr Flavio De Melo of University of Liverpool for their invaluable friendship and support. I also thank Mr Joe Cummings, Mr Michael Denn, Dr Ian McConnell, Mr David Hubberstey, Mr Richard Griffin, Mr Jonathan Brady, Mr Amjad Shaheed and Mr Tommy Wong of Denbridge Marine Ltd. for their invaluable support, guidance and assistance during my industrial placement at the company.

Finally, I would like to thank the Engineering and Physical Sciences Research Council and the Smith Institute for Industrial Mathematics and System Engineering, not only for facilitating and providing the iCase Studentship award which allowed me to undertake this research, but also for giving me the opportunity to attend conferences and meet so many interesting people.

Contents

CHAPTER 1

Introduction

1.1 Motivation & Scope

Maritime Traffic Surveillance (MTS) is a crucial field for a large body of international institutions and agencies. By definition, MTS involves the effective observation and understanding of all activities carried out at sea. As such, MTS incorporates the surveillance of regions of interest (i.e. ports), using various deployments of sensors, with the optimal aim of ensuring the efficiency and safety of vessel navigation as well as the effective protection of the maritime environment. Vessel Traffic Services (VTS) then constitute MTS systems which have been developed by VTS providers and established by the relevant harbour or port authorities aiming to achieve the above tasks[1].

Multi-Target Tracking (MTT) systems form a crucial component of VTS systems. Their role is to accurately track the positions (and possibly other information, such as speed and course) of all vessels, cruising through the regions of interest, at any given point in time. As a result, a great amount of research has been undertaken over the years, aimed at identifying MTT techniques that can provide the means for dealing with the involved uncertainty, with the objective of making efficient use of all the available data, as well as any prior knowledge,

[1]Denbridge Marine Ltd., the company that co-funds the PhD project undertaken by the author, are a local Small-to-Medium sized Enterprise (SME) who focus on the development and deployment of VTSs across the globe.

received from a given deployment of sensors, so as to effectively estimate and track the state of both single and multiple moving targets.

Sea clutter, obscured targets and relatively low resolution in both range and azimuth, constitute some of the major barriers to maritime X/S-band radar detection and tracking [1, 2]. The combined effects of the above pose a great burden on the effective plot association of targets with data, leading to problems such as track merging, false tracks and even missed targets. Conventional MTT systems perform detection by thresholding the data to exclude spurious (low-SNR) tracks, before tracking is carried out. Nonetheless, such approaches fail to consider the spatial behaviour of clutter and targets, meaning that they are still prone to detection errors. A solution to this problem is provided by Track-Before-Detect (TkBD) algorithms, where detection and tracking are performed simultaneously on the raw data, allowing for clutter models to be directly incorporated in the process. Implementation of such techniques however is quite challenging due to their inherent non-linear/non-Gaussian nature.

Technology based on the Kalman Filtering framework is in widespread use in modern operational target tracking systems [3, 4, 5]. At a research level, there has also been a significant interest in Particle Filters [6, 7], which are widely researched but far less widely applied to deliver an operational advantage. Perhaps surprisingly, objects are typically modelled as moving with a nearly constant Cartesian velocity (i.e. integrated independent Brownian motion in latitude and longitude) and not as moving according to a nearly constant polar velocity (i.e. speed and heading) [8, 9, 10]. In actual fact, it is possible to devise high-accuracy solutions to such SDEs and so derive non-linear non-Gaussian dynamic models suitable for use in multi-target fusion systems. Initial scoping work [11] indicates that, while standard multi-target tracking systems (e.g. based on the Kalman Filter) are degraded by the use of such models, Monte-Carlo techniques (e.g. the Particle Filter) can embrace the non-linear non-Gaussian nature of the models.

The engineering challenge tackled by the PhD project reported in this thesis is to study and experiment with models that are well placed to capitalise on the abilities of Particle Filters and to develop solutions that make use of such models to deliver a direct operational advantage in real applications within the commercial maritime domain. To develop high performance multi-target tracking algorithms, one needs to: understand single-target tracking; understand multi-target tracking; estimate the parameters of models used; have a semi-automated method for evaluating performance using an independent data source; understand which parts of the algorithm are responsible for any improvement in performance. This motivates the work comprising this thesis and is accomplished by:

1. Presenting an in-depth background and review of existing algorithms and

methodologies that pertain to the problem of Single-Target Tracking.

2. Experimenting with non-linear, non-Gaussian models and demonstrating that an operational advantage can be achieved with the use of such models in a Particle Filtering context.

3. Presenting an in-depth background to Multi-Target Tracking methods and drawing relations between conventional approaches and the state-of-the-art.

4. Presenting new applications of state-of-the-art algorithms, with particular focus on applications relating to maritime radar and electro-optical sensors. Demonstrating the good performance of such algorithms for these applications using a mixture of real and simulated data.

5. Actively engaging with the target tracking community to develop solutions that improve the ability of future researchers to develop and evaluate the performance of new target tracking solutions.

1.2 Organisation of the Thesis

Chapter 2 formulates of the fundamental framework of state-space models as considered in this thesis and introduces readers to the more specific examples of models that shall be utilised in subsequent chapters. Section 2.2 starts by describing the format of stochastic dynamic models that describe the evolution of a target's state over time, while Section 2.3 concerns itself with the measurement models used to transform between target and measurement spaces, while modelling the stochastic characteristics of the associated measurement noise. Continuing, Section 2.4 discusses the notion of detection models which are used to model target detectability across a given surveillance space. Finally, clutter models that describe the statistical characteristics of clutter are presented and discussed in Section 2.5.

Chapter 3 builds on the framework of state-space models presented in Chapter 2 and aims to delve into the specifics of the aforementioned Bayesian Filtering and Data Association problems in the context of Single Target Tracking. Section 3.2 outlines the fundamental concepts of the standard Bayesian Filtering framework, which are then utilised to formulate the Kalman and Particle Filtering algorithms, while discussing the various advantages and pitfalls of each approach. Section 3.3 then proceeds to define the Data Association problem in the context of Single-Target Tracking, while presenting some well founded algorithms that can be used to solve it. Finally, performance evaluations are performed in Section 3.4, to test the performance of the presented algorithms.

Chapter 4 builds on the concepts of state-space modelling and Single-Target Tracking presented in the previous chapters, and extends the discussion to the case of Multi-Target Tracking. Section 4.2 introduces the Multi-Target Bayes Filter, while drawing relation to the Standard Bayes Filter. Section 4.3 provides a summary and brief derivation of the algorithms that form the main components of conventional Multi-Target trackers, with special focus to the utilised Data Association and Track Management methods. In the same section, the author presents a discussion on the relation between the Joint Probabilistic Data Association (JPDA) and Joint Integrated PDA algorithms, with the aim of highlighting how the latter can be performed using the same constructs. Section 4.5 continues by discussing the concept of Random Finite Sets (RFS) and presents a formulation of the Probability Hypothesis Density (PHD) filter as an approximation to the Multi-target Bayes Filter. Section 4.6 discusses a state-of-the-art radar track initiation technique which utilises a PHD filter to model the density of uninitiated targets and consecutively propose tracks for initiation on the basis of target existence probabilities. Preliminary simulation results are presented in Section 4.6.1 to showcase the performance benefits of the PHD track initiator compared to other mainstay approaches using synthetic data, while Section 4.7 presents a case study performed on real data collected from a commercial radar, whereby a more thorough qualitative analysis is performed on a pair of challenging scenarios, with the aim of demonstrating a real operational advantage.

Chapter 5 elaborates on the problem of parameter estimation in Linear Dynamical Systems (LDSs) and demonstrates the applicability of Expectation Maximisation to parameter estimation for LDSs with control inputs. The objective is to present a brief derivation of the complete set of equations required to perform parameter estimation for a LDS with control inputs, as well as a straightforward formulation of the relevant algorithmic EM steps. Section 5.2 begins by defining the complete log-likelihood equation for a generic LDS with control inputs, and proceeds by presenting a brief derivation of the EM equations, used to achieve the local optima. Section 5.3 introduces a case study conducted on a segway platform to demonstrate the applicability and achieved performance of the proposed method. Simulated numerical results are also shown in Section 5.3, along with graphs indicating the existing correlations between different parameters

Chapter 6 provides a thorough description and review of the literature in the context of vessel detection and tracking, using video data from active Electro-Optical (EO) sensors. Section 6.2 initiates the discussions by introducing the reader to the basic concepts and notation used in this Chapter. Section 6.3 is then focused on the engineering challenges relating to development of accurate real-time ship detectors with the use of active EO sensors (cameras). A review of existing approaches is presented, followed by an introduction to the state-of-the-art detectors that base their operation on Convolutional Neural Networks (CNNs). Under the same section, a custom CNN-based ship detector is introduced, followed by brief report of the followed methodologies and a comprehensive performance analysis. Continuing, Section 6.4 demonstrates an effective method for estimating and accounting for the errors induced by the camera motion in real-time. The use of an active (Pan-Tilt-Zoom) camera, introduces errors due to the inherent ability of the camera to exhibit motion. Thus, in addition to the multi-target tracking complexities discussed in Chapter 4, further measures must be employed in order to ensure that the positions of targets are estimated accurately, even after the camera has changed its orientation.

Chapter 7 shall focus on presenting work done by the author on developing and contributing towards the development of open-source frameworks for tracking and state estimation. Section 7.3 discusses Stone Soup [12, 13, 14], an open-source Python framework that is currently under development, stemming from combined international efforts led by the Defence Science and Technology Laboratory (UK) in direct collaboration with the Defence Research and Development Canada (Canada), the Air Force Research Laboratory (US), and the University of Liverpool (UK), to which the author has been an major contributor. Section 7.2 introduces an open-source object-oriented MATLAB toolbox, developed by the author over the course of the PhD project and comprised of efficiently coded implementations of target tracking algorithms, with the aim of assisting and accelerating future research within the field.

1.3 Original Contributions

- Contributions to Multi-Target Tracking:
  - Presentation and derivation of the relations between the Joint Probabilistic Data Association (JPDA) and Joint Integrated Probabilistic Data Association (JIPDA) algorithms. (Section 4.4)
  - Implementation of a state-of-the-art radar track initiation scheme with the use of a PHD filter and demonstration of results using a mixture of real and simulated data. (Sections 4.6 & 4.7)
- Contributions to Parameter Estimation (Chapter 5)[2]:
  - Brief derivation of the complete set of equations required to perform parameter estimation for a LDS with control inputs, using Expectation

[2]This work is part of a co-authored paper, to which the author is the leading author.

Maximisation, as well as a straightforward formulation of the relevant algorithmic EM steps.

- Presentation of a method for obtaining initial estimates, under the assumption that the hidden states are fully observable.
- Demonstration of the performance implications stemming from bad choice of initial estimates, given that no parameters are constrained throughout the learning process.
- Test results demonstrating the effectiveness of Expectation Maximisation for parameter estimation of an LDS in a mixed real/simulation context.

- Contributions to Maritime Video Detection and Tracking:
  - Development of a robust detector for improved detection of ships in Electro-Optical imagery data, using Convolutional Neural Network models, and demonstration of the achieved operational benefits. (Section 6.3)
  - Demonstration of an effective method for estimating and accounting for the errors induced by the camera motion in real-time with the use of Optical Flow and Affine Transformation estimation. (Section 6.4)
- Contributions to Modular Frameworks for Tracking and State Estimation:
  - Active engagement and contribution towards international efforts for the development of an open-source Python framework (Stone Soup [14]) that aims to provide researchers and practitioners with the ability to develop and implement new and existing tracking and state estimation algorithms for ease of comparison[3]. (Section 7.3)
  - Development of a new unified, re-usable, highly-modular and open-source object-oriented MATLAB toolbox (TrackingX), comprised of efficient implementations of state-of-the-art target tracking and state estimation algorithms. (Section 7.2)

[3]This work is part of collaborative efforts with scientists and developers from across the globe, leading to the publication of [14], to which the author has been a major contributor and co-author.

CHAPTER 2

State-space models

2.1 Introduction

State estimation can be viewed as a time series analysis problem, where the goal is to estimate the state of a (set of) target(s), over a given period of time. As such, the problem can be conveniently formulated with the use of probabilistic state-space models. Such probabilistic models can be utilised to describe the various processes that are taken into account when attempting to estimate the state of a number of targets within a given surveillance region of interest, based on corrupted and potentially cluttered or incomplete measurements.

Although real systems are generally considered in continuous time, it is a general convention to use discrete time notation to describe the state-space models. Thus, without loss of generality, we assume that time is discretised and times of interest fall at time instants t_k, $k = 1, 2, ...$, but the time between timesteps $\Delta t = t_k - t_{k-1}$ can vary. At any time instant t_k there can exists a variable number of targets within the surveillance region observed by a fixed deployment of sensors. In this context, the state of a target at time t_k, $\mathrm{x}_k \in \mathcal{X}$, forms the hidden variable of interest and encapsulates the complete information pertaining to the kinematic characteristics of the target (such as position, speed, orientation, etc).

In a similar manner to the target state, the state of a sensor $\mathrm{s}_k \in \mathcal{S}$ contains all kinematic information available for that sensor, which may vary over time, but is assumed to be known in the context of this thesis. Unless stated otherwise, it is assumed that only a single sensor generates measurements at any time instant. The data received from a sensor come in the form of measurement scans Y_k, where each measurement scan can be viewed as a set of measurements $Y_k = \{\mathrm{y}_k^j\}$, $j = 0, ..., M$, where for $j = 0$ we have that $Y_k = \emptyset$. Each measurement $\mathrm{y}_k^j \in \mathcal{Y}$ can either be generated by an existing target, in which case it is viewed as a perturbed transformation of the target state, or can be the result of background interference (a.k.a. clutter) and/or internal interference of the sensor. In any case, unless explicitly stated, the origin of each measurement is assumed to be unknown in the context of this thesis.

This chapter aims to formulate of the fundamental framework of state-space models as considered in this thesis, as well as present readers to the more specific examples of models that shall be utilised in subsequent chapters. Section 2.2 starts by describing the format of stochastic dynamic models that describe the evolution of a target's state over time, while Section 2.3 concerns itself with the measurement models used to transform between target and measurement spaces, while modelling the stochastic characteristics of the associated measurement noise. Continuing, Section 2.4 discusses the notion of detection models which are used to model target detectability across a given surveillance space. Finally, clutter models that describe the statistical characteristics of clutter are presented and discussed in Section 2.5.

2.2 Dynamic models

Dynamic models are used to describe the evolution of the target state with time. In its generic formulation, a dynamic model encapsulates the statistical process and the uncertainties involved in the process since the beginning of time (i.e. $1 : k - 1$) and is expressed as:

$$\mathrm{x}_k = f(\mathrm{x}_{1:k-1}, \mathrm{q}_{1:k}, \mathrm{y}_{1:k-1}) \sim p(\mathrm{x}_k | \mathrm{x}_{1:k-1}, \mathrm{y}_{1:k-1}) \tag{2.1}$$

where $p(\mathrm{x}_k|.)$ denotes the dynamic model pdf, which describes the stochastic dynamics of the tracked process, $f : \mathbb{R}^{n_\mathrm{x}} \times \mathbb{R}^{n_q} \rightarrow \mathbb{R}^{n_\mathrm{x}}$ is a possibly nonlinear function of the states $\mathrm{x}_{1:k-1}$ and the process noise $\mathrm{q}_{1:k}$, with n_x and n_q denoting the dimensions of the state and process noise vectors, respectively. In addition, the states of targets are defined on the target state-space $\mathcal{X}$, such that $\mathrm{x}_k \in \mathcal{X}$.

As is standard in the literature [15, 16], it is common to assume that the processes described by dynamic models are Markovian. Thus, the Markov property [17, 18] that is applicable to such models dictates that the state x_{k-1} is a sufficient statistic of the history of states $\mathrm{x}_{1:k-1}$. Thus, the joint distribution of states up to, and including time t_k can be expressed as:

$$p(\mathrm{x}_{1:\mathrm{k}}) = p(\mathrm{x}_1) \prod_{k'=2}^{k} p(\mathrm{x}_{k'}|\mathrm{x}_{k'-1}) \tag{2.2}$$

This fact allows for a simplification of the dynamic model equation (2.1), as follows:

$$\mathrm{x}_k = f_{k|k-1}(\mathrm{x}_{k-1}, \mathrm{q}_k) \sim p(\mathrm{x}_k|\mathrm{x}_{k-1}) \tag{2.3}$$

where $f_{k|k-1} : \mathbb{R}^{n_\mathrm{x}} \times \mathbb{R}^{n_q} \to \mathbb{R}^{n_\mathrm{x}}$ is the Markovian transition kernel.

2.2.1 Random Walk (RW)

The simplest model that one can consider is that of a Random Walk [19], where the state at a given time is the state at a previous time plus random noise, assumed to evolve according to a Brownian motion. Thus, when considering a 1-dimensional (1-D) scenario, the state vector can be represented simply by the positional component:

$$\mathrm{x}_k = x_k \tag{2.4}$$

Velocity can be viewed as white noise, typically considered to be zero-mean Gaussian over some period $\Delta t = t_k - t_{k-1}$, leading to a position that is following a Random Walk, which is the integral of white noise. The model can then be described by the following Stochastic Differential Equation (SDE):

$$dx_k = \sigma dw_k \tag{2.5}$$

In the above, w_k denotes a standard Brownian motion, with $w_k \sim \mathcal{N}(0, t_k)$. Formulating the above in a similar manner to (2.3), leads to the following expression:

$$\mathrm{x}_k = f_{k|k-1}(\mathrm{x}_{k-1}, \mathrm{q}_k) = \mathrm{x}_{k-1} + \mathrm{q}_k,\ \mathrm{q}_k \sim \mathcal{N}\left(0, \sigma^2 \Delta t\right) \tag{2.6}$$

Extending the Random Walk model to n_d dimensions can be achieved by vertically cascading the state vector, i.e. $\mathrm{x}_k = [\mathrm{x}_k^1; \mathrm{x}_k^2; ...; \mathrm{x}_k^{n_d}]$, and process noise, i.e. $\mathrm{q}_k = [q_k^1, q_k^2, ..., q_k^{n_d}]^T$, where the exponent is used as a dimension identifier, leading to a vectorised expression of (2.6).

2.2.2 Constant Velocity (CV)

As opposed to the previous model, most targets can be assumed to exhibit some consistency of motion, i.e. they tend to maintain a constant velocity. A common way of representing this is to model the acceleration as white noise, leading to a velocity that follows Random Walk and a position that is the integral of velocity. In this model, the state vector for the fundamental 1-D case takes the following form:

$$\mathrm{x}_k = [x_k, \dot{x}_k]^T \tag{2.7}$$

and the SDEs that describe the model are given below [19]:

$$\begin{aligned} dx_k &= \dot{x}_k dt \\ d\dot{x}_k &= \sigma dw_k \end{aligned} \tag{2.8}$$

Under the assumption of zero-mean Gaussian noise, the model can be expressed in terms of (2.3) as follows:

$$\mathrm{x}_k = f_{k|k-1}(\mathrm{x}_{k-1}, \mathrm{q}_k) = F_{CV}\mathrm{x}_{k-1} + \mathrm{q}_k,\ \ \mathrm{q}_k \sim \mathcal{N}\left(0_{2\times 1}, Q_{CV}\right) \tag{2.9}$$

where:

$$\begin{aligned} F_{CV} &= \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \\ Q_{CV} &= \begin{bmatrix} \frac{\Delta t^3}{3} & \frac{\Delta t^2}{2} \\ \frac{\Delta t^2}{2} & \Delta t \end{bmatrix} \sigma^2 \end{aligned} \tag{2.10}$$

To extend the Constant Velocity model to n_d dimensions, the combined state vector is formed by vertically cascading the state vectors for each dimension, i.e. $\mathrm{x}_k = [\mathrm{x}_k^1; \mathrm{x}_k^2; ...; \mathrm{x}_k^{n_d}]$, where the exponent is used as a dimension identifier. Then, to comply with (2.9), the combined F_{CV} and Q_{CV} matrices are composed as block diagonal combinations of the fundamental matrices of (2.10), e.g.:

$$\begin{aligned} F_{CV} &= diag\{F_{CV}^1, F_{CV}^2, ..., F_{CV}^{n_d}\} \\ Q_{CV} &= diag\{Q_{CV}^1, Q_{CV}^2, ..., Q_{CV}^{n_d}\} \end{aligned} \tag{2.11}$$

where, once again, the exponent values $1, ..., n_d$ are used to denote the dimension identifiers, rather than powers.

2.2.3 Integrated Ornstein-Uhlenbeck (IOU)

The Integrated Ornstein-Uhlenbeck model [16] (a.k.a. Constant Velocity model with Drag [19]) is similar to the previous (CV) model, but also includes a damping term on the velocity component that leads to an exponential decay of the velocity over time. This is preferred in some instances, particularly in simulations, as it admits a steady state velocity, whereas the variance of velocity under the CV model grows unbounded with time.

The state vector in IOU is identical to that of (2.7), while the model SDEs are given below [19]:

$$\begin{aligned} dx_k &= \dot{x}_k dt \\ d\dot{x}_k &= -\alpha\dot{x}_k dt + \sigma dw_k \end{aligned} \tag{2.12}$$

where $\alpha > 0$ is the damping (a.k.a. drag) coefficient. As it should be evident from (2.12), for $\alpha = 0$ the IOU model reduces to a CV.

Under the common assumption of zero-mean Gaussian noise, the model can be formulated in terms of (2.3) as follows:

$$\mathrm{x}_k = f_{k|k-1}(\mathrm{x}_{k-1}, \mathrm{q}_k) = F_{IOU}\mathrm{x}_{k-1} + \mathrm{q}_k,\ \mathrm{q}_k \sim \mathcal{N}\left(0_{2\times1}, Q_{IOU}\right) \tag{2.13}$$

where we have that [16]:

$$\begin{aligned} F_{IOU} &= \begin{bmatrix} 1 & \frac{1-e^{-\alpha\Delta t}}{\alpha} \\ 0 & e^{-\alpha\Delta t} \end{bmatrix} \\ Q_{IOU} &= \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix}\sigma^2 \end{aligned} \tag{2.14}$$

and

$$\begin{aligned} Q_{11} &= \alpha^{-2}[\Delta t - 2\alpha^{-1}(1-e^{-\alpha\Delta t}) + \frac{1}{2}\alpha^{-1}(1-e^{-2\alpha\Delta t})] \\ Q_{12} &= Q_{21} = \alpha^{-2}[(1-e^{-\alpha\Delta t}) - \frac{1}{2}(1-e^{-2\alpha\Delta t})] \\ Q_{22} &= \frac{1}{2}\alpha^{-1}(1-e^{-2\alpha\Delta t}) \end{aligned} \tag{2.15}$$

Extension of the OU model to multiple dimensions can be achieved using the same process described previously for the CV model.

2.2.4 Constant Heading (CH)

The Constant Heading model [11] describes the motion of targets on a two-dimensional (Cartesian) plane, under the assumption of nearly constant heading (hence the name) and absolute velocity. In other words, the CH model assumes that the absolute velocity and heading of a target follow a Random Walk. Thus, the absolute acceleration and turn-rate are modelled as white noise components that evolve according to two independent Brownian motions.

The state vector for the CH model is formed as follows:

$$\mathrm{x}_k = [x_k, y_k, s_k, \phi_k]^T \tag{2.16}$$

where x_k, y_k denote the position on the x and y axes, respectively, $s_k = \sqrt{\dot{x}_k^2 + \dot{y}_k^2}$ is the absolute velocity and ϕ_k is the heading. The system can then be described by the following set of SDEs:

$$\begin{aligned} dx_k &= s_k \cos\phi dt \\ dy_k &= s_k \sin\phi dt \\ ds_k &= \sigma_s dw_k \\ d\phi_k &= \sigma_\phi db_k \end{aligned} \tag{2.17}$$

Due to the non-linear nature of the CH model, a closed-form expression in terms of linear algebra operations, similar to that of the previous models, does not exist. It is possible to obtain an approximation under certain assumptions [11], however the expression and derivation are rather lengthy and thus not considered here. Nevertheless, we can still express the model in the form of (2.3), as shown below:

$$\mathrm{x}_k = f_{k|k-1}(\mathrm{x}_{k-1}, \mathrm{q}_k) = \begin{bmatrix} x_{k-1} + s_{k-1}\cos\phi_{k-1}\Delta t \\ y_{k-1} + s_{k-1}\sin\phi_{k-1}\Delta t \\ s_{k-1} \\ \phi_{k-1} \end{bmatrix} + \mathrm{q}_k,\ \mathrm{q}_k \sim \mathcal{N}\left(0_{4\times1}, Q_{CH}\right) \tag{2.18}$$

where $Q_{CH} = diag\{0, 0, \sigma_s^2\Delta t, \sigma_\phi^2\Delta t\}$.

A great advantage of the CH model lies in the offered decoupling between a target's speed and relative heading. This fact allows for the noise parameters of each individual state variable to be configured independently from one another, allowing for finer parameterisation of the model.

2.3 Measurement models

Measurement models are used to describe the transformation/relationship between the measurement and target state-spaces. In addition to the above, a measurement model also describes the statistical characteristics of the noise that is present in the received measurements.

The generic form and structure of a measurement model can be expressed as:

$$\mathrm{y}_k = h(\mathrm{x}_{1:k}, \mathrm{r}_{1:k}, \mathrm{y}_{1:k-1}) \sim p(\mathrm{y}_k|\mathrm{x}_{1:k}, \mathrm{y}_{1:k-1}) \tag{2.19}$$

where $p(\mathrm{y}_k|.)$ is the measurement model pdf, $h_k : \mathbb{R}^{n_\mathrm{x}} \times \mathbb{R}^{n_r} \rightarrow \mathbb{R}^{n_\mathrm{y}}$ is a possibly nonlinear function of the state x_k and a measurement noise sequence r_k[1], with n_y, n_r being dimensions of the measurement and measurement noise vectors, respectively. What is more, measurements are defined on the measurement state-space $\mathcal{Y}$, such that $\mathrm{y}_k \in \mathcal{Y}$.

Similar to the case of dynamic models, it is often assumed that measurements are conditionally independent of both past states $\mathrm{x}_{1:k-1}$ and measurements $\mathrm{y}_{1:k-1}$, thus leading to a simplification of (2.19):

$$\mathrm{y}_k = h_k(\mathrm{x}_k, \mathrm{r}_k) \sim p(\mathrm{y}_k|\mathrm{x}_k) \tag{2.20}$$

It is also worth noting here that the pdf $p(\mathrm{y}_k|\mathrm{x}_k)$ in (2.20) also describes the measurement likelihood function, used to evaluate the validity of states given a measurement.

While (2.20) defines the transformation from state-space to measurement-space ($\mathcal{X} \rightarrow \mathcal{Y}$), it is often desirable to define the reverse function that allows to perform the transformation from measurement-space to state-space ($\mathcal{Y} \rightarrow \mathcal{X}$), i.e.:

$$\mathrm{x}_k = h_k^{-1}(\mathrm{y}_k, \mathrm{r}_k) \tag{2.21}$$

Although the relation of (2.21) can be very convenient in a number of scenarios, it is worth noting that it is not always possible to define a closed-form and/or deterministic expression of the inverse function $h_k^{-1}(.)$.

[1]To avoid confusion in following discussions, a distinction is made between the notations r_k, used to denote the measurement noise samples, and r_k that will be introduced in Section 2.3.2 to denote range.

2.3.1 Standard Linear-Gaussian (LG)

Measurements are often considered to be linear transformations of the state, that are corrupted by additive zero-mean Gaussian noise. The class of models used to describe this relation are referred to as Linear-Gaussian and are typically expressed in terms of (2.20) as follows:

$$\mathrm{y}_k = h_k(\mathrm{x}_k, \mathrm{r}_k) = H\mathrm{x}_k + \mathrm{r}_k,\ \mathrm{r}_k \sim \mathcal{N}\left(0_{n_\mathrm{y}\times 1}, R\right) \tag{2.22}$$

where H is a time-invariant $n_\mathrm{x} \times n_\mathrm{y}$ measurement matrix and R is a $n_\mathrm{y} \times n_\mathrm{y}$ measurement noise covariance matrix.

The inverse relation of (2.21) for this class of models can also be defined as:

$$\hat{\mathrm{x}}_k = h_k^{-1}(\mathrm{y}_k, \mathrm{r}_k) = H^{+}(\mathrm{y}_k - \mathrm{r}_k) \tag{2.23}$$

where H^{+} denotes the pseudo-inverse of H. The notation $\hat{\mathrm{x}}_k$ is used due to the fact that a solution to (2.23) may not exist, or if one exists, it may not be unique. If, and only if, H is invertible, then $H^{+} = H^{-1}$ and $\hat{\mathrm{x}}_k = \mathrm{x}_k$.

When the received measurements form a 1-to-1 representation of (part of) the state, i.e. $\mathcal{Y} \subseteq \mathcal{X}$, the measurement matrix H takes the form of an binary map matrix. For example, considering an exemplar scenario where the target state is of the form (2.7), where only the positional component of the state is observable, the measurement matrix takes the form

$$H = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{2.24}$$

Furthermore, it is common to assume that the noise on each measurement coordinate is independent of all other coordinates, meaning that the covariance matrix R can be viewed as a block diagonal combination of the variance on each dimension. For the 2-D expansion of (2.7), where only the positional components of the state are observable, this would lead to $R = diag\{\sigma_x^2, \sigma_y^2\}$, with σ_x, σ_y denoting the standard deviation on the x and y axes, respectively.

2.3.2 Gaussian Azimuth-Range (GAR)

As is often the case in real target tracking applications, measurements are generated in terms of polar (azimuth, range) coordinates, relative to the state $\mathrm{s}_k \in \mathcal{S}$ of some sensor, where the noise is defined as zero-mean Gaussian on the polar coordinate frame. This representation is particularly useful when dealing with data received from active radar or sonar systems.

The measurement state-space $\mathcal{Y}$ for the model is defined on the Polar coordinate system, whose pole position is aligned with the positional $(x_{\mathrm{s},k}, y_{\mathrm{s},k})$ Cartesian coordinates in the sensor state s_k, and thus can vary over time. For purposes of simplicity, we assume that $\mathcal{X}$ and $\mathcal{S}$ are defined on a common Cartesian system and the polar axis of $\mathcal{Y}$ aligns with the x-axis of that system. Thus, the vector of measurements generated by the GAR model is of following form:

$$\mathrm{y}_\mathrm{k} = [\theta_k, r_k]^T \tag{2.25}$$

where θ_k an r_k denote the azimuth and range reported at time t_k, respectively.

Based on the assumption that the target state-space $\mathcal{X}$ and sensor state-space $\mathcal{S}$ are defined as super-sets of the same 2-D positional Cartesian space, the model of interest can be expressed in terms of (2.20) as follows:

$$\begin{aligned}\mathrm{y}_k &= h_k(\mathrm{x}_k, \mathrm{r}_k)\\ &= \begin{bmatrix}\mathrm{arctan2}(\bar{y}_k, \bar{x}_k)\\ \sqrt{\bar{x}_k^2 + \bar{y}_k^2}\end{bmatrix} + \mathrm{r}_k,\ \mathrm{r}_k \sim \mathcal{N}\left(0_{2\times1}, \begin{bmatrix}\sigma_\theta^2 & 0\\ 0 & \sigma_r^2\end{bmatrix}\right)\end{aligned} \tag{2.26}$$

where $\bar{x}_k = x_k - x_{s,k}$, $\bar{y}_k = y_k - y_{s,k}$, with $(x_k, y_k) \subseteq \mathrm{x}_k$ and $(x_{s,k}, y_{s,k}) \subseteq \mathrm{s}_k$ denoting the Cartesian positional coordinates of the target and sensor respectively, while σ_θ, σ_r denote the standard deviation of the noise on for the azimuth and range components, respectively.

The inverse relation of (2.21) for the GAR model can be defined as:

$$\begin{aligned}\hat{\mathrm{x}}_k &= h_k^{-1}(\mathrm{y}_k, \mathrm{r}_k)\\ &= \begin{bmatrix}\hat{r}_k \cos(\hat{\theta}_k) + x_{s,k}\\ \hat{r}_k \sin(\hat{\theta}_k) + y_{s,k}\end{bmatrix},\end{aligned} \tag{2.27}$$

where $[\hat{\theta}_k, \hat{r}_k]^T = \mathrm{y}_k - \mathrm{r}_k$. It is worth noting that $\hat{\mathrm{x}}_k = [x_k, y_k]^T$ and $\hat{\mathrm{x}}_k \subseteq \mathrm{x}_k$, as the solution of (2.27) only returns positional information about the target state x_k. Furthermore, the relation of (2.27) is generally known to give biased and inconsistent estimates for certain levels of the cross-range measurement error [20]. Unbiased methods for inverting bearing and range have been demonstrated in [21].

2.4 Detection models

When tracking targets using non-ideal sensors, a common problem that one has to consider is that of target detectability. Targets can either be detected, and thus generate a detection which is reported by a received measurement scan, or can go undetected. To model this behaviour it is common practice to use the notion of a detection model, which can be used to validate the extent to which any target is detectable.

In their most generic form, detection models can be described by a probability distribution function (pdf), such that

$$p_{d_k} = p(d_k = 1|.) \tag{2.28}$$

where $d_k \in [0, 1]$ is a binary random variable, with $d_k = 1$ denoting the successful detection event at time t_k, while p_{d_k} measures the probability of detection, evaluated through the detection model pdf $p(d_k = 1|.)$. The use of a dotted notation in the definition of the pdf is intentional, as the pdf can be defined to be conditional on a number of different variables.

2.4.1 Constant Detection Rate (CDR)

The simplest and most common case of a detection model is one that models the detection as being constant across the entire search space. In this case (2.28) reduces to the following equation:

$$p_{d_k} = P_D \tag{2.29}$$

where $P_D \in [0, 1]$ is the chosen constant detection probability.

The main advantage of the CDR model is its relative simplicity and speed of evaluation, as it circumvents the evaluation of any pdfs and removes the necessity of calculating any state or measurement related quantities. As such, CDR is the most common choice of detection model for simulation purposes. However, a major drawback of the CDR model is its inability to incorporate any information regarding known regions of low and/or detectability, thus making it a sub-optimal choice for real systems.

2.4.2 State Dependent Detection Rate (SDDR)

As opposed to the previous model, the SDDR model allows the detection rate to vary based on a given quantity of interest. In its most common formulation, SDDR is defined such that the detection probability is dependent on the state of a single target. Thus, the model can be expressed in terms of (2.28) as follows:

$$p_{d_k} = p(d_k = 1|\mathrm{x}_k) \tag{2.30}$$

where $p(d_k|\mathrm{x}_k)$ can take the form of any pdf or pmf, which when evaluated returns values strictly in the range $[0,1]$.

A simplistic example of such a model would be in the case where there exists a well defined detection grid, over a given surveillance region $\mathcal{V}$. Assuming a 1-D problem, where targets can only be detected within $\mathcal{V} \in [0,1000]$ with probability of 1, and 0 elsewhere, one could define $p(d_k|\mathrm{x}_k)$ as the following pmf:

$$p(d_k|\mathrm{x}_k) = \begin{cases} 1, & \text{if } d_k = 1 \text{ and } \mathrm{x}_k \in \mathcal{V} \\ 1, & \text{if } d_k = 0 \text{ and } \mathrm{x}_k \notin \mathcal{V} \\ 0, & \text{otherwise} \end{cases} \tag{2.31}$$

It is also possible to define the SDDR model such that it is dependent on the state of the sensor s_k, as done in [22], and/or the state of a given measurement y_k, however such definitions fall outside of the scope of this thesis and thus shall not be considered any further.

A major benefit of using the SDDR model is that it allows for the incorporation of existing maps of the environment that can be used to determine the probability of detection for any given hypothetical target, measurement and/or sensor position. This is particularly useful for applications where prior knowledge exists about regions of the search space, where targets can go undetected and/or be detected with high probability [23, 22].

2.5 Clutter models

Another common intricacy encountered when tracking multiple targets with non-ideal sensors is the presence of clutter. Measurement scans received from the sensor(s) may contain measurements originating from existing targets, i.e. true detections, as well as spurious measurements, a.k.a. clutter, that can originate from a number of different sources: background interference and/or internal interference of the sensors. Thus, a received measurement scan Y_k can viewed as the union of the sets of true and clutter detections, i.e.:

$$Y_k = Y_k^C \cup Y_k^\Delta \tag{2.32}$$

where Y_k^C contains the set of M_k^C clutter generated detections, while Y_k^Δ contains the set of M_k^Δ true detections generated by the targets.

The statistical characteristics of clutter can be represented by appropriately defined clutter models. Such clutter models are typically defined in the form of a pair of spatial distribution and spatial density functions:

- A spatial pdf is used to represent the spatial distribution of clutter measurement positions over a given measurement space:

$$p_C(\mathrm{y}_k) \tag{2.33}$$

  where $p_C(\mathrm{y}_k)$ defines the probability that a given measurement y_k originated from clutter.

- A cardinality pdf is used to model the number of received clutter measurements over the entire surveillance region, such that:

$$p_{M^C}(M) = p(\phi = M) \tag{2.34}$$

  where $p(\phi = M)$ defines the probability that the number of clutter measurements is equal to M.

2.5.1 Poisson Rate with Uniform Position (PRUP)

The model that is most commonly used in literature to model clutter assumes that the number of clutter measurements observed at each scan evolves according to a Poisson distribution with mean clutter rate λ_{FA}, while the measurements are uniformly distributed across the entire search space. When expressed in terms of the generalised model described by (2.33)-(2.34), this yields the following expressions:

- Spatial pdf: The spatial pdf is uniform across the space $\mathcal{V}$, with volume V, i.e.

$$p_C(y_k) = \mathcal{U}(y_k; \mathcal{V}) = \frac{1}{V} \tag{2.35}$$

- Cardinality pdf: The cardinality pdf is modeled as Poisson, with a mean λ_{FA} of total received clutter measurements, i.e.

$$p_{M^C}(M) = Pois(M; \lambda_{FA}) = \frac{\lambda_{FA}^{M} e^{-\lambda_{FA}}}{M!} \tag{2.36}$$

2.6 Conclusion

This chapter has presented the set of dynamic, measurement, detection and clutter models that will be used extensively in the chapters that follow. This has been done with the aim of formulating the fundamental framework of state-space model equations as considered in this thesis, as well as present readers to the general notation that will be utilised in subsequent chapters.

CHAPTER 3

Non-Linear, Non-Gaussian Single-Target Tracking

3.1 Introduction

For any given surveillance system to serve its purpose, target tracking is a fundamental requirement to obtain a clear and complete view of the entities present in the surveillance region. In the context of Single-Target Tracking (STT), the basic assumption holds that only a single target of interest is present within the surveillance area, at any given point in time, and this target is assumed to exist throughout the entire length of the surveillance process.

A series of noisy measurement scans are reported from a deployed sensor (e.g. radar), where each scan can contain measurements that may originate from a number of different sources: the target of interest, background interference (a.k.a. clutter) and/or internal interference of the sensor. In addition to the above, the target may go undetected for random intervals of time, thus adding to the overall complexity of the problem in hand. Based on the above, the problem to be solved by STT systems involves the process of identifying the measurements that are likely to have originated from the target, if any, and then utilising such measurements in order to recursively estimate the state of the target, which typically encapsulates certain quantities of interest, such as target position and velocity.

There exists a large body of literature which has studied, and in turn reviewed, how to develop such target tracking systems [24, 25]. The STT problem in the presence of clutter and missed detections can typically be broken in two parts. First is the process of distinguishing which measurements originated from the target and which were generated by clutter. In the tracking literature this is referred to as Data Association and there has been extensive research conducted over the years on developing methods that solve this problem both efficiently and effectively [26, 27, 28]. Once such valid measurements have been identified, it is still necessary to process the measurements, so as to filter out the noise that is inherently introduced by the sensors. A solution to this second problem can be conveniently provided by Bayesian Filters that can utilise appropriately defined statistical state-space models to perform Bayesian inference on the information provided by the measurements.

This chapter builds on the framework of state-space models presented in Chapter 2 and aims to delve into the specifics of the aforementioned Bayesian Filtering and Data Association problems in the context of Single Target Tracking. Section 3.2 outlines the fundamental concepts of the standard Bayesian Filtering framework, which are then utilised to formulate the Kalman and Particle Filtering algorithms, while discussing the various advantages and pitfalls of each approach. Section 3.3 then proceeds to define the Data Association problem in the context of Single-Target Tracking, while presenting some well founded algorithms that can be used to solve it. Finally, performance evaluations are performed in Section 3.4, to test the performance of the presented algorithms.

3.2 Bayesian Filtering

Bayesian Filtering is only concerned with the simplified problem of estimating the state x_k of a single target, under the fundamental assumption that exactly one measurement y_k is available at each of the discrete time steps t_k. As such, the problem can be conveniently formulated and expressed in terms of Bayesian inference rules [29] applied on a state-space model defined solely on the dynamic and measurement model presented in Sections 2.2 and 2.3, respectively. Thus, making use of the notation introduced in Chapter 2, the goal here is to form an estimate of the state x_k, at every time step t_k, using noisy observations $\mathrm{y}_{1:k}$, acquired up to time t_k.

Therefore, at each time t_k, Bayesian Filtering is applied with the intent of obtaining an expression for the posterior distribution of the state of the target at t_k, conditional on all measurements received up to and including time t_k. In mathematical terms, the quantity to be computed is a probability density function (pdf) of the form

$$p(\mathrm{x}_k|\mathrm{y}_{1:k}) \tag{3.1}$$

For most problems, a state estimate is desirable every time that a measurement is received, in which case a recursive Bayesian Filter becomes a convenient solution. A recursive filtering approach allows for the received measurements to be processed sequentially, rather than as a batch. Such an approach has the advantage of circumventing the necessity of maintaining the complete set of estimates, as well as reprocessing of existing estimates when a new measurement becomes available.

3.2.1 The Standard Bayes Filter

Recursive Bayesian Filtering can be performed under three main assumptions: the system transition function f in (2.1) describes a first-order Markov process [17, 18], leading to a simplification of (2.1) as (2.3); measurements are assumed to be conditionally independent of each other over time, leading to (2.20); some initial information about the state is available, in the form a prior pdf $p(\mathrm{x}_0)$. Given that the aforementioned assumptions are satisfied, the pdf $p(\mathrm{x}_k|\mathrm{y}_{1:k})$ can be obtained, recursively, using two steps: prediction and update.

The prediction step propagates the last estimate of the state (a.k.a. the prior) through the dynamic model equation (2.3) so as to generate a prediction of the state pdf between measurement times. Since the state is usually subjected to unknown disturbances (modelled as random noise), prediction generally translates, deforms, and spreads the state pdf. Next, the update stage makes use of the latest received measurement to update/correct the prediction pdf, through application of the Bayes theorem, which is the mechanism that allows updating one's knowledge about the target state in light of the new measurement.

Algorithm 3.2.1 (Standard Bayesian Filter) *The set of recursive equations necessary to compute the predicted $p(\mathrm{x}_k|\mathrm{x}_{k-1}, \mathrm{y}_{1:k-1})$ and posterior $p(\mathrm{x}_k|\mathrm{y}_{1:k})$ distributions are given by the following Bayesian filtering equations.*

- **Initialisation:** *The filter is initialised with the prior distribution $p(\mathrm{x}_0)$.*
- **Prediction:** *Given the dynamic model of the target's motion, $p(\mathrm{x}_k|\mathrm{x}_{k-1})$, and the previous filtered estimate, $p(\mathrm{x}_{k-1}|\mathrm{y}_{1:k-1})$, the Chapman-Kolmogorov equation [30] can be used to compute the predicted distribution of x_k:*

$$p(\mathrm{x}_k|\mathrm{x}_{k-1}, \mathrm{y}_{1:k-1}) = \int p(\mathrm{x}_k|\mathrm{x}_{k-1})p(\mathrm{x}_{k-1}|\mathrm{y}_{1:k-1})d\mathrm{x}_{k-1} \tag{3.2}$$

- **Update:** *Given the observation model,* $p(\mathrm{y}_k|\mathrm{x}_k)$*, the measurement* y_k *and the above computed predicted distribution, the target posterior distribution* $p(\mathrm{x}_k|\mathrm{y}_{1:k})$ *can be calculated using Bayes' rule:*

$$p(\mathrm{x}_k|\mathrm{y}_{1:k}) = \frac{1}{Z_k} p(\mathrm{y}_k|\mathrm{x}_k) p(\mathrm{x}_k|\mathrm{x}_{k-1}, \mathrm{y}_{1:k-1}) \tag{3.3}$$

where Z_k *is a normalisation constant, given as:*

$$Z_k = \int p(\mathrm{y}_k|\mathrm{x}_k) p(\mathrm{x}_k|\mathrm{x}_{k-1}, \mathrm{y}_{1:k-1}) d\mathrm{x}_k \tag{3.4}$$

The recurrence relations of (3.2) and (3.3) form the basis for the Standard Bayes Filter (SBF), which is the optimal Bayesian solution to the problem of recursively calculating the exact posterior density $p(\mathrm{x}_k|\mathrm{y}_{1:k})$. However, the definition of the SBF is generally theoretical, as any attempt of direct analytical evaluation proves to be intractable[15]. As a result, popular algorithms which have been developed over the years to provide a solution to the Bayesian Filtering problem, base their operation on either constraining the problem to a subset of the state-space model, or on the use of approximation methods, or both. Such techniques will be analysed in the following subsections.

3.2.2 Kalman Filters

3.2.2.1 Linear Kalman Filter

The Kalman Filter (KF) constitutes the optimal closed-form solution to the Bayesian filtering equations, where the posterior density $p(\mathrm{x}_k|\mathrm{y}_{1:k})$ is assumed to be Gaussian. For this assumption to hold throughout the recursions, one further constraint of the standard Kalman Filter is that the dynamic and observation models are required to be linear and Gaussian[1]:

$$\begin{aligned} p(\mathrm{x}_k|\mathrm{x}_{k-1}) &= \mathcal{N}(\mathrm{x}_k; F_{k-1}\mathrm{x}_{k-1}, Q_k), \\ \therefore \mathrm{x}_k &= F_{k-1}\mathrm{x}_{k-1} + \mathrm{q}_k, \\ p(\mathrm{y}_k|\mathrm{x}_k) &= \mathcal{N}(\mathrm{y}_k; H_k\mathrm{x}_k, R_k), \\ \therefore \mathrm{y}_k &= H_k\mathrm{x}_k + \mathrm{r}_k \end{aligned} \tag{3.5}$$

where F_{k-1} is the transition matrix of the linear dynamic model, H_k is the linear measurement model matrix, $\mathrm{q}_k \sim \mathcal{N}(0, Q_k)$ and $\mathrm{r}_k \sim N(0, R_k)$ are the process

[1]For the sake of simplicity, only the cases of additive noise, for the process and measurement models, will be considered here. See [15] for derivations in non-additive cases.

and measurement noises, which are both assumed to be zero-mean Gaussians with covariances of Q_k and R_k, respectively. The closed form solution to the Bayesian filtering equations for the linear models of (3.5) can then be evaluated and the resulting distributions are:

$$\begin{aligned} p(\mathrm{x}_k|\mathrm{x}_{k-1}, \mathrm{y}_{1:k-1}) &= \mathcal{N}(\mathrm{x}_k; \bar{\mathrm{x}}_{k|k-1}, \bar{V}_{k|k-1}) \\ p(\mathrm{y}_k|\mathrm{y}_{1:k-1}) &= \mathcal{N}(\mathrm{y}_k; H_k\bar{\mathrm{x}}_{k|k-1}, S_k) \\ p(\mathrm{x}_k|\mathrm{y}_{1:k}) &= \mathcal{N}(\mathrm{x}_k; \bar{\mathrm{x}}_k, \bar{V}_k) \end{aligned} \tag{3.6}$$

Accordingly, the KF recursion can be described in terms of standard Bayesian Filtering steps, as shown in the Algorithm 3.2.2. The full derivation of the above equations in a Bayesian context can be found in [15].

Algorithm 3.2.2 (Kalman Filter) *The set of recursive equations necessary to compute the predicted $p(\mathrm{x}_k|\mathrm{x}_{k-1}, \mathrm{y}_{1:k-1})$ and posterior $p(\mathrm{x}_k|\mathrm{y}_{1:k})$ distributions are given by the following Kalman Filtering equations.*

- **Initialisation:** *The filter is initialised with the prior $p(\mathrm{x}_0) = \mathcal{N}(\mathrm{x}_0; \bar{\mathrm{x}}_0, \bar{V}_0)$ and the main recursion is initiated.*

- **Prediction:** *The predicted pdf $p(\mathrm{x}_k|\mathrm{x}_{k-1}, \mathrm{y}_{1:k-1}) = \mathcal{N}(\mathrm{x}_k; \bar{\mathrm{x}}_{k|k-1}, \bar{V}_{k|k-1})$ can be calculated as:*

$$\begin{aligned} \bar{\mathrm{x}}_{k|k-1} &= F_{k-1}\bar{\mathrm{x}}_{k-1}, \\ \bar{V}_{k|k-1} &= F_{k-1}\bar{V}_{k-1}F_{k-1}^T + Q_k \end{aligned} \tag{3.7}$$

- **Update:** *The posterior pdf $p(\mathrm{x}_k|\mathrm{y}_{1:k}) = \mathcal{N}(\mathrm{x}_k; \bar{\mathrm{x}}_k, \bar{V}_k)$ can be found:*

$$\begin{aligned} \bar{\mathrm{y}}_k &= H_k\bar{\mathrm{x}}_{k|k-1}, \\ S_k &= H_k\bar{V}_{k|k-1}H_k^T + R_k, \\ K_k &= \bar{V}_{k|k-1}H_k^T S_k^{-1}, \\ \bar{\mathrm{x}}_k &= \bar{\mathrm{x}}_{k|k-1} + K_k(\mathrm{y}_k - \bar{\mathrm{y}}_k), \\ \bar{V}_k &= \bar{V}_{k|k-1} - K_k S_k K_k^T \end{aligned} \tag{3.8}$$

Since their introduction by Kalman [31, 32], KFs have been among the most commonly researched and employed tracking algorithms. Out of the box implementations of the Kalman Filter are based on a small set of linear algebra equations, which makes them relatively simple and incredibly efficient. Applications of the KF in avionic and maritime target tracking range from GPS tracking of vessels [33], to airborne radar tracking [34] and missile guidance systems[35].

In the case of linear, Gaussian dynamic and measurement models, as long as the prior probability density is Gaussian, the Kalman Filter ensures that all subsequent predicted and updated probability densities will also be Gaussian. Furthermore, the means and covariances of these densities can be calculated using basic matrix algebra operations. However, if the problem in hand involves non-linear models, the above assertion does not hold, thus posing a prohibitive limitation to the applicability of the filter in problems involving non-linear behaviour.

3.2.2.2 Extended Kalman Filter

As mentioned above, the linearity constraints on the dynamics and measurement models of the standard Kalman Filter prevent direct application of the algorithm in problems involving non-linear behaviour. One first solution to the problem is provided by the use of local-point linearisation techniques [33], through application of the popular Taylor Series first-order approximation [36]. The term Extended Kalman Filter (EKF)[24, 37] was later adopted for reference to the specific incarnation of the Kalman Filter algorithm[2].

Given that the dynamic and measurement models are non-linear, and based on the assumption that the process and measurement noises are additive, the new model (equivalent to (3.5)) for the EKF can be written as:

$$\begin{aligned} \mathrm{x}_k &= f_{k-1}(\mathrm{x}_{k-1}) + \mathrm{q}_k, \\ \mathrm{y}_k &= h_k(\mathrm{x}_k) + \mathrm{r}_k \end{aligned} \tag{3.9}$$

Again, $\mathrm{q}_k \sim \mathcal{N}(0, Q_k)$ and $\mathrm{r}_k \sim \mathcal{N}(0, R_k)$ are the zero-mean Gaussian process and observation noises, with covariances of Q_k and R_k, respectively. However, the non-linear state-space models can no longer be expressed in terms of matrices. As such, the respective non-linear transfer functions, $f(.)$ and $h(.)$, are used instead. Next, assuming that the posterior density $p(\mathrm{x}_k|\mathrm{y}_{1:k})$ can be approximated by a Gaussian, the solutions to the Bayesian Filtering equations can also be approximated in the same fashion as in (3.6) and the resulting algorithm is shown below.

Algorithm 3.2.3 (Extended Kalman Filter) *The set of recursive equations necessary to compute/approximate the predicted $p(\mathrm{x}_k|\mathrm{x}_{k-1}, \mathrm{y}_{1:k-1})$ and posterior $p(\mathrm{x}_k|\mathrm{y}_{1:k})$ distributions are given by the following Bayesian filtering equations.*

[2]It should be noted here that a second-order approximation EKF exists (namely EKF2 [38]), but the additional complexity has prohibited its widespread use and, thus, has not been considered here.

- **Initialisation:** *The filter is initialised with the prior pdf* $p(\mathrm{x}_0) = \mathcal{N}(\mathrm{x}_0; \bar{\mathrm{x}}_0, \bar{V}_0)$ *and the main recursion is initiated.*

- **Prediction:** *The predicted pdf* $p(\mathrm{x}_k|\mathrm{x}_{k-1}, \mathrm{y}_{1:k-1}) = \mathcal{N}(\mathrm{x}_k; \bar{\mathrm{x}}_{k|k-1}, \bar{V}_{k|k-1})$ *can be calculated as:*

$$\begin{aligned} \bar{\mathrm{x}}_{k|k-1} &= f_{k-1}(\bar{\mathrm{x}}_{k-1}), \\ \bar{V}_{k|k-1} &= \hat{F}_{k-1}\bar{V}_{k-1}\hat{F}_{k-1}^T + Q_k \end{aligned} \tag{3.10}$$

- **Update:** *The posterior pdf* $p(\mathrm{x}_k|\mathrm{y}_{1:k}) = \mathcal{N}(\mathrm{x}_k; \bar{\mathrm{x}}_k, \bar{V}_k)$ *can be found:*

$$\begin{aligned} \bar{\mathrm{y}}_k &= h_k(\bar{\mathrm{x}}_{k|k-1}), \\ S_k &= \hat{H}_k\bar{V}_{k|k-1}\hat{H}_k^T + R_k, \\ K_k &= \bar{V}_{k|k-1}\hat{H}_k^T S_k^{-1}, \\ \bar{\mathrm{x}}_k &= \bar{\mathrm{x}}_{k|k-1} + K_k(\mathrm{y}_k - \bar{\mathrm{y}}_k), \\ \bar{V}_k &= \bar{V}_{k|k-1} - K_k S_k K_k^T \end{aligned} \tag{3.11}$$

where $\hat{F}_{k-1}$ *and* $\hat{H}_k$ *are Jacobian matrices, resulting from the local linearisations of the non-linear model functions,* $f_{k-1}(.)$ *and* $h_k(.)$ *respectively, and can be computed using Tayor series approximation:*

$$\begin{aligned} \hat{F}_{k-1} &= \left.\frac{df_{k-1}(x)}{dx}\right|_{x=\bar{\mathrm{x}}_{k-1}} \\ \hat{H}_k &= \left.\frac{dh_k(x)}{dx}\right|_{x=\bar{\mathrm{x}}_{k|k-1}} \end{aligned} \tag{3.12}$$

Aside from the computation of the Jacobian matrices and the direct propagation of the prior and predicted means through the respective non-linear models, the EKF implementation is almost identical to the standard KF approach. A major advantage of EKF over other non-linear filters is its relative simplicity compared to the achieved performance. For this reason, EKF has provided solutions to an number of inherently complex and non-linear problems, including robot localisation and mapping [39], UAV navigation and landing control systems [40], as well as detection and tracking of moving targets from a mobile UAV platform [41].

However, this offered convenience does come at a cost, as there are two major drawbacks which need to be considered [42]. Firstly, the computation of the desired Jacobian matrices is often a non-trivial task, which can lead to significant implementation difficulties in cases where the dynamic and measurement model functions are not (easily) differentiable. The second, and possibly most significant, drawback of the EKF can manifest itself when the algorithm is applied to problems that involve highly non-linear models. In such circumstances, the assumptions of local linearity can be extensively violated, leading to substantial performance degradation and filter instabilities.

3.2.2.3 Unscented Kalman Filter

The Unscented Kalman Filter (UKF) [43, 44] was developed with the intent of addressing the inherent errors and inaccuracies introduced by EKF, due to the use of first-order analytic linearisation. Instead of performing analytical linearisation around a single point, the UKF makes use of statistical linear regression to linearise the non-linear dynamic and measurement functions, through an appropriately selected set of regression points [45].

To achieve the aforementioned, the UKF utilises a method termed as the Unscented Transform (UT) [42]. The idea behind the UT is to deterministically select a fixed number of points, referred to as sigma points, from a distribution of interest, that can be appropriately weighted and used to provide an exact representation of its mean and covariance. The chosen set of points is then passed through the desired non-linear function and the resulting set is used to approximate the mean and covariance of the transformed variable. Using the above mentioned technique, the UKF is able to accurately capture the posterior mean and covariance of the transformed distributions, up to the 4rd order of the Taylor series expansion (compared to 1st order for the EKF) [42]. More on the UT can be found in [42, 43, 46].

The dynamic and measurement models for the UKF can be defined in the same manner as in (3.9), where the assumption has been made that the process and measurement noises are additive. Again, the solutions to Bayesian filtering equations can be expressed just as in (3.6), where the equalities ("=") are replaced by approximations ("≈"). Considering all the above, the description of UKF in terms of the 3 standard Bayesian filtering steps is provided in the Algorithm 3.2.4.

Algorithm 3.2.4 (Unscented Kalman Filter) *The set of recursive equations necessary to compute/approximate the predicted $p(\mathrm{x}_k|\mathrm{x}_{k-1}, y_{1:k-1})$ and posterior $p(\mathrm{x}_k|\mathrm{y}_{1:k})$ distributions are given by the following Bayesian filtering equations.*

- **Initiation:** *The filter is initialised with the prior pdf $p(\mathrm{x}_0) = \mathcal{N}(\mathrm{x}_0; \bar{\mathrm{x}}_0, \bar{V}_0)$ and the main recursion is initiated.*
- **Prediction:** *The predicted pdf $p(\mathrm{x}_k|\mathrm{x}_{k-1}, \mathrm{y}_{1:k-1}) = \mathcal{N}(\mathrm{x}_k; \bar{\mathrm{x}}_{k|k-1}, \bar{V}_{k|k-1})$ can be calculated in 3 sub-steps derived from the UT:*

1. *A set of 2n+1 sigma points ($\mathcal{X}_{k-1}^{(0:2n)}$), including the respective set of mean and covariance weights ($\mathcal{W}_m^{(0:2n)}$ and $\mathcal{W}_c^{(0:2n)}$), is formed as follows:*

$$\begin{aligned}\mathcal{X}_{k-1}^{(0)} &= \bar{\mathrm{x}}_{k-1},\\ \mathcal{X}_{k-1}^{(i)} &= \bar{\mathrm{x}}_{k-1} + \sqrt{n+\lambda}\Big[\sqrt{\bar{V}_{k-1}}\Big]_i,\\ \mathcal{X}_{k-1}^{(i+n)} &= \bar{\mathrm{x}}_{k-1} - \sqrt{n+\lambda}\Big[\sqrt{\bar{V}_{k-1}}\Big]_i, \quad i=0,1,....,2n\end{aligned} \tag{3.13}$$

$$\begin{aligned}\mathcal{W}_m^{(0)} &= \frac{\lambda}{n+\lambda},\\ \mathcal{W}_c^{(0)} &= \mathcal{W}_m^{(0)} + (1-\alpha^2+\beta),\\ \mathcal{W}_m^{(i)} &= \mathcal{W}_c^{(i)} = \frac{1}{2(n+\lambda)}, \quad i=1,2,....,2n\end{aligned} \tag{3.14}$$

2. *The propagated set of sigma points $\hat{\mathcal{X}}_k^{(0:2n)}$ is computed using the dynamic model function $f_{k-1}(.)$:*

$$\hat{\mathcal{X}}_{k-1}^{(i)} = f_{k-1}(\mathcal{X}_{k-1}^{(i)}), \quad i=0,1,....,2n \tag{3.15}$$

3. *The predicted mean $\bar{\mathrm{x}}_{k|k-1}$ and covariance $\bar{V}_{k|k-1}$ are finally computed as:*

$$\begin{aligned}\bar{\mathrm{x}}_{k|k-1} &= \sum_{i=0}^{2n} \mathcal{W}_m^{(i)}\hat{\mathcal{X}}_k^{(i)},\\ \bar{V}_{k|k-1} &= \sum_{i=0}^{2n} \mathcal{W}_c^{(i)}(\hat{\mathcal{X}}_k^{(i)} - \bar{\mathrm{x}}_{k|k-1})(\hat{\mathcal{X}}_k^{(i)} - \bar{\mathrm{x}}_{k|k-1})^T + Q_k\end{aligned} \tag{3.16}$$

- **Update:** *Finally, a slight alteration of the previous 3 UT sub-steps is used to find the posterior pdf $p(\mathrm{x}_k|\mathrm{y}_{1:k}) = \mathcal{N}(\mathrm{x}_k; \bar{\mathrm{x}}_k, \bar{V}_k)$ as follows:*

  1. *A set of 2n+1 sigma points ($\tilde{\mathcal{X}}_k^{(0:2n)}$), including the respective set of mean and covariance weights ($\tilde{\mathcal{W}}_m^{(0:2n)}$ and $\tilde{\mathcal{W}}_c^{(0:2n)}$), is formed as fol-*

lows[3]*:*

$$\begin{aligned}\tilde{\mathcal{X}}_k^{(0)} &= \bar{\mathrm{x}}_{k|k-1},\\ \tilde{\mathcal{X}}_k^{(i)} &= \bar{\mathrm{x}}_{k|k-1} + \sqrt{n+\lambda}\left[\sqrt{\bar{V}_{k|k-1}}\right]_i,\\ \tilde{\mathcal{X}}_k^{(i+n)} &= \bar{\mathrm{x}}_{k|k-1} - \sqrt{n+\lambda}\left[\sqrt{\bar{V}_{k|k-1}}\right]_i, \quad i = 0,1,....,2n\end{aligned} \tag{3.17}$$

$$\begin{aligned}\tilde{\mathcal{W}}_m^{(0)} &= \frac{\lambda}{n+\lambda},\\ \tilde{\mathcal{W}}_c^{(0)} &= \tilde{\mathcal{W}}_m^{(0)} + (1-\alpha^2+\beta),\\ \tilde{\mathcal{W}}_m^{(i)} &= \tilde{\mathcal{W}}_c^{(i)} = \frac{1}{2(n+\lambda)}, \quad i = 1,2,....,2n\end{aligned} \tag{3.18}$$

2. *The propagated set of sigma points $\hat{\mathcal{Y}}_k^{(0:2n)}$ is computed using the observation model function $h_k(.)$:*

$$\hat{\mathcal{Y}}_k^{(i)} = h_k(\tilde{\mathcal{X}}_k^{(i)}), \quad i = 0,1,....,2n \tag{3.19}$$

3. *The updated mean m_k and covariance P_k are finally computed as:*

$$\begin{aligned}\bar{\mathrm{y}}_k &= \sum_{i=0}^{2n} \mathcal{W}_m^{(i)} \hat{\mathcal{Y}}_k^{(i)},\\ S_k &= \sum_{i=0}^{2n} \mathcal{W}_c^{(i)} (\hat{\mathcal{Y}}_k^{(i)} - \bar{\mathrm{y}}_k)(\hat{\mathcal{Y}}_k^{(i)} - \bar{\mathrm{y}}_k)^T + R_k,\\ C_k &= \sum_{i=0}^{2n} \mathcal{W}_c^{(i)} (\tilde{\mathcal{X}}_k^{(i)} - \bar{\mathrm{x}}_{k|k-1})(\hat{\mathcal{Y}}_k^{(i)} - \bar{\mathrm{y}}_k)^T,\\ K_k &= C_k S_k^{-1},\\ \bar{\mathrm{x}}_k &= \bar{\mathrm{x}}_{k|k-1} + K_k(\mathrm{y}_k - \bar{\mathrm{y}}_k),\\ \bar{V}_k &= \bar{V}_{k|k-1} - K_k S_k K_k^T\end{aligned} \tag{3.20}$$

where n is the dimensionality of the estimated state x*, $\lambda = \alpha^2(n+\kappa) - n$ is a scaling factor, α and κ are used in order to control the spread of the sigma points around the mean and β is used to incorporate previous knowledge about the distribution of* x*. Typical values for the above parameters are: $\alpha = 10^{-3}$, $\kappa = 0$ and $\beta = 2$(Optimal for Gaussian distributions).*

[3]Some UKF implementations bypass this step and simply set $\tilde{\mathcal{X}}_k = \hat{\mathcal{X}}_{k-1}$ [47]

3.2.3 Particle Filter

All incarnations of the Kalman Filter suffer from the same fundamental limitation stemming from the assumption that the state of interest is normally distributed. If this assumption is violated and some multi-modality, skewness or other departure from being Gaussian, is introduced to the pdf, then the distribution of interest cannot be effectively captured by it's first two moments (mean and covariance) and any attempt to estimate it using any of the aforementioned filters will lead to extensive errors and possibly tracker failure.

The Particle Filter (PF) [48, 6] makes use of Monte-Carlo approximation methods to form an approximation to the posterior distribution $p(\mathrm{x}_k|\mathrm{y}_{1:k})$, by utilising the diversity of an appropriately weighted set of N_p samples $\{\mathrm{x}^i, \mathrm{w}^i\}_{i=1}^{N_p}$, a.k.a particles, where x_k^i and w_k^i are respectively the state and weight of the i^{th} particle. Given that the weights of the particles are defined such that they sum to unity, an estimation of the posterior can be conveniently evaluated as a weighted sum of all the particles, i.e.:

$$p(\mathrm{x}_k|\mathrm{y}_{1:k}) \approx \sum_{i=1}^{N_p} \mathrm{w}_k^i \delta(\mathrm{x}_k - \mathrm{x}_k^i) \tag{3.21}$$

where $\delta(.)$ represents the Delta-Dirac function and the weights are normalised such that $\sum_i \mathrm{w}_k^i = 1$.

The approximation of (3.21) means that it is generally not possible to sample from this distribution. To circumvent this issue, Particle Filters make use of importance sampling [6, 49], whereby samples are drawn from a proposal density[4], instead of the target density, and are subsequently weighted. Let $p(x)$ be the target density from which it is hard to draw samples but can be evaluated, and let $\pi(x)$ be the proposal density, which is both easy to sample from and to evaluate. Thus, given a set of samples $x^i \sim \pi(x)$, their importance weights can be calculated as

$$\mathrm{w}^i \propto \frac{p(x^i)}{\pi(x^i)} \tag{3.22}$$

To derive a recursive expression for the weight update equation, let us start by considering the full posterior distribution of states $\mathrm{x}_{0:k}$ given the measurements $\mathrm{y}_{1:k}$. By using the Markov properties of the model, we get the following recursion

[4]With slight abuse of notation, the term *density* will be interchangeably used throughout the thesis to refer to either a distribution or a density, as also done in [6].

for the posterior distribution:

$$\begin{aligned} p(\mathrm{x}_{0:k}|\mathrm{y}_{1:k}) &\propto p(\mathrm{y}_k|\mathrm{x}_{0:k}, \mathrm{y}_{1:k-1})p(\mathrm{x}_{0:k}|\mathrm{y}_{1:k-1}) \\ &= p(\mathrm{y}_k|\mathrm{x}_k)p(\mathrm{x}_k|\mathrm{x}_{0:k-1}, \mathrm{y}_{0:k-1})p(\mathrm{x}_{0:k-1}|\mathrm{y}_{1:k-1}) \\ &= p(\mathrm{y}_k|\mathrm{x}_k)p(\mathrm{x}_k|\mathrm{x}_{k-1})p(\mathrm{x}_{0:k-1}|\mathrm{y}_{1:k-1}) \end{aligned} \tag{3.23}$$

If the samples x_k^i are drawn from an importance density $\pi(\mathrm{x}_{0:k}|\mathrm{y}_{1:k})$, given (3.23), the importance weights can be calculated as follows:

$$\mathrm{w}_k^i \propto \frac{p(\mathrm{x}_{0:k}^i|\mathrm{y}_{1:k})}{\pi(\mathrm{x}_{0:k}^i|\mathrm{y}_{1:k})} = \frac{p(\mathrm{y}_k|\mathrm{x}_k^i)p(\mathrm{x}_k^i|\mathrm{x}_{k-1}^i)p(\mathrm{x}_{0:k-1}^i|\mathrm{y}_{1:k-1})}{\pi(\mathrm{x}_{0:k}^i|\mathrm{y}_{1:k})} \tag{3.24}$$

Continuing, if the importance density is also assumed to be Markovian then we can re-write $\pi(\mathrm{x}_{0:k}|\mathrm{y}_{1:k})$ as follows:

$$\pi(\mathrm{x}_{0:k}|\mathrm{y}_{1:k}) = \pi(\mathrm{x}_k|\mathrm{x}_{k-1}, \mathrm{y}_{1:k})\pi(\mathrm{x}_{0:k-1}|\mathrm{y}_{1:k-1}) \tag{3.25}$$

which leads to the following expansion of (3.24):

$$\mathrm{w}_k^i \propto \frac{p(\mathrm{y}_k|\mathrm{x}_k^i)p(\mathrm{x}_k^i|\mathrm{x}_{k-1}^i)}{\pi(\mathrm{x}_k^i|\mathrm{x}_{k-1}^i, \mathrm{y}_{1:k})} \frac{p(\mathrm{x}_{0:k-1}^i|\mathrm{y}_{1:k-1})}{\pi(\mathrm{x}_{0:k-1}^i|\mathrm{y}_{1:k-1})} \tag{3.26}$$

Let us now assume that a set of samples $\mathrm{x}_{0:k-1}^i$ have already been drawn from the importance density $\pi(\mathrm{x}_{0:k-1}|\mathrm{y}_{1:k-1})$, with corresponding weights

$$\mathrm{w}_{k-1}^i \propto \frac{p(\mathrm{x}_{0:k-1}^i|\mathrm{y}_{1:k-1})}{\pi(\mathrm{x}_{0:k-1}^i|\mathrm{y}_{1:k-1})} \tag{3.27}$$

Finally, by considering the recursive formulation of the importance density shown in (3.25), it is possible to obtain samples $\mathrm{x}_{0:k}^i$ from $\pi(\mathrm{x}_{0:k}|\mathrm{y}_{1:k})$ by drawing samples as $\mathrm{x}_k^i \sim \pi(\mathrm{x}_k|\mathrm{x}_{k-1}^i, \mathrm{y}_{1:k})$. Therefore, substitution of (3.27) into (3.26), leads to the final recursive expression for the weight update step:

$$\mathrm{w}_k^i \propto \frac{p(\mathrm{y}_k|\mathrm{x}_k^i)p(\mathrm{x}_k^i|\mathrm{x}_{k-1}^i)}{\pi(\mathrm{x}_k^i|\mathrm{x}_{k-1}^i, \mathrm{y}_{1:k})}\mathrm{w}_{k-1}^i \tag{3.28}$$

By employing the above techniques, the Sequential Importance Sampling (SIS) Particle Filter [6] can fully exploit non-linear/non-Gaussian models and distributions. This enables the algorithm to form a direct approximation to the Bayesian Filtering equations of (3.2) and (3.3). However, as shown in [49], a common problem with the SIS variant is that the variance of the importance weights increases with time, leading to most of the particles having negligible weights.

This phenomenon is known in the literature as particle degeneracy [49, 6] and implies that large computational efforts are devoted to updating particles whose contribution to the approximation of (3.21) becomes increasingly small.

A solution to the degeneracy problem is provided by adding a resampling step to the SIS recursion, leading to the Sequential Importance Resampling (SIR) or, as commonly known, the Particle Filter recursion. During the resampling process, particles with high importance weights are duplicated and used to suppress particles with low weights. There exists a number of different resampling algorithms, however experience has shown that performance is not significantly affected by this choice [6]. For more information on resampling strategies, including performance evaluations and comparisons, readers are advised to refer to [50, 51]. As resampling is known to be the bottleneck of the SIR recursion, the process can either be executed on each iteration, or may be set to execute only when the number of effective samples falls below a given threshold, as shown in (3.33).

Algorithm 3.2.5 (Particle Filter) *The set of recursive equations necessary to compute/approximate the predicted* $p(\mathrm{x}_k|\mathrm{x}_{k-1}, \mathrm{y}_{1:\mathrm{k}-1})$ *and posterior* $p(\mathrm{x}_k|\mathrm{y}_{1:k})$ *distributions are given by the following Bayesian filtering equations.*

- **Initiation:** *An initial set of* N_p *samples is drawn from the prior* $p(\mathrm{x}_0)$*, with uniform weights as:*

$$\begin{aligned} \mathrm{x}_0^i &\sim p(\mathrm{x}_0), \\ \mathrm{w}_0^i &= \frac{1}{N}, \qquad i = 1, 2, ..., N_p \end{aligned} \tag{3.29}$$

- **Prediction:** *A new set of N samples is drawn from the importance density:*

$$\mathrm{x}_k^i \sim \pi(\mathrm{x}_k|\mathrm{x}_{k-1}^i, \mathrm{y}_{1:k}), \qquad i = 1, 2, ..., N_p \tag{3.30}$$

- **Update:** *The update step can be broken down to the following 4 sub-steps:*

  1. *The non-normalised weight for each particle can be calculated using* (3.28)*:*

$$\tilde{\mathrm{w}}_k^i = \mathrm{w}_{k-1}^i \frac{p(\mathrm{y}_k|\mathrm{x}_k^i)p(\mathrm{x}_k^i|\mathrm{x}_{k-1}^i)}{\pi(\mathrm{x}_k^i|\mathrm{x}_{k-1}^i, \mathrm{y}_{1:k})}, \qquad i = 1, 2, ..., N_p \tag{3.31}$$

2. *All the weights are normalised such that they sum to unity:*

$$\mathrm{w}_k^i = \frac{\tilde{\mathrm{w}}_k^i}{\sum_{i=1}^{N_p} \tilde{\mathrm{w}}_k^i}, \qquad i = 1, 2, ..., N_p \tag{3.32}$$

3. *The number of effective samples(N_{eff}) is calculated as:*

$$N_{eff} = \frac{1}{\sum_{i=1}^{N_p} \left(\mathrm{w}_k^i\right)^2} \tag{3.33}$$

4. *If the number of effective samples is lower than the set threshold (i.e. $N_{eff} < N_{eff_{min}}$), then resampling can be performed.*

A generic PF recursion, involves the same two general steps (i.e. prediction and update), during which a new posterior particle set $\{\mathrm{x}_k^i, \mathrm{w}_k^i\}_{i=1}^{N_p}$ for time k, is typically constructed from the set $\{\mathrm{x}_{k-1}^i, \mathrm{w}_{k-1}^i\}_{i=1}^{N_p}$ obtained from the previous timestep $k-1$, using the defined dynamic and measurement models, as well as the current observation y_k. Upon resampling, the weights for all particles are uniformly distributed and the new particle set $\{\mathrm{x}_k^i, \mathrm{w}_k^i\}_{i=1}^{N_p}$ is formed. A formal representation of the above is shown in Algorithm 3.2.5.

The choice of appropriate proposal density $\pi(\mathrm{x}_k|\mathrm{x}_{k-1}^i, \mathrm{y}_{1:k})$ in (3.30)-(3.31) is a crucial step in the successful design of a Particle Filter, which also contributes in reducing the effect of particle degeneracy [6]. Ideally, a proposal density should be selected that minimises the variance of the importance weights [6, 7]. The optimal proposal has been shown in [49] to be $p(\mathrm{x}_k|\mathrm{x}_{k-1}^i, \mathrm{y}_{1:k})$, however it requires the ability to sample from $p(\mathrm{x}_k|\mathrm{x}_{k-1}^i, \mathrm{y}_{1:k})$ and to evaluate the integral over the new state, both of which are generally not straightforward. Examples of cases where this is possible can be found in [6]. A widespread choice over the years has been the prior density $p(\mathrm{x}_k|\mathrm{x}_{k-1}^i)$, giving rise to an incarnation of the algorithm known as Bootstrap Particle Filter [48, 52, 53, 54], which leads to the following convenient simplification of (3.31):

$$\mathrm{w}_k^i = \mathrm{w}_{k-1}^i p(\mathrm{y}_k|\mathrm{x}_k^i) \tag{3.34}$$

Particle Filters provide significant advantages over the Kalman filtering framework, by discarding the assumptions of Gaussianity of the state and local linearity of the models. As such, they can be applied to accurately estimate processes with complex state distributions (e.g. multimodal Gaussians) and non-linear/non-Gaussian models (e.g. K-distribution and variants [55]), which would otherwise be highly challenging, if not impossible, to track. The lack of linearisation of the dynamic and/or measurement models, combined with the Monte-Carlo nature of the algorithm, implies that PFs are (optimally) able to fully exploit all the orders of non-linearity involved in such models, which provides an even greater improvement in terms of estimation performance, a fact which becomes even more evident, as the number of particles is increased.

Inevitably, the PF framework has it's own disadvantages [15]. The higher the number of particles used to describe the posterior density, the higher the computational overhead of the algorithm becomes. This fact poses limitations to the applicability of the algorithm in cases where speed and efficiency are paramount. A further disadvantage of PFs is a phenomenon known as particle impoverishment. This problem is introduced by the resampling step of the algorithm and is known to manifests itself when the process noise in the dynamic model is relatively low. As such, pure recursive estimation, such as the estimation of static parameters, using the PF could lead to very poor results. A solution to this issue is provided by the use of techniques such as regularisation and Rao-Blackwallization [49], however such approaches tend to add to the overall complexity of the algorithm.

3.3 Data Association

In the presence of missing reports and/or false detections, the standard Bayesian Filtering equations cannot be directly applied to obtain an estimate of the target state. This is because Bayesian Filters base their operation on the assumption that, at any given measurement time, only one association hypothesis is generated for the target, which relates to the association event between the target and exactly one received measurement. This poses an issue when a target goes undetected (i.e. no measurement is available), as well as when more than one measurements are received. Thus, data association techniques become necessary in order to deal with the added complexity.

3.3.1 Optimal Solution

At each measurement time t_k, a measurement scan $Y_k = \mathrm{y}_k^1, \ldots, \mathrm{y}_k^{M_k}$ is received. Each measurement scan consists of a measurement originating from the target, if the target is detected, as well as spurious measurements (clutter), that can originate from background interference and/or internal interference of the sensors.

Provided that a set of M_k measurements is received, there exist $M_k + 1$ possible association hypotheses[5] for the target; a single hypothesis for each received measurement, plus an additional hypothesis for the target going undetected. This is based on the assumption that, if detected, a target can generate at most one true measurement in any given scan, and, conversely, each measurement may have originated from at most one target.

Based on the above definition, let α_k be a random variable that denotes the set of all possible association hypotheses for the target at time t_k, that takes values

$$\alpha_k = \begin{cases} 0 & \text{no measurement originated from the target} \\ j & \text{if measurement } \mathrm{y}_k^j \text{ originated from the target} \end{cases} \tag{3.35}$$

Based on the above, a series of exhaustive and mutually exclusive association hypotheses θ_k^j can be defined, which consider the event that the target is associated with a given measurement, i.e.

$$\theta_k^j \triangleq [\alpha_k = j] \tag{3.36}$$

where the Iverson notation is used; if P is a statement, [P] equals 1 if the statement is true and 0 otherwise.

Therefore, the posterior state distribution $p(\mathrm{x}_k|Y_{1:k})$, conditional on all measurement scans received up to time t_k, can be computed by marginalising over the entire set of association hypotheses, i.e:

$$p(\mathrm{x}_k|Y_{1:k}) = \sum_{\theta_k^j} p(\mathrm{x}_k|\theta_k^j, \mathrm{y}_k^j, Y_{1:k-1})p(\theta_k^j|Y_{1:k}) \tag{3.37}$$

where $p(\mathrm{x}_k|\theta_k^j, \mathrm{y}_k^j, Y_{1:k-1})$ is equivalent to (3.1), albeit presented using different notation to emphasise that the posterior distribution is conditional on the marginal association hypothesis θ_k^j.

Based on the above definitions, the computation of the association probabilities $p(\theta_k^j|Y_{1:k})$ can be done as shown in [28], albeit using a slightly different

[5]It is assumed that no gating is performed to reduce the number of possible associations. Gating will be discussed further in Chapter 4

notation:

$$p(\theta_k^j|Y_{1:k}) \propto \begin{cases} (1-p_d(\mathrm{x}_k))p_{M^c}(M_k) \displaystyle\prod_{i\in[1,\dots,M_k]} p_C(\mathrm{y}_k^i), & j=0 \\ \dfrac{p_d(\mathrm{x}_k)p(\mathrm{y}_k^j|\mathrm{x}_k)p_{M^c}(M_k-1)}{M_k} \displaystyle\prod_{i\in[1,\dots,M_k]\backslash j} p_C(\mathrm{y}_k^i), & j=1,\dots,M_k \end{cases} \tag{3.38}$$

Finally, using the notation of equation (3.37), it should be clear to see that the standard Bayesian Filtering equations (3.2)-(3.4) can now be employed to evaluate the conditional distributions $p(\mathrm{x}_k|\theta_k^j, \mathrm{y}_k^j, Y_{1:k-1})$ independently for each association event θ_k^j, thus allowing for any of the presented Kalman or Particle Filter algorithms to be applied, subject to linearity and/or Gaussianity restrictions.

3.3.2 Nearest Neighbour

It is often desirable, for computational purposes, to only consider the most likely hypothesis, while ignoring the remaining hypotheses. The computational benefit is thus achieved by circumventing the necessity of calculating the posterior distributions relating to the less likely hypotheses. This can be achieved using the so called Nearest Neighbour (NN) data association algorithm.

In general, the NN algorithm can be formulated in 2 ways: i) probabilistic; and ii) distance-based. In both formulations, the first step of the algorithm involves identifying the value of $\hat{\theta}_k^j$ that represents the most likely association hypothesis. Once this is done, the new posterior is calculated by simplifying equation (3.37) as follows:

$$p(\mathrm{x}_k|Y_{1:k}) = p(\mathrm{x}_k|\hat{\theta}_k^j, \mathrm{y}_k^j, Y_{1:k-1}) \tag{3.39}$$

The probabilistic formulation of NN is obtained by computing $p(\theta_k^j|Y_{1:k})$ for all $\alpha_k = 0, \dots, M_k$ and then selecting the most likely hypothesis on the basis of maximising $p(\theta_k^j|Y_{1:k})$

$$\hat{\theta}_k^j = arg \max_{0\leq j\leq M_k} \Big[p(\theta_k^j|Y_{1:k})\Big] \tag{3.40}$$

The distance-based formulation of NN [56] is realised by defining a distance measure $\mathcal{D}(\mathrm{x}_k, \mathrm{y}_k^j)$ between the target state and a given measurement to be used as a logarithmic approximation of the association probabilities $p(\theta_k^j|Y_{1:k})$. It is also necessary to define the threshold value $\mathcal{D}(\mathrm{x}_k, \emptyset)$ (where we have used the notation $\mathrm{y}_k^0 = \emptyset$) that defines the maximum distance for the missed detection hypothesis. Commonly used distance measures include the Euclidian or Mahalanobis distances, but other measures can also be employed. Thus, the most likely association hypothesis is selected on the basis of minimising the defined distance measure, i.e.:

$$\hat{\theta}_k^j = arg \min_{0 \leq j \leq M_k} \left[\mathcal{D}(\mathrm{x}_k, \mathrm{y}_k^j) \right] \tag{3.41}$$

In general, the distance-based NN algorithm can be significantly faster than probabilistic NN, as it replaces the (generally) costly evaluation of all the pdfs in equation (3.38), with a typically cheap, in terms of computation, distance calculation. However, as is often the case with approximation techniques, the distance-based NN algorithm can only be guaranteed to be exact, in terms of the calculated association probabilities, under certain assumptions. For example, distance-based NN using a Mahalanobis distance metric is only exact in cases where the state and measurement predictions are unimodal Gaussians.

Nevertheless, both formulations of the NN algorithm suffer from the same fundamental problem: making a hard decision about only the most-likely hypothesis does not guarantee that it is also the correct one. This makes the NN algorithm a somewhat naive approach to the data association problem. In cases of heavily cluttered and/or populated environments, NN has proven to be inadequate in a number of different occasions, causing tracker convergence and stability issues [27, 57].

3.3.3 Probabilistic Data Association

In contrast to the hard-decision mechanism employed by NN methods, the Probabilistic Data Association (PDA) [58] algorithm adopts a soft decision approach. In PDA, all association hypotheses contribute to the calculation of the target posterior, depending on the extent to which they satisfy the prediction. This is achieved by computing an association probability $p(\theta_k^j | Y_{1:k})$ for each measurement hypothesis, which is then used as a weighting factor in calculating the target posterior pdf as a mixture over the pdfs produced by considering each of the association hypotheses independently:

$$p(\mathrm{x}_k | Y_{1:k}) = \sum_{\theta_k^j} p(\mathrm{x}_k | \theta_k^j, \mathrm{y}_k^j, Y_{1:k-1}) p(\theta_k^j | Y_{1:k}) \tag{3.42}$$

Computing the association probabilities $p(\theta_k^j | Y_{1:k})$ is done in the same manner as the probabilistic NN algorithm, but instead of selecting a single hypothesis to be used in generating the posterior pdf, all the pdfs are utilised. Thus, the probabilistic NN algorithm can be viewed as a reduced case of the standard PDA method. The operation described by equation (3.42) can be performed by means of mixture reduction algorithms [59, 60] to reduce a mixture of association hypothesis pdfs, weighted by their respective association probabilities, down to a unimodal distribution.

3.3.3.1 PDA in Kalman Filters

The Probabilistic Data Association Filter was seminally derived within a Kalman Filtering (KF) context (KFPDA), and provided a soft decision mechanism for updating the target state posterior distribution $p(\mathrm{x}_k|Y_{1:k})$ under the standard assumptions of linear Gaussian models. As such, each of the conditional posterior distributions of (3.38) and (3.42) are Gaussian densities of the form

$$p(\mathrm{x}_k|\theta_k^j, \mathrm{y}_k^j, Y_{1:k-1}) = \mathcal{N}(\mathrm{x}_k; m_k^j, P_k^j) \tag{3.43}$$

and the posterior density of (3.38) becomes a Gaussian Mixture, with mixture weights $\beta_k^j = p(\theta_k^j|Y_{1:k})$, i.e.

$$p(\mathrm{x}_k|Y_{1:k}) = \sum_{j=1}^{M_k} \beta_k^j \mathcal{N}(\mathrm{x}_k; m_k^j, P_k^j) \tag{3.44}$$

The parametric version of PDA assumes Poison distributed clutter with spatial density λ. This assumption is equivalent to the Poisson model of Section 2.5.1. What is more, a typical assumption, made mostly for notational simplicity, is that the target detection probability is constant and equal to P_D for all target, similar to the CDR model of Section 2.4.1. As such, a quick derivation of the base association probability calculation equations (37)-(38) of [58] can be obtained by plugging (2.29), (2.35), (2.36) into (3.38), as follows:

$$\begin{aligned}
\beta_k^j &\propto \begin{cases} (1-p_d(\mathrm{x}_k))p_{M^c}(M_k) \displaystyle\prod_{i\in[1,\dots,M_k]} p_C(\mathrm{y}_k^i), & j=0 \\ \dfrac{1}{M_k}p_d(\mathrm{x}_k)p(\mathrm{y}_k^j|\mathrm{x}_k)p_{M^c}(M_k-1) \displaystyle\prod_{i\in[1,\dots,M_k]\setminus j} p_C(\mathrm{y}_k^i), & j=1,\dots,M_k \end{cases} \\
&\propto \begin{cases} (1-P_D)\dfrac{\lambda_{FA}^{M_k}e^{-\lambda_{FA}}}{M_k!}V^{-M_k}, & j=0 \\ \dfrac{1}{M_k}P_D p(\mathrm{y}_k^j|\mathrm{x}_k)\dfrac{\lambda_{FA}^{M_k-1}e^{-\lambda_{FA}}}{(M_k-1)!}V^{-M_k+1}, & j=1,\dots,M_k \end{cases} \\
&\propto \frac{\lambda_{FA}^{M_k}e^{-\lambda_{FA}}}{M_k!}V^{-M_k} \begin{cases} (1-P_D), & j=0 \\ P_D p(\mathrm{y}_k^j|\mathrm{x}_k)\dfrac{V}{\lambda_{FA}}, & j=1,\dots,M_k \end{cases}
\end{aligned} \tag{3.45}$$

Therefore, the normalised PDA mixture weights can be computed as

$$\beta_k^j = \begin{cases} \dfrac{1-P_D}{1-P_D+\sum_{i=1}^{M_k}\mathcal{L}_k^j}, & j=0 \\ \dfrac{\mathcal{L}_k^j}{1-P_D+\sum_{i=1}^{M_k}\mathcal{L}_k^j}, & j=1,\dots,M_k \end{cases} \tag{3.46}$$

where

$$\mathcal{L}_k^j \triangleq \frac{\mathcal{N}(\mathrm{y}_k^j;\bar{\mathrm{y}}_k,S_k)P_D}{\lambda} \tag{3.47}$$

and $p(\mathrm{y}_k^j|\mathrm{x}_k) = \mathcal{N}(\mathrm{y}_k^j;\bar{\mathrm{y}}_k,S_k)$ is the likelihood of measurement y_k^j given the Gaussian measurement prediction with mean $\bar{\mathrm{y}}_k$ and innovation covariance S_k, where we have used the fact that $\lambda_{FA} = \lambda V$.

In the original derivation of PDA [58], the presented algorithm requires that the measurement update (3.8) of the standard Kalman Filter is modified, such as to reduce the mixture over the measurement innovation terms associated with each hypothesis. Nevertheless, as demonstrated by (3.44), the same process can be more conveniently performed once the individual posterior estimates of (3.43) have been obtained, by using the standard Kalman Filter recursion to update each hypothesis, and then reducing the mixture using the weights in (3.46), as follows:

$$\bar{\mathrm{x}}_k = \sum_{j=0}^{M_k} \beta_k^j \bar{\mathrm{x}}_k^j \tag{3.48}$$

$$\bar{V}_k = \sum_{j=0}^{M_k} \beta_k^j (\bar{V}_k^j + (\bar{\mathrm{x}}_k^j - \bar{\mathrm{x}}_k)^T (\bar{\mathrm{x}}_k^j - \bar{\mathrm{x}}_k)) \tag{3.49}$$

It is worth noting here that even when PDA is applied in the context of linear-Gaussian models, the over-arching PDA algorithm is ultimately a nonlinear estimator. Although the estimate update in (3.48) may appear linear, it is in fact nonlinear because the association probabilities β_k^j depend on the innovations according to (3.46). What is more, the mixture reduction process performed on each update leads to an increase of the covariance of the updated state, a phenomenon also known as the effect of the measurement origin uncertainty [58].

3.3.3.2 Extension of PDA to Particle Filtering

As mentioned in the previous section, the PDA algorithm is inherently non-linear. This fact, combined with the Gaussian Mixture reduction that is necessitated by the nature of the Kalman Filtering PDA approach can introduce significant errors in the estimation performance of the Kalman Filter based PDA, even when the models considered are linear and Gaussian. A typical example can be illustrated when all the posterior distributions resulting from the update of each measurement hypothesis appear to have comparable weights, but their means fall relatively far apart. In this case, even if the individual posteriors are highly informative, the mixture reduction process can result in much less informative combined update.

This is where the very nature of Expected Likelihood Particle Filters [28] can provide huge benefits. From (3.31), it can be observed that the particle weights depend strongly on $p(\mathrm{y}_k|\mathrm{x}_k^i)$, i.e. the likelihood of the received observation y_k with respect to the particle state x_k^i. However, in the standard Particle Filter recursion, the weight update step assumes that only one measurement is available at each time step and thus evaluation of $p(\mathrm{y}_k|\mathrm{x}_k^i)$ is fairly straightforward, given a well defined measurement model.

In contrast to the scenario of a single target with no-clutter, for which the standard Particle Filter is suitable, when tracking targets in clutter, the tracker may be presented with a variable number of different measurements, including none. In terms of the weight update process, this means that the likelihood $p(Y_k|\mathrm{x}_k^i)$ over all measurement hypotheses needs to be evaluated. Thus, the weight update equation (3.31) can be redefined as follows:

$$\mathrm{w}_k^i = \mathrm{w}_{k-1}^i \frac{p(Y_k|\mathrm{x}_k^i)p(\mathrm{x}_k^i|\mathrm{x}_{k-1}^i)}{\pi(\mathrm{x}_k^i|\mathrm{x}_{k-1}^i, \mathrm{y}_{1:k})}, \qquad i = 1, 2, ..., N_p \tag{3.50}$$

This is easily achieved by noting that, for the each particle, the joint likelihood of the measurements at time t_k is a mixture distribution, where the expected likelihood is the weighted sum of the individual likelihoods with the weights given by PDA. Recalling the definition from the previous section, PDA utilises the mixture weights β_k^j, where each entry β_k^j represents the association probability $p(\theta_k^j|Y_{1:k})$, which is shown in (3.38) to be proportional to the likelihood of each hypothesis. Thus, it can easily be shown that the expected likelihood $p(Y_k|\mathrm{x}_k^i)$ can be computed via a process akin to the PDA equations (3.45)-(3.46) as

$$p(Y_k|\mathrm{x}_k^i) \propto \sum_{j=0}^{M_k} \beta_k^j \tag{3.51}$$

The weights β_k^j can be computed according to (3.45), but a slight modification needs to be applied to (3.47) in order to consider the fact that the likelihood is now evaluated for each individual particle as

$$\mathcal{L}_k^j \triangleq \frac{\mathcal{N}(\mathrm{y}_k^j; \bar{\mathrm{y}}_k^i, R_k)P_D}{\lambda} \tag{3.52}$$

where $\bar{\mathrm{y}}_k^i$ denotes the predicted measurement for the i-th particle and R_k is the measurement noise covariance at time t_k. It is worth noting here that the use of a Gaussian likelihood function in (3.52) is only used to maintain similar notation to (3.47).

Finally, by substituting (3.51) and (3.52) in (3.50), we can obtain a new expression for the particle weights update equation

$$\mathrm{w}_k^i \propto \mathrm{w}_{k-1}^i \frac{\left[\lambda(1-P_D) + \sum_{j=1}^{M_k} \mathcal{N}(\mathrm{y}_k^j; \bar{\mathrm{y}}_k^i, R_k)P_D\right] p(\mathrm{x}_k^i|\mathrm{x}_{k-1}^i)}{\pi(\mathrm{x}_k^i|\mathrm{x}_{k-1}^i, \mathrm{y}_{1:k})} \tag{3.53}$$

The above formulation gives rise to an algorithm known as the Expected Likelihood (EL) Particle Filter [28], which is essentially a PDA implementation in a Particle Filtering context. In other literature, this PDA implementation is alternatively reffered to as Monte-Carlo PDA (MCPDA)[61, 62].

There exist a number of benefits associated to the use of MCPDA over the standard KFPDA. The first benefit stems from the ability of MCPDA to avoid the necessity for performing the mixture reduction of (3.48)-(3.49). In essence, even though MCPDA still exhibits an overall increased uncertainty in the produced posterior, caused by the effect of the measurement origin uncertainty, the likelihood evaluated for new measurements at each iteration is performed over the true likelihood density, rather than its Gaussian approximation. Furthermore, as the core recursion of MCPDA is built upon the standard Particle Filter recursion, the algorithm exhibits the same benefits over KFPDA when applied to non-linear, non-Gaussian models, as outlined for the case of Kalman and Particle Filters in the context of standard Bayesian Filtering in Section 3.2.

3.4 Performance analysis

This section presents results generated by simulating and comparing the performance achieved by the various algorithms presented in this chapter. The content presented in this section aims to demonstrate two things: i) the ability of non-linear models to describe the motion of objects with greater accuracy in a chosen scenario, ii) the ability of Particle Filters to outperform any Kalman Filtering approaches to tracking with such non-linear models.

3.4.1 Non-linear dynamic models

For the purpose of simulations, we consider the scenario of tracking the position of a single manoeuvring ship, on a two-dimensional Cartesian plane (i.e. with x, y coordinates). The trajectory followed by the target (see Fig. 3.1) contains 3 non-linear manoeuvres at times $t_{m_1} = 7s$, $t_{m_2} = 73s$ and $t_{m_3} = 93s$, with linear (straight-line) motion between the manoeuvres. The target maintains a steady absolute speed of 36 knots (≈ 42.5 mph) throughout the entire trajectory. Noisy positional measurements are received every second and are generated by adding random, zero-mean Gaussian noise to the ground-truth data, with standard deviation σ_m. That is the measurement(s) available at time k are given by a Linear-Gaussian model, of the form:

$$\mathrm{y}_k = \underbrace{[x_k, y_k]^T + \mathrm{r}_k}_{h_k(\mathrm{x}_k,\mathrm{r}_k)}, \ \mathrm{r}_k \sim N(0, \sigma_m^2 I_{2x2}) \tag{3.54}$$

where x_k, y_k correspond to the positional coordinates of the target on the 2-dimensional Cartesian plane and σ_m denotes the standard deviation of the i.i.d. Gaussian noise on each coordinate.

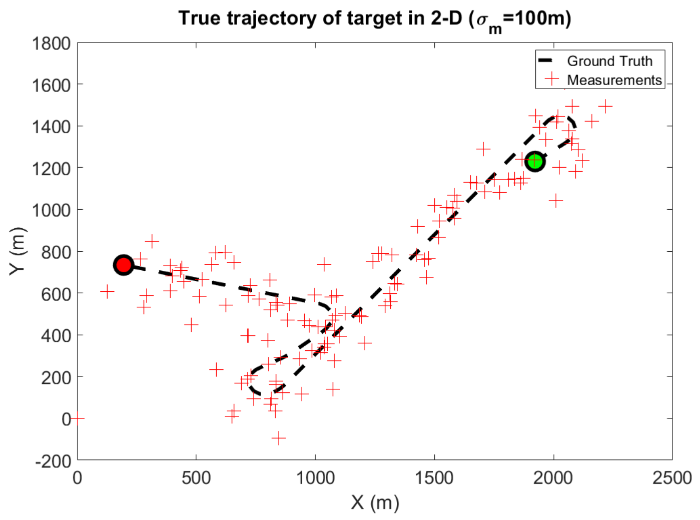


Figure 3.1: True trajectory followed by the ship during our experiments. The start and end positions are denoted by the green and red filled circles respectively, the ground-truth path of the target is denoted by the dashed line and the red points show an example set of generated measurements for one simulation run, with $\sigma_m = 100m$.

3.4.1.1 Single Target, no clutter

The first test-case scenario aims to investigate the benefits of using non-linear dynamic models under the standard Bayesian Filtering context, in combination with Particle Filters. Thus, it is assumed that only a single target is present in the surveillance region, prior information about the target's initial position and velocity is available, and at each timestep exactly one noisy measurement is obtained (i.e. there exist no clutter or missed detections). The trajectory followed by the target can be observed in Fig. 3.1.

The models employed for the purposes of evaluation are the following:

- A nearly-Constant Velocity (CV) model, with noise diffusion coefficient $\sigma = 1m/s$ (see Section 2.2.2). The model assumes that the target's velocity on each axis remains nearly-constant between consecutive iterations. It is described by a set of linear Stochastic Differential Equations (SDEs), where the rates of change on each dimension are assumed to evolve according to Brownian motions.

- The nearly-Constant Heading (CH) model, with $\sigma_s = 1m/s$ and $\sigma_\phi = 0.16rad/s$ (see Section 2.2.4). The model decouples the absolute speed and heading of a target and assumes that both quantities remain nearly constant between consecutive iterations. The SDEs that govern the CH model differ to those for the CV in that they are non-linear. Thus, direct application to a KF context requires the use of one of the two approximation variants, namely EKF and UKF.

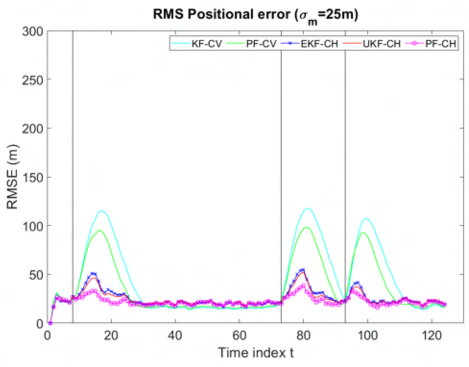
The tracking performance of 5 different algorithm-model pairs has been evaluated, as listed below:

- KF-CV: A standard Kalman Filter (KF) using the CV model. EKF, UKF reduce to the standard KF under linear models.
- PF-CV: A Particle Filter (PF) employing 5000 particles, using the CV model.
- EKF-CH: A first-order Extended Kalman Filter (EKF) using the CH model.
- UKF-CH: An Unscented Kalman Filter (UKF) using the CH model.
- PF-CH: A Particle Filter (PF) employing 5000 particles, using the CH model.

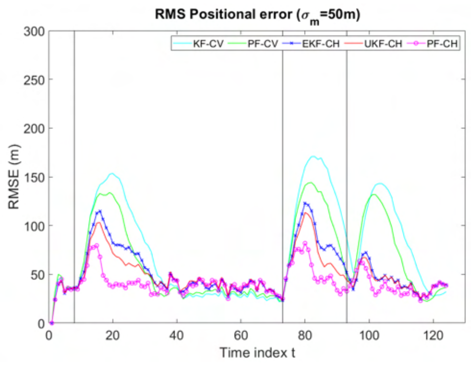
The performance evaluation is achieved by comparing the positional Root Mean Squared Error (RMSE) for each algorithm-model pair, averaged over 50 Monte-Carlo simulation runs, with variable values of measurement noise intensity σ_m. Figures 3.2(a)-3.2(d) show the relevant results received from examining four different variations, by allowing σ_m to take the values of $25m$, $50m$, $75m$ and $100m$, respectively.

As it can be observed, during the linear parts of the trajectory, the RMSE for KF-CV is marginally lower than that for PF-CV, while both are superior to any CH based algorithms, all of which produce similar levels of RMSE. This observation implies that the steady-state error of the CV model is lower than that of the CH model. Following each of the manoeuvres, we can compare the performance for each of the algorithms in terms of the overshoot error (i.e. max RMSE following a manoeuvre) and settling time (i.e. time to converge following a manoeuvre). In this context, PF-CH is found to exhibit superior performance across all variations of σ_m, by demonstrating the lowest overshot error and lowest settling time. The rest of the algorithms rank in the following order: UKF-CH, EKF-CH, PF-CV, KF-CV. As σ_m is increased, the RMSE for EKF-CH and UKF-CH can be observed to increase relative to the RMSE of PF-CH, while approaching (and even surpassing) the RMSE for KF-CV and PF-CV. Such a behaviour is expected for EKF and UKF in general, since an increase in σ_m implies an amplification of the involved non-linearities, already induced by the target manoeuvres, which has a detrimental effect to the linear/Gaussian approximations inherent to both algorithms.

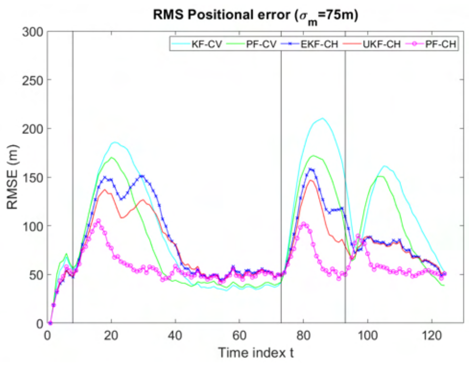
Overall, the above results confirm the expectations relating to the advantages



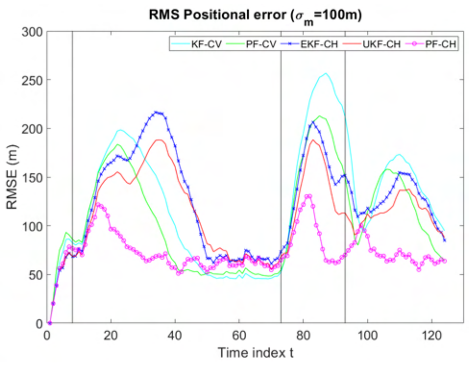
(a) $\sigma_m = 25m$



(b) $\sigma_m = 50m$



(c) $\sigma_m = 75m$



(d) $\sigma_m = 100m$

Figure 3.2: RMSE vs Time for each of the 5 algorithms, with 4 different values of noise intensity σ_m. The black vertical lines, drawn at $t_{m_1} = 7s$, $t_{m_2} = 73s$ and $t_{m_3} = 93s$, depict the respective times when each of the 3 manoeuvres is initiated.

of Particle Filters and non-linear dynamic models. It can be observed that, even when examining a very simple scenario, a more accurate (non-linear) dynamic model can yield significantly better tracking performance, whose benefits are further complemented by the use of a PF. The advantages of combining PFs with non-linear models become more obvious when clutter is introduced in the measurements. Such a scenario will be examined next.

3.4.1.2 Single Target, with clutter

Building up from the results observed in the previous section, this test-case scenario is very much identical in terms of the trajectory followed by the target. However, in contrast to before, additional (clutter) measurements are introduced at each measurement scan, while target detectability is modelled according to a Constant Detection Rate model (see Section 2.4.1, with probability of detection

$P_D = 0.9$. Clutter is assumed to evolve according to a Poisson Rate with Uniform Position model (see Section 2.5.1), where the number of clutter measurements per scan is assumed to be Poisson distributed, with mean λ_{FA} across the surveillance volume V, while their positions are uniformly distributed over the entire region. Figure 3.3 shows a snapshot of an example measurement scan extracted from one of the simulations.

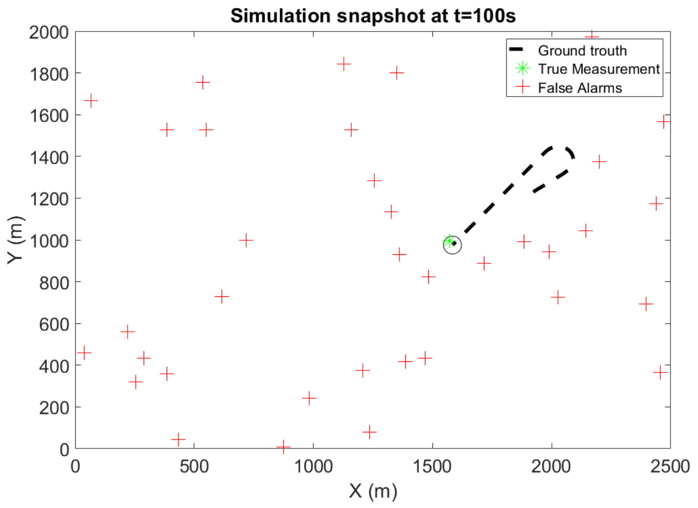


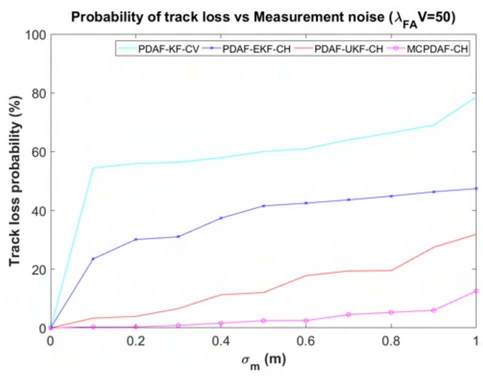
Figure 3.3: Snapshot of an example measurement scan, with $\lambda_{FA} = 50$. Clutter measurements are shown in red, and the true target measurement is shown in green.

The tracking performance of 4 different algorithms has been evaluated, as follows:

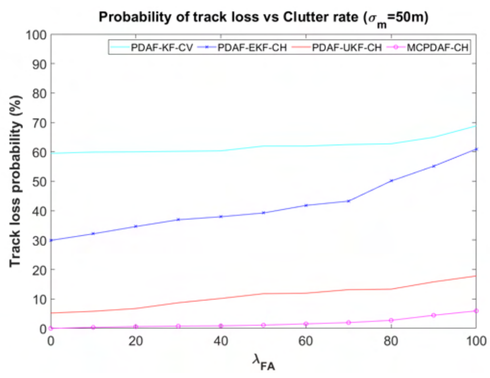
- PDAF-KF-CV: A Probabilistic Data Association Filter (PDAF) using the CV model and a Kalman Filter (KF) to perform prediction and update of tracks. EKF, UKF reduce to the standard KF under linear models, and thus have not been evaluated.
- PDAF-EKF-CH: A Probabilistic Data Association Filter (PDAF) using the CH model and a first-order Extended Kalman Filter (EKF) to perform prediction and update of tracks.
- PDAF-UKF-CH: A Probabilistic Data Association Filter (PDAF) using the CH model and an Unscented Kalman Filter (UKF) to perform prediction and update of tracks.
- MCPDAF-CH: A Monte-Carlo (MC) Probabilistic Data Association Filter (PDAF) using the CH model and a Particle Filter (PF) employing 5000 particles to perform prediction and update of tracks..

Due to the increased uncertainty, added to the problem through the inclusion of missed detections and clutter measurements, it becomes possible for tracks to diverge from the true trajectory and begin tracking false alarms, leading to a phenomenon known as track loss. When an algorithm loses track of a target, the amount by which the estimate diverges from the true trajectory can vary to a great extent and, thus, the computed RMSE is no longer directly relevant to the tracking performance of the algorithm. For this reason, performance evaluation of the individual algorithms is performed using a different metric to the one employed for the previous test case.

Given a measurement noise intensity σ_m, we consider that an algorithm loses track of the target when the RMSE of its estimate is higher than $3\sigma_m$. A check is performed on each time step, which indicates whether a track loss event has occurred for each of the algorithms. Continuing, once a single simulation has been executed, the track loss probability for each algorithm can be computed as the fraction of time steps during which a track loss event has occurred, over the total number of time steps for the simulation. When averaged over a number of Monte Carlo simulations, the track loss probability gives us a measure of how likely an algorithm is to lose track of a target, given a set of chosen parameters. Figures 3.4(a)-3.4(b) show the results obtained by performing two experiments to compute the track loss probability for each algorithm, with varying measurement noise intensity σ_m and clutter rate λ_{FA}, respectively for each experiment, averaged over 50 Monte Carlo simulation runs.



(a)



(b)

Figure 3.4: Track loss probability for each of the 4 algorithms, for variable values of (a) measurement noise intesnity (σ_m) and (b) mean clutter rate over the entire surveillance region (λ_{FA}).

The ranking order of the four algorithms is the same in both experiments, which also coincides with the ranking obtained in the previous section for the equivalent algorithms. The PDAF-KF-CV algorithm has the highest levels of track loss probability, making it the worst choice out of the four. Second to last was the PDAF-EKF-CH algorithm, followed by PDAF-UKF-CH. One observation that can be made here is that, even though EKF-CH and UKF-CH had very similar RMSE in the single-target, no clutter scenario, the performance improvement of PDAF-UKF-CH over PDAF-EKF-CH is much more noticeable here. This can be explained by the ability of UKF to better capture the posterior's higher-order moments relative to EKF, combined with the fact that PDAF, by definition, produces multi-modal likelihoods (assuming multiple measurements are available). Finally, the best performing algorithm is the MCPDAF-CH, which demonstrates incredibly low track loss probability, due to the ability of the underlying PF to fully capture the multi-modalities explained above, while also avoiding any linear approximation of the models.

As it can be observed, the advantages stemming from the combination of particle filtering methods, with non-linear dynamic models, become increasingly noticeable as the involved uncertainty grows. This is a strong indication that the performance improvement will be even greater when the complexity of the tracking problem is increased, by extending the problem to involve tracking of multiple (closely-spaced) targets, with unknown initial positions and lifespan, under highly non-linear/non-Gaussian observation and clutter models. Work conducted towards achieving this extension is presented in the next section.

3.4.2 Nonlinear measurement models

3.4.2.1 Single Target, no clutter

For the purposes of simulations, we consider the scenario of tracking the position of a single vessel, on a two-dimensional Cartesian plane, using synthetic radar data. The simulated trajectory followed by the target (see Figure 3.5) contains 3 non-linear manoeuvres, with linear (straight-line) motion between the manoeuvres, and the vessel maintains a steady absolute speed of approx. 36 knots (≈ 42.5 mph), throughout the entire trajectory. The dynamics of the target are assumed to be governed by a Constant Heading (CH) model (see Section 2.2.4), with $\sigma_s = 1m/s^2$ and $\sigma_\phi = 0.16rad/s$.

A Radar sensor is placed at coordinates $(0m, 0m)$, which generates observations of the target position in terms of azimuth & range, over 185 scans, at a rate of 3 sec/scan. Measurement noise is modelled using a Gaussian Azimuth-Range measurement model (see Section 2.3.2), having Gaussian noise in both azimuth and range, with standard deviations σ_θ and σ_r, respectively. That is, the measurement(s) available at time t_k are given by:

$$\mathrm{y}_k = \underbrace{\begin{bmatrix} \arctan2(y_k, x_k) \\ \sqrt{x_k^2 + y_k^2} \end{bmatrix}}_{h_k(\mathrm{x}_k, \mathrm{r}_k)} + \mathrm{r}_k, \; \mathrm{r}_k \sim \mathcal{N}\left(0, \begin{bmatrix} \sigma_\theta^2 & 0 \\ 0 & \sigma_r^2 \end{bmatrix}\right) \tag{3.55}$$

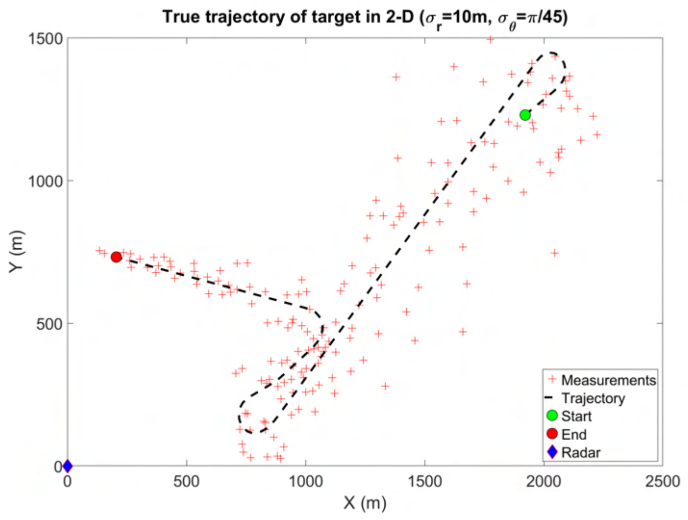


Figure 3.5: True trajectory followed by the vessel during our experiments. An example of the entire history of measurements, generated in a single experiment, with noise parameters $\sigma_r = 10\ m$ and $\sigma_\theta = \frac{\pi}{45}\ rad$, is also depicted.

The tracking performance of 3 different algorithms has been evaluated, as follows:

- EKF: An Extended Kalman Filter (UKF)[24, 37].
- UKF: An Unscented Kalman Filter (UKF)[43].
- PF: A Particle Filter (PF)[6] employing 5000 particles.

The performance evaluation is achieved by means of comparing the positional Root Mean Squared Error (RMSE) for each algorithm, averaged over 50 Monte-Carlo simulation runs, with variable values of noise intensity σ_r and σ_θ. From the results of Table 3.1, it can be observed that, irrespective of the measurement noise intensity, the PF provides a substantial improvement compared to the other two algorithms, with UKF ranking second and EKF following closely behind. The performance differences between the three algorithms become increasingly apparent as the measurement noise, in either range and/or bearing, is increased.

The observed outcome can be attributed to the linearisation and Gaussian approximations involved in both EKF (Taylor-Series expansion) and UKF (Unscented Transform), respectively, which introduce estimation errors in both algo-

Params / Filter	Mean RMSE (m) $\sigma_r = 10\ m$, $\sigma_\theta = \frac{\pi}{180}\ rad$	$\sigma_r = 50\ m$, $\sigma_\theta = \frac{\pi}{180}\ rad$	$\sigma_r = 10\ m$, $\sigma_\theta = \frac{\pi}{90}\ rad$	$\sigma_r = 10\ m$, $\sigma_\theta = \frac{\pi}{45}\ rad$
No filter	13.0688	47.3391	47.4761	432.0758
EKF	5.9532	22.8774	14.8999	79.6216
UKF	5.6949	20.8812	14.0779	73.3490
PF	4.7842	15.3307	12.7059	58.5702

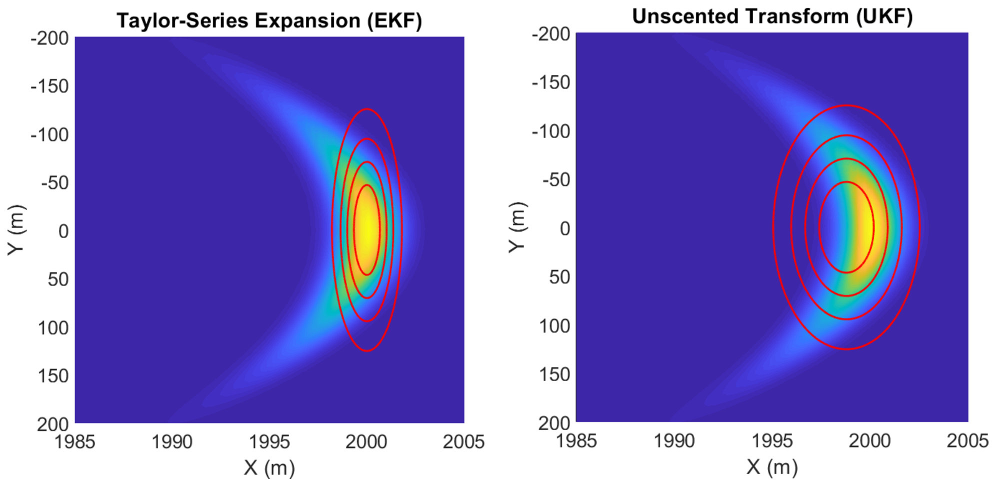
Table 3.1: Performance comparison between EKF, UKF and PF for range of measurement noise parameter values. The computed mean RMSE, averaged over 50 Monte-Carlo simulations is presented, for varying values σ_r and σ_θ.

rithms. An example illustration of these phenomena is shown in Figure 3.6. More specifically, the involved first-order analytic linearisation process causes the EKF to underestimate the covariance of the measurement noise, leading to a similar effect on the covariance of its estimates. On the other hand, UKF overcomes this issue through its utilisation of a deterministicaly selected set of (non-linearly) transformed (sigma) points to perform statistical linear regression. Even so, the performance of UKF deteriorates significantly as the approximated density becomes increasingly non-Gaussian. Finally, we can see that the PF, by utilising a Monte-Carlo approximation of the uncertainty, is able to fully capture the measurement noise distribution, thus providing a significant performance benefit over the other algorithms, as demonstrated by the results.

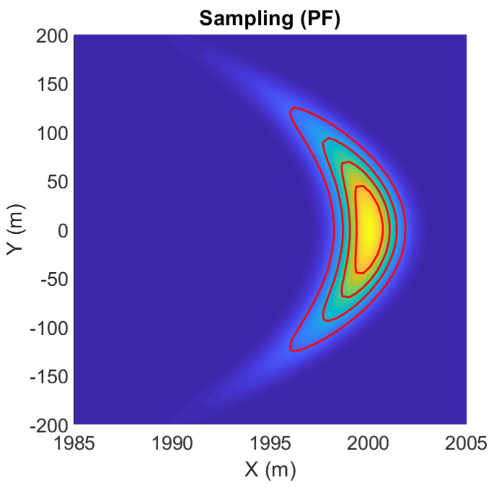
3.4.2.2 Single Target, with clutter

This section extends the previous experiment by considering clutter and missed detections, thus necessitating data association. The scenario of interest is the same as before, however target detectability is modelled according to a Constant Detection Rate model (see Section 2.4.1), with probability of detection $P_D = 0.9$. Measurement noise is modelled using a Gaussian Azimuth-Range measurement model (see Section 2.3.2), having Gaussian noise in both azimuth and range, with standard deviations $\sigma_\theta = \frac{\pi}{180} rad$ and $\sigma_r = 10m$, respectively. Additionally, clutter is modelled using to a Poisson Rate with Uniform Position model (see Section 2.5.1), where the number of clutter measurements per scan is assumed to be Poisson distributed, with mean λ_{FA} across the surveillance volume V, while their positions are uniformly distributed over the polar search space defined on the azimuth and range coordinate system, such that $p_C(\mathrm{y}_k^j) = U([0, 2500], [0, 2\pi])$.

The tracking performance of 3 different algorithms has been evaluated, as follows:



(a) EKF (b) UKF



(c) PF

Figure 3.6: Visualisation of radar measurement noise approximation in EKF, UKF and PF. The first 4 contours of each distribution, are plotted with red lines.

- PDA-EKF: Probabilistic Data Association, using an EKF as the underlying filter.
- PDA-UKF: PDA, using a UKF as the underlying filter.
- PDA-PF: PDA, using a PF (with 5000 particles) as the underlying filter.

The performance evaluation is achieved by means of comparing the value of the Root Mean Squared Error (RMSE) achieved by each algorithm, averaged over 50 Monte-Carlo simulation runs, with variable values of clutter rate λ_{FA}. From the results of Table 3.2, it can be observed that the merits of combining

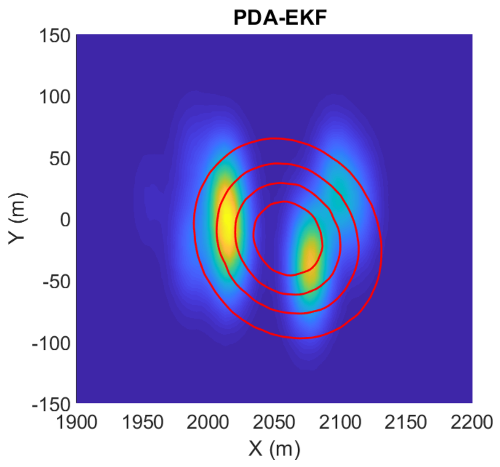
PFs with non-linear, non-Gaussian models become increasingly apparent as the involved uncertainty grows. The jumps of RMSE values, observed in the case of both PDA-EKF and PDA-UKF, indicates that, for the given clutter rate, these algorithms lost track of the target, leading to a false track that drifts off and gets driven solely by clutter measurements.

	Mean RMSE (m)			
Params / Filter	$\lambda_{FA} = 0$	$\lambda_{FA} = 50$	$\lambda_{FA} = 250$	$\lambda_{FA} = 500$
PDA-EKF	5.8925	56.4789	108.6974	184.7950
PDA-UKF	5.6689	5.9847	74.6874	98.8756
PDA-PF	4.7925	5.3145	5.5432	7.2534

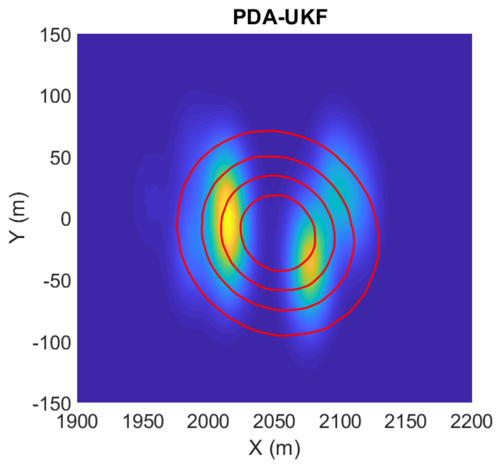
Table 3.2: Performance comparison between PDA-EKF, PDA-UKF and PDA-PF for range of clutter parameter values. The computed mean RMSE, averaged over 50 Monte-Carlo simulations is presented, for varying values λ_{FA}.

When expressed in its true form, PDA leads to multi-modal estimates, as the posterior distribution for each target is acquired on the basis of calculating a weighted average over a set of joint hypotheses, formed by assuming that: i) a single measurement originated from at most one target; and ii) each target produces at most a single measurement. Even under the basic assumption of a Gaussian prior and linear-Gaussian models, the resulting posterior distribution is a Gaussian mixture, with individual weights corresponding to the joint association probabilities generated by PDA. In the case of PDA-EKF and PDA-UKF, the mixture is reduced to a single Gaussian at each timestep, leading to approximation errors even in the most basic scenario. As previously shown (see Chapter 2), when non-linear, non-Gaussian models are introduced, the resulting posterior is no longer guaranteed to result in a Gaussian, thus inducing further approximation errors in the two algorithms.

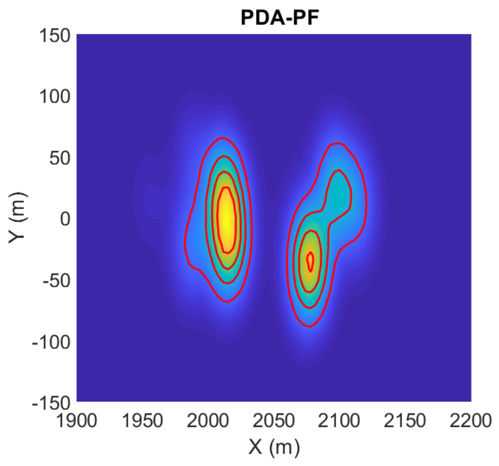
The above mentioned phenomenon is illustrated in Figure 3.7, where we can see that PDA-PF is able to capture the full multi-modality and non-Gaussianity of the resulting posterior, while the other two algorithms approximate the density as a reduced single Gaussian, with the mean of both estimates falling in a region of low probability in the true density. It is also worth noting that the covariance of the estimate formed by PDA-EKF is slightly tighter than that of PDA-UKF, which is a direct result of the issues associated with EKF underestimating the measurement (as well as dynamic) model noise covariance, as explained and demonstrated in Chapter 2.



(a) EKF



(b) UKF



(c) PF

Figure 3.7: Visualisation of an example single target posterior distribution, as estimated by PDA-EKF, PDA-UKF and PDA-PF, respectively. The background intensity of each subfigure shows the true distribution, while the red lines show the first 4 contours of each distribution.

3.5 Conclusion

This chapter delved into the specifics of the Bayesian Filtering and Data Association problems in the context of Single Target Tracking, building on the framework of state-space models presented in Chapter 2, and presented a review of existing mainstay algorithms. An introduction was provided to the fundamental concepts of the standard Bayesian Filtering framework, which were then utilised to formulate the Kalman and Particle Filtering algorithms, while discussing the various advantages and pitfalls of each approach. Furthermore, the Data Association problem was considered in the context of Single-Target Tracking. This discussion was followed by a brief discussion on the Nearest Neighbour and Probabilistic Data Association algorithms, while in the latter case a clear distinction was made between the application of the algorithm in a Kalman and Particle Filtering context, respectively. Finally, a review of the algorithms was performed by comparing the performance achieved by each algorithm in a range of simulated scenarios. The benefits of combining Particle Filters with non-linear, non-Gaussian dynamic and measurement models were demonstrated, while in both cases emphasis was given to the amplification of the achieved enhancement when the problem is extended to consider clutter and missed detections.

CHAPTER 4

Non-Linear, Non-Gaussian Multi-Target Tracking

4.1 Introduction

When the number of considered targets is greater than one, the problem of tracking is referred to as Multi-Target Tracking (MTT). Multi-Target Tracking in the presence of clutter and sensor probability of detection less than unity is more intricate than the Single-Target Tracking problem. In addition to process and measurement noise in the dynamic and observation models, respectively, one has to contend with much more complex sources of uncertainty, such as the measurement origin uncertainty, multi-target data association, false alarms, missed detections, as well as births and deaths of targets.

In contrast to the Single-Target Tracking case, Multi-Target Tracking aims to jointly estimate both the number of targets that are present within the surveillance region and their respective states. Both the states of targets, as well as their number may vary over time, as targets can move unexpectedly and they can appear or disappear at random. Again, a set of measurements is assumed to be received at discrete intervals, where each measurement can potentially originate from clutter, from existing targets that are already known to exist, and/or from new targets that have appeared in the surveillance area since the last report time. Hence, it becomes apparent that the data association problem in the case of Multi-Target Tracking becomes increasingly complex, not only because multiple targets may share measurements, but also due to the added uncertainty of whether such measurements should be attributed to the appearance of new or fading targets.

Conventional approaches to solving the Multi-Target Tracking problem involve breaking the problem down to a set of well-defined sub-system components, each tasked with a particular operation [1, 27] (Figure 4.1). This paradigm has proven successful in providing a solution in a multitude of applications [63, 64, 65, 66]. Recent advancements in the field of Multi-Target Tracking have given birth to algorithms that circumvent the necessity of decomposing the problem as is done by convention [67, 68]. Such approaches view the Multi-Target Tracking problem as a joint Bayesian estimation problem, and derive their formulation from the Multi-Target Bayes Filter, building upon the now well established theory of Random Finite Sets [69, 70].

This chapter builds on the concepts of state-space modelling and Single-Target Tracking presented in the previous chapters, and extends the discussion to the case of Multi-Target Tracking. Section 4.2 introduces the Multi-Target Bayes Filter, while drawing relation to the Standard Bayes Filter. Section 4.3 provides a summary and brief derivation of the algorithms that form the main components of conventional Multi-Target trackers, with special focus to the utilised Data Association and Track Management methods. In the same section, the author presents a discussion on the relation between the Joint Probabilistic Data Association (JPDA) and Joint Integrated PDA algorithms, with the aim of highlighting how the latter can be performed using the same constructs. Section 4.5 continues by discussing the concept of Random Finite Sets (RFS) and presents a formulation of the Probability Hypothesis Density (PHD) filter as an approximation to the Multi-target Bayes Filter. Section 4.6 discusses a state-of-the-art radar track initiation technique which utilises a PHD filter to model the density of uninitiated targets and consecutively propose tracks for initiation on the basis of target existence probabilities. Preliminary simulation results are presented in Section 4.6.1 to showcase the performance benefits of the PHD track initiator compared to other mainstay approaches using synthetic data, while Section 4.7 presents a case study performed on real data collected from a commercial radar, whereby a more thorough qualitative analysis is performed on a pair of challenging scenarios, with the aim of demonstrating a real operational advantage.

4.2 The Multi-Target Bayes Filter

Suppose that at time step $t_k \in \mathbb{N}$, there exist N_k targets with states $\mathrm{x}_k^1, \dots, \mathrm{x}_k^{N_k}$ taking values in the single-target state space $\mathcal{X} \subseteq \mathbb{R}^{n_x}$. Then the multi-target state can be represented as a finite set X_k of the form:

$$X_k = \{\mathrm{x}_k^1, \dots, \mathrm{x}_k^{N_k}\} \in \mathcal{F}(\mathcal{X}) \tag{4.1}$$

where $\mathcal{F}(\mathcal{X})$ is the collection of all finite subsets of $\mathcal{X}$. At each timestep, new targets may appear in the vicinity of the surveillance region $\mathcal{S}$, while existing targets may either continue to exist or disappear from $\mathcal{S}$. Furthermore, targets may be detected in a given timestep, but it is also possible that they may go undetected.

Similarly, suppose that at t_k a set of M_k measurements $\{\mathrm{y}_k^1, ..., \mathrm{y}_k^{M_k}\}$ are received, each taking values in the single-target measurement space $\mathcal{Y} \subseteq \mathbb{R}^{n_y}$. Then the multi-target measurement is represented by the finite set Y_k as:

$$Y_k = \{\mathrm{y}_k^1, \dots, \mathrm{y}_k^{M_k}\} \in \mathcal{F}(\mathcal{Y}) \tag{4.2}$$

where $\mathcal{F}(\mathcal{Y})$ is the collection of all finite subsets of $\mathcal{Y}$.

Measurement scans are composed of detections originating from targets, a.k.a. true detections, as well as spurious detections, a.k.a. clutter, that can originate from a number of different sources: background interference and/or internal interference of the sensors. Thus, Y_k can be decomposed as follows:

$$Y_k = Y_k^C \cup Y_k^\Delta \tag{4.3}$$

where Y_k^C contains the set of M_k^C clutter measurements, and similarly Y_k^Δ contains the set of M_k^Δ detections generated by targets.

Therefore, the objective of the Multi-Target Bayes Filter [71] is then to obtain an expression for the posterior probability density of the multi-target state, conditional on all measurements received up to and including time t_k. In mathematical terms, the quantity to be computed is a probability density function (pdf) of the form:

$$p(X_k|Y_{1:k}) \tag{4.4}$$

The Multi-Target Bayes Filter recursion can be obtained by generalising the standard Bayes Filter recursion discussed in Section 3.2. Thus, the recursion is again performed using two steps: prediction and update. The prediction step propagates the last estimate of the multi-target state (a.k.a. the prior) through the multi-target transition density such as to generate a prediction of the state pdf between measurement times. Since the state is usually subject to unknown disturbances (caused by target birth/death and transition noise), prediction generally translates, deforms, and spreads the state pdf. Next, the update stage makes use of the latest received measurement to update/correct the prediction pdf, through application of the Bayes theorem, which is the mechanism that allows updating one's knowledge about the target state in light of the new measurement.

Algorithm 4.2.1 (Multi-Target Bayes Filter) *The set of recursive equations necessary to compute the predicted $p(X_k|Y_{1:k-1})$ and posterior $p(X_k|Y_{1:k})$ distributions are given by the following Bayesian filtering equations.*

- **Initialisation:** *The filter is initialised with the prior distribution $p(X_0)$.*
- **Prediction:** *Given the multi-target transition density $p(X_k|X_{k-1})$ and the prior $p(X_{k-1}|Y_{1:k-1})$, the Chapman-Kolmogorov equation [30] can be used to compute the predicted distribution of X_k:*

$$p(X_k|X_{k-1}, Y_{1:k-1}) = \int p(X_k|X_{k-1})p(X_{k-1}|Y_{1:k-1})\delta X_{k-1} \tag{4.5}$$

- **Update:** *Given the multi-target measurement likelihood $p(Y_k|X_k)$, the measurement set Y_k and the above computed predicted distribution, the target posterior distribution $p(X_k|Y_{1:k})$ can be calculated using Bayes' rule:*

$$p(X_k|Y_{1:k}) = \frac{p(Y_k|X_k)p(X_k|X_{k-1}, Y_{1:k-1})}{\int p(Y_k|X_k)p(X_k|X_{k-1}, Y_{1:k-1})\delta X_k} \tag{4.6}$$

The recurrence relations of (4.5) and (4.6) form the basis for the Multi-Target Bayes Filter (MBF), which is the optimal Bayesian solution to the problem of recursively calculating the exact multi-target posterior density $p(X_k|Y_{1:k})$. It is important to note that this recursion is a non-trivial generalisation of (3.2) and (3.3). This is because the transition density $p(X_k|X_{k-1})$ needs to also consider the uncertainty introduced by the appearance and disappearance of targets, while the multi-target likelihood $p(Y_k|X_k)$ needs to consider the uncertainty relating to missed detections and false alarms. Furthermore, the integrals in the relations (4.5) - (4.6) are nonstandard set integrals. Finally, since the multi-target posterior density $p(X_k|Y_{1:k})$ is defined over $\mathcal{F}(\mathcal{X})$, practical implementations of MBF for non-linear, non-Gaussian models are challenging and usually limited to a small number of targets [72, 73]. Methods that have been developed to provide a solution to the above problems will be analysed in the following subsections.

4.3 Conventional Multi-Target Tracking

Conventional approaches to solving the Multi-Target Tracking problem involve breaking the problem down to a set of well-defined sub-system components, each tasked with a particular operation [1, 27]. This paradigm has proven successful in providing a solution in a multitude of applications [63, 64, 65, 66]. A structural overview of an exemplary conventional Multi-Target Tracking system is shown in Figure 4.1.

Similar to the case of Single-Target Tracking, standard Bayesian Filtering is applied in order to predict and update the states of targets, based on detection hypotheses generated via Data Association techniques. However, as already emphasised, the Data Association process is now much more involved, as it must be applied over the joint data association problem of interacting targets, with the added uncertainty of target existence. To reduce the computational complexity of the necessary Data Association, an optional Measurement Gating step is performed to limit the number of possible association hypotheses for each target. In addition to the above, Track Management methods are employed to handle the appearance and disappearance of targets in the surveillance region.

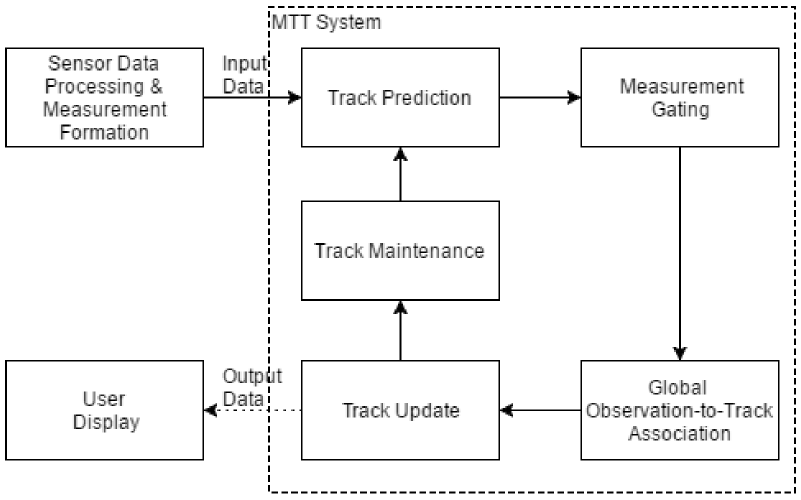


Figure 4.1: Basic components of a conventional Multi-Target Tracking system.

4.3.1 Measurement Gating

Measurement gating is an approximation technique, introduced as a pre-processing step on the detections, with the main objective of reducing the computational complexity of the data association process. In heavily cluttered environments, a fringe benefit of gating is to limit the clutter measurement density fluctuations within the gate. This is typically achieved by down-selecting only the most likely measurements that can be associated to each track, by drawing a validation region around each target, termed as the target's gate. The validation region then allows for a hard decision to be made about the different valid measurements. Although the concept of a validation region does not guarantee that a measurement in the gate has originated from the corresponding target, it provides a likely hypothetical candidate for the association problem.

There exist a multitude of gating methods in the tracking literature [74, 75, 76]. The most commonly employed gating technique uses an ellipsoid to represent the gate region. At each time step t_k, the previous state estimate x_{k-1} of a given target is predicted forward using an appropriately defined dynamic model, as discussed in Section 2.2, giving rise to a prediction $\mathrm{x}_{k|k-1}$. Making use of the measurement models discussed in Section 2.3, the target's expected measurement $\bar{\mathrm{y}}_k$ mean and the associated covariance matrix S_k can be computed. It is assumed that the true measurement y_k, conditioned on the set of all measurements up to t_{k-1}, is normally distributed with pdf

$$p(\mathrm{y}_k|\mathrm{x}_{k|k-1}, Y_{k-1}) = \mathcal{N}(\mathrm{y}_k; \bar{\mathrm{y}}_k, S_k) \tag{4.7}$$

Thus, the gate region $\mathcal{V}_G(k, \gamma)$ can be defined by an ellipse having a centroid $\bar{\mathrm{y}}_k$ and volume $V_G(k)$ as

$$\begin{aligned}\mathcal{V}_G(k, \gamma_G) &= \{\mathrm{y} : (\mathrm{y} - \bar{\mathrm{y}}_k)^T S_k^{-1} (\mathrm{y} - \bar{\mathrm{y}}_k) \leq \gamma_G\} \\ V_G(k) &= c_{n_\mathrm{y}} \gamma_G^{\frac{n_\mathrm{y}}{2}} |S_k|^{\frac{1}{2}}\end{aligned} \tag{4.8}$$

where n_y denotes the dimensionality of the measurement vector, c_{n_y} is the volume of the n_y-dimensional unit hypershpere and γ_G is the set gate threshold. The set $\mathcal{V}$ is defined by a region on the state-space also known as ellipsoid of probability concentration; that is, the minimum volume that contains a given probability mass, under the assumption that the uncertainty is normally distributed. The semi-axes of the ellipsoid are the square-roots of the eigenvalues of $\gamma_G S$.

The left hand side of the inequality condition of (4.8) is the Mahalanobis distance between some measurement y and the ellipse centroid $\bar{\mathrm{y}}_k$. As such, γ_G describes the maximum acceptable distance, in terms of the number of standard deviations from $\bar{\mathrm{y}}_k$, that measurements will be considered as valid association hypotheses for any given track. Provided that the target state and the measurement model are correct, γ_G is $\mathcal{X}^2$-distributed and can be easily computed given the desired probability of gating (P_G) and the dimensionality of the tracked state.

The process of determining the size of the gate can be crucial for tracking in highly cluttered environments. The introduction of a relatively small gate can lead to the exclusion of measurements originating from clutter, but also bears the danger of ignoring true measurements when the target does not follow closely the dynamic model assumptions (e.g. during a manoeuvre). At the same time, using a large gate may ensure that true measurements are captured by the gate, but can lead to the inclusion of large number of false or low-probability detections, which can hinder the computational performance of the system. This thesis will not delve into the specifics of gate size optimisation, however more information can be found in [75, 76].

4.3.2 Track Management

As is often the case, prior knowledge of the number of targets, as well as their initial target position and velocity, is not available to Multi-Target Tracking systems. Thus, Track Management is essential in order to identify and maintain tracks for persisting targets. A persisting target is most generally defined as an object that will persist in the surveillance region for several scans, while satisfying certain constraints regarding its dynamic behaviour and detectability.

Track Management is typically split into three sub-processes: i) Track Initiation; ii) Track Confirmation; and iii) Track Deletion. A notable exception of this convention are Track-Before-Detect methods, where Track Initiation and Confirmation are typically considered together, although this is not necessarily always true.

4.3.2.1 Track Initiation

The initiation step is employed in order to set up new tracks from measurements. This is usually accomplished by spawning new tracks for all measurements that are deemed as unlikely to have been associated with any existing targets.

In systems where measurement gating is performed, candidate measurements are typically chosen as ones that have not fallen withing the validation gate of any existing tracks; that is, they do not satisfy the inequality of (4.8) for any track. However, this can generally prove problematic in a number of situations. One example is when tracking closely spaced targets, where measurements from newly appeared targets will have a high probability of falling within the gates of existing tracks, meaning that these targets may take a long time to be initiated by the tracker, or even be missed altogether. Another potential problem can exhibit itself when the measurement noise generated by the sensors is relatively large, or when measurement reports are received at sparse intervals. Both scenarios can lead to a high uncertainty associated with the measurement prediction of targets, which (as seen from (4.8)) can lead to large target gates, thus posing similar issues to the previous example.

An alternative approach, that is also applicable to systems that do not employ gating measures, is to weight the impact of measurements by the probability that they are unused by existing tracks. As such, a threshold can be defined on the joint probability that a measurement is not associated to any targets. Let $p(\alpha_k^i = j|Y_k)$ denote the probability that y_k^j is associated to the i-th track, then the probability that measurement y_k^j is unused by the existing tracks can be approximated by

$$p(\mathrm{y}_k^j\ unused) \approx \prod_{i=1}^{N_k}(1 - p(\alpha_k^i = j|Y_k)) \tag{4.9}$$

An obvious limitation of this approach is that the joint association probabilities $p(\alpha_k^i = j|Y_k)$ are generally costly to compute. Thus, the above method can only typically be applied to systems that employ soft data association methods, such as (J)PDA and variants, as the quantities of interest are computed as part of the standard algorithm and thus can be readily extracted; the sole purpose of hard data association, such as (G)NN, is to circumvent the necessity of evaluating these quantities.

4.3.2.2 Track Confirmation and Deletion

In a large number of cases, especially in highly cluttered scenarios, many of the tracks spawned during the track initiation process may have originated from false alarms, thus necessitating the existence of a method for classifying newly spawned tracks. The concept of Track Confirmation is employed for this exact purpose. As such, newly initiated tracks, also known as *tentative* tracks, are typically dealt separately to persisting, or *confirmed*, tracks. What is more, once a tentative track is confirmed, it is not guaranteed to exist forever, and thus, Track Deletion schemes can be employed to identify and kill off fading tracks.

M-out-of-N

Early ad-hoc track management techniques based their operation on maintaining a count of M subsequent iterations, out of a N sized window of report intervals, during which a track has been successfully associated to at least one measurement. Due to their heuristic rule-based approach, such techniques are commonly referred to as *M-out-of-N* methods [77, 10]. With this M/N measure in hand, confirmation (M_{conf}) and deletion (M_{del}) thresholds can be set, such as to confirm tracks which have been successfully detected in M_{conf} out N iterations, and respectively delete tracks which have not been detected in M_{del} out of N iterations. As it is obvious, such techniques do not allow for the direct incorporation of clutter and detection models, such as the ones discussed in Sections 2.5 and 2.4, meaning that pre-processing of the sensor data (e.g. using clutter maps) is compulsory for such techniques to be applied in scenarios involving dense clutter.

Log Likelihood Ratio

Another commonly used technique is a score-based approach, known as the *Log Likelihood Ratio Test* [1, 78]. According to this method, each track is assigned a log likelihood ratio (LLR) score defined as

$$LLR = \log \frac{p(\mathrm{y}|H_1)}{p(\mathrm{y}|H_0)} = \log \frac{P_T}{P_F} \tag{4.10}$$

where H_1, H_0 denote the hypotheses that the track corresponds to a true or false target, with respective probabilities P_T and P_F, and $p(\mathrm{y}|H_i)$ is the probability density function evaluated with the associated measurement y, under the assumption that H_i is true. Upon initiation, each track is assigned an initial score and, on consecutive iterations, the score of each track is increased, or decreased, depending on the computed likelihood of its associated measurements as

$$LLR_k = LLR_{k-1} + \Delta L \tag{4.11}$$

where

$$\Delta L = \log \frac{p(\mathrm{y}_\mathrm{k}^\mathrm{j}|H_1)}{p(\mathrm{y}_\mathrm{k}^\mathrm{j}|H_0)} \tag{4.12}$$

Using similar notation to (4.23), an expression for ΔL can be obtained as follows:

$$\Delta L = \log \begin{cases} (1 - p_d(\mathrm{x}_k)P_G)p_{M^c}(M_k) \displaystyle\prod_{i\in[1,\dots,M_k]} p_C(\mathrm{y}_k^i), & j = 0 \\ \dfrac{p_d(\mathrm{x}_k)p(\mathrm{y}_k^j|\mathrm{x}_k)p_{M^c}(M_k - 1)}{M_k} \displaystyle\prod_{i\in[1,\dots,M_k]\backslash j} p_C(\mathrm{y}_k^i), & j \neq 0 \end{cases} \tag{4.13}$$

In the case of Poisson distributed clutter with mean λ, and uniform probability of detection P_D, (4.13) can be simplified using a similar process and notation to the derivation of (4.24) as follows:

$$\Delta L = \log \begin{cases} (1 - P_D P_G), & j = 0 \\ \dfrac{P_D p(\mathrm{y}_k^j|\mathrm{x}_k)}{\lambda}, & j \neq 0 \end{cases} \tag{4.14}$$

Then, carefully selected high and low thresholds (T_{high}, T_{low}) are used as conditions for confirming and deleting tentative tracks, while a deviation (THD) from the maximum LLR is used for the deletion of confirmed tracks [1]. Following the standard LLR definition, these thresholds are defined as follows:

$$T_{low} = \log \frac{\beta}{1-\alpha}, \quad T_{high} = \log \frac{1-\beta}{\alpha} \tag{4.15}$$

where α is the false track confirmation probability and β the true track deletion probability. As stated in [1], α can be defined from the system requirements on false track initiation. For example, if the system generates λ_{FA} false alarms per scan and only N_{FC} false track confirmations are allowed within N_S scans, then one can set

$$\alpha = \frac{\lambda_{FA}}{N_S N_{FC}} \tag{4.16}$$

A typical choice of β is $\beta \leq 0.1$. The choice of β is less important since its effect on the track confirmation threshold is much smaller. The incorporation of probability concepts within this approach makes it possible for dynamic and clutter models to be incorporated within the process, although it can be highly a challenging task [79].

One main disadvantage of both above described approaches, lies in the fact that tentative (unconfirmed) tracks are treated in very much the same way as normal tracks. In most cases, a tentative track is initiated for each unassociated measurement, which, in highly cluttered environments, bears the threat of devoting the majority of computational resources to tracking spurious tracks. Recent research [80] has demonstrated the use of the notion of existence probability [81], logarithmically proportional to LLR, and a modified PF search track, which circumvents the above problems, but has the limitation that only one track can be initiated at any time. Further work by the same authors [82], presents an even more appealing, solution by substituting the search track with a PHD filter and deriving a simple rule for extracting multiple true tracks from the PHD, at any given time. A similar approach to the one presented in [82] was also developed in parallel in [83].

4.3.3 Data association

4.3.3.1 Problem Formulation

Once standard gating procedures have been performed, the set of valid association hypotheses between a given track and all of its candidate measurements, can be conveniently expressed in terms of a validation vector Ω_k^i, composed of M_k binary elements:

$$\Omega_k^i = [\omega_k^{i,1}, ...\omega_k^{i,M_k}] \tag{4.17}$$

where each $\omega_k^{i,j}$ for $j > 0$ serves as an indicator for the event of measurement y_k^j falling inside the gate of i-th target

$$\omega_k^{i,j} = \begin{cases} 1, & \text{if } \mathrm{y}_k^j \text{ gated} \\ 0, & \text{otherwise} \end{cases} \tag{4.18}$$

Finally, the joint validation matrix Ω_k is established as the vertical concatenation of all vectors Ω_k^i, and lists all valid measurement-to-track associations. As such, Ω_k can be defined as follows:

$$\Omega_k = \begin{bmatrix} \Omega_k^1 \\ \vdots \\ \Omega_k^{N_k} \end{bmatrix} \tag{4.19}$$

For the purposes of this section, unless stated otherwise, the assumption is posed that the number of targets N_k is known a priori and remains constant throughout the data association (i.e. no target births/deaths take place). Let α_k^i be a random variable representing the measurement associated with track i, or 0 if the track is not detected at time t_k

$$\alpha_k^i = \begin{cases} j, & \text{if measurement } \mathrm{y}_k^j \text{ associated with track } i \\ 0, & \text{otherwise} \end{cases} \tag{4.20}$$

Thus, for each track, α_k^i can take values in $j = 1, \ldots, M_k$, for which $\omega_k^{i,j} = 1$, while $\alpha_k^i = 0$ is used to denote the missed detection event.

Similarly, let $d_k^i = [\alpha_k^i \neq 0]$ be a random variable indicating if target i is detected at time t_k

$$d_k^i = \begin{cases} 1, & \text{if target } i \text{ was detected} \\ 0, & \text{otherwise} \end{cases} \tag{4.21}$$

In this context, the notation D_k^i shall be used to denote the event where $d_k^i = 1$, while $\bar{D}_k^i$ will denote the opposite.

Based on the above, a series of exhaustive and mutually exclusive single-target association hypotheses $\theta_k^{i,j}$ can be defined, which consider the events that a target

i is associated with a given measurement j, i.e.

$$\theta_k^{i,j} \triangleq \left[\alpha_k^i = j\right] \tag{4.22}$$

where the Iverson notation is used, as defined in (3.36).

For each of the defined single-target hypotheses, the prior association probability $p(\theta_k^{i,j}|Y_{1:k-1})$ can be computed as:

$$p(\theta_k^{i,j}|Y_{1:k-1}) \propto \begin{cases} (1-p_d(\mathrm{x}_k^i)P_G)p_{M^c}(M_k) \displaystyle\prod_{j'\in[1,\dots,M_k]} p_C(\mathrm{y}_k^{j'}), & j=0 \\ \dfrac{p_d(\mathrm{x}_k^i)p(\mathrm{y}_k^j|\mathrm{x}_k)p_{M^c}(M_k-1)}{M_k} \displaystyle\prod_{j'\in[1,\dots,M_k]\setminus j} p_C(\mathrm{y}_k^{j'}), & j=1,\dots,M_k \end{cases} \tag{4.23}$$

where, under the assumptions of constant detection probability $p_d(x_k^i) = P_D$ and Poisson clutter with mean λ, it can be proven in a similar manner to (3.45) that (4.23) simplifies to

$$p(\theta_k^{i,j}|Y_{1:k-1}) \propto \begin{cases} (1-P_D P_G), & j=0 \\ \dfrac{P_D p(\mathrm{y}_k^j|\mathrm{x}_k^i)}{\lambda}, & j=1,\dots,M_k \end{cases} \tag{4.24}$$

Let Θ_k^n be a feasible joint association hypothesis, defined as a realisation of the set of valid track association hypotheses $\theta_k^{i,j}$, for which the following conditions must hold:

1. Each measurement y_k^j can be assigned to at most one track, i.e,

$$\sum_{i=1}^{N_k} \theta_k^{i,j} \leq 1, \quad \forall \theta_k^{i,j} \in \Theta_k^z \tag{4.25}$$

2. Each track is associated to exactly one measurement, including the missed detection, i.e.,

$$\sum_{j=1}^{M_k} \theta_k^{i,j} = 1, \quad \forall \theta_k^{i,j} \in \Theta_k^z \tag{4.26}$$

In other words, a feasible association hypothesis Θ_k^n can be viewed as a valid combination of the unit elements in the validation matrix Ω_k, which satisfy the conditions of (4.25)-(4.26). Considering this fact, it becomes obvious that for any given Ω_k, there can exist N_k^Θ possible joint hypotheses Θ_k^n, where $n \in [1, N_k^\Theta]$.

Thus, the set Θ_k of all valid association hypotheses can thus be defined as follows:

$$\Theta_k = [\Theta_k^1, \Theta_k^2, ...\Theta_k^{N_k^{\Theta}}] \tag{4.27}$$

The probability for a valid association hypothesis Θ_k^n to occur is proportional the joint probability over all the marginal association events that pertain to that hypothesis, i.e.,

$$p(\Theta_k^n|Y_{1:k}) \propto \prod_{\theta_k^{i,j} \in \Theta_k^n} p(\theta_k^{i,j}|Y_{1:k}) \tag{4.28}$$

For a given joint association Θ_k^n, let $\mathcal{T}_D$ be the set of tracks hypothesised as been successfully detected and let $\mathcal{T}_{ND}$ denote the set of targets that have been missed. It is then possible to rewrite (4.28) as

$$p(\Theta_k^n|Y_{1:k}) \propto \prod_{i \in \mathcal{T_{ND}}} p(\theta_k^{i,0}|Y_{1:k-1}) \prod_{i \in \mathcal{T_D}} p(\theta_k^{i,j(i)}|Y_{1:k-1}) \tag{4.29}$$

which, under the assumptions of (4.24), leads to

$$p(\Theta_k^n|Y_{1:k}) \propto \prod_{i \in \mathcal{T_{ND}}} (1 - P_D P_G) \prod_{i \in \mathcal{T_D}} \frac{P_D p(\mathrm{y}_k^{j(i)}|\mathrm{x}_k^i)}{\lambda} \tag{4.30}$$

where $j(i)$ is used to denote the index of the measurement assigned to target i, under the joint association Θ_k^n.

Finally, the posterior association probabilities $p(\theta_k^{i,j}|Y_{1:k})$ can be computed by applying the law of total probability over the entire set of joint association hypotheses, where a given association event $\theta_k^{i,j}$ is valid, i.e.

$$p(\theta_k^{i,j}|Y_{1:k}) = \frac{1}{C} \sum_{\Theta_k^n \in \Theta_k:\ \theta_k^{i,j} \in \Theta_k^n} p(\Theta_k^n|Y_{1:k}) \tag{4.31}$$

where C is the total probability normalisation constant

$$C = \sum_{\Theta_k^n \in \Theta_k} p(\Theta_k^n|Y_{1:k}) \tag{4.32}$$

The optimal goal for data association is then to compute the conditional association probabilities $p(\theta_k^{i,j}|Y_{1:k})$ for all feasible events $\theta_k^{i,j}$, which can be utilised to update the posterior state distributions $p(\mathrm{x}_k^i|Y_k)$ of all targets, where

$$p(\mathrm{x}_k^i|Y_k) = \sum_{\theta_k^{i,j}} p(\mathrm{x}_k^i|\theta_k^{i,j}, \mathrm{y}_k^j, Y_{k-1}) p(\theta_k^{i,j}|Y_{1:k}) \tag{4.33}$$

The term $p(\mathrm{x}_k^i|\theta_k^{i,j}, \mathrm{y}_k^j, Y_{k-1})$ is equivalent to the standard Bayesian Filtering posterior $p(\mathrm{x}_k^i|\mathrm{y}_{1:k})$, albeit expressed using different notation. Thus, the standard Bayesian Filtering equations (3.2)-(3.4) can be employed to evaluate these densities independently for each association event $\theta_k^{i,j}$.

4.3.3.2 Global Nearest Neighbour

The Global Nearest Neighbour (GNN) algorithm forms an extension to the Nearest Neighbour approach. In Section 3.3.2 it was shown that the standard NN algorithm aims to identify the most likely single-target hypothesis $\hat{\theta}_k^j$, which is then utilised to update the track. GNN extends the same concept to the multi-target case, by identifying the most likely joint association hypothesis $\hat{\Theta}_k^n$, and then using the marginal association hypotheses $\theta_k^{i,j} \in \hat{\Theta}_k^n$ to update all existing tracks.

The GNN algorithm circumvents the necessity of evaluating all valid joint association hypotheses by posing the problem of finding the most likely association hypothesis as a constrained optimisation problem [25]. In this context, a problem instance is described by a cost matrix of the form:

$$\mathcal{C} = \begin{bmatrix} \mathcal{C}_{1,0} & \dots & \mathcal{C}_{1,M_k} \\ \vdots & \ddots & \vdots \\ \mathcal{C}_{N_k,0} & \dots & \mathcal{C}_{N_k,M_k} \end{bmatrix} \tag{4.34}$$

Each element $\mathcal{C}_{i,j}$ corresponds to the cost of associating track i with measurement j, where $j = 0$ is used to denote the missed detection. In other words, the costs can be computed by means of a function defined on the set of single-target association hypotheses, i.e.

$$\mathcal{C}_{i,j} = f_c(\theta_k^{i,j}) \tag{4.35}$$

Then, a solution to the data association problem can be obtained by means of identifying the complete assignment of tracks to measurements that give rise to minimal cost. Formally, let X be a boolean matrix where $X_{i,j} = 1$ if row i is assigned to column j. Then the optimal assignment has cost

$$\min \sum_i \sum_j \mathcal{C}_{i,j} X_{i,j} \tag{4.36}$$

such that each row is assigned to at most one column, and each column to at most one row, with the exception of column $j = 0$.

The above generalisation makes it possible for (4.36) to be computed by solving a linear sum assignment problem, also known as minimum weight matching in bipartite graphs. Linear sum assignment algorithms have been an active field of research over the last century, which has given rise to popular methods such as the Kuhn-Munkres [84, 85] (aka. Hungarian), Auction [86] or JVC [87] algorithms, that are well-studied and known to provide significant gains in terms computational complexity.

Similar to the case of its single-target equivalent, there generaly exist two formulations of the GNN algorithm: i) probabilistic; and ii) distance-based. The distinction between the two formulation is drawn on the basis of how each approach defines the cost function $f_c(\theta_k^{i,j})$ of (4.35). In the probabilistic case, the cost is defined as a function of the prior association probabilities $p(\theta_k^{i,j}|Y_{1:k-1})$ as

$$f_c(\theta_k^{i,j}) = -\log p(\theta_k^{i,j}|Y_{1:k-1}) \tag{4.37}$$

The distance-based formulation of GNN [56] is realised by defining a distance measure $\mathcal{D}(\mathrm{x}_k^i, \mathrm{y}_k^j)$ between the target state and a given measurement, such that

$$f_c(\theta_k^{i,j}) = \mathcal{D}(\mathrm{x}_k^i, \mathrm{y}_k^j) \tag{4.38}$$

Commonly used distance measures include the Mahalanobis or Euclidian distances, but other measures can also be employed.

Once the optimal joint association hypothesis $\hat{\Theta}_k^n$ has been determined, it's member association hypotheses are used to update each track. Thus, only a single measurement hypothesis is assumed to hold for each track, effectively reducing the update process to a standard Bayesian update step

$$p(\mathrm{x}_k^i|Y_{1:k}) = p(\mathrm{x}_k^i|\theta_k^{i,j} \in \hat{\Theta}_k^n, \mathrm{y}_k^j, Y_{1:k-1}) \tag{4.39}$$

Despite the computational efficiency of the GNN algorithm, it constitutes a myopic solution to the data association problem, as it only selects and utilises a single association hypothesis, which is assumed to be optimal. Studies have shown that this approach suffers from tracking reliability issues, even in environments with relatively low clutter densities [26, 88, 89].

4.3.3.3 Joint Probabilistic Data Association

The Joint Probabilistic Data Association Filter [58] (JPDAF) is an extension of the single-target PDA of Section 3.3.3, which provides a soft decision mechanism for updating the target state distributions $p(\mathrm{x}_k^i|Y_k)$. In JPDAF, all valid association events $\theta_k^{i,j}$ contribute to the update process proportionately to their respective conditional association probabilities $p(\theta_k^{i,j}|Y_{1:k})$, giving rise to the posterior density formulation (4.33).

Enumerating the joint hypotheses

The bottleneck of JPDAF lies in the enumeration and management of all hypotheses Θ_k^n that satisfy the validity conditions of (4.25) and (4.26). This can be achieved by enumerating all the possible realisations of the validation matrix of (4.19) that satisfy the assumptions of (4.25) and (4.26). However, being closely related to the calculation of a matrix permanent, the time complexity of this approach grows exponentially with the number of targets and measurements. Thus, application of this method becomes prohibitive in situations that involve a high number of tracks and/or measurements.

Another common approach is to build a tree of associations possible for each track. The nodes in the top-most (first) layer in the tree represent the valid set of measurement assignments for the first target. Similarly, the second layer of nodes represent the valid measurement assignments for the second target, given the assignments for the first target. Accordingly, the i-th layer of nodes represents the valid measurement assignments for the i-th target, given the assignments, up to and including the $(i-1)$-th target. So, each node stores the information of which measurements have been used by all layers down to the layer of the target being analysed. In order to construct the tree, the targets are processed sequentially and, for each node of the previous layer, a new node for the current layer is created for each measurement that does not result in any violation of the constraints. From the tree of associations, the set of feasible joint association hypotheses is enumerated by starting from the root and, considering each target (layer) in turn while navigating the tree towards the leaves. Computing the association probabilities follows naturally from the structure of the tree, by considering that each path down the tree corresponds to one feasible joint association hypothesis Θ_k^n, where the edges along the path represent the marginal association hypotheses $\theta_k^{i,j}$ that pertain to Θ_k^n. This fact allows for the direct application of the relations in (4.28)-(4.32), while parsing each path, starting from the root node.

Although the tree-based approach provides significant computational advantages, the method suffers from repetitions of identical computations, stemming from the existence of morphological redundancies in the generated tree structure, due to the existence of joint association hypotheses that share common association events. To better understand the above, consider an example validation matrix

$$\Omega_k = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \tag{4.40}$$

defined according to (4.18)-(4.19). Based on Ω_k, it can be derived that there exist 3 tracks (rows) and 3 measurements (columns), where the first measurement is gated by the first and second track, the second measurement is gated by the second and third tracks and the third measurement is on only gated by the third track. For the given problem, there exist $N_k^{\Theta} = 13$ possible joint association hypotheses that satisfy the conditions of (4.25)-(4.26).

Application of the tree-based approach on the validation matrix of (4.40) yields to the generation of the hypothesis tree depicted in Figure 4.2. Each node is labelled with the identifier of the track-layer to which it relates, edge labels denote the index $\{i, j\}$ of the measurement hypothesis $\theta_k^{i,j}$ that was considered to generate each child-node of the i-th layer and leaf-nodes are also labelled with the identifier of the joint association hypothesis corresponding to each path in the tree. The redundancies inherent to the tree-based approach can be seen in Figure 4.2, where many subtrees are systematically repeated to account for the same remaining association possibilities, but for different associations already made. This is clear when we notice that, for instance, the trees departing from the layer of nodes for $i = 2$ (second layer), are essentially copies of 2 sets of identical sub-trees.

Efficient Hypothesis Management

It is in fact possible to circumvent the morphological redundancies in the tree-based approach, while still ensuring an exact solution to the problem is obtained. An efficient method that accomplishes the above is the Efficient Hypothesis Management (EHM) algorithm, presented in [90] and further extended in [91]. The fundamental idea behind EHM is to simplify the enumeration and weight calculation processes, by eliminating redundant computations related to hypotheses which share common association events.

In order to eliminate the association redundancies, given the nodes for the i-th layer, the EHM algorithm enumerates the valid and unambiguous associations that are possible for the remaining $N_k - i$ layers. This is different to the standard tree approach that enumerates the set of all valid associations for the

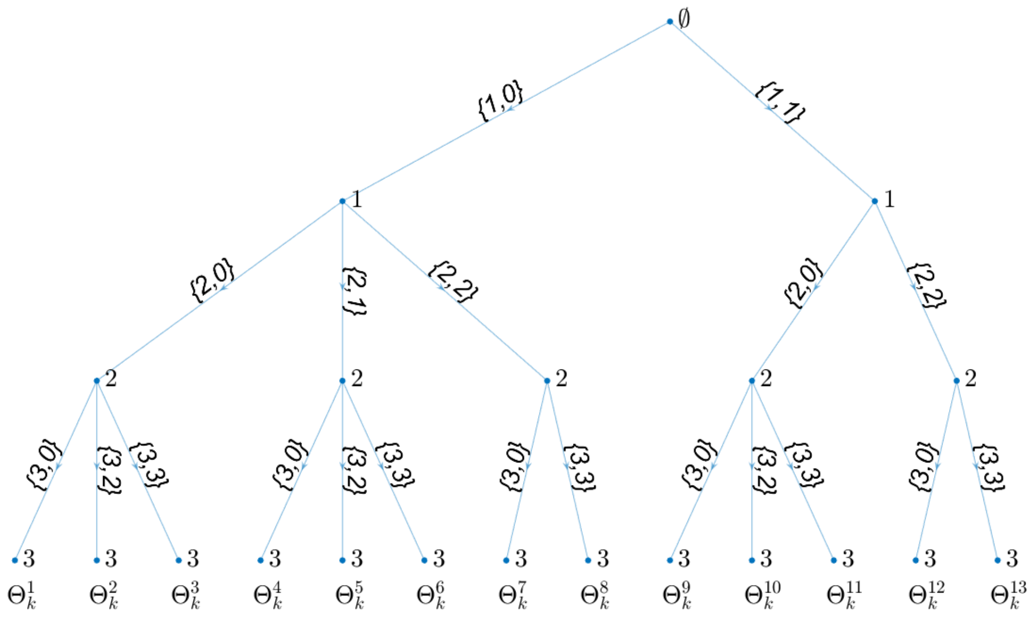


Figure 4.2: Hypothesis tree generated by applying the tree-based hypothesis enumeration approach to the example validation matrix (4.40).

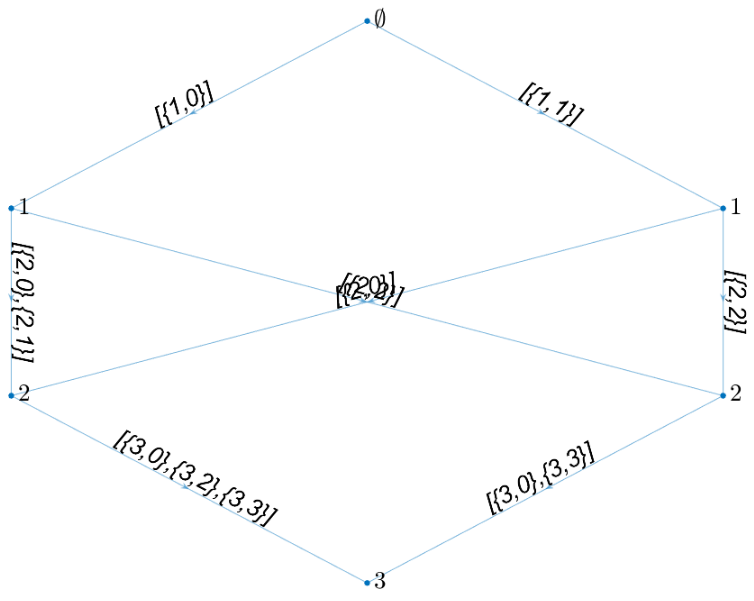


Figure 4.3: Hypothesis net generated by applying EHM to the example validation matrix (4.40).

$(i+1)$-th layer, for each of the nodes generated up to the i-th layer, which renders exponential grow of the number of nodes as a function of the number of measurements and targets. The resulting structure obtained by applying EHM is a network that, during its construction, merges the nodes that share the same subset of possible association events for the $N_k - i$ remaining tracks. The computational saving becomes clear when the computation of the marginal probabilities is explicitly formulated as a sequence of operations over the association nodes. Ultimately, the complexity for computing the marginal probabilities is directly dependent on the number of nodes (#P-complete), which in turn depend on the number of measurements and targets.

Computing the association probabilites in the case of EHM is slightly more intricate. The intent is to perform a sum over all descents of the net that include a given element equal to the unity in the validation matrix Ω_k. All descents must pass through one of the nodes of the i-th net layer of the network. Thus, the sum over the joint feasible events can be represented as a sum over the nodes according to

$$\beta_k^{i,j} = p(\theta_k^{i,j}|Y_{1:k}) \propto \sum_{n_i=1}^{N_{n_i}} p_T(j|n_i) \tag{4.41}$$

where $p_T(j|n_i)$ are the joint likelihoods for descents that include an edge representing the association event $\theta_k^{i,j}$ and go through the n_i-th node of the $i-th$ track layer. N_{n_i} is the total number of nodes for the j-th layer.

The joint likelihood $p_T(j|n_i)$ is a quantity calculated for each of the nodes in the network. For each node n_i , it can be effciently calculated as a product of three terms:

- a term $p_{D,T}(i, n_i)$ related to the descents from the node's parents that have the measurement j incorporated in the parent-child relationship
- a term $p(\theta_k^{i,j}|Y_{1:k-1})$ related to the prior association probability for each single association hypothesis
- a term $p_U(n_i)$ related to the possible paths for the remaining $N_k - i$ tracks

Based on the above information, the joint likelihoods $p_T(j|n_i)$ can be computed by the following equations

$$p_T(j|n_i) = p_U(n_i)p(\theta_k^{i,j}|Y_{1:k-1})p_{D,T}(i, n_i) \tag{4.42}$$

$$p_{D,T}(i, n_i) = \sum_{n_{i-1}\in\mathcal{P}(n_i),\{i-1,j\}\in\mathcal{E}_{n_{i-1},n_i}^{i-1,i}} p_D(n_i - 1) \tag{4.43}$$

where $\mathcal{P}(n_i)$ denotes the set of parents of a node n_i, while $\mathcal{E}^{i-1,i}_{n_{i-1},n_i}$ denotes the edges that connect the parent node n_{i-1} of the $(i-1)$-th layer, to the child node n_i of the i-th layer.

Continuing, the terms $p_D(n_i)$ and $p_U(n_i)$ are then calculated by application of the forward-backward algorithm, performed as a sequential application of a forward (downward) and backward (upward) pass through the net layers as:

$$p_D(n_i) = \sum_{n_{i-1}\in\mathcal{P}(n_i),\{i-1,j\}\in\mathcal{E}^{i-1,i}_{n_{i-1},n_i}} p(\theta^{i,j}_k|Y_{1:k-1})p_D(n_{i-1}) \tag{4.44}$$

$$p_U(n_i) = \sum_{n_{i+1}\in\mathcal{C}(n_i),\{i,j\}\in\mathcal{E}^{i,i+1}_{n_i,n_{i+1}}} p(\theta^{i+1,j}_k|Y_{1:k-1})p_U(n_{i+1}) \tag{4.45}$$

where $\mathcal{C}(n_i)$ denotes the set of children of a node n_i and an expression for the prior association probabilities $p(\theta^{i,j}_k|Y_{1:k-1})$ has been obtained in (4.23)-(4.24). It is worth noting here that the same node may appear more than once in the sets of children and parent of nodes indexed by some n_i.

The forward pass is performed first, which involves the calculation of the downward probabilities $p_D(.)$, followed by the backward pass to calculate the upward probabilities $p_U(.)$. These probabilities are combined with the prior association probabilities $p(\theta^{i,j}_k|Y_{1:k-1})$ of each single association to give the sum of probabilities of all descents through the net for each single association event $\theta^{i,j}_k$. Making use of this information, it is now possible to compute the posterior association probabilities of (4.41).

Figure 4.3 depicts the hypothesis net generated by applying EHM on the validation matrix of (4.40). Each node is labelled with the identifier of the track-layer to which it related. Edge labels denote the set of indices $\{i,j\}$ of the measurement hypotheses $\theta^{i,j}_k$ that were considered to generate each node in the i-th layer. By comparing the net of Figure 4.3 to the tree of Figure 4.2, it should be clear to see the effects of the node redundancy introduced by EHM. The number of nodes in the original tree amount to 20, while the nodes in the EHM net have been effectively reduced to 6.

4.4 Joint Integrated Probabilistic Data Association

The JPDA algorithm provides an effective solution the problem of multi-target data association under the presence of clutter and missed detection. However, JPDA is defined under the assumption that the number of targets is known apriori and remains constant throughout the data association process [58]. Furthermore, the entire set of JPDA equations are derived with the implicit condition on target existence, meaning that any information about the existence of targets is effectively removed.

As a direct consequence to the above, JPDA necessitates the use separate track management methods that must run along the algorithm to deterministically initiate, confirm and delete tracks, between iterations. As already discussed in Section 4.3.2, such track management approaches typically rely on heuristic methods (e.g. *M-out-of-N*) that pose certain assumptions that are not guaranteed to hold in a real-life scenario, or require the definition of certain thresholds for confirmation and deletion that are inherently hard to define and tune (e.g. *LLR Test*).

The Integrated PDA (IPDA) algorithm has been derived and formulated in [92, 93], which removes the assumption of target existence and provides data association formulae that expose the equations required to recursively compute the probability of target existence in a single-target tracking context. Continuing, the same authors extended the equations of IPDA to a multi-target tracking context, giving rise to the Joint IPDA (JIPDA) algorithm [94]. The JIPDA algorithm, is developed in a similar fashion to the IPDA algorithm. It draws a clear differentiation between the probability of existence calculation, and the evaluation of the association probabilities, thus resulting in recursive expressions for the probability of target existence and data association coefficients, over the set of joint association hypotheses. The inclusion of the existence probabilities in JIPDA allows for track confirmation and deletion to be incorporated within the main algorithm recursion, by setting a threshold on the existence quantities computed for each track.

Since its initial formulation, JIPDA has been successfully applied to solve a number of problems [95, 80, 82], albeit [80, 82] using an alternative formulation of the algorithm, derived in [81]. Despite the differences in notation between [94] and [81], both approaches eventually conclude with the same set of equations for updating the target posteriors and existence probabilities as:

$$p(\mathrm{x}_k^i|E_k^i, Y_{1:k}) = \sum_{j=0}^{M_k} \beta_{i,j} p(\mathrm{x}_k^i|\theta_k^{i,j}, \mathrm{y}_k^j, E_k, Y_{1:k-1}) \tag{4.46}$$

$$\beta_{i,j} = p(\theta_k^{i,j}|E_k^i, Y_{1:k}) = \frac{p(\theta_k^{i,j}, E_k^i|Y_{1:k})}{p(E_k^i|Y_{1:k})} \tag{4.47}$$

$$p(E_k^i|Y_{1:k}) = \sum_{j=0}^{M_k} p(\theta_k^{i,j}, E_k^i|Y_{1:k}) \tag{4.48}$$

where a similar notation to [81] is used, although the substitution $\theta_k^{i,j} \triangleq [\alpha_k^i = j]$ from (4.22) is applied to maintain notational consistency with the rest of the thesis. The relation of (4.47) is equivalent to (10)-(11) of [94], while (4.48) aligns with (9) of [94]. A clear expression equivalent to (4.46) is not presented in [94], although the statement is made that the quantities of (4.47) can be used to perform track estimation in the same way as JPDA, which is exactly what (4.46) aims to convey.

4.4.1 Relation between JPDA and JIPDA

As shown above, [94] and [81] eventually advocate the same JIPDA recursion. However, both manuscripts seem to separate the data association process, or more specifically the process of calculating the joint association probabilities, performed by JIPDA and JPDA. For example, the authors in [94] emphasise on the fact that the expressions for the joint association events in JIPDA incorporate the probabilities of track existence of individual tracks, in contrast to JPDA. It is possible that this distinction is not intentional, but rather stems from the approach followed by the different authors to solve the problem of obtaining an expression for the quantities of (4.46)-(4.48).

In this section, the author aims to show that the very same approach used by JPDA to compute the association probabilities of (4.31), which are conditional on target existence, can be re-used in the context of JIPDA, yielding exact solutions to (4.46)-(4.48). By proving that the above holds true, it is the possible to apply the same performance optimisation methods introduced in Section 4.3.3.3 for JPDA (e.g. EHM), to speed up the recursion of JIPDA.

4.4.2 Prior Association Probabilities

Using similar notation to before, equations (32)-(34) of [80], which draws its notation directly from [81], used to compute the prior association probabilities, can be re-written as :

$$\begin{aligned}
\bar{\beta}_{i,0,0} &= p(\theta_k^{i,0}, \bar{E}_k^i|Y_{1:k-1}) = \underbrace{p(\bar{D}_k^i|\bar{E}_k^i, Y_{1:k-1})}_{=1} p(\bar{E}_k^i|Y_{1:k-1}) = p(\bar{E}_k^i|Y_{1:k-1}) \\
&= (1 - p(E_k^i|Y_{1:k-1})) && (4.49) \\
\bar{\beta}_{i,0,1} &= p(\theta_k^{i,0}, E_k^i|Y_{1:k-1}) = \underbrace{p(\bar{D}_k^i|E_k^i, Y_{1:k-1})}_{=1-p_d(\mathrm{x}_k^i)P_G} p(E_k^i|Y_{1:k-1}) \\
&= (1 - p_d(\mathrm{x}_k^i)P_G)p(E_k^i|Y_{1:k-1}) && (4.50) \\
\bar{\beta}_{i,j,1} &= p(\theta_k^{i,j}, E_k^i|Y_{1:k-1}) = p(\theta_k^{i,j}|D_k^i, E_k^i, Y_{1:k-1}) \underbrace{p(D_k^i|E_k^i, Y_{1:k-1})}_{=p_d(\mathrm{x}_k^i)P_G} p(E_k^i|Y_{1:k-1}) \\
&= \frac{p(\mathrm{y}_k^j|\mathrm{x}_k^i)p_d(\mathrm{x}_k^i)P_G}{\lambda} p(E_k^i|Y_{1:k-1}) && (4.51) \\
\bar{\beta}_{i,j,0} &= p(\theta_k^{i,j}, \bar{E}_k^i|Y_{1:k-1}) = 0 && (4.52)
\end{aligned}$$

The above equations define a three-dimensional (3-D) table $\bar{\beta}$, which represents the joint association likelihood $p(\theta_k^{i,j}, e_k^i|Y_{1:k-1})$ over the association $\theta_k^{i,j}$ and existence variables e_k^i, i.e:

$$\bar{\beta} \triangleq p(\theta_k^{i,j}, e_k^i|Y_{1:k-1}) \tag{4.53}$$

4.4.3 Marginalising the Prior Association Probabilities

By recalling the relation of (4.23), it can be observed that JPDA performs its operation on the prior association probabilities of the form $p(\theta_k^{i,j}|Y_{k-1})$. Thus, making use of (4.53), we can define the likelihood matrix $\mathcal{L}$, which serves as an input to the joint hypothesis enumeration process of JPDA (e.g. EHM), by marginalising over the existence variable e_k^i in (4.53), i.e.:

$$\begin{aligned}
\mathcal{L} &\triangleq p(\theta_k^{i,j}|Y_{1:k-1}) = \sum_{e_k^i} p(\theta_k^{i,j}, e_k^i|Y_{1:k-1}) \\
&= \begin{bmatrix} \mathcal{L}_{1,0} & \dots & \mathcal{L}_{1,M_k} \\ \vdots & \ddots & \vdots \\ \mathcal{L}_{N_k,0} & \dots & \mathcal{L}_{N_k,M_k} \end{bmatrix} && (4.54)
\end{aligned}$$

where each matrix element $\mathcal{L}_{i,j}$ is defined as:

$$\begin{aligned}\mathcal{L}_{i,j} &= \begin{cases}\bar{\beta}_{i,0,0}+\bar{\beta}_{i,0,1} & j=0\\ \bar{\beta}_{i,j,0}+\underbrace{\bar{\beta}_{i,j,1}}_{=0} & j\neq 0\end{cases}\\ \overset{(4.49)-(4.52)}{\Rightarrow}\quad \mathcal{L}_{i,j} &= \begin{cases}(1-p_d(\mathrm{x}_k^i)P_G p(E_k^i|Y_{1:k-1})) & j=0\\ \dfrac{p(\mathrm{y}_k^j|\mathrm{x}_k^i)p_d(\mathrm{x}_k^i)P_G}{\lambda}p(E_k^i|Y_{1:k-1}) & j\in[1,\dots,M_k]\end{cases}\end{aligned} \tag{4.55}$$

It is worth noting here that, even after marginalising, all columns in $\mathcal{L}$, except the first, are equivalent to the joint likelihoods $\bar{\beta}_{i,j,1}$, since from (4.52) we know that $\bar{\beta}_{i,j,0}=0$. On the contrary, the first column of $\mathcal{L}$ represents the marginal probability of the target not being detected. This is important since, as we show in the following sections, it will have an effect on how we compute both the existence probabilities $p(E_k^i|Y_{1:k})$ and posterior distributions $p(\mathrm{x}_k^i|E_k^i,Y_{1:k})$.

4.4.4 Interpreting the Posterior Association Probabilities

Once provided with the table of marginal prior association probabilities, defined by (4.54), the produced output is a matrix $\hat{\beta}$ of posterior association probabilities

$$\hat{\beta}=\begin{bmatrix}\hat{\beta}_{1,0} & \hat{\beta}_{1,1} & \dots & \hat{\beta}_{1,M_k}\\ \vdots & \vdots & \ddots & \vdots\\ \hat{\beta}_{N_k,0} & \dots & \dots & \hat{\beta}_{N_k,M_k}\end{bmatrix} \tag{4.56}$$

where $\hat{\beta}$ is normalised over each row, and each element $\hat{\beta}_{i,j}$ is still marginalised over the existence variable e_k^i

$$\begin{aligned}\hat{\beta}_{i,j} &= p(\theta_k^{i,j}|Y_{1:k})\\ &= p(\theta_k^{i,j},E_k^i|Y_{1:k})+p(\theta_k^{i,j},\bar{E}_k^i|Y_{1:k})\end{aligned} \tag{4.57}$$

In a similar fashion to (4.52), by considering that the association probability of a non-existing target being associated to a measurement is zero, i.e. $p(\theta_k^{i,j},\bar{E}_k^i|Y_{1:k})=0$ for all $j\neq 0$, we can re-write (4.57) as:

$$\hat{\beta}_{i,j}=\begin{cases}p(\theta_k^{i,0},E_k^i|Y_{1:k})+p(\theta_k^{i,0},\bar{E}_k^i|Y_{1:k}) & j=0\\ p(\theta_k^{i,j},E_k^i|Y_{1:k}) & j\in[1,\dots,M_k]\end{cases} \tag{4.58}$$

4.4.5 Computing the Existence Probabilities

From [80, 81], with slight abuse of notation, we can write that:

$$p(E_k^i|Y_{1:k}) = \sum_{j=0}^{M_k} p(\theta_k^{i,j}, E_k^i|Y_{1:k}) \tag{4.59}$$

Thus, it becomes clear that, in order to compute the association probabilities, it is necessary to calculate the respective sum components $p(\theta_k^{i,j}, E_k^i|Y_{1:k})$. Given that we have obtained the output matrix $\hat{\beta}$, this can be achieved by solving (4.57) as such:

$$p(\theta_k^{i,j}, E_k^i|Y_{1:k}) = \begin{cases} \hat{\beta}_{i,0} - p(\theta_k^{i,0}, \bar{E}_k^i|Y_{1:k}) & j = 0 \\ \hat{\beta}_{i,j} & j \in [1, \dots, M_k] \end{cases} \tag{4.60}$$

Therefore, for all $j \in [1, \dots, M_k]$, the respective sum components of (4.48) are equal to the respective association probabilities $\hat{\beta}_{i,j}$. However, the same cannot been done for $j = 0$, as we need to first calculate the joint $p(\theta_k^{i,0}, \bar{E}_k^i|Y_{1:k})$.

A first step towards computing $p(\theta_k^{i,0}, \bar{E}_k^i|Y_{1:k})$ can be achieved by noting that:

$$\frac{p(\theta_k^{i,0}, \bar{E}_k^i|Y_{1:k})}{p(\theta_k^{i,0}, E_k^i|Y_{1:k})} = \frac{\bar{\beta}_{i,0,0}}{\bar{\beta}_{i,0,1}} = \frac{1 - p(E_k^i|Y_{1:k-1})}{(1 - p_d(\mathrm{x}_k^i)P_G)p(E_k^i|Y_{1:k-1})} = w_k^i \tag{4.61}$$

where $\bar{\beta}_{i,0,0}$ and $\bar{\beta}_{i,0,1}$ have been computed in (4.49) and (4.50), respectively. The above can be understood better by considering that, as explained in Section 4.4.3, JPDA operates on the combined likelihoods, marginalised over the existence variable e_k^i. This means that the data association process preserves the ratio of the joint likelihoods/probabilities, conditional on the target not being detected.

Having explained the validity of relation (4.61), we can obtain an expression for $p(\theta_k^{i,0}, \bar{E}_k^i|Y_{1:k})$ as follows:

$$p(\theta_k^{i,0}, \bar{E}_k^i|Y_{1:k}) = w_k^i \times p(\theta_k^{i,0}, E_k^i|Y_{1:k}) \tag{4.62}$$

and substituting the above in (4.60) leads to

$$p(\theta_k^{i,j}, E_k^i|Y_{1:k}) = \begin{cases} \dfrac{\hat{\beta}_{i,0}}{1 + w_k^i} & j = 0 \\ \hat{\beta}_{i,j} & j \in [1, \dots, N_M] \end{cases} \tag{4.63}$$

Finally, by combining (4.59)-(4.62) we can obtain an expression for the existence probability

$$p(E_k^i|Y_{1:k}) = \frac{\hat{\beta}_{i,0}}{1+w_k^i} + \sum_{j=1}^{N_M} \hat{\beta}_{i,j} \tag{4.64}$$

4.4.6 Updating the Track Posteriors

It has already been explained that the updated target posterior distributions, conditional on target existence, can be written as weighted mixtures over the measurement association hypotheses, i.e.:

$$p(\mathrm{x}_k^i|E_k^i, Y_{1:k}) = \sum_{j=0}^{N_M} \beta_{i,j} p(\mathrm{x}_k^i|\theta_k^{i,j}, \mathrm{y}_k^j, E_k, Y_{1:k-1}) \tag{4.65}$$

where

$$\beta_{i,j} = p(\theta_k^{i,j}|E_k^i, Y_{1:k}) \tag{4.66}$$

and $p(\mathrm{x}_k^i|\theta_k^{i,j}, \mathrm{y}_k^j, E_k, Y_{1:k-1})$ represents the updated target posterior distribution, obtained by performing a standard Bayesian Filtering update, using the respective measurement j.

Thus, in order to evaluate (4.65), it is necessary to obtain an expression for the mixture weights $\beta_{i,j}$. This easily achieved by applying Bayes rule to (4.63)

$$p(\theta_k^{i,j}|E_k^i, Y_{1:k}) = \frac{p(\theta_k^{i,j}, E_k^i|Y_{1:k})}{p(E_k^i|Y_{1:k})} \tag{4.67}$$

Finally, we can expand the above expression to obtain

$$\beta_{i,j} \propto \begin{cases} \dfrac{\beta_{i,0}}{1+w_k^i} & j = 0 \\ \beta_{i,j} & j \in [1, \ldots, N_M] \end{cases} \tag{4.68}$$

which gives us everything we need to evaluate (4.65).

4.5 The Probability Hypothesis Density Filter

The Probability Hypothesis Density (PHD) filter is a tractable alternative to the optimal Multi-Target Bayes Filter, seminally derived in a Random Finite Set

(RFS) and Finite Set Statistics (FISST)[96] context [69, 70]. The idea behind the PHD filter is to approximate and propagate only the first-order moment (i.e. the Probability Hypothesis Density) of the multi-target state density. In plain terms, the PHD filter estimates the mean number of targets per given area/volume of the plane.

Suppose that X_{k-1} is a realisation of the multi-target state from time t_{k-1}. Then the multi-target state for t_k can be modelled as an RFS Ξ_k of the form

$$\Xi_k = S_k(X_{k-1}) \cup B_k(X_{k-1}) \cup \Gamma_k \tag{4.69}$$

where $S_k(.)$ is the RFS of surviving targets, $B_k(.)$ is the RFS of new targets that have been spawned from existing targets, and Γ_k is the RFS of targets that have spontaneously appeared at time t_k. For the RFSs of surviving and spawned targets, the transition kernel $f_{k|k-1}(X_k|X_{k-1})$ can be used in a manner analogous to the single-target Markov transition density of (2.3).

Similar to above, given a realisation X_k of the multi-target state, the multi-target measurement density can be modelled as an RFS Σ_k of the form

$$\Sigma_k = \Delta_k(X_k) \cup C_k(X_k) \tag{4.70}$$

where $\Delta_k(.)$ is the RFS of measurements relating to detected targets in X_k, while $C_k(.)$ denotes the RFS of clutter generated measurements. The likelihood kernel $p(Y_k|X_k)$ can be used to describe the statistical behaviour of Σ_k, in a manner similar to the likelihood function of (2.20).

The PHD of a RFS is analogous to the expectation of a random vector. To overcome the computational limitations of the Multi-Target Bayes Filter, the PHD filter propagates only the PHD of the multi-target state pdf $p(X_k|Y_{1:k})$, defined as:

$$D_k(\mathrm{x}) = D_{\Xi_k}(x) = \int \delta_X(\mathrm{x}) p(X|Y_{1:k}) \delta X \tag{4.71}$$

Given a measurable region $A \subseteq \mathcal{X}$, the PHD is defined such that the measure

$$\int_A D_k(\mathrm{x}) d\mathrm{x} \tag{4.72}$$

is equal to the expected number of targets in A.

It should be noted that $D_k(\mathrm{x})$ does not describe a probability density, since its integral over the entire target space is not strictly equal to unity, but instead represents the expected number of targets. Similar to before, the PHD filter involves a pair of prediction and update steps that propagate the intensity function

$D_{k-1}(\mathrm{x})$ recursively in time. A generic PHD filter recursion is shown in Algorithm 4.5.1.

Algorithm 4.5.1 (Probability Hypothesis Density Filter) *The set of recursive equations necessary to compute/approximate the predicted $D_{k|k-1}(\mathrm{x})$ and posterior $D_k(\mathrm{x})$ PHD densities are given by the following filtering equations.*

1. **Initiation:** *The filter is initialised with the prior PHD $D_0(\mathrm{x})$, and the recursion is initiated.*

2. **Prediction:** *The predicted PHD density $D_{k|k-1}(\mathrm{x})$ is computed as follows:*

$$D_{k|k-1}(\mathrm{x}) = \gamma_k(x) + \langle\ b_{k|k-1}(\mathrm{x}|.), p_S D_{k-1}, f_{k|k-1}(\mathrm{x}|.)\ \rangle \tag{4.73}$$

where:

- $\gamma_k(\mathrm{x})$ *denotes the intensity function of the spontaneous birth targets RFS*
- $b_{k|k-1}(\mathrm{x}|\mathrm{x}')$ *denotes the intensity function of the spawned targets RFS*
- $p_S(\mathrm{x}')$ *is the probability that a target still exists at time t_k given that it has previous state* x'
- $f_{k|k-1}(\mathrm{x}|\mathrm{x}')$ *is the single-target transition density*
- $\langle\ f, g, h\ \rangle = \int f(\mathrm{x})g(\mathrm{x})h(\mathrm{x})d\mathrm{x}$

3. **Update:** *The updated PHD density $D_k(\mathrm{x})$ is computed as follows:*

$$D_k(\mathrm{x}) = [1 - p_d(\mathrm{x})]\, D_{k|k-1}(\mathrm{x}) + \sum_{\mathrm{y}\in Y_k} \frac{p_d(\mathrm{x})p(\mathrm{y}|\mathrm{x})D_{k|k-1}(\mathrm{x})}{\kappa(\mathrm{x}) + \langle\ p_d p(\mathrm{y}|.), D_{k|k-1}\ \rangle} \tag{4.74}$$

where:

- $\kappa(\mathrm{y})$ *is the intensity function of the clutter RFS*
- $p_d(\mathrm{x})$ *is the state-dependent probability of detection*
- $p(\mathrm{y}|\mathrm{x})$ *is the single-target measurement likelihood*

The PHD filter inherently provides a partial solution to the multi-target tracking problem, while circumventing the necessity for many complex procedures, such as measurement gating, data association and track management (see Section 4.3). The term "partial" is used, due to the inability of the PHD filter to label, and thus distinguish between, different targets. Examples of previous work on incorporating such a functionality to PHD filters can be found in [97, 68], however such approaches are generally based on either clustering or peak-extraction methods, which are prone to errors. There exist a number of different algorithmic implementations of the PHD filter, with the most prominent being the closed-form Gaussian Mixture PHD (GM-PHD) [67, 98] and the approximate Sequential Monte Carlo PHD (SMC-PHD) [68, 99, 100] algorithms. Due to their relative flexibility, PHD filters have quickly been adopted to solve a host of practical problems. These include tracking using bi-static radar data [101], tracking in sonar images [102, 103], as well as tracking in video [104, 105].

4.6 Track Management using Random Finite Sets and Existence Probabilities

The content presented herein aims to demonstrate the application of a radar track initiation technique which utilises a PHD Filter to model the density of uninitiated targets and consecutively flag "good" quality tracks for initiation. The method presented here is not new; the seminal work that formulates the algorithm was performed in [80] and [23], where the authors demonstrated how a Particle Filter and PHD Filter, respectively, can be utilised to perform track initiation in a generic multi-target tracking scenario.

In summary, the track initiation scheme presented in [23] utilises a modified SMC-PHD [68] filter to track and initiate tentative tracks, while confirmed tracks are tracked via a JIPDA process, akin to the one discussed in Section 4.4, whereby a Particle Filter is used to generate the posteriors conditional on each measurement hypothesis. Once measurements are passed through the above process, the probability ρ_j that the j-th measurement is unused by any of the confirmed tracks is computed as per (4.9). The authors use this probability as a weighting factor in the modified SMC-PHD recursion, to account for the fact that some measurements are used by confirmed tracks.

The difference between the modified SMC-PHD filter, presented in [23], and a standard SMC-PHD lies in the update step of the filter. More specifically, the authors propose a modified version of (4.74), that makes use of the weighting probability ρ_j, as follows:

$$\begin{aligned} D_k(\mathrm{x}) &= D_k^0(\mathrm{x}) + \sum_{j=1}^{M_k} D_k^j(\mathrm{x}) \\ &= [1 - p_d(\mathrm{x})]\, D_{k|k-1}(\mathrm{x}) + \sum_{j=1}^{M_k} \frac{\rho_j p_d(\mathrm{x}) p(\mathrm{y}_k^i|\mathrm{x}) D_{k|k-1}(\mathrm{x})}{\kappa(\mathrm{x}) + \langle\ p_d p(\mathrm{y}_k^i|.), D_{k|k-1}\ \rangle} \end{aligned} \tag{4.75}$$

where $D_k^0(\mathrm{x})$ and $D_k^j(\mathrm{x})$ are the updated PHD densities, conditional on the null measurement hypothesis and the j-th measurement respectively.

Using the above notation, it is therefore possible to define the probability p_j that a given measurement relates to an unconfirmed track as follows:

$$p_j = \int D_k^j(\mathrm{x}) dx \tag{4.76}$$

The above is justified by noting that $\int D_k^j(\mathrm{x}) dx \leq 1$ for $j = 1, \ldots, M_k$, since each measurement can only originate from at most one target. Furthermore, the expected number of targets producing measurement y_k^j is equal to the probability that y_k^j originated from a true target.

Finally, tracks to be initiated are extracted from the PHD density on the basis of the following condition:

$$p_j \geq P_{conf} \tag{4.77}$$

where P_{conf} is the selected confirmation probability threshold.

The track extraction process is performed prior to the PHD update step. First, candidate measurements to be used for track initiation are identified via (4.77). Continuing, new (confirmed) tracks are formed for each respective density $D_k^j(\mathrm{x})$, with existence probability p_j, relating to each identified candidate measurement. Finally, the PHD is updated using (4.75), for the remaining measurement hypotheses that do not satisfy (4.77).

4.6.1 Results

For the purpose of simulations, we consider the scenario of tracking the position of 10 vessels, on a two-dimensional Cartesian plane (i.e. with x, y coordinates). The simulated trajectories followed by the vessels (see Figure 4.4) all have the same lifespan and their initial positions are set, but the initial course and speed of each vessel is chosen randomly. As before, the dynamics of targets are assumed to be governed by a Constant Heading (CH) model (see Section 2.2.4), with $\sigma_s = 1\ m/s^2$ and $\sigma_\phi = 0.07\ rad/s$.

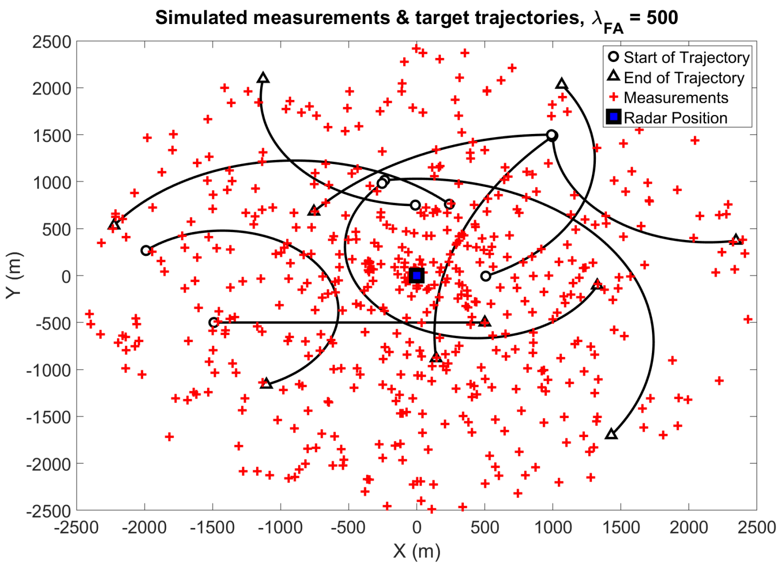


Figure 4.4: True trajectories followed by the vessels in a single experiment, with clutter rate $\lambda_{FA} = 500$. The output of the last measurement scan is also depicted.

A Radar sensor is placed at coordinates $(0m,0m)$, which detects targets according to a Constant Detection Rate model (see Section 2.4.1) with probability $P_D = 0.9$ and generates observations of their positions in terms of range & bearing, over 200 scans, at a rate of 3 sec/scan. Measurement noise is modelled using a Gaussian Azimuth-Range measurement model (see Section 2.3.2), having Gaussian noise in both azimuth and range, with standard deviations $\sigma_r = 10\ m$ and $\sigma_\theta = \frac{\pi}{180}\ rad$, respectively. Clutter is modelled using a Poisson Rate with Uniform Position model (see Section 2.5.1), where the number of clutter measurements per scan is assumed to be Poisson distributed, with mean λ_{FA} accross the surveillance volume V, while their positions are uniformly distributed over the polar search space defined on the azimuth and range coordinate system, such that $p_C(\mathrm{y}_k^j) = U([0, 2500], [0, 2\pi])$.

The tracking performance of 3 different algorithms has been evaluated, as follows:

- PDAF-MofN: *M-out-of-N* based track management approach, with confirmation threshold of $T_{conf} = 8/10$ scans and deletion threshold of $T_{del} = 2/3$ scans. Tentative tracks are initiated from any measurements that are not associated with any known tracks. Once initiated, tentative tracks are processed using a Probabilistic Data Association Particle Filter (PDA-PF), while confirmed tracks are process using a Joint PDA-PF (JPDA-PF). A set of 5000 particles is used to approximate the pdf of individual tracks.

- PDAF-LLR: *Log Likelihood Ratio* (LLR) based track management approach, with false track confirmation probability $\alpha = 10^{-6}$, true track deletion probability $\beta = 0.1$ and track deletion threshold $THD = -6.7$ (see Section 4.3.2). Tentative tracks are initiated from any measurements that are not associated with any known tracks. A PDA-PF and a JPDA-PF are used to perform tracking of tentative and confirmed tracks, respectively. A set of 5000 particles is used to approximate the pdf of individual tracks.
- SMC-PHD-EP: A Sequential Monte-Carlo PHD (SMC-PHD) [68] filter is used to model the intensity of unconfirmed targets. The PHD density is approximated using $5 * 10^4$ particles and a uniform track birth intensity $\gamma_k(x) \sim 0.2\ U(2\pi\ rad, 2500\ m)$[1] is used. A track is confirmed when its estimated probability of existence is higher than 0.9. Confirmed tracks are then processed using a Joint Integrated PDA-PF (JIPDA-PF), with 5000 particles allocated to each track. Target existence is modelled according to a Markov Chain One [92, 94] model, with survival probability $P_S = 0.95$, and a track is deleted when its existence probability becomes lower than 0.1.

The estimation performance evaluation is achieved by means of comparing the value of the Optimal SubPattern Assignement (OSPA)[106] metric achieved by each algorithm, averaged over 50 Monte-Carlo simulation runs, with variable values of clutter rate λ_{FA}. The combination of both the localisation and cardinality error components of OSPA allow us to compare the different approaches based on both how well they approximate the correct number of targets and how quickly they respond to the birth or death of targets (cardinality), as well as how well each method approximates the posterior of each initiated track (localisation)[2]. The results of this evaluation are shown in Table 4.1. Furthermore, the computation time for each approach (only the track management process execution time is measured), averaged over 50 Monte-Carlo simulation runs, is also evaluated for varying values of λ_{FA}. The simulations were run on a desktop computer with a 3.7GHz 8-core Intel i7 6900k processor with 32GB of maximum available RAM. Figure 4.5 depicts the evolution of the execution time a each approach as λ_{FA} increases.

[1]U(x,y) is used do denote a uniform distribution over the space $[0, x] \times [0, y]$.

[2]Without loss of generality, we assume that the OSPA localisation error relating to the estimation of confirmed tracks is the same accross the different methods, due to the fact that all methods use the same algorithm (i.e. JPDA-PF) to track the confirmed tracks. Thus, any variation of OSPA between the different methods is assumed to be related to errors in the respective track management processes.

Method \ Params	Mean OSPA [lower is better]			
	$\lambda_{FA} = 0$	$\lambda_{FA} = 100$	$\lambda_{FA} = 250$	$\lambda_{FA} = 500$
PDAF-MofN	31.846	44.597	72.541	141.874
PDAF-LLR	27.605	32.876	42.125	61.675
SMC-PHD-EP	14.541	18.954	23.785	30.756

Table 4.1: Performance comparison between PDAF-MofN, PDAF-LLR and SMC-PHD-EP for range of clutter parameter values. The computed mean OSPA, averaged over 50 Monte-Carlo simulations is presented, for varying values λ_{FA}.

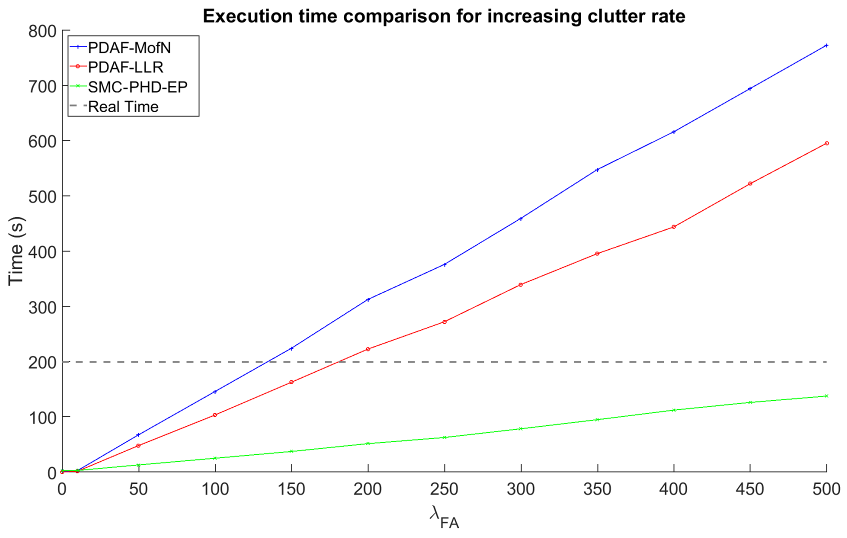


Figure 4.5: Execution time comparison between PDAF-MofN, PDAF-LLR and SMC-PHD-EP, averaged over 50 Monte-Carlo simulations, for varying values λ_{FA}.

From the results of Table 4.1, it can be observed that the error achieved by SMC-PHD-EP is consistently lower than that of the other two approaches. Furthermore, as the clutter intensity is increased, the improvement margin becomes increasingly more evident. It should be noted that for the case of PDAF-MofN and PDAF-LLR, better results can be achieved by employing a JPDA for the tentative tracks, however such an attempt would introduce additional computational costs, to what are effectively already very costly approaches. This can be clearly observed by analysing the results of Figure 4.5. As the clutter rate increases, both PDAF-MofN and PDAF-LLR can be seen to exhibit significantly higher computation times than SMC-PHD-EP. This can be attributed to the fact that both PDAF-MofN and PDAF-LLR initiate tentative tracks for all unassociated measurements, meaning that data association (PDA) must be performed for an increasing number of false tracks, as clutter is increased. On the contrary, as the SMC-PHD filter, which is used to track the density of tentative tracks in SMC-PHD-EP, effectively circumvents the necessity for data association, able to handle the increase in clutter rate with higher efficiency.

4.7 Specifics of Applications to Real Data

This section presents a qualitative analysis of the performance achieved by various multi-target tracking approaches, applied on a dataset collected from real radar sensor. Specific focus is given to the track management aspect of the compared algorithms and a range of parameters is tested for each one. Furthermore, the performance of each algorithm is qualitatively evaluated on the basis of two opposing scenarios. The first scenario involves an obscured region, where the radar sensor does not have a line of sight to the targets. The second scenario is concerned with a region of high levels of clutter, artificially induced by lowering the detection threshold of the radar.

4.7.1 Dataset

The dataset used in this section is formed by extracting a radar plot recording a real Vessel Traffic Service (VTS) site in the UK, serviced by Denbridge Marine[3]. The deployed radar unit from which the plots were extracted is a 25kW X-Band ($\sim$ 10GHz, 3cm wavelength) Magnetron Radar, equipped with an 8ft antenna that performs full rotations every 2-3 seconds. The length of the dataset extends over 1 hours, with Sensitivity Time Control (STC) and Constant False Alarm Rate (CFAR) amplitude thresholds turned on. A plot of the entire dataset is shown in Figure 4.6.

4.7.2 Algorithms

The tracking performance of 3 different algorithms has been evaluated, as follows:

[3]Denbridge Marine Ltd. are a Small-to-Medium sized Enterprise (SME), based in Liverpool, who focus on the development and deployment of VTSs across the globe and also co-funded the PhD project undertaken by the author, via an iCASE EPSRC award facilited by the Smith Institute for Industrial Mathematics and Systems Engineering

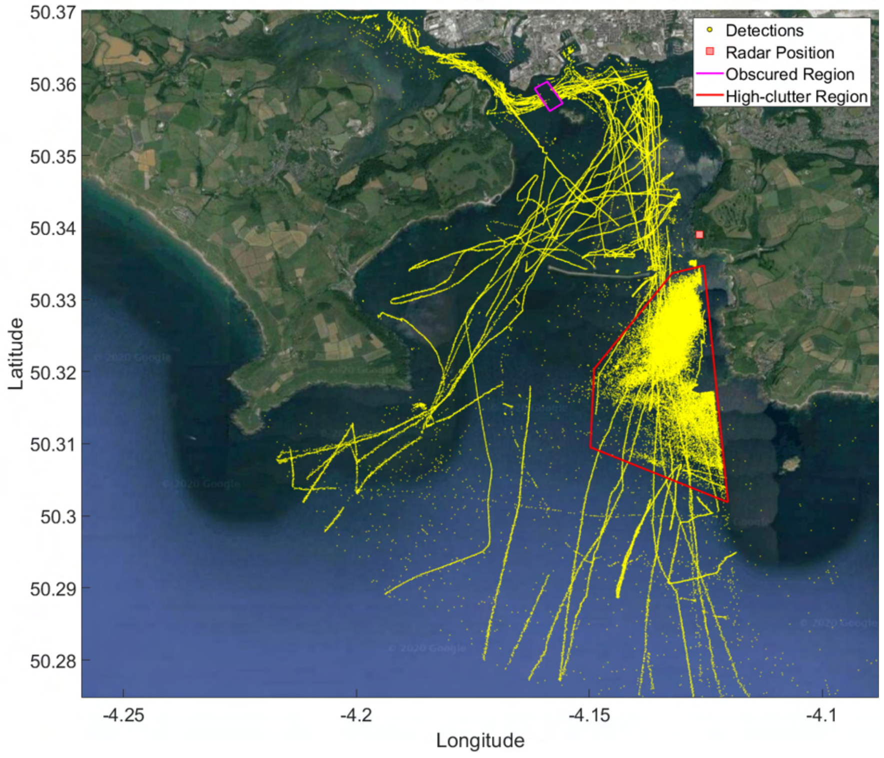


Figure 4.6: Full dataset extracted for the purposes of experimentation

- M-out-of-N: Track Management is performed using a *M-out-of-N* based approach, whereby Tentative tracks are initiated from any measurements that are not associated with any known tracks. Data association for confirmed tracks is performed using JPDA, while the data association algorithm used to process tentative tracks will vary, depending on the scenario.
- Log-Likelihood Ratio: Track Management is performed using a *Log Likelihood Ratio* (LLR) based approach, whereby tentative tracks are initiated from any measurements that are not associated with any known tracks. Data association for confirmed tracks is performed using JPDA, while the data association algorithm used to process tentative tracks will vary, depending on the scenario.
- SMC-PHD-EP: A Sequential Monte-Carlo PHD (SMC-PHD) [68] filter is used to model the intensity of unconfirmed targets. The PHD density is approximated using $5 * 10^4$ particles and a uniform track birth intensity $\gamma_k(x) \sim 0.2\ U(2\pi\ rad, 2500\ m)$ is used. Confirmed tracks are processed using JIPDA, where target existence is modelled according to a Markov Chain One [92, 94] model.

4.7.3 Scenarios

In the considered dataset, there exist two regions of the surveillance area, which exhibit artefacts that are typical of real radar tracking scenarios: i) obscured regions; and ii) regions of high-clutter density. These regions are illustrated in Figure 4.6 using polygons that encompass the related areas. Due to the fact that the underlying sensor characteristic in those areas vary to a great extent when compared to the rest of the surveillance area, successful tracking of targets in said regions becomes highly challenging. Therefore, specific focus will be given to studying the behaviour of the various algorithms, when considering these regions. As such, performance evaluations will be performed based on the following two scenarios:

1. **Tracking through obscurations**: Figure 4.7 shows the focus area for this scenario. The obscured area is marked using a purple rectangle. As it can be observed, the island that is situated south-west from the highlighted region falls in the line of sight of the sensor, meaning that targets go undetected when they cross behind it. Due to this fact, tracks for targets that travel behind the island will typically be prematurely terminated. The goal in this case will be to identify a parameter setting that allows for such tracks to survive and maintain the posterior for targets as they emerge from behind the island.

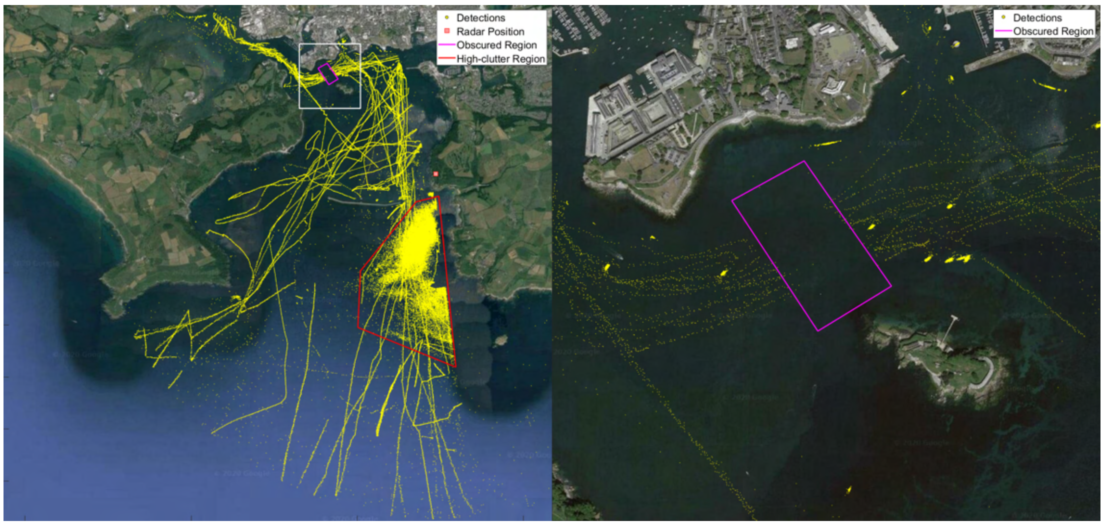


Figure 4.7: Focus area for obscuration scenario (Scenario 1)

2. **Tracking through dense clutter**: Figure 4.8 shows the focus area for this scenario. The area marked using a red polygon envelops a region which ex-

hibits heavy levels of clutter in comparison to the rest of the surveillance area. The clutter inherent to this area results in challenges in track initiation and termination, as well as hindered tracking performance for targets that pass through. Therefore, a comparison will be made to qualitatively measure the ability of the various algorithms to maintain sufficient tracking performance.

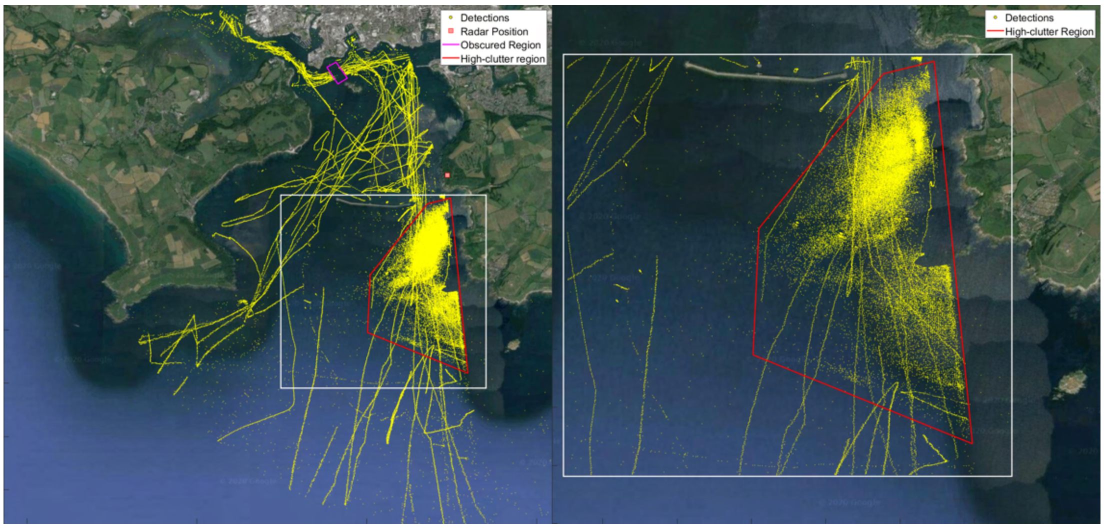


Figure 4.8: Focus area for dense-clutter scenario (Scenario 2)

4.7.4 Models

The dynamic motion of targets is assumed to evolve according to an Integrated Ornstein Uhlenbeck model (see Section 2.2.3), with velocity noise $\sigma = 1m/s$ on each coordinate and a damping coefficient of $\alpha = 10^{-2}$. Radar noise is modelled using a Gaussian Azimuth Range model (see Section 2.3.2), with standard deviations $\sigma_r = 10\ m$ and $\sigma_\theta = \frac{\pi}{180}\ rad$, respectively. Clutter is modelled using a Poisson Rate with Uniform Position model (see Section 2.5.1), where the number of clutter measurements per scan is assumed to be Poisson distributed with mean $\lambda_{FA} = 10$ across the surveillance volume V, while their positions are uniformly distributed over the polar search space defined on the azimuth and range coordinate system, such that $p_C(\mathrm{y}_k^j) = U([0, 2500], [0, 2\pi])$. Target detectability is modelled according to a Constant Detection Rate model (see Section 2.4.1) with probability $P_D = 0.8$. Finally, to reduce the computational complexity of the data association processes, ellipsoidal gating is performed (see Section 4.3.1), with $\gamma_G = 5$, leading to a gating probability $P_G \approx 0.92$.

4.7.5 Results

The results will be presented in line with the algorithms and scenarios discussed in Sections 4.7.2 and 4.7.3, respectively. For each of the discussed scenarios, the performance of the algorithms will be evaluated against a selection of parameters with the aim of addressing the problems and complications inherent to each case, while making comparisons between the best performance achieved by each algorithm. Due to the dataset being collected from a real sensor, rather than being simulated, the is no ground-truth data available, which would allow us to perform quantitative evaluations and comparisons. For this reason, a qualitative approach will be followed, based on observations made on the underlying data.

4.7.5.1 Scenario 1 - Tracking through obscurations

The experiments performed in this section will be based around the first scenario discussed in Section 4.7.3. The focus area for this scenario and the extent of the dataset used are illustrated in Figure 4.7. At a first stage, the performance of each of the algorithms discussed in Section 4.7.2 will be quantified using the base set of model parameters shown in Section 4.7.4, after which different parameters will be tested for each algorithm with the aim of alleviating the track continuity issues that arise.

M-out-of-N

As outlined previously, *M-out-of-N* track management is performed by maintaining a count of M subsequent iterations, out of a N sized window of report intervals, during which a track has been successfully associated to at least one detection. With this M/N measure in hand, confirmation ($T_{conf} = M_{conf}/N_{conf}$) and deletion ($T_{del} = M_{del}/N_{del}$) thresholds can be set, ss as to confirm tracks which have been successfully detected in M_{conf} out N_{conf} iterations, and respectively delete tracks which have not been detected in M_{del} out of N_{del} iterations.

The minimal presence of clutter in this scenario allows for the use of Joint Probabilistic Data Association (JPDA) methods to process tentative and confirmed tracks. Hence, to ease notation, the term *JPDA-MofN* will be used to refer to the overall tracker. The universal threshold for track confirmation is set as $T_{conf} = 4/5$, while the threshold for track deletion was configured as $T_{del} = 8/10$. Although the two threshold ratios seem similar, it is important to notice the track confirmation window N_{conf} is configured to be smaller than the deletion window N_{del}. A small detection window for track confirmation ensures that tracks can

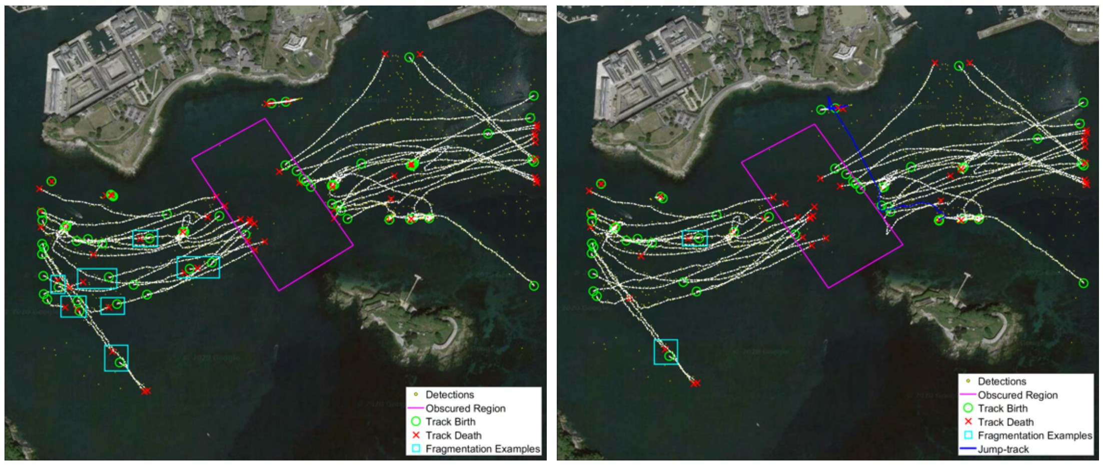


Figure 4.9: Comparison between PDAF-MofN tracker output with deletion thresholds $T_{del} = 4/5$ (left) and $T_{del} = 8/10$ (right) for the dataset of Scenario 1.

be confirmed, and therefore be shown to the operator, relatively fast. By examining Figure 4.7 it can be observed that the levels of clutter are relatively low in this scenario, meaning that the confirmation window can be kept low. On the other hand, although not evident in Figure 4.7, missed detections are a frequent phenomenon. Therefore, use of a larger window for track deletion ensures that tracks are not prematurely terminated.

To demonstrate the effects of varying the window size, a pair of examples is generated that examine the tracker output for $T_{del} = 4/5$ and $T_{del} = 8/10$, respectively. The result are illustrated in Figure 4.9. At can be seen that a deletion threshold of 4/5 results in an increased amount of fragmented tracks. On the other hand, although increasing the detection window improves track fragmentation, it is possible that *track jumping* can occur, whereby a track's covariance is allowed to grow to such an extent that it ends up being misassociated with a distant detection and leaps to that location. An important observation in both examples shown in Figure 4.9 is that the existence of the obscuration region results in all tracks that enter the area to be terminated and then re-initiated once they emerge on the opposite end. However, as is expected in the case of $T_{del} = 8/10$, the tracks can be seen to survive for longer, which can be easily justified by considering that M_{del} is larger in this case.

The discussion on the effects of varying the chosen value of T_{del}, and mainly the observation about the survival of tracks when entering the obscuration region, naturally hints to a potential solution to the problem of maintaining tracks which cross that area. One first option would be to further increase the value of M_{del}, however, as already noted, doing so for the entire surveillance region bears the risk

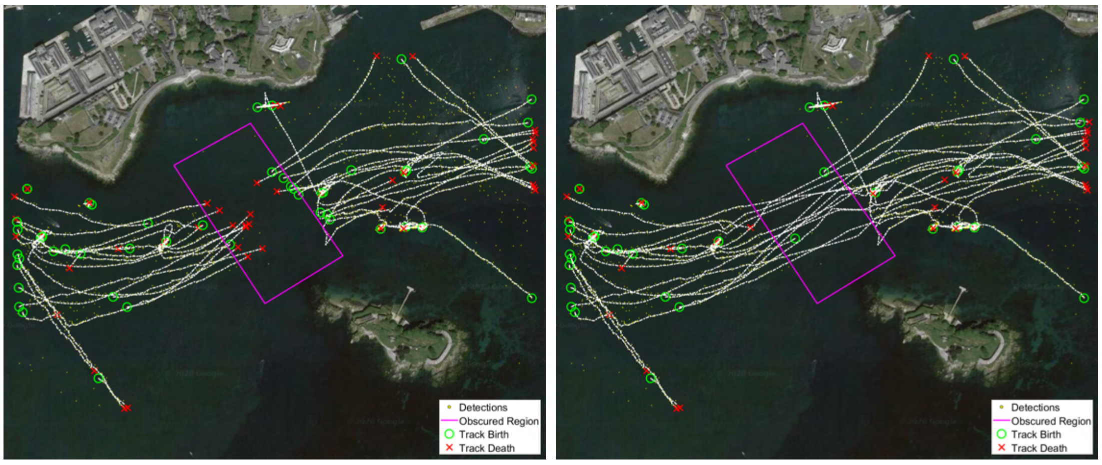


Figure 4.10: Comparison between JPDAF-MofN tracker output with (right) and without (left) the use of a dynamic deletion threshold for the dataset of Scenario 1.

of introducing more *track jumping*. Instead, a sensible solution is to introduce a longer detection window for tracks that are located withing the obscured region.

The correct choice of detection window depends on the length of time that a given target is obscured. As is obvious, this time is directly related to the speed of the target as well as the length of its followed path. Unfortunately, neither of these quantities can be accurately known, however it is possible to obtain an approximation by introducing certain assumptions. If the assumption is made that a target maintains nearly constant velocity and heading while travelling through the obscured region, one can calculate both an estimated path (i.e. a straight line between the two edges of the obscured region) and consequently estimate the time T_p taken to follow that path. With this time estimate in hand, and noting that the time T_s between consecutive scans is constant in rotating radars, it is therefore possible to dynamically calculate the threshold M_{del} for each track as $M_{del} = T_p/T_s$. Finally, the value for N_{del} is computed as $N_{del} = M_{del} + 0.1M_{del}$.

The output from a tracker that utilises the process outlined in the previous paragraph to dynamically estimate a deletion threshold, applied while a target travels through the obscured region, is shown in Figure 4.10. The deletion threshold outside the obscured region is set to $T_{del} = 8/10$. For reference purposes, the output from the equivalent tracker without the use of a dynamic threshold is also shown (this is the same as the output of Figure 4.9 for $T_{del} = 8/10$). As it can be observed, the addition of a dynamic threshold is successful in maintaining tracks through the obscured region, while the tracking performance outside this region is kept identical to its counterpart. It is worth noting here that due to the absence of ground-truth data it is not possible to evaluate quantitatively whether track switching occurs.

Log-Likelihood Ratio

The notion of Log-Likelihood Ration (LLR) for track confirmation and deletion was discussed in Section 4.3.2. Instead of using heuristically defined parameters for the confirmation and deletion thresholds, the LLR thresholds are defined based on a set of parameters that can generally be defined from the system requirements. More importantly, the calculations for LLR take into account the detection and clutter characteristics, meaning that the process can be more finely tuned.

Once again, the minimal presence of clutter in this scenario allows for the use of Joint Probabilistic Data Association (JPDA) methods to process tentative and confirmed tracks, while the term *JPDA-LLR* will be used to refer to this particular tracker configuration. As per general convention [1], we set $\beta = 0.1$. When choosing a value for α, we make use of (4.16). From our model definition of Section 4.7.4 we have assumed $\lambda_{FA} = 10$. Continuing, since we want to minimise false track confirmations, we allow $N_{FC} = 1$ false track confirmation per $N_S = 10^7$ scans. Therefore, solving (4.16) for this set of chosen parameters yields $\alpha = 10^{-6}$. By utilising (4.15), the above choice of α and β leads to the following thresholds $T_{low} \approx -2.3$ and $T_{high} \approx 13.7$. The value for the deletion threshold THD is defined by assuming that a track should be deleted if it has not been detected in 5 consecutive scans. From equation (4.14), and using our model parameter values for P_D and P_G from Section 4.7.4, we can derive that ΔL_0 for a single missed detection is given by:

$$\Delta L_0 \triangleq \log \frac{p(\mathrm{y}_k^0|H_1)}{p(\mathrm{y}_k^0|H_0)} = \log(1 - P_D P_G) \approx -1.34 \tag{4.78}$$

Therefore, we can obtain a value for THD as $THD \triangleq 5\Delta L_0 = 6.7$.

The output of a JPDAF-LLR tracker configured with the above parameter settings is shown in the left sub-figure of Figure 4.11. The tracker can be seen to exhibit satisfactory level of performance, without any obvious signs of the track fragmentation or jumping instances observed in the case of the JPDAF-MofN configuration. However, the algorithm is not able to maintain tracks that cross the obscured region. This is an expected outcome, since no provisioning has been made to account for the fact that targets are significantly more likely to go undetected while cruising through that area.

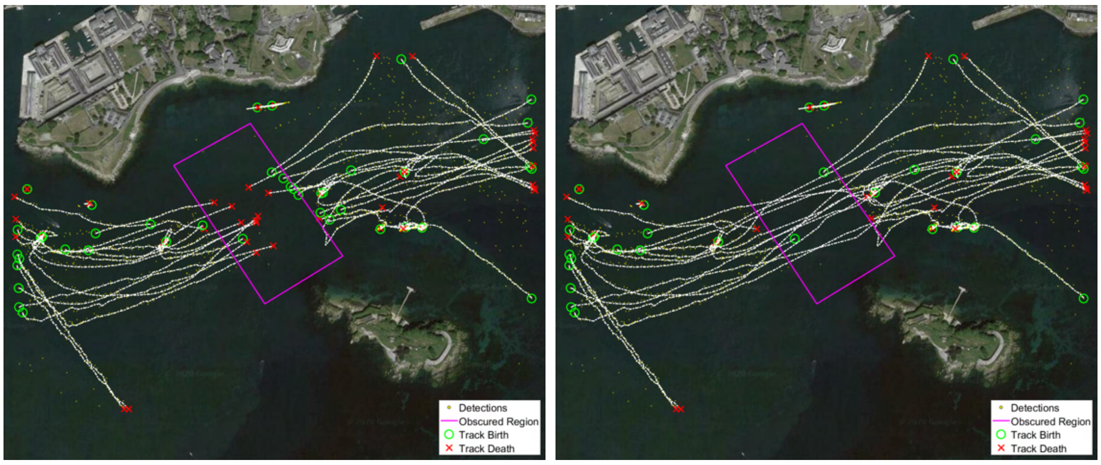


Figure 4.11: Comparison between JPDAF-LLR tracker output with CDR (left) and SDDR (right) detection model for the dataset of Scenario 1.

One potential solution to the problem of maintaining track of targets through the obscured region, would be to modify the value of THD in that area, in a manner similar to the approach followed for the JPDAF-MofN tracker. However, due to the fact that the LLR computation equations take into account the model parameters, it is not necessary to resort to such heuristics. Instead, the more statistically correct approach taken here is to modify the detection probability for targets that pass through the obscured region. Therefore, a State Dependent Detection Rate (SDDR) model is employed (see Section 2.4.2), such that:

$$p_d(\mathrm{x}_k) = p(d_k = 1|x_k) = \begin{cases} 0.1, & \text{if } \mathrm{x}_k \in \mathcal{V}_o \\ 0.8, & \text{otherwise} \end{cases} \tag{4.79}$$

where $\mathcal{V}_o$ is used to denote the obscured region. The output of the JPDAF-LLR tracker configured to utilise the above SDDR model is illustrated alongside its CDR counterpart in Figure 4.11. As it can be observed, the use of a SDDR model is both an effective and relatively simple (compared to the use of heuristics) method for ensuring track continuity through the obscured region.

SMC-PHD with Existence Probabilities

The use of a SMC-PHD filter for track initiation, coupled with the notion of existence probabilities for track confirmation and deletion, was discussed in Section 4.6. For notational purposes we will refer to this algorithm as *SMC-PHD-EP*. The method requires two parameters in order to operate, both of which are exis-

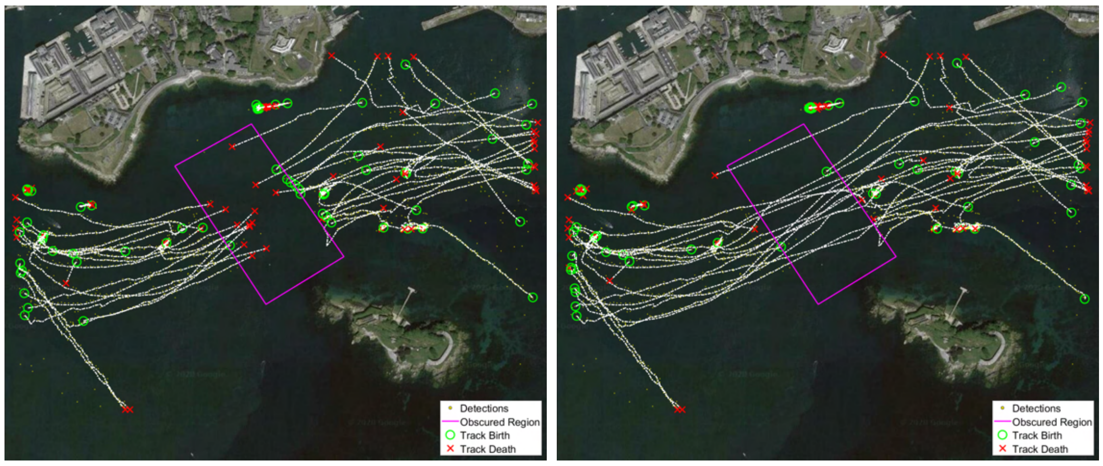


Figure 4.12: Comparison between SMC-PHD-EP tracker output with CDR (left) and SDDR (right) detection model for the dataset of Scenario 1.

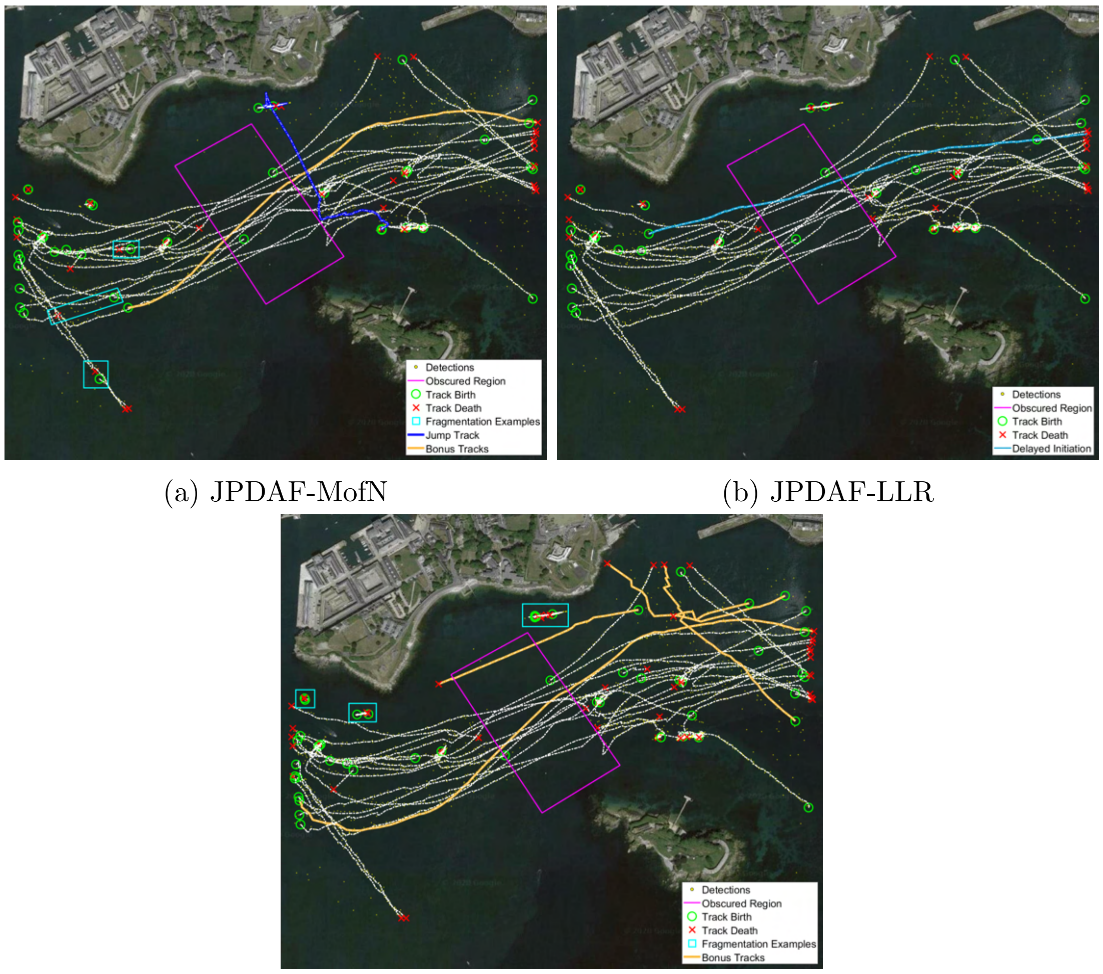
tence probability thresholds. The track confirmation probability P_{conf} and track deletion probability P_{del} are configured as $P_{conf} = 0.8$ and $P_{del} = 0.1$, respectively. Since the definition of these thresholds is fairly intuitive and the expected outcome of varying their values is relatively obvious, further analysis will not be performed here.

As done for the previous algorithms, the tracker is first run on the dataset using the model parameters specified in Section 4.7.4. The output of the tracker is shown in the left sub-figure of Figure 4.12. In order to deal with the track continuity issues caused by the obscured region, the same approach is followed as in the case of the JPDAF-LLR algorithm. Accordingly, the default CDR detection model is replaced with the SDDR model outlined in (4.79) and the tracker is re-run on the same dataset. The resulting tracks are illustrated in the right sub-figure of Figure 4.12, where it can once again be observed that the algorithm is able to successfully maintain track continuity for the obscured targets.

Comparison

A comparison is performed between the tracks produced by each of the algorithms considered above. As there is no ground-truth data, it is not possible to produce a quantitative comparison. Therefore, a qualitative comparison is performed by means of visual comparison and an attempt has been made to identify and highlight observable differences between the different tracker outputs.

Figure 4.20 contains a sub-figure illustrating the tracks produced by each



(a) JPDAF-MofN (b) JPDAF-LLR

(c) SMC-PHD-EP

Figure 4.13: Visualisation of the resulting tracks produced by the JPDAF-MofN, JPDAF-LLR and SMC-PHD-EP algorithms for the dataset of Scenario 1.

of the algorithms. The advantages and deficiencies for each tracker are also highlighted. To better understand the illustrated content, the used terminology is hereby explained:

- The term *Bonus Tracks* is used to denote tracks that are generated by the respective algorithm, but are not found in at least one of the remaining algorithms. It is noted that a closer observation of the data indicates that these tracks can be found to conform to data produced by a real target, rather than clutter.

- The term *Track Fragmentation* is used to indicate that a track is prematurely terminated and then re-initiated at a later stage.

- The term *Jump Track* refers to an instance of a track whose covariance is allowed to grow to such an extent that it ends up being misassociated with a distant detection and falsely leaps to that location.
- The term *Delayed Initiation* is used to an instance of a track which is found to be initiated significantly later than the time indicated by the data, as well as in comparison to the initiation time of other algorithms.

Overall, the SMC-PHD-EP tracker can be seen to produce tracks that are at worst as reliable as the tracks produced by the other two trackers. More specifically, multiple examples of *Bonus Tracks* are found for faint targets that are not picked up by the other algorithms. It is worth noting that, by closely examining tracks that are commonly found in all outputs, no obvious signs are identified to suggest that the particular algorithm initiates targets earlier that other algorithms. A minor short-coming of the algorithm is a marginally elevated number of *Track Fragmentation* example for tracks relating to what is seemingly static targets (buoys), as highlighted in Figure 4.13c. A closer examination of the evolution of the process over time shows that this is caused by missed and merged detections, which lead to tracks being pre-maturely terminated by the JIPDA algorithm and re-initiated at a later stage.

Continuing, the JPDAF-MofN is found to exhibit signs of *Track Fragmentation* for moving targets, while at the same time an example of a *Jump Track* can be observed. These results hint at the inadequacies of the heuristic track deletion approach followed by the algorithm. More specifically, a potential solution to tackling the track fragmentation problem would be to further increase the deletion threshold. However, the presence of a *Jump Track* antithetically suggests that the deletion threshold should be reduced. It is noted that test have been performed for a range of other deletion thresholds, for varying values of M_{del} and N_{del}, which did not yield any improvement. That being said, one advantage of the JPDAF-MofN, compared to JPDAF-LLR, is that the tracker is able to initiate and track a *Bonus Track*, while its overall track initiation performance for all remaining tracks is found to be comparable to that of JPDAF-LLR for the particular scenario.

Finally, the JPDAF-LLR algorithm can be seen to overcome the track fragmentation issues identified in the other two trackers, however is found to suffer from cases of *Delayed Initiation* and an inability to initiate tracks for faint targets. When it comes to track fragmentation the JPDAF-LLR algorithm is able to handle the presence of missed and merged detections more gracefully compared to the other two algorithms, therefore avoiding the premature termination of tracks. Overall, the algorithm can be found to be more conservative than the other two algorithms in terms of track initiation. As discussed previously, the track initiation threshold T_{high} is inversely proportional on the choice of the false track confirmation probability α. Further tests performed for marginally higher values of the specific parameter lead to a modest improvement of the *Delayed Initiation* example, the tracker was stick unable to acquire any *Bonus Tracks*, while an introduction of false tracks was also observed.

4.7.5.2 Scenario 2 - Tracking through dense clutter

The experiments performed in this section will be based around the second scenario discussed in Section 4.7.3. The focus area for this scenario and the extent of the dataset used are illustrated in Figure 4.8. Let us denote the region of dense clutter as $\mathcal{V}_{HC}$. At a first stage, the performance of each of the algorithms discussed in Section 4.7.2 will be quantified using the base set of model parameters shown in Section 4.7.4, after which different parameters will be tested for each algorithm with the aim of alleviating the issues that arise.

The presence of a region of dense clutter in this scenario necessitates that an estimate of the mean clutter rate in that region is obtained. From an empirical observation of the dataset, it has been observed that the ratio of the number of clutter generated detections M_k^C to the number of true detections M_k^Δ at each timestep is significantly high. As it is not possible to obtain an exact value for this ratio, the value $\gamma = M_k^C/M_k^\Delta = 10$ is used as an approximation. Using this value, it is possible to obtain an approximation to the mean number of clutter measurements $\bar{\lambda}_{HC}$ at each scan within that region, as follows:

$$\bar{\lambda}_{HC} = \lambda_{HC} V_{HC} = \frac{1}{N_S} \sum_{k=1}^{N_S} (1 - \gamma^{-1}) M_k \approx 50 \tag{4.80}$$

where, λ_{HC} is the estimated clutter density per unit volume of $\mathcal{V}_{HC}$, V_{HC} is the total volume of $\mathcal{V}_{HC}$, and N_S is the total number of measurement scans. The above value for $\bar{\lambda}_{HC}$ is important, as it will be used to parameterise the clutter model for tracks that travel within $\mathcal{V}_{HC}$ in the following subsections.

M-out-of-N

The presence of dense clutter in this scenario renders the use of Probabilistic Data Association (PDA) methods to process tentative and confirmed tracks impossible. This is due to the fact that new tracks are initiated for all unassociated measure-

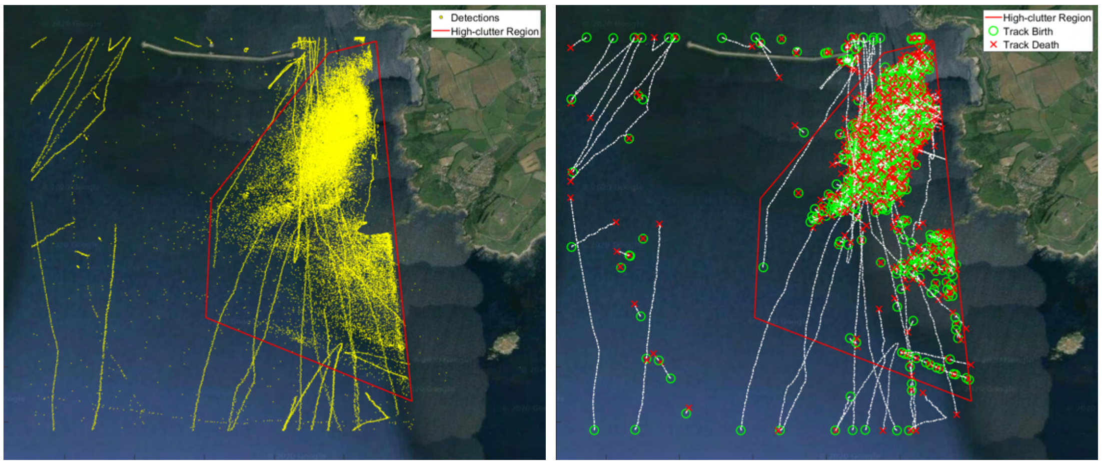
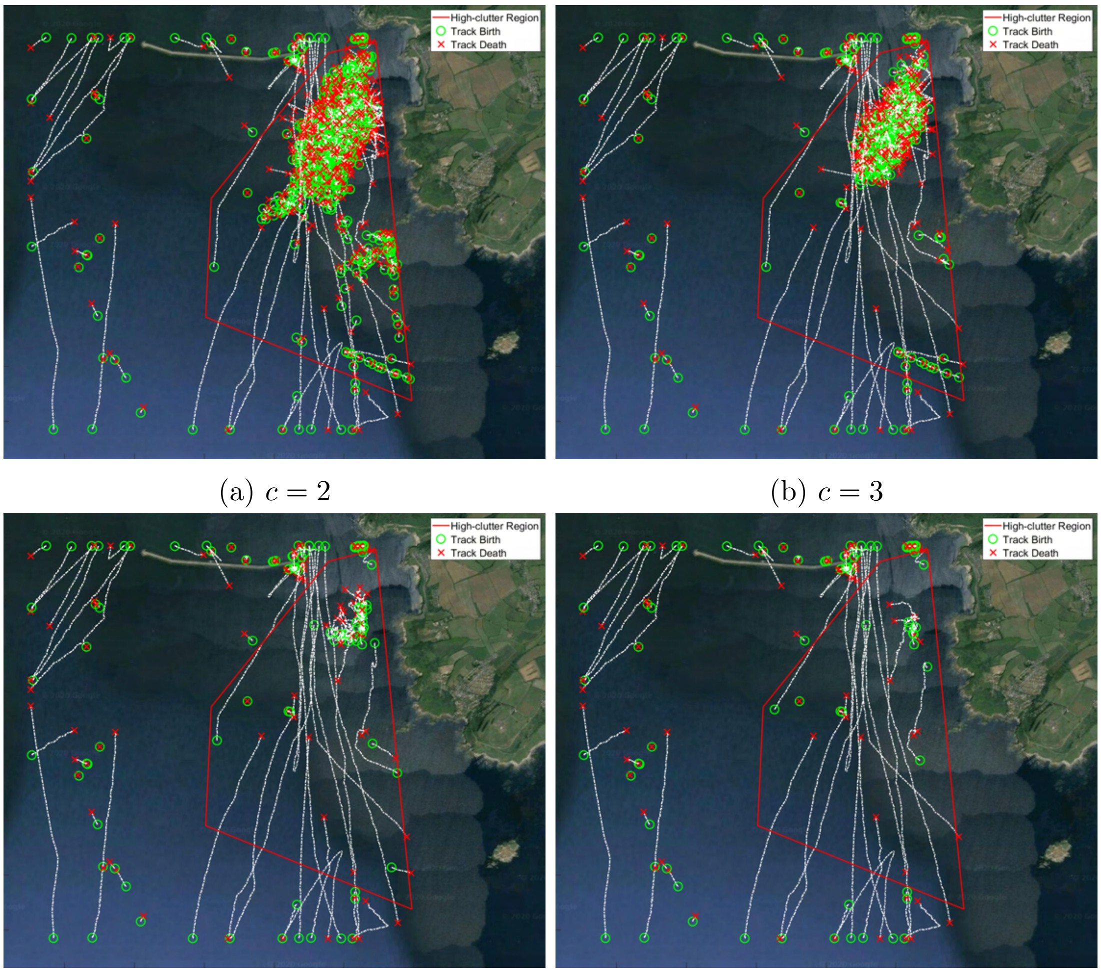


Figure 4.14: Visualisation of the examined dataset (left) and the GNN-MofN tracker output with uniform clutter rate (right).

ments, leading to a combinatorial explosion in the case of these algorithms. For this reason, a Global Nearest Neighbour approach is used to process tentative tracks. To ease notation, the term *GNN-MofN* will be used to refer to this particular tracker configuration. As before, the confirmation and deletion thresholds are initially set to $T_{conf} = M_{conf}/N_{conf} = 4/5$ and $T_{del} = M_{del}/N_{del} = 8/10$, respectively.

The output of the tracker configured with the initial set of parameters specified above, and uniform clutter rate $\lambda_{FA} = 10$, is shown in Figure 4.14. From the results of Figure 4.14 it quickly becomes evident that the dense clutter results in the introduction of a high number of false tracks in the marked region. The presence of a high number of false tracks indicates that the track confirmation threshold needs to be appropriately adjusted to avoid premature confirmation of tracks. This is necessary for *M-out-of-N* based approaches, since the track initiation process does not take into account the clutter model parameters.

In order to identify an optimal setting for the T_{conf}, a set of trial runs was performed for a range of different values. The candidate values were selected by setting $M_{conf} = c{*}4$ and $N_{conf} = c{*}5$, where c is a proportionality constant, taking values $c = 2, 3, 4, 5$. In each of the trial runs, the mean clutter rate for tracks within $\mathcal{V}_{HC}$ is also set to $\bar{\lambda}_{HC} = 50$. The tracks generated by the tracker for the various confirmation thresholds are shown in Figure . As expected, increasing the threshold values leads to a progressive reduction of the amount of false tracks. However, at the same time, the tracker can also be seen to gradually become unable to confirm some of the true tracks, while a subset of the successfully confirmed tracks are found to have their estimated trajectory shortened. The best trade-off parameter was found to be $c = 4$, where the majority of the false tracks are eliminated, while maintaining a satisfactory quality for confirmed tracks.



(a) $c = 2$ (b) $c = 3$

(c) $c = 4$ (d) $c = 5$

Figure 4.15: Visualisation of GNN-MofN tracker output for various values of confirmation threshold $T_{conf} = (c * 4)/(c * 5)$ in the high-clutter region. The confirmation threshold for the remainder of the surveillance area is kept constant at $T_{conf} = 4/5$ for all runs.

Log-LikeLihood Ratio

Since the LLR approach shares the same track initiation scheme as the M-out-of-N method, whereby new tentative tracks are initiated from all unassociated measurements, the use of JPDA for processing the tentative tracks leads to in-

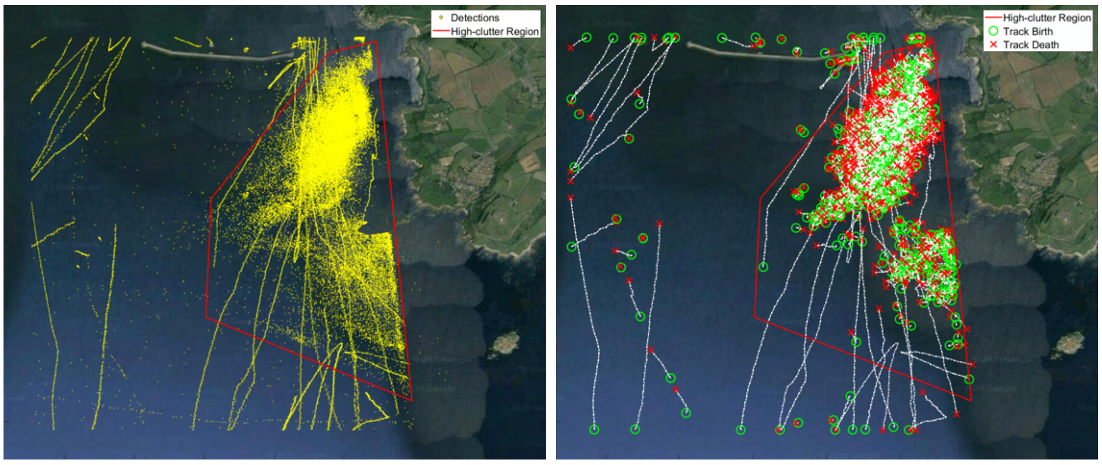


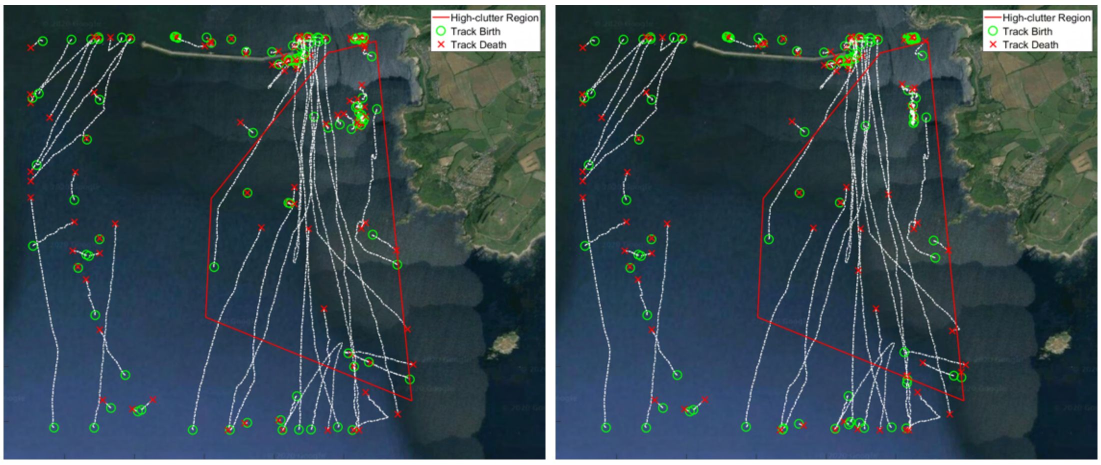
Figure 4.16: Visualisation of the examined dataset (left) and the GNN-LLR tracker output with uniform clutter rate (right).

tractable results. Hence, a GNN algorithm is once again utilised here to maintain tractable performance, and the notation *GNN-LLR* will be used to refer to this particular tracker configuration.

The tracker output for uniform clutter rate $\lambda_{FA} = 10$ and track management parameters set as $\alpha = 10^{-6}$, $\beta = 0.1$ and $THD = 6.7$ is shown in Figure 4.16. As expected, the tracker exhibits an elevated number of false tracks in the high-clutter region, caused by the increased number of false alarms, which are not captured correctly by the existing clutter model. In the interest of reducing the number of false tracks in the region of dense clutter, we first apply the regional clutter rate $\bar{\lambda}_{HC} = 50$ for that area. The resulting tracker output is shown in Figure 4.17a, where it can be observed that the majority of the false tracks are removed, without affecting greatly the quality of the true tracks. Continuing, an attempt is made to further improve on the performance of the tracker by decreasing the value of of the false track confirmation probability α to 10^{-8}, thus resulting in an increase of the confirmation threshold, which now becomes $T_{high} = 20.61$. The tracks generated by the tracker using the modified confirmation threshold are visualised in Figure 4.17b. As expected, this modification has resulted in a reduction of the amount of false tracks, at the expense however of some cases of delayed track initiation and potentially missed tracks.

SMC-PHD with Existence Probabilities

Unlike the previous track management approaches, which necessitated the use of data association algorithms to process tentative tracks and were therefore greatly



(a) $\bar{\lambda}_{HC} = 50$ and $\alpha = 10^{-6}$ (b) $\bar{\lambda}_{HC} = 50$ and $\alpha = 10^{-8}$

Figure 4.17: Comparison between the output of the GNN-LLR tracker for the examined dataset of Scenario 2 for a range of parameters: (a) Regional clutter rate $\bar{\lambda}_{HC} = 50$ and $\alpha = 10^{-6}$; (b) Regional clutter rate $\bar{\lambda}_{HC} = 50$ and $\alpha = 10^{-8}$.

hindered computationally by the presence of dense clutter, this method manages to maintain tractable performance through its use of an SMC-PHD filter to track the density of tentative tracks. This is due to the inherent ability of the SMC-PHD filter, like all PHD filters, to circumvent the computationally intensive data association procedures (see Section 4.5). As before, the notation *SMC-PHD-EP* will be used to refer to this particular tracker configuration.

The tracker output for uniform clutter rate $\lambda_{FA} = 10$ and track confirmation and deletion probabilities set as $P_{conf} = 0.8$ and $P_{del} = 0.1$, respectively, is shown in Figure 4.18. Once again, it can be observe that the use of a uniform clutter rate results in a high number of false tracks being generated by the tracker in with the region of dense clutter. Therefore, the tracker is configured with the regional clutter rate $\bar{\lambda}_{HC} = 50$ for the region of clutter and the trial is repeated. The tracks generated by the tracker using this particular parameterisation is shown in Figure 4.19a, where it can be clearly seen that the amount of false tracks has been significantly reduced. In an attempt to further minimise the presence of false tracks, a final trial run is performed for which the confirmation probability is set to $P_{conf} = 0.9$ for tracks that are evaluated for confirmation within the region of high clutter. Figure 4.19b illustrates the tracker output for this run. Although it is not clearly obvious from the respective figure, the increased confirmation threshold does result in a minor delay of track confirmation, however the reduction of false tracks is evidently more noticeable. Therefore, it is concluded this particular setting of parameters yields the most reliable results.

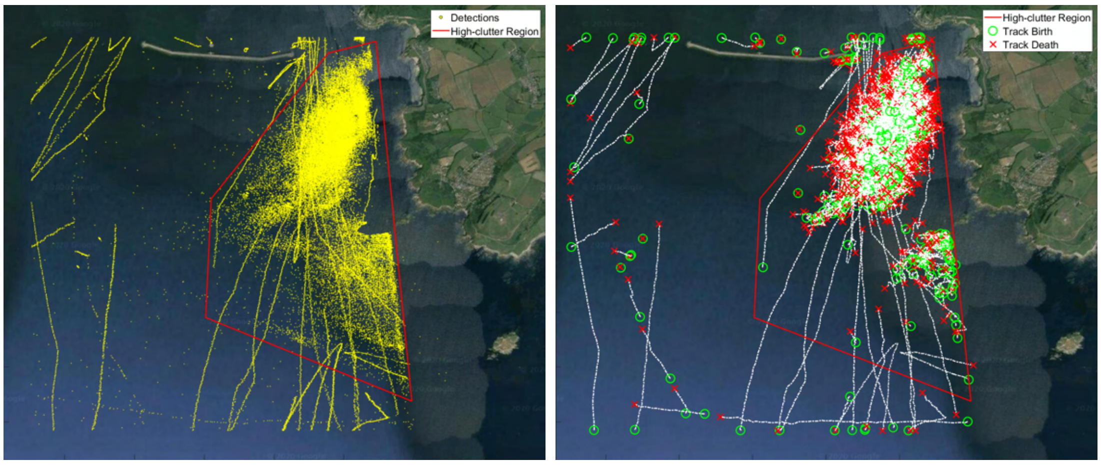
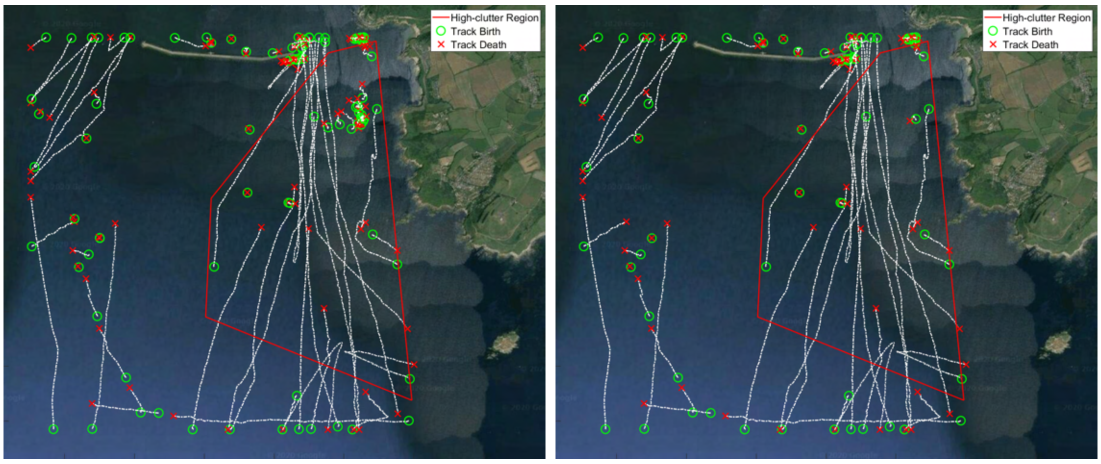


Figure 4.18: Visualisation of the examined dataset (left) and the SMC-PHD-EP tracker output with uniform clutter rate (right).



(a) $\bar{\lambda}_{HC} = 50$ and $P_{conf} = 0.8$ (b) $\bar{\lambda}_{HC} = 50$ and $P_{conf} = 0.9$

Figure 4.19: Comparison between the output of the SMC-PHD-EP tracker for the examined dataset of Scenario 2 for a range of parameters: (a) Regional clutter rate $\bar{\lambda}_{HC} = 50$ and $P_{conf} = 0.8$; (b) Regional clutter rate $\bar{\lambda}_{HC} = 50$ and $P_{conf} = 0.9$.

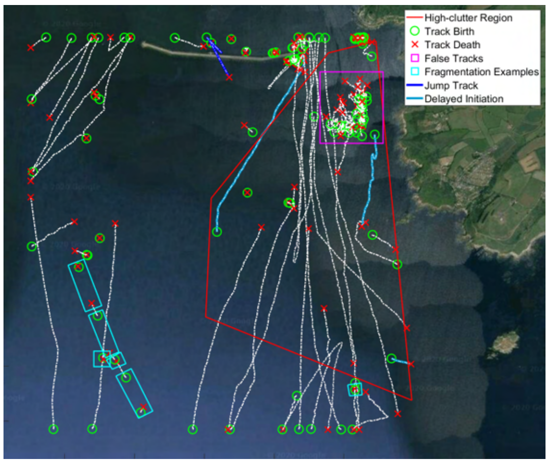
Comparison

A comparison is performed between the tracks produced by each of the algorithms considered above, for the respective set of optimal parameters in each case. As before, since there is no ground-truth data, it is not possible to produce a quantitative comparison. Therefore, a qualitative comparison is performed by means of visual evaluation and an attempt has been made to identify and highlight observable differences between the different tracker outputs. Figure **??** contains a sub-figure illustrating the tracks produced by each of the algorithms. The advantages and deficiencies for each tracker are also highlighted. To avoid repetition, readers are advised to refer to the explanation provided in the Comparison subsection of Section 4.7.5.1 for a reference to the terminology used to refer to the various identified features.

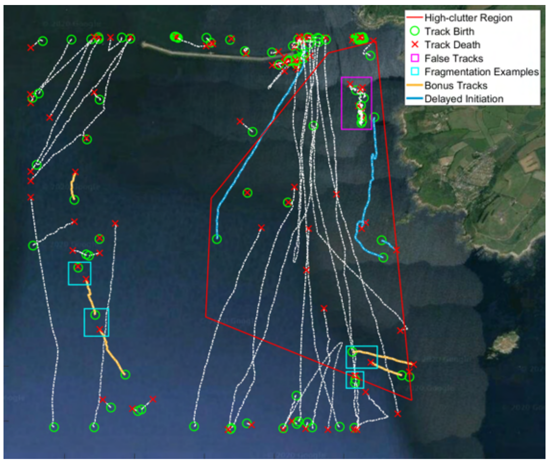
Overall, the GNN-MofN tracker (Figure 4.20a) can be seen to produce inferior results compared to the other two algorithms. The tracker is found to exhibit the highest number of false tracks, while also suffering from elevated levels of delayed initiation and missed tracks. As previously discussed and illustrated (Figure 4.15), any attempt to increase the confirmation threshold, so as to further reduce the amount of generated false tracks, results in an increase in the number of missed true tracks, as well as a performance deterioration due to delayed initiation. Furthermore, even outside the region of high clutter, the algorithm can be seen to produce a substantial amount of fragmentation, while at the same time demonstrating an inability to correctly initiate and maintain tracks for faint targets.

The GNN-LLR tracker (Figure 4.20b) manages to overcome some of the issues met by the GNN-MofN tracker. The algorithm shows a significant improvement, compared to the GNN-MofN case, in terms of the amount of generated false tracks in the region of dense clutter. Continuing, although some cases of delayed initiation are still observable, their effect is marginally alleviated, while no obvious cases of jump tracks were observed. Some cases of bonus tracks are identified for seemingly faint targets which were not picked up by the GNN-MofN tracker, while there exists (at least) one bonus track that is not found in either of the other two trackers. Finally, track fragmentation issues are observed, including some missed true tracks, which are seen to be less severe than those observed in the case of GNN-MofN, but more prominent when compared to SMC-PHD-EP.

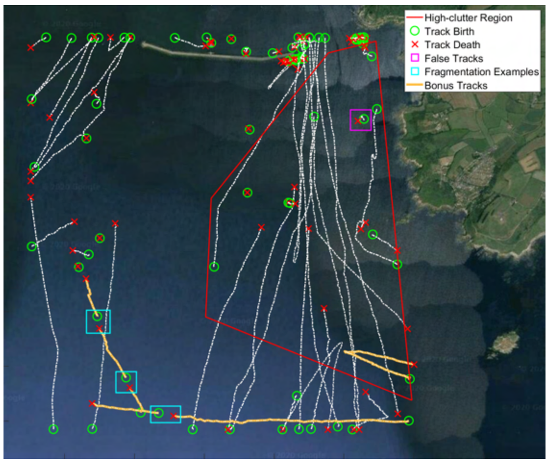
The best performance is found to be demonstrated by the SMC-PHD-EP algorithm (Figure 4.20c). Firstly, the amount of false tracks in the region of dense clutter has been effectively reduced to a single case, which could potentially also be related to a true static target (buoy). Furthermore, the tracker was found to not exhibit any obvious signs of delayed initiation, especially when compared to the other two approaches. Tracking performance in relation to faint targets is also found to be superior to the other two algorithms. The most prominent example is seen in the presence of a well maintained bonus track at the bottom of Figure (Figure 4.20b), which, although marginally fragmented, cannot be found in any of the outputs from the other trackers. Finally, some examples of track fragmentation can be observed, however all cases can be seen to occur for tracks generated from faint targets, which can either also be found in the other plots or they relate to true tracks not generated by the other algorithms altogether.



(a) GNN-MofN



(b) GNN-LLR



(c) SMC-PHD-EP

Figure 4.20: Visualisation of the resulting tracks produced by the GNN-MofN, GNN-LLR and SMC-PHD-EP algorithms for the dataset of Scenario 2.

4.8 Conclusion

This chapter extended the discussion from Chapter 3 to the case of Multi-Target Tracking. An introduction to the Multi-Target Bayes Filter was presented, while drawing relation to the Standard Bayes Filter. A structural overview of conventional Multi-Target trackers was presented, with special focus to the utilised Data

Association and Track Management methods. Along the same discussion, the author presented drew a relation between the Joint Probabilistic Data Association (JPDA) and Joint Integrated PDA algorithms, highlighting how the latter can be performed using the same constructs. Furthermore, a formulation of the Probability Hypothesis Density (PHD) filter as an approximation to the Multi-target Bayes Filter was briefly discussed. Finally, the author demonstrated the application of a radar track initiation technique which utilises a PHD filter to model the density of uninitiated targets. Simulation results were shown that showcased the merits of the approach, followed by a demonstration of results achieved by applying the method to real data extracted from a commercial radar.

CHAPTER 5

Parameter Estimation in Dynamical Markov Models

5.1 Introduction

Kalman Filters [31] are in widespread use in a plethora of fields, from the estimation of stock prices and monetary aggregates, to the navigation, tracking and control of mobile robots. A highly challenging task in all of the above applications is the accurate modelling of the dynamics that govern the evolution of the processes to be estimated. Linear Gaussian state-space models [107] are widely used to describe the dynamics of Linear Dynamical Systems (LDSs), and the term "modeling" in this context refers to the estimation of the optimal state-space parameters, which define a given model.

In general, Linear Gaussian state-space models can be described by the following equations:

$$\mathrm{x}_k = F\mathrm{x}_{k-1} + B\mathrm{u}_k + \mathrm{q}_k, \quad \mathrm{q}_k \sim \mathcal{N}(0, Q) \tag{5.1}$$

$$\mathrm{y}_k = H\mathrm{x}_k + \mathrm{r}_k, \quad \mathrm{r}_k \sim \mathcal{N}(0, R) \tag{5.2}$$

where y_k is the corrupted measurement at time t_k, x_k is the hidden state variable, u_k is a control input, q_k and r_k are the process and measurement noise components, assumed to be i.i.d. zero-mean Gaussians with corresponding covariances Q and R, while F, H and B denote the time-invariant transition, measurement and control gain matrices, respectively.

Ghahramani & Hinton [108] and Digalakis et al. [109] demonstrate the applicability of the EM algorithm [110] to parameter estimation for a LDS, but only provide a derivation of the equations in the case of no control inputs (i.e. $B\mathrm{u}_k \triangleq 0$). Holmes [111] do include a control input in their notes, but only consider the special case of a stochastic exponential growth model, where F and H reduce to identity matrices, and assumes $B\mathrm{u}_k = B$ to be constant. Gibson & Ninness [112] present a robust method for estimating the parameters of a more generic case of multivariable dynamical systems using EM, however a derivation of the reduced set of equations, which conform to the more specific scenario with which this paper is concerned, is not straight-forward. Similar work, to the one presented herein, has also been done by Blocker [113], however some terms appear to be missing from the proposed solutions to the EM equations, potentially resulting from the use of approximations, which have not been explicitly stated.

The content presented herein is derived from and incorporates text from a paper that has been prepared for submission to the IEEE Signal Processing Letters, for which the author is the main author. In this chapter, the objective is to present a brief derivation of the complete set of equations required to perform parameter estimation for a LDS with control inputs, as well as a straightforward formulation of the relevant algorithmic EM steps. Section 5.2 begins by defining the complete log-likelihood equation for a generic LDS with control inputs, and proceeds by presenting a brief derivation of the EM equations, used to achieve the local optima. Section 5.3 introduces a case study conducted on a segway platform to demonstrate the applicability and achieved performance of the proposed method. Simulated numerical results are also shown in Section 5.3, along with graphs indicating the existing correlations between different parameters. Finally, conclusions are drawn and discussed in Section 5.4.

5.2 Expectation Maximization

5.2.1 Maximum Likelihood Estimation

The process of learning a LDS using the EM algorithm involves finding the set of parameters $\theta = \{F, B, H, Q, R\}$ that maximize the log-likelihood function $\mathcal{L}(\theta)$:

$$\mathcal{L}(\theta) \triangleq \log p_\theta(\mathrm{y}_{1:N}, \mathrm{x}_{1:N} | \mathrm{u}_{1:N}) \tag{5.3}$$

where $\mathrm{y}_{1:N} = \mathrm{y}_1 \dots \mathrm{y}_N$, and similarly for $\mathrm{x}_{1:N}$ and $\mathrm{u}_{1:N}$.

Making use of the Markov property implicit in the models shown in (5.1) and (5.2), an expression for $\mathcal{L}(\theta)$ can be obtained in the form of (5.4), as shown below:

$$\begin{aligned}\mathcal{L}(\theta) = &\log p(\mathrm{x}_1) + \sum_{k=2}^{N} \log p_\theta(\mathrm{x}_k|\mathrm{x}_{k-1}, \mathrm{u}_k) \\ &+ \sum_{k=1}^{N} \log p_\theta(\mathrm{y}_k|\mathrm{x}_k) \\ = &-\frac{1}{2}(\mathrm{x}_1 - \mu_1)V_1^{-1}(\mathrm{x}_1 - \mu_1)^T - \frac{1}{2}\log|V_1| \\ &- \sum_{k=2}^{N} \frac{1}{2}(\mathrm{x}_k - F\mathrm{x}_{k-1} - B\mathrm{u}_k)Q^{-1}(\mathrm{x}_k - F\mathrm{x}_{k-1} - B\mathrm{u}_k)^T \\ &- \sum_{k=1}^{N} \frac{1}{2}(\mathrm{y}_k - H\mathrm{x}_k)R^{-1}(\mathrm{y}_k - H\mathrm{x}_k)^T - \frac{N}{2}\log|R| \\ &- \frac{N-1}{2}\log|Q| - \frac{N(D_x + D_y)}{2}\log 2\pi \end{aligned} \tag{5.4}$$

where $p_\theta(.)$ denotes a pdf evaluated based on the set of parameters in θ, $p(\mathrm{x}_1) = \mathcal{N}(\mathrm{x}_1; \mu_1, V_1)$ is the initial hidden state distribution, $p_\theta(\mathrm{y}_k|\mathrm{x}_k) = \mathcal{N}(\mathrm{y}_k; H\mathrm{x}_k, R)$ is the measurement model pdf, $p_\theta(\mathrm{x}_k|\mathrm{x}_{k-1}, \mathrm{u}_k) = \mathcal{N}(\mathrm{x}_k; F\mathrm{x}_{k-1} + B\mathrm{u}_k, Q)$ is the dynamic model pdf and D_x, D_y denote the dimensionality of the hidden state variable and the measurement data, respectively.

Once an initial set of parameters θ_1 is identified, application of the EM algorithm is performed by recursive repetition of Estimation and Maximisation steps, until the algorithm converges to the optimal set of parameters. On each iteration of the EM algorithm, the optimised set of parameters $\hat{\theta} = \{\hat{F}, \hat{H}, \hat{B}, \hat{Q}, \hat{R}\}$ can be estimated on the basis of maximising the lower-bound $\mathcal{Q}(\theta)$ of $\mathcal{L}(\theta)$, i.e. :

$$\mathcal{Q}(\theta) = \mathbb{E}[\mathcal{L}(\theta)|\mathrm{y}_{1:N}] \tag{5.5}$$

$$\hat{\theta} = \arg\max_{\theta}\ \mathcal{Q}(\theta) \tag{5.6}$$

Sections 5.2.2-5.2.3 introduce the main set of equations and operations involved in executing the Estimation (E) and Maximisation (M) steps, respectively. In Section 5.2.4, a method is presented for obtaining a good set of initial estimates θ_1, while Section 5.2.5 discusses an approach to determine when the algorithm has converged. For an in-depth explanation and derivation of (5.4)-(5.6) the reader is advised to refer to [114]. Assuming that all parameters within θ are independent random variables, the above maximisation process can be simplified and computed separately for each parameter, as we show in Section 5.2.3.

5.2.2 E Step

The Estimation (E) step is employed with the aim of obtaining maximum likelihood estimates of the hidden state distribution $p_\theta(\mathrm{x}_{1:k}|\mathrm{y}_{1:k}, \mathrm{u}_{1:k})$, over the entire time epoch $k = 1 : N$. In the case of Linear Gaussian state-space models, a closed-form solution to the above estimation problem is provided by the well known Kalman Filter & Smoother equations. These are described below.

Firstly, a feed-forward pass is performed, using the standard Kalman Filter [31] equations to compute the filtered estimates, as follows:

$$\bar{\mathrm{x}}_{k|k-1} = F\bar{\mathrm{x}}_{k-1} + B\mathrm{u}_k \tag{5.7}$$

$$\bar{V}_{k|k-1} = F\bar{V}_{k-1}F^T + Q \tag{5.8}$$

$$K_k = \bar{V}_{k|k-1}H^T(H\bar{V}_{k|k-1}H^T + R)^{-1} \tag{5.9}$$

$$\bar{\mathrm{x}}_k = \bar{\mathrm{x}}_{k|k-1} + K_k(\mathrm{y}_k - H\bar{\mathrm{x}}_{k|k-1}) \tag{5.10}$$

$$\bar{V}_k = (I - KH)\bar{V}_{k|k-1} \tag{5.11}$$

where $\bar{\mathrm{x}}_{k|k-1}, \bar{V}_{k|k-1}$ denote the predicted estimates, while $\bar{\mathrm{x}}_k, \bar{V}_k$ denote the filtered estimates at time t_k, and the recursion is initialised by setting $\bar{\mathrm{x}}_1 = \mu_1$, $\bar{V}_1 = V_1$.

Once the filtered estimates have been attained, a backward recursion can be performed, using the Rauch-Tung-Striebel (RTS) smoothing equations [115], in order to compute the maximum likelihood (a.k.a. smoothed) estimates $\hat{\mathrm{x}}_k = \mathbb{E}[\mathrm{x}_k]$, $\hat{V}_k = Var(\mathrm{x}_k)$, $P_k = \mathbb{E}[\mathrm{x}_k\mathrm{x}_k^T] = \hat{V}_k + \hat{\mathrm{x}}_k\hat{\mathrm{x}}_k^T$ and $P_{k,k-1} = \mathbb{E}[\mathrm{x}_k\mathrm{x}_{k-1}^T] = \hat{V}_{k,k-1} + \hat{\mathrm{x}}_k\hat{\mathrm{x}}_{k-1}^T$, as follows:

$$J_{k-1} = \bar{V}_{k-1}F^T(\bar{V}_{k|k-1})^{-1} \tag{5.12}$$

$$\hat{\mathrm{x}}_{k-1} = \bar{\mathrm{x}}_{k-1} + J_{k-1}(\hat{\mathrm{x}}_k - \bar{\mathrm{x}}_{k|k-1}) \tag{5.13}$$

$$\hat{V}_{k-1} = \bar{V}_{k-1} + J_{k-1}(\hat{V}_k - \bar{V}_{k|k-1}){J_{k-1}}^T \tag{5.14}$$

$$\hat{V}_{k,k-1} = \bar{V}_k{J_{k-1}}^T + J_k(\hat{V}_{k+1,k} - F\bar{V}_k){J_{k-1}}^T \tag{5.15}$$

where $\hat{V}_{k,k-1}$ is the lag one covariance, with:

$$\hat{V}_{N,N-1} = (I - K_N H)F\bar{V}_{N-1} \tag{5.16}$$

5.2.3 M Step

The Maximization (M) step is performed with the aim of estimating the set of parameters θ that maximize the log-likelihood function $\mathcal{Q}(\theta)$. With the assumption that all parameters within θ are independent random variables and that the likelihood, conditional on each parameter, is Gaussian, i.e. concave, its global maximum can be identified by taking the partial derivative, with respect to each parameter, and setting it equal to zero. Below we present the equations used to update the parameters on each EM iteration:

- Prior mean μ_1 and covariance V_1:

$$\begin{aligned} \mu_1 &= \hat{\mathrm{x}}_1 \\ V_1 &= \hat{V}_1 \end{aligned} \tag{5.17}$$

- State Transition Matrix F:

$$\begin{aligned} \frac{\partial \mathcal{Q}}{\partial F} &= \sum_{k=2}^{N} (P_{k,k-1} - F P_{k-1} - B \mathrm{u}_k \hat{\mathrm{x}}_{k-1}^T) \\ \therefore \hat{F} &= \left(\sum_{k=2}^{N} (P_{k,k-1} - B \mathrm{u}_k \hat{\mathrm{x}}_{k-1}^T) \right) \left(\sum_{k=2}^{N} P_{k-1} \right)^{-1} \end{aligned} \tag{5.18}$$

- Similarly for the Measurement Matrix H and Control Matrix B:

$$\hat{H} = \left(\sum_{k=1}^{N} \mathrm{y}_k \hat{\mathrm{x}}_k^T \right) \left(\sum_{k=1}^{N} P_k \right)^{-1} \tag{5.19}$$

$$\hat{B} = \left(\sum_{k=2}^{N} (\hat{\mathrm{x}}_k \mathrm{u}_k^T - F \hat{\mathrm{x}}_{k-1} \mathrm{u}_k^T) \right) \left(\sum_{k=2}^{N} \mathrm{u}_k \mathrm{u}_k^T \right)^{-1} \tag{5.20}$$

- Process Variance Matrix Q:

$$\begin{aligned}\frac{\partial \mathcal{Q}}{\partial Q^{-1}} =& -(N-1)Q + \sum_{k=2}^{N}\Big(P_k + FP_{k-1}F^T \\ & - P_{k,k-1}F^T - FP_{k-1,k} - \hat{\mathrm{x}}_k\mathrm{u}_k^T B^T - B\mathrm{u}_k\hat{\mathrm{x}}_k^T \\ & + F\hat{\mathrm{x}}_{k-1}\mathrm{u}_k^T B^T + B\mathrm{u}_k\mathrm{u}_k^T B^T + B\mathrm{u}_k\hat{\mathrm{x}}_k^T F^T\Big) \\ \therefore\ \hat{Q} =& \frac{1}{N-1}\sum_{k=2}^{N}\Big(P_k + FP_{k-1}F^T - P_{k,k-1}F^T \\ & - FP_{k-1,k} - FP_{k-1,k} - \hat{\mathrm{x}}_k\mathrm{u}_k^T B^T - B\mathrm{u}_k\hat{\mathrm{x}}_k^T \\ & + F\hat{\mathrm{x}}_{k-1}\mathrm{u}_k^T B^T + B\mathrm{u}_k\mathrm{u}_k^T B^T + B\mathrm{u}_k\hat{\mathrm{x}}_k^T F^T\Big)\end{aligned} \tag{5.21}$$

- Similarly for the Measurement Variance Matrix R:

$$\hat{R} = \frac{1}{N}\sum_{k=1}^{N}(\mathrm{y}_k\mathrm{y}_k^T - \mathrm{y}_k\hat{\mathrm{x}}_k^T H^T - H\hat{\mathrm{x}}_k\mathrm{y}_k^T + HP_kH^T) \tag{5.22}$$

where, in (5.18)-(5.22) we have used the fact that $\frac{\partial \mathbb{E}[x]}{\partial y} = \mathbb{E}\left[\frac{\partial x}{\partial y}\right]$. The reader should note that (5.17)-(5.22) are similar but different to the results in [113] and equivalent to those in [112], albeit articulated using the form considered in [108, 109, 111, 115].

5.2.4 Initial Parameter Settings

The effectiveness of EM depends highly on the provided set of initial parameters $\theta_1 = \{F_1, H_1, B_1, Q_1, R_1\}$. Bad setting of initial parameters may lead to incorrect results, while suitable initial value setting could significantly improve convergence speed. Thus, according to good statistical practice, if some prior knowledge about the system's parameters exists, the best course of action is to perform initialisation based on this information.

Below we present a method for obtaining initial estimates, assuming the hidden states are fully observable, i.e. $D_x = D_y$.

- State Transition Matrix F_1: As the time interval between timesteps is minimised, the inter-step state variation is also reduced, causing the state transition matrix to approach the identity matrix. Thus, we set F_1 as follows:

$$F_1 = \mathbb{I}_{D_x} \tag{5.23}$$

where $\mathbb{I}_{D_x}$ is an identity matrix of size D_x.

- Measurement Matrix H_1: As is often the case, raw data from sensors are utilised without scaling. Thus, H_1 can be initialised as follows:

$$H_1 = \mathbb{I}_{D_y} \tag{5.24}$$

- Control Matrix B_1: Utilising (5.23)-(5.24) and assuming that $w_k = 0$, $\mathrm{r}_k = 0$ in (5.1)-(5.2), B_1 can be computed as follows:

$$B_1 = \frac{1}{N-1} \sum_{k=1}^{N-1} \frac{\mathrm{y}_{k+1} - \mathrm{y}_k}{\mathrm{u}_k + \epsilon} \tag{5.25}$$

- Process & Measurement Noise Matrices Q_1 & R_1 [111]:

$$\tilde{y}_k = \sum_{i=k}^{k+3} \mathrm{y}_i, \forall k \in [1, \tilde{N}], \tilde{N} = N - 3 \tag{5.26}$$

$$Q_1 = \frac{1}{3} \left[Var(\tilde{y}_{4:\tilde{N}} - \tilde{y}_{1:\tilde{N}-3}) - Var(\tilde{y}_{2:\tilde{N}} - \tilde{y}_{1:\tilde{N}-1}) \right] \tag{5.27}$$

$$R_1 = \frac{1}{2} \left[Var(\tilde{y}_{2:\tilde{N}} - \tilde{y}_{1:\tilde{N}-1}) - Q_1 \right] \tag{5.28}$$

- Prior mean μ_1 and covariance V_1:

$$\mu_1 = \mathrm{y}_1 \tag{5.29}$$

$$V_1 = R_1 \tag{5.30}$$

It should also be possible to use (5.27) and (5.28) to estimate Q_1 and R_1 first, then proceed to estimate B_1 by performing independent Monte-Carlo simulations to sample w_k, r_k, thus removing the $w_k = 0$ and $\mathrm{r}_k = 0$ assumption. The same process could also be followed in cases where either Q_1 or R_1 are known a priori. Although possible, this would lead to a slightly more complex form of (5.25), while potentially introducing a correlation between the estimation accuracy of B_1 and that of Q_1 and R_1, and therefore this approach has not been considered here.

It is important to also note that the above parameter settings are derived based on the assumption that hidden states are fully observable and, consequently, on the identity assumptions for F_1 and H_1. Even though this can be a reasonable assumption for a vast number of low-dimensional problems, the effectiveness of the above process can be greatly impacted by both the number and the dimensionality of the parameters to be estimated.

5.2.5 Checking for convergence

Convergence of EM has been proven in [110, 116] where it was shown that, under mild conditions, the algorithm is guaranteed to converge to a local maximum on the log-likelihood plane. Despite this fact, the time required for optimal convergence can vary greatly depending on the problem, or could even be infeasible, due to machine precision errors that may occur during the log-likelihood computation.

To avoid such pitfalls, a mechanism can be put in place to detect when the algorithm tends to converge. One very common criterion, and also the one utilised in this paper, is based on setting a convergence threshold dl_{min} on the difference dl_i between the incomplete data likelihood $l(\theta_k)$, evaluated at two consecutive EM iterations, i and $i-1$, such that the algorithm is halted when dl_i falls bellow dl_{min}:

$$\begin{aligned} l(\theta_i) \triangleq \sum_{k=1}^{N} \log p_{\theta_k}(\mathrm{y}_k|\mathrm{x}_{k|k-1}) &\propto -\frac{N}{2}\log|S_k| \\ &- \frac{1}{2}\sum_{k=1}^{N}(\mathrm{y}_k - H_k\bar{\mathrm{x}}_{k|k-1})S_k^{-1}(\mathrm{y}_k - H_k\bar{\mathrm{x}}_{k|k-1})^T \end{aligned} \tag{5.31}$$

where $\bar{\mathrm{x}}_{k|k-1}$ is defined in (5.7), $S_k = H_k\bar{V}_{k|k-1}H_k^T + R_k$ is the innovation (predicted measurement) covariance at time t_k, and the algorithm is allow to run while:

$$dl_i = l(\theta_i) - l(\theta_{i-1}) \geq dl_{min} \tag{5.32}$$

However, it is worth noting that convergence of the EM algorithm does not guarantee globally optimal results, unless the problem is such that the log-likelihood function is concave. As shown in [116], under certain conditions, the algorithm may not even converge to local optima, instead getting trapped at existing stationary points. The above presented convergence criteria may make this problem even more profound, by causing premature convergence in cases where the log-likelihood function contains regions of negligible slope. Nevertheless, this is considered a reasonable trade-off between speed and accuracy, which can be avoided all together by a carefully selected convergence threshold.

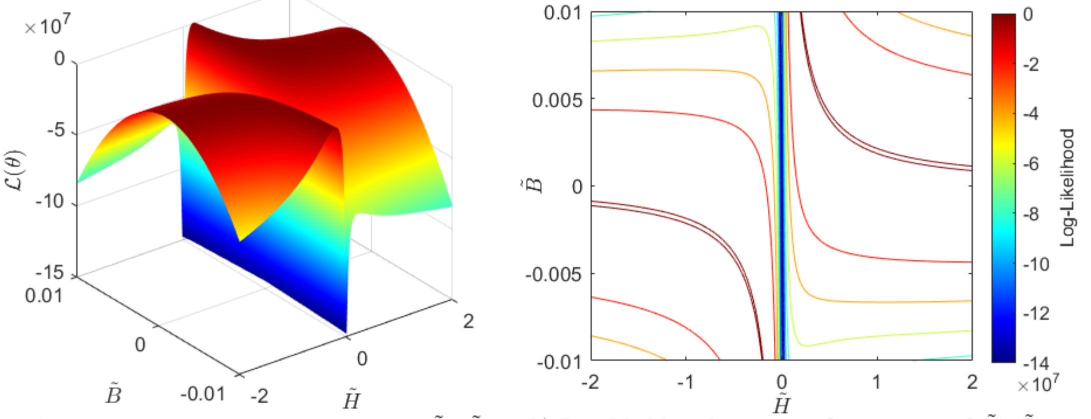
5.2.6 Uniqueness of parameter estimates

By analysing (5.1) and (5.2), it should become evident that, given x_{k-1} and u_k, there exist an infinite number of parameter combinations that can yield the same measurement y_k. To demonstrate this fact, consider the case where a (non-zero) scaling parameter $\tilde{c}$ is introduced in (5.1), such that $\tilde{\mathrm{x}}_k = \tilde{c}\mathrm{x}_k$. Then, (5.1) - (5.2) can be rewritten as follows:

$$\tilde{\mathrm{x}}_k = (\tilde{c}F)\,\mathrm{x}_{k-1} + (\tilde{c}B)\,\mathrm{u}_k + \tilde{\mathrm{q}}_k, \quad \tilde{\mathrm{q}}_k \sim \mathcal{N}(0, \tilde{c}^2 Q) \tag{5.33}$$

$$\mathrm{y}_k = \frac{H}{\tilde{c}}\tilde{\mathrm{x}}_k + \tilde{\mathrm{r}}_k, \quad \tilde{\mathrm{r}}_k \sim \mathcal{N}(0, \frac{R}{\tilde{c}^2}) \tag{5.34}$$

This phenomenon results in the introduction of multiple extended modes in the log-likelihood plane, the number of which is related to the number of possible values of $\tilde{c}$. An example visualisation of this phenomenon is depicted in Fig. 5.1, where it should be clear to see that the correlation between $\tilde{H}$ and $\tilde{B}$ follows closely the relations expressed in (5.33)-(5.34).



a) Log-likelihood plot for a range of $\tilde{H}$, $\tilde{B}$ b) Log-likelihood contours for a range of $\tilde{H}$, $\tilde{B}$

Figure 5.1: Log-likelihood plot (a) and contours (b) evaluated at a range of values for parameters $\tilde{H}$ and $\tilde{B}$, given $H = 1$ and $B = 0.002$.

With this in mind, we can see that if none of the parameters are known a priori, the EM algorithm can converge to any of an infinite number of possible combinations of the parameters. What is more, the choice of initial parameter settings will have a significant effect on the results, as EM will naturally converge to the nearest local maximum.

To circumvent this issue, one of the parameters in θ can be constrained throughout the learning process, thus placing a bound on the value of $\tilde{c}$. In many cases, this can be achieved if the sensor characteristics are known (e.g. from a specification document), in which case $\tilde{H}$ and/or $\tilde{R}$ can be inferred.

5.3 Experimental Results

5.3.1 Case Study: Segway System

To benchmark the proposed algorithm, a real segway SISO system is used to generate a series of control inputs u_k, corresponding to the power supplied to the servo-motors to ensure that the platform maintains a near upright position (such that the model is well approximated as linear). The state-space model is formed such that the hidden state x_k corresponds to the angular (tilt) acceleration of the platform, while y_k relates to noisy measurements of (a function of) x_k, received via an accelerometer mounted on the platform.

For the purposes of benchmarks, measurements y_k are in fact generated via simulation. Although it would be possible to extract real measurements from the platform, evaluating the performance of the algorithm would be challenging: the very reason for developing this method is that, for the platforms we have access to, we do not have precise sensor characteristics. To overcome this issue, the extracted control inputs are used, along with preset values for θ_{true}, to simulate the system and extract both y_k and ground truth data for x_k.

The set of initial parameters, used to initialise the algorithm in each of the experiments, is generated according to the process described in Section 5.2.4. Although we believe that this approach may have some generic utility, we do recognise that it might be specific to the models considered herein. Finally, convergence of the algorithm is judged based on (5.32), with $dl_{min} = 10^{-5}$.

5.3.2 Unconstrained Parameters

The first set of tests is performed based on the assumption that no prior information about the parameters is available. Thus, all parameters are left unconstrained throughout the learning process. Each row in Table 5.1, corresponds to a different experiment, each performed over 100 Monte-Carlo runs, using a distinct combination of ground truth parameter settings. For each experiment, both the initial and estimated parameter values are shown, as well as a 95% confidence interval.

A first observation can be made with respect to the generated initial parameter estimates, which can be seen to fall in close proximity to the true values in all but the last experiment, where both H_{true} and F_{true} violate the identity assumption of (5.23)-(5.24). Continuing, we observe that the parameter estimation performance varies between experiments. This can be attributed to the issues discussed Section 5.2.6. More specifically, as all parameters are left unconstrained, the algorithm can converge to a number of different parameter combinations, the choice of which is highly influenced by the initial parameters. As a result, we can observe that the worst estimation performance is achieved by the last experiment, where the initial estimates fall far from the true values.

Table 5.1: Parameter Estimation with Unconstrained Parameters

	F	**H**	**B**	**Q**	**R**
True	1.0000	1.5000	-0.0020	0.0100	0.0100
Initial	1.0000	1.0000	$0.010078 \pm 118.1959\%$	$0.0005624 \pm 4.2004\%$	$0.13866 \pm 3.4714\%$
Estimated	$0.99994 \pm 0.019496\%$	$0.95575 \pm 4.7373\%$	$-0.0031427 \pm 5.2108\%$	$0.024897 \pm 11.5474\%$	$0.0099215 \pm 7.9157\%$
True	1.0000	0.5000	-0.0020	0.0100	0.100
Initial	1.0000	1.0000	$0.0034921 \pm 99.4673\%$	$0.00019431 \pm 4.9819\%$	$0.11409 \pm 3.231\%$
Estimated	$0.99996 \pm 0.015934\%$	$0.92417 \pm 7.5458\%$	$-0.0010899 \pm 9.2494\%$	$0.0029329 \pm 19.9756\%$	$0.10001 \pm 3.1343\%$
True	0.5000	0.5000	0.2000	0.1000	0.1000
Initial	1.0000	1.0000	$0.013139 \pm 2.7159\%$	$0.098559 \pm 0.25174\%$	$39.057 \pm 0.23256\%$
Estimated	$0.50007 \pm 0.14394\%$	$4.9347 \pm 1.7161\%$	$0.020267 \pm 1.7325\%$	$0.0010173 \pm 17.5247\%$	$0.1 \pm 5.6792\%$

5.3.3 Constrained Measurement Matrix H

In this set of experiments, it is assumed that the measurement matrix H is known, and therefore it is constrained to its true value throughout the learning process. Thus, the goal here is to estimate all other parameters, given that H is fixed. The same experiments as before are repeated and the results are shown in Table 5.2. Once again, both the initial and estimated parameter values are shown, complemented by a 95% confidence interval.

From the achieved results, it becomes evident that the performance of the system has greatly improved. All parameters, including B and Q are estimated much more accurately and the correlation between initial and final estimates has been removed. Although some estimation errors still exist, their magnitude has largely decreased compared to Section 5.3.2.

Table 5.2: Parameter Re-estimation with Constrained H

	F	**H**	**B**	**Q**	**R**
True	1.0000	1.5000	-0.0020	0.0100	0.0100
Initial	1.0000	1.5000	$0.0078661 \pm 144.3946\%$	$0.00056028 \pm 4.1486\%$	$0.13834 \pm 3.2502\%$
Estimated	$0.99992 \pm 0.018887\%$	1.5000	$-0.0019989 \pm 1.63\%$	$0.0099151 \pm 5.1499\%$	$0.010109 \pm 6.936\%$
True	1.0000	0.5000	-0.0020	0.0100	0.1000
Initial	1.0000	0.5000	$0.0035615 \pm 102.8949\%$	$0.00019377 \pm 4.5276\%$	$0.11397 \pm 2.9011\%$
Estimated	$0.9999 \pm 0.021135\%$	0.5000	$-0.0020143 \pm 3.9625\%$	$0.0096162 \pm 13.9098\%$	$0.09993 \pm 2.6947\%$
True	0.5000	0.5000	0.2000	0.1000	0.1000
Initial	1.0000	0.5000	$0.013116 \pm 2.6062\%$	$0.098566 \pm 0.24239\%$	$39.0574 \pm 0.22044\%$
Estimated	$0.50014 \pm 0.13681\%$	0.5000	$0.19996 \pm 0.096441\%$	$0.097487 \pm 17.6467\%$	$0.10055 \pm 5.8327\%$

5.4 Conclusion

In conclusion, this chapter demonstrates the applicability of EM to parameter estimation for LDSs with control inputs. A derivation of the complete set of equations required to perform the estimation process has been presented, as well as a straightforward formulation of the relevant algorithmic steps. Also, an intuitive way of generating initial parameter estimates was introduced, given that certain assumption hold, and the performance implications of such initial estimates was discussed, given that no parameters are constrained throughout the learning process. The performance of the presented algorithm has been evaluated, based on a case study performed on a real segway system. The presented results show that the proposed technique achieves acceptable results, where the estimation performance can be greatly improved, if one (or more) of the parameters are known a priori.

CHAPTER 6

Maritime Video Detection & Tracking

6.1 Introduction

One major component of modern VTS systems is the video component. Research into video surveillance systems is one of the most active topics in computer vision [117], which has received increasing attention over the past decade [118]. Over this period, there have been a plethora of applications stemming from the need to develop efficient maritime video surveillance systems. The Autonomous Maritime Surveillance System (AMASS)[119] is European project aiming at the development of an autonomous surveillance system equipped with FLIR cameras. The AVITRACK[120], MAAW[121] and ARGOS[117] systems are other examples of surveillance systems based on cameras. Other applications have also emerged, which allow for the fusion of visual information with data acquired from a deployment of other sensors (e.g. radars, AIS e.t.c.). The AMFIS[122], ASV[123] and AIVS3[124] systems are all examples of such applications. Although there has been an increased focus on performing vessel detection and tracking through the use of port surveillance cameras, little work has been done, to the best of the author's knowledge, on developing systems that perform automated image capturing of vessels that are present in a given port.

Automatic Identification and Data Capture (AIDC) systems have grown increasingly popular over the years, with numerous successful applications in the Defence and Security sector, due to their inherent ability to accumulate, organise and distribute valuable information and intelligence, with the added benefit of accelerating, or even avoiding altogether, the labourious and heavily-beaurocratic manual data collections, as well as removing the elements of human error and health risk. Aside from the immediate benefit in terms of crime control and prevention in the present time, AIDC systems facilitate the means for gradually gathering and constructing a substantial dataset - in our case imagery - which can be further utilised in the research and development of ever more effective automated (vessel) classification and identification systems in the future.

This chapter aims to summarise the work performed by the author in the domain of Maritime Video Detection & Tracking. Section 6.2 initiates the discussions by introducing the reader to the basic concepts and notation used in this Chapter. Section 6.3 is then focused on the engineering challenges relating to development of accurate real-time ship detectors with the use of active Electro-Optical sensors (cameras). A review of existing approaches is presented, followed by an introduction to the state-of-the-art detectors that base their operation on Convolutional Neural Networks (CNNs). Under the same section, a custom CNN-based ship detector is introduced, followed by brief report of the followed methodologies and a comprehensive performance analysis. Continuing, Section 6.4 demonstrates an effective method for estimating and accounting for the errors induced by the camera motion in real-time. The use of an active (Pan-Tilt-Zoom) camera, introduces errors due to the inherent ability of the camera to exhibit motion. Thus, in addition to the multi-target tracking complexities discussed in Chapter 4, further measures must be employed in order to ensure that the positions of targets are estimated accurately, even after the camera has changed its orientation.

6.2 Problem Formulation

We define our state-space system on the 2-dimensional coordinate frame defined by the image plane of received video frames. Following standard practice, given a video frame I_k of dimensions (I_{width}, I_{height}), the origin $(0, 0)$ of the coordinate system is placed in the top-left corner of the image, with the positive X-axis running from left-to-right, while the positive Y-axis running in a top-to-bottom orientation. An example frame, along with appropriate illustrations of the employed coordinate system is provided in Figure 6.1.

Based on the above mindset, target states at time t_k are assumed to be represented by 2D bounding boxes, defined on the aforementioned coordinate frame,

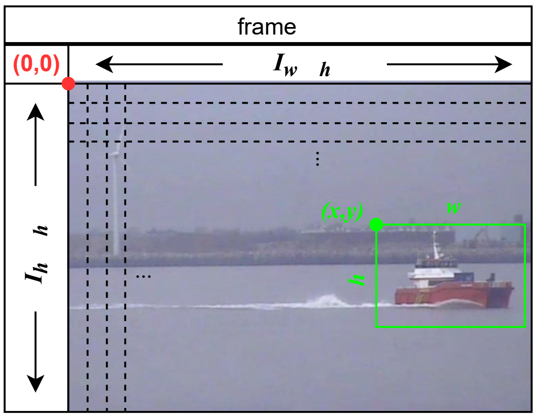


Figure 6.1: Example image frame and detection

as follows:

$$\mathrm{x}_k = [x_k, \dot{x}_k, y_k, \dot{y}_k, w_k, h_k]^T \tag{6.1}$$

where x, y denote the X-Y coordinates of the bounding box top-left corner, w, h denote the width and height of the box and $\dot{\xi}$ is used to denote the first order derivative of a given value ξ. Without loss of generality, target motion on each axis can be assumed to follow a 2-D nearly Constant Velocity model (see Section 2.2.2), while the size (width, height) of a target can be modelled according to i.i.d. Random Walk (see Section 2.2.1) processes.

At each time t_k, a set of noisy detections $Y_k = \{\mathrm{y}_k^1, \ldots, \mathrm{y}_k^{M_k}\}$ is received, where each y_k^j is of the form

$$\mathrm{y}_k = [x_k, y_k, w_k, h_k]^T \tag{6.2}$$

Thus, the transformation between measurements and target states, can be assumed to be described by a 4-D Linear-Gaussian model (see Section 2.3.1), with a pair of i.i.d. Gaussian noise components for the positional (x_k, y_k) and size (w_k, h_k) dimensions.

False detections are modelled according to a Poisson Rate with Uniform Position (see Section 2.5.1) clutter model, with Poisson distributed number of false alarms per frame, with expected rate of λ_{FA}, and Uniformly distributed position and size across the entire frame I_k. What is more, target detectability is modelled according to a Constant Detection Rate (see Section 2.4.1) model, with detection probability P_D.

6.3 A Robust CNN-based Ship Detector

Most common approaches to vessel detection depend on either motion estimation [125] and/or background segmentation [123]. However, motion-based vessel detection can experience difficulties when a boat is anchored or is on direct course toward the camera, due to the small amount of inter-frame variation, while in the case of background segmentation, the stochastic motion of waves tends to introduce statistical irregularities in the background, causing increased false alarms. Both of the aforementioned techniques have the strict requirement of the camera(s) being stationary, meaning that a trade-off needs to be made between maximum resolution and area coverage, in single-sensor scenarios.

In [126, 127, 128] the authors present a method for visual ship detection and tracking in open sea, specifically designed to work with streams acquired from non-stationary passive cameras mounted on buoys. Although effective in the given scenario, the method is based on two limiting assumptions: i) the camera is placed in the open sea where the background contains mostly the horizon, meaning that it is clear of other objects which may be falsely detected as ships; and ii) detected objects are assumed to always be present above the horizon, which, although true in the case of buoys floating on the the sea surface, it is not a valid assumption on the case of overlooking PTZ cameras. Continuing, other approaches exist that have addressed the issue of non-stationary automated active cameras [129, 130], however they suffer from certain limitations: i) they involve computation of complex features, i.e. HOG, requiring the use of hardware acceleration with programmable components, such as FPGAs, to achieve real-time performance; ii) they impose certain assumptions, e.g. detection is performed based on only the cabin of ships [130].

An alternative method for vessel detection has been demonstrated via the application of ensemble classifiers, trained offline with Haar wavelet features [131, 132], to achieve real-time detection of vessels in video frame sequences, received from a user-controlled (non-automated) Pan-Tilt-Zoom (PTZ) camera. Although Haar-feature cascade classifiers [133] have been reasonably successful in detecting objects such as faces [134] and pedestrians [135], these objects have generally very little geometrical shape variation. On the other hand, boats can come in a vast variety of sizes and shapes, meaning that (optimally) a different cascade needs to be trained for each different vessel type, such as to avoid over-generalising the trained cascade, which in turn leads to increased amounts of false detections. Continuing, even in cases of generally homogeneous objects, an additional disadvantage of Haar classifiers is their sensitivity to out-of-plane rotation. This is caused by the fact that, when rotated, the aspect ratio changes for most 3-D objects. Thus, training a single detector to handle all orientations does not work, and, once again, the general approach is to train a detector for each orientation of the object.

6.3.1 Object Detection using CNNs

The rapid development of high performance parallel computing over the past two decades has lead to the rebirth of Artificial Neural Networks (ANNs) as an active research topic, with relatively recent advancements in the field giving rise to the use of Convolutional Neural Networks (CNNs) for image classification and object detection. Such recent advancements build upon the extraction and incorporation of powerful deep features [136], which came to light and grew to prominence through ImageNet classification competition [137]. Object detection algorithms that lead this wave, while demonstrating impressive results on PASCAL Visual Object Classes (VOC) [138] challenge, included algorithms such as R-CNN[139] and OverFeat [140].

There exist two main frameworks of generic object detectors: i) Region Proposal based; and ii) Regression/Classification based. The former follows the traditional two-stage object detection pipeline, which starts by generating region proposals and then classifying each proposal into different object categories. The later perceives object detection as a regression/classification problem and adopts a fully convolutional approach to generate categories and locations in a single-stage. Computationally efficient approaches, such as You Only Look Once (YOLO) [141][142] and Single Shot Detector (SSD) [143], adopt the single-stage paradigm to achieve object detection in a single step. When initially perceived, such detectors sacrificed small amounts of accuracy for a substantial increase in computation speed. In more recent years, the introduction of even larger datasets, such as the Microsoft Common Objects in Context (COCO)[144], has lead to the development of faster two-stage detectors that can be efficiently trained and applied on these datasets. Examples of such detectors include Fast Region-based CNN (Fast-RCNN) [145], in which convolutions are shared across different region proposals to achieve computational gains, and Faster-RCNN [146] that extends Fast-RCNN by introducing a Region Proposal Network (RPN), employed to generate high quality region proposals shown to improve detection performance. A thorough analysis and trade-off comparisons, in terms of speed and accuracy, of modern CNN object detectors can be found in [147, 148].

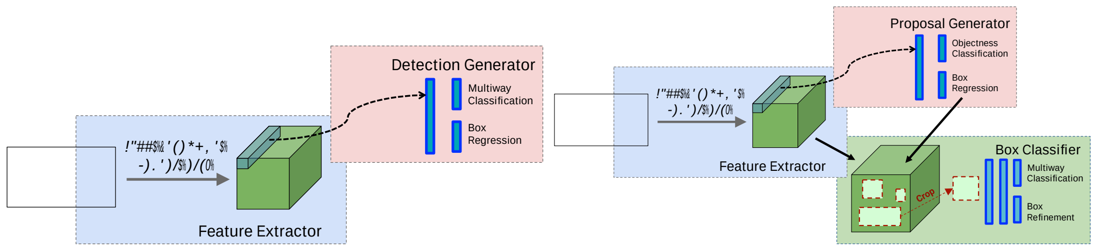
6.3.2 Transfer Learning

As elaborated in the previous section, deep learning in the domain of object detection has made considerable progress in recent years. There exist various object detection networks, with state-of-the-art performance, that have been developed and tested over the years. This has enabled scientists to tackle complex problems and yield astounding results.

Traditionally deep learning networks are designed to work in isolation. They are trained to perform a specific task, while the models have to be rebuilt from scratch once the feature-space distribution changes. However, as is typical for deep learning systems, a vast amount of data and training time are required to perform training, while, generally, data availability is scarce and comprehensive data acquisition is non-trivial. To tackle this problem, the concept of transfer learning has been widely employed by researchers and applied practitioners. Transfer learning is the idea of overcoming the isolated learning paradigm and utilising knowledge acquired from solving one task, to solve other related ones. It is a popular approach in deep learning where pre-trained models are used as the starting point for training models that solve a different problem. Therefore, the idea is that one can re-purpose, or transfer, learned knowledge (e.g. features, weights etc.) from previously trained models for training newer models on a different target dataset and/or task.

Most modern object detection CNNs can generally be decoupled as a series of inner (convolutional) layers that perform feature extraction, and 'meta-architecture' layer(s) being responsible for generating the weights pertaining to each object class (classification) and the image locations of detected objects (localisation) [148]. An example of this decoupling is shown in Figure 6.2 for the SSD and Faster-RCNN algorithms. More specifically, both SSD and Faster-RCNN used VGG-16 [149] (up to "Conv5" layer) as their base network for feature extraction in their respective seminal papers [143, 146]. Therefore, although the classification and localisation processes may vary across different algorithms, the feature extraction layers are a common theme shared by all algorithms.

Popular feature extraction networks include VGG-16 [149], Inception v1-v4 [150, 151, 152] and ResNet [153]. Studies have shown that in such networks the first layers tend to learn a standard set of general features, resembling Gabor filters or colour blobs, which occur regardless of the cost function and image dataset [136, 154]. As one progresses further into the network layers, the features tend to become more tightly coupled to the chosen dataset and task, with the final layer features being specific to the objective in hand. For example, in a network an N-dimensional output layer that has been trained toward a supervised classification problem, each of the outputs will be the specific weights attributed



(a) SSD (b) Faster RCNN

Figure 6.2: High level diagrams of the detection meta-architectures [148]

to each class. The above notions of general and specific features form the basis for transfer learning.

Transfer learning is usually performed by copying the first (base) layers of a pre-trained network, i.e. the layers that carry the general features, to the first layers of a new target network. Once this is done, the remaining layers of the target network are randomly initialised and trained using the target dataset. During the re-training, the transferred feature layers can be left frozen, meaning that their weights are not affected by the training process. Alternatively, it is possible to fine-tune the base layers by allowing the errors from the re-training process to be back-propagated into these layers. Fine-tuning, if done correctly, can lead to performance improvements, however it is also possible that allowing a network to be fine tuned can cause over-fitting.

The choice of whether or not to fine-tune the first layers of the target network depends on the size of the target dataset and the number of parameters [154]. If the target dataset is small and the number of parameters is large, fine-tuning may result in over-fitting. Since this is typically the case, the features are often left frozen. On the other hand, if a large target dataset is available and/or the number of parameters is relatively small, meaning that over-fitting is unlikely to occur, then the base features can be fine-tuned to improve performance. Of course, if the target dataset is very large, there would be little need to perform transfer learning in the first place, because the lower level filters could just be learned from scratch on the target dataset.

	Open Images v4	Self-collected	Total
Train	22,927	1,143	24,070
Test	2,548	126	2,674
Total	25,475	1,269	26,744

Table 6.1: Breakdown of the dataset used for training and evaluation

6.3.3 Experimental methods

6.3.3.1 Dataset

The process of experimentation and development of object detectors necessitates the existence of a comprehensive dataset of images that forms the basis for training and evaluation. For the purposes of this task, two main sources of data have been acquired. Firstly, the latest release of the Open Images [155] (v4) dataset was utilised and, following appropriate pre-processing, a set of 25,475 pre-labeled images of boats were extracted to complement the training process. In addition, a custom collection of 1,269 images was captured using a commercial maritime surveillance camera owned and operated by Denbridge Marine Ltd., located at Fort Perch Rock, New Brighton.

Acquisition of the custom dataset was performed via an automated process, which utilised the output of a pre-existing tracking system, developed by Denbridge Matine Ltd, which fused radar and AIS data to obtain accurate track position estimates. The developed automated image acquisition process consisted of the following steps: 1) Unobscured targets, within a radius of 1km from the camera, were identified; and 2) The camera was panned to the estimated location of the unobscured closest target, taking photos at 4 different levels of zoom (0, 20, 40 and 60%) and 10 distinct distance levels (100, 200,..., 1000m). Identification of unobscured targets was performed by utilising land polygons, extracted form Keyhole Markup Language (KML) map files, and asserting that no land polygons intersect the line of sight to a given target. Once the custom dataset had been collected, a manual process of labelling was performed, such that the locations of the ships are marked within each image.

The final combined dataset constisted of 26,744 annotated images. A training dataset was the formed by randomly selecting 90% of the total images, while the remaining 10% was set aside for testing and evaluation purposes. Figure 6.3 demonstrates some samples extracted from the generated dataset and Table 6.1 shows a summary of the utilised training and test datasets.

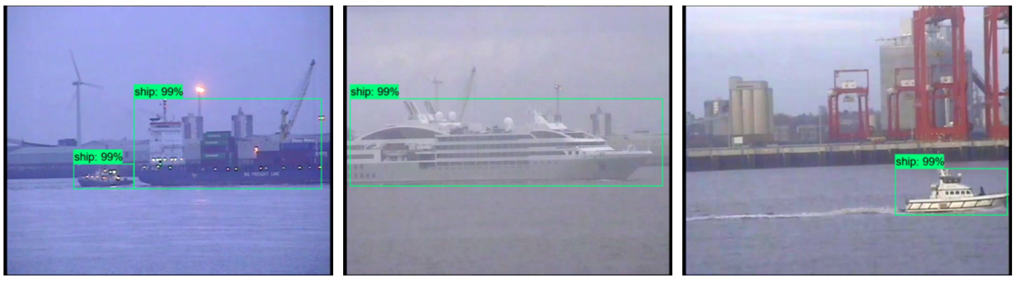


Figure 6.3: Samples extracted from the training and evaluation dataset. The ground truth annotations are depicted using green rectangles.

6.3.3.2 TensorFlow

Training and performance evaluation of the various object detection networks was performed using TensorFlow [156]. TensorFlow is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms, recently developed by Google. As part of the framework, TensorFlow offers a range of pre-trained object detection networks that allow scientists and practitioners to swiftly experiment with and develop custom object detectors. A list of all available pre-trained models can be found in [157]. The majority of the available models have been trained on either the COCO [144] or Open Images [155] datasets, both of which include a collection of images that contain annotations for boats, among a wide variety of other classes, such as vehicles (boats, cars, motorcycles, trains, etc.), household appliances (tv, laptop, fridge, etc.) and even animals. This ensures that detectors trained on the above datasets, have the potential of identifying such object in images, but more importantly the inner network layers contain classifiers that have been trained on features pertaining to these classes of objects.

6.3.3.3 Baseline Model Evaluation

Before committing to a specific detector on which to perform training, the performance of a number of existing detectors was evaluated, in terms of their respective ability to detect ships. The models selected for evaluation were chosen on the basis of providing a satisfactory trade-off between speed and accuracy, as documented in [148, 157]. As such, three models were selected for evaluation:

- *SSD Inception*: A SSD meta-architecture using the Inception V2 feature extractor.
- *SSD ResNet*: A SSD meta-architecture using the ResNet-50 feature extrac-

tor.

- *Faster-RCNN Inception*: A Faster-RCNN meta-architecture using the Inception V2 feature extractor.

Based on the review presented in [148], the SSD and Faster-RCNN detectors have been selected as they represent the two ends of the scale when it comes to accuracy and speed, with SSD being faster but also marginally less accurate, while R-FCN [158] can be seen to always fall between the two. Continuing, the Inception V2 feature extractor has been found to be significantly faster than its counterparts, while still demonstrating relative accuracy. Nevertheless, as the ResNet feature extractor was shown to provide a significant advantage over the other feature extractors, with comparable accuracy performance across SSD, Faster-RCNN and F-RCN, the combination of SSD and ResNet has also been considered, due to the significant speed advantages of the SSD. A clear demonstration of the above arguments can be found in the performance graphs of Figures 4 and 7 in [148].

6.3.3.4 Model Re-training

Having evaluated the detection performance of the baseline models on the training dataset, we proceed by re-training the models using our custom dataset. For each model, we perform training in two distinct ways:

1. Fine-tuning (FT): The models are fine-tuned by allowing the errors to be back-propagated through the entire network of each model.
2. Feature-extraction (FE): The feature extraction layers for each model are frozen, meaning that only the layers responsible for classification and localisation are trained.

TensorFlow uses a modular approach to structuring object detection networks. Most notably, a clear distinction is made between Feature Extractors and Box Predictors. As their name suggests, Feature Extractors are responsible for extracting features from images. They accept an image tensor as an input and output a list of feature map tensors. On the other hand, Box Predictors are tasked with the objective of performing classification and localisation of objects within the image. They take a high level image feature map as input and produce two predictions: a tensor encoding box locations; and a tensor encoding classes for each box. This modularity is particularly convenient when performing FE training. As we explain in more detain further down, FE training can be achieved by freezing all layers pertaining to the Feature Extractor of each respective model.

The training parameters for each of the networks were mostly kept identical to the default TensorFlow settings used during pre-training, matching the parameter settings used in [148]. All models were configured to employ three different data augmentation methods: random image crop, random horizontal flip and random colour distortion. Other training parameters, specific to each model, are summarised below:

- SSD Inception:
  - Features are extracted from *Mixed 4c* and *Mixed 5c*, and four additional convolutional layers are appended with decaying resolution with depths 512, 256, 256, 128 respectively. ReLU6 is used as the non-linear activation function for each convolutional layer.
  - Loss optimisation is performed using an RMSprop optimiser.
  - The initial learning rate is set to 0.001, with an exponential decay factor of 0.95 every 800k steps.
  - During FE training, the variables pertaining to layers included under *FeatureExtractor/InceptionV2/InceptionV2* are frozen.
- SSD ResNet:
  - Features are extracted from the last layer of the *conv5* (*block4*) block and five convolutional layers with decaying spatial resolution are appended with depths 512, 512, 256, 256, 128, respectively.
  - Loss optimisation is performed using an Momentum optimiser.
  - A warm up stage of 2k steps is initially employed, during which the learning rate grows linearly from 0.0013333 to 0.004, followed by a cosine decay schedule until step 25k.
  - During FE training, the variables pertaining to layers inside *FeatureExtractor/ResNet_v1_50* are frozen.
- Faster-RCNN Inception:
  - Features are extracted from the *Mixed 4e* layer with stride size of 16 pixels, following which the feature maps are cropped and resized to 14x14.
  - Loss optimisation is performed using an Momentum optimiser.
  - A manual schedule is used for adjusting the learning rate. The initial learning rate is set to 2e-5, which is reduced to 2e-6 after 900k steps.

– During FE training, the feature extractors of both the RPN and the box predictor are frozen by freezing all variables pertaining to layers under *FirstStageFeatureExtractor/InceptionV2* and *SecondStageFeatureExtractor/InceptionV2* respectively.

6.3.4 Results

Performance evaluation of the various models is performed by means of two metrics. Firstly, the mean Average Precision (mAP) [159, 160] with an Intersection over Union (IoU) threshold of 0.5 is computed, which provides a measure of detection accuracy. Secondly, the computational cost of each method was evaluated by calculating the average computation time per detection, per image. The evaluation was performed on a desktop computer with a 3.7GHz 8-core Intel i7 6900k processor, with 32GB of maximum available RAM, and a nVidia GTX1070Ti graphics card, clocked at 1683 MHz, with 8GB of VRAM.

6.3.4.1 Baseline Models

At a first stage, the performance of all three pre-trained detectors was evaluated against the test dataset of Table 6.1. The results of the above evaluation process are depicted in Figure 6.4. Our results confirm our expectations and follow closely the results of [148]. More specifically, the best accuracy is achieved by the *SSD ResNet* detector, followed by *SSD Inception* and lastly *Faster-RCNN Inception*, while in terms of speed, the *SSD Inception* detector comes first, with the *SSD ResNet* coming second and *Faster-RCNN* falling last. However, it is important to note that the detection performance is relatively poor, as the minimun mAP value can be seen to come at 11%, with a maximum value that falls just below 30%.

6.3.4.2 Re-Trained Models

The performance of the re-trained detectors has been evaluated using the same metrics as in the case of the pre-trained detectors and the results are shown in Figure 6.5. From the results, it becomes apparent that the performance achieved by the new detectors is significantly superior to the pre-trained versions. As expected, all detectors show an increase in detection accuracy, while at the same time exhibiting faster computation times.

In both the FE and FT re-trained models, training of the box prediction layers results in the models being better placed at classifying and localising the objects

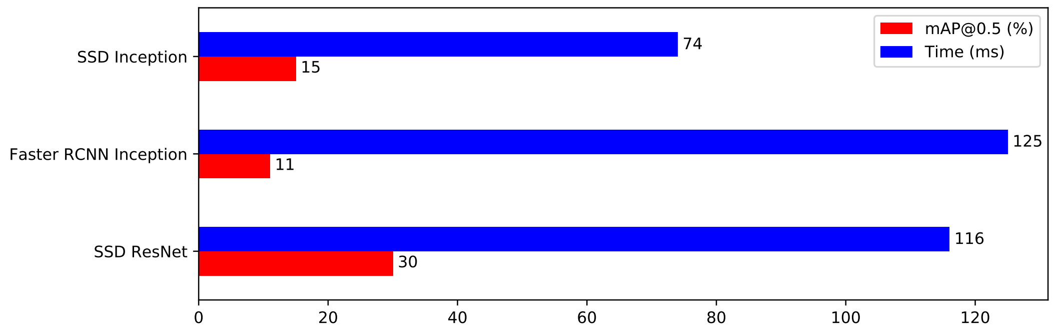


Figure 6.4: Plot of mean Average Precision (mAP) vs computation time for the pre-trained models.

of interest, when compared to the pre-trained models. The localisation benefits achieved in the case of the FE models follows from the training performed on the box prediction and RPN layers in the SSD and Faster-RcNN meta-architectures, respectively. Furthermore, it also becomes evident that all FT models exhibit superior performance compared to their FE trained counterparts. This observation can be attributed to the fact that in the case of the FT models the layers of the feature extractors are also allowed to be trained on the dataset. Therefore, the features extracted by these layers form a better representation of the inherent features of ships, which results in the generation of more accurate predictions, in turn resulting in significant classification performance gains.

The gains in computational performance can be attributed to the reduced complexity of the classification layers in the network. In the re-trained models, the classification problem is effectively reduced to detecting a single class, hence the activation maps produced by the extracted features lead to a reduced number of predictions that need to be processed by each algorithm (convolution filters for SSD and RPN for Faster-RCNN). Furthermore, as there is only a single class to be detected, the amount of computations performed by the final Softmax function, which calculates the classification probabilities, is effectively minimised to a single weighted sum of the outputs from the previous layer, as opposed to approximately 90 for the pre-trained models.

From the presented results, it can be concluded that the *SSD Inception (FT)* detector provides the optimal trade-off between accuracy and speed, with a mAP of 87% and average computation time of 25ms. Although the *SSD ResNet (FT)* detector can be seen to exhibit the best accuracy, the achieved speed of 80ms limits the applicability of the method for real-time detection. Example detections generated using the developed detector are shown in Figure 6.6.

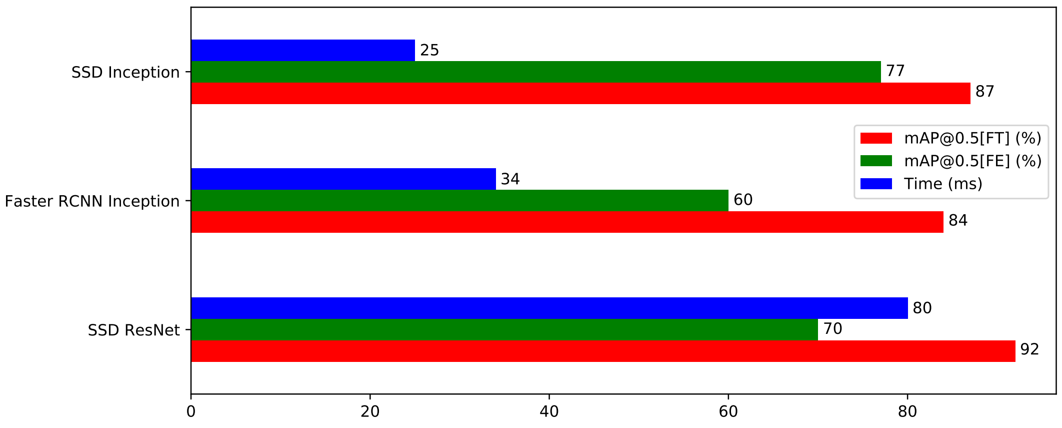


Figure 6.5: Plot of mean Average Precision (mAP) vs computation time for the re-trained models.

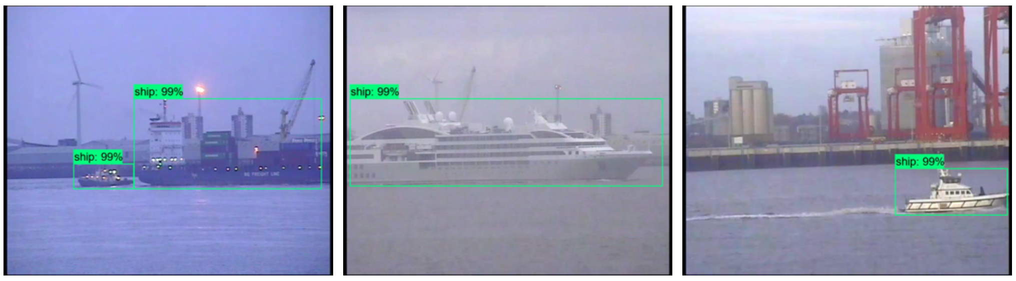


Figure 6.6: Example ship detections achieved using the developed Re-Trained SSD Inception v2 detector.

6.3.5 Conclusions

In conclusion, this section demonstrates a custom trained CNN ship detector, that can be readily applied to detect ships in imagery data from an off-the-shelf commercial maritime surveillance camera. A brief review of the various components of CNN object detection networks has been performed, followed by a discussion on the different transfer learning approaches commonly employed by practitioners. A custom dataset has been collected from a commercial maritime camera and used to train a set of 3 mainstay object detection networks. The training parameters used for each algorithm has been documented for future references and a quantitative comparison has been performed between the performance achieved by each network. The survey concluded that fine-tuning of the networks produced significantly better results, across all examined algorithms, when compared to constrained training of the feature extraction layers for the considered dataset. Furthermore, it has been found that the combination of an SSD meta-architecture, combined with the Inception v2 feature extraction network, exhibits the optimal trade-off between speed and accuracy. The particular ship detector is able to detect ships in video streams in real-time, achieving speeds up to 40 frames per second. This speed surpasses the frame rate of most commercial camera units, whose typical rate is at 30 frames per second. The achieved accuracy performance of 87% mAP, ensures that the detector exhibits minimal missed detections and false alarms, thus setting a firm detection foundation on which tracking and state estimation algorithms can be applied.

6.4 Real-Time Camera Motion Error Correction using Optical Flow

Tracking multiple targets with the use active, a.k.a. Pan-Tilt-Zoom (PTZ), cameras in a commercial maritime environment introduces a significant challenge. When the orientation and/or focal-length of the camera are adjusted by an operator, the space that is observable by the camera changes, meaning that any information pertaining to the state of tracks becomes invalid. This problem is frequently formulated in terms of a closed-loop feedback system, where the orientation (Pan-Tilt) and Focal Length (zoom) of the sensor are readily available. However, in most commercial security applications the sensors are often placed at remote locations, while the central processing is performed on a mainframe server, typically located in a secure location that is not in close proximity to the sensors. Thus, communication between the server and the sensor relies on the use of physical network infrastructure at best, while wireless communication mediums, such as microwave links, are more that often employed. What is more, most off-the-shelf surveillance cameras are developed with the intent of being operated by a human operator, where tele-operation is achieved via relative speed commands generated by a joy-stick. Thus, increased focus is devoted to the development of efficient and low-latency video feeds to the operator, while feedback from the camera is seldom guaranteed to be delivered quickly and accurately.

The delay between the received video stream and the feedback reported by the camera can vary significantly due to network delays, while video frames can also be dropped due to connectivity problems. The combination of network induced lag, and the lack of accurate and/or timely feedback introduces significant delays between the receival intervals of video data and camera feedback. Thus, it becomes near impossible to correct the errors in real-time by relying on the camera feedback information, meaning that significant errors in the tracking process are inevitable. This has a detrimental effect on tracking performance, since even the slightest change of the state of the camera can have a massive effect on the track positions between any two consecutive frames. A typical example of this problem is demonstrated in Figure 6.7, where two consecutive video frames are shown, along with the respective track estimates, corresponding to the observed surveillance region before and after a camera motion command is executed.

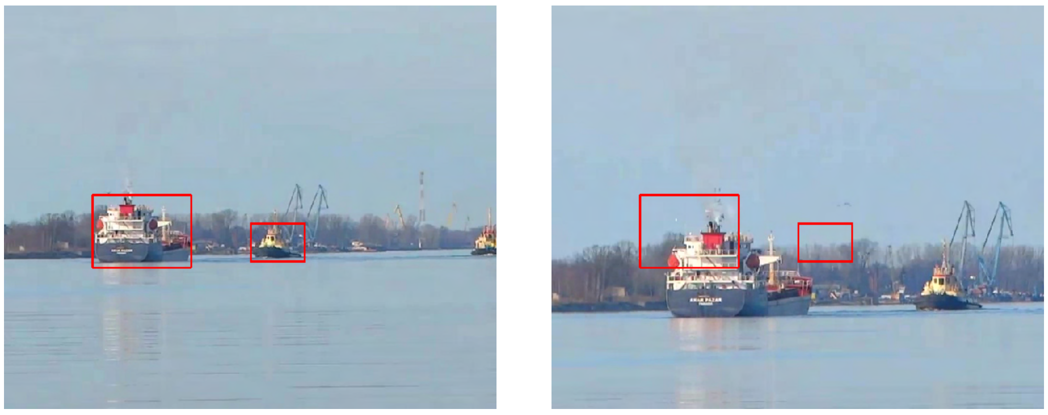


Figure 6.7: Example frames, along with annotated track estimates, before and after a camera command execution.

In this section, a method is presented for estimating and correcting the errors relating to motion of the camera in real-time, by formulating the problem of frame-to-frame variation as an affine transformation problem. First, a feature detector is applied to extract a set of "good" features, along with their respective locations, from the previous frame. Then, Optical Flow is employed to track and locate the detected feature locations in the new frame, following which the two sets of matching features are utilised in order to estimate the affine transformation between the two frames. Finally, the estimated affinity coefficients are used to correct the posterior state distributions of known tracks, such as to compensate for the errors introduced by the movement of the camera.

6.4.1 Feature Detection

In the context of image processing, feature detection is the process of defining a set of features which provide an efficient and meaningful representation of the image, thus enabling the reduction of resources necessary to accurately describe the contained information. The goal is to locate points in the image that lie along some defined boundaries. There generally exist three different types of features: textural (such as spatial frequency, patterns, homogeneity, etc.), spectral (such as colour, size, ratio etc.) and geometric features (corners, edges, blobs etc.).

The chosen feature extraction approach, outlined in this section, is based on detecting corner features. Corners are among the most useful features that can be extracted from an image. Formally defined, corners are regions in the image with large variation in intensity in all the directions. They typically correspond to well-distinguishable, often unique, characteristics corresponding to discontinuities in the physical, photometrical and geometrical properties of objects. For this reason, corner features are widely used in a multitude of detection and tracking applications [161, 162, 163].

An effective and efficient method for corner feature detection was presented in [164] and further improved in [165]. Without loss of generality, let I denote the image of interest and assume that is a 2-D grayscale image. Now, consider taking an image patch over some area (x, y) of I and displacing it by (dx, dy) in all directions. Then, the difference in intensity caused by the performed displacements can be computed as a weighted sum of squared differences, between the patches formed by the displacements, as

$$\mathcal{E}(x,y) = \sum_{dx,dy} w(dx,dy)\,[I(x+dx,y+dy) - I(dx,dy)]^2 \tag{6.3}$$

where $w(.)$ denotes a window function (e.g. Box or Gaussian filter) that slides over the image, giving weights to pixels underneath. By applying Taylor-series Expansion [36] to the term $I(x + dx, y + dy)$, (6.3) can be re-writen as

$$\mathcal{E}(x,y) \approx [x,y]\,\mathcal{M}\,[x,y]^T \tag{6.4}$$

where $\mathcal{M}$ is the structure tensor, given by

$$\mathcal{M} = \sum_{dx,dy} w(dx,dy) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix} \tag{6.5}$$

with I_x, I_y denoting the partial derivatives of I.

In general, corners can be identified as points that exhibit a large variation of the function $\mathcal{E}$ in all directions. More specifically, a closer analysing of the eigenvalues λ_1, λ_2 of the tensor $\mathcal{M}$ for a given point p with coordinates (x, y), yields the following scenarios:

1. If $\lambda_1 \approx 0$ and $\lambda_2 \approx 0$, then p has not features of interest.
2. If $\lambda_1 \approx 0$ and $\lambda_2 \gg \lambda_1$, then the point lies on an edge.
3. If both $\lambda_1 \gg 0$ and $\lambda_2 \gg 0$, then the point is a corner point.

Finally, making use of the above defined criteria, a corner point can be identified if the following condition holds [165]:

$$\min(\lambda_1, \lambda_2) > \lambda_T \tag{6.6}$$

where λ_T is defined such that $\lambda_1 \gg 0$ and $\lambda_2 \gg 0$.

The above documented process then allows us to extract a set of distinguishable feature points $\mathcal{F}_{k-1}$ from a given frame I_{k-1} at time t_{k-1}. The next section discusses how the locations of a set of matching feature points $\mathcal{F}_k$ can be efficiently estimated in the next frame I_k at time t_k, using Optical flow.

6.4.2 Feature Matching using Optical Flow

Optical flow is defined as the pattern of apparent motion of pixels between two consecutive frames, caused by the movement of an object on the image plane. The method provides a succinct representation of both the regions of the image undergoing motion and the velocity of motion in those region. Optical flow is a well established concept that has been extensively researched over the past few decades [166, 167], while a definitive account of the mathematical concepts relating to Focus of Expansion (FoE), forming the basis of Optical Flow, can be dated back to 1980 [168]. Since then, the method has been widely applied in the tracking domain, with particular interest in problems involving the detection of moving objects [169, 170, 171] and background estimation/segmentation [172, 173]. Furthermore, Optical flow methods have recently been applied to perform video stabilisation and mosaicing in problems involving non-stationary cameras [174, 175].

For the purposes of this section, let I_k and I_{k-1} denote the two consecutive image frames and, without loss of generality, assume that they are 2-D grayscale images where the quantity $I_k(x, y)$ returns the grayscale value of I_k at the frame coordinates (x, y). Now, consider that a point (x, y) in I_{k-1} has moved by a distance (dx, dy) and is now located at coordinates $(x + dx, y + dy)$ of I_k. Also let $dt = t_k - t_{k-1}$ denote the time elapsed between times t_k and t_{k-1}. The goal of Optical Flow is then to find the velocity vector

$$V = [V_x, V_y]^T = \left[\frac{dx}{dt}, \frac{dy}{dt}\right]^T \tag{6.7}$$

which defines the image velocity, a.k.a. the optical flow, such that

$$I_{k-1}(x, y) = I_k(x + dx, y + dy) \tag{6.8}$$

If the assumption is made that the elapsed time dt is relatively small, taking Taylor-series expansion of the term relating to I_k leads to

$$I_k(x+dx, y+dy) = I_{k-1}(x,y) + \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt \tag{6.9}$$

which can then be substituted in (6.8), giving rise to the Optical Flow equation:

$$I_x V_x + I_y V_y = -I_t \tag{6.10}$$

or making use of vector notation:

$$SV = -I_t \tag{6.11}$$

where $S = \nabla I^T$, while V_x, V_y are the velocity vectors defined in (6.7) and

$$I_x = \frac{\partial I}{\partial x};\ I_y = \frac{\partial I}{\partial y};\ I_y = \frac{\partial I}{\partial t} \tag{6.12}$$

The relation of (6.11) defines a single system with two unknowns, meaning that it cannot be directly solved. One popular method that solves the above problem is the so called Lucas-Kanade method [176], where the point (x, y) is expanded using a $N \times N$ patch of N^2 points defined around (x, y), which are all assumed to have the same velocity. By doing so, the previously problematic system is now expanded to an overdetermined system of N^2 equations with 2 unknowns, meaning that S and I_t of (6.11) are expanded to N^2 rows.

Finally, a Least Squares solution to the expanded system of (6.11) can be found by multiplying the equation by S^T, followed by an inversion of the product $S^T S$, such that:

$$V = -(S^T S)^{-1} S^T I_t \tag{6.13}$$

Therefore, given a set of feature points $\mathcal{F}_{k-1}$ in I_{k-1}, with positions $\mathcal{P}(\mathcal{F}_{k-1})$, it is possible to obtain a set of matched points $\mathcal{F}_k$ in I_k, whose coordinates $\mathcal{P}(\mathcal{F}_k)$ are approximated as:

$$\mathcal{P}(\mathcal{F}_k) = \mathcal{P}(\mathcal{F}_{k-1}) + V dt \tag{6.14}$$

6.4.3 Affine Transformation

An affine transformation (or affinity) is defined as a transformation that preserves collinearity and ratios of distances; that is, all points that form a line prior to an affinity, still lie on a line after transformation (collinearity) and the midpoint of a line segment remains the midpoint after transformation. In this context, the term affine indicates a special class of projective transformations that do not move any objects from the affine space $\mathbb{R}^3$ to the plane at infinity or conversely. Thus, affine transformations can be used to express rotation, translation and/or scale operations between two vectors in terms of a system of linear equations. More specifically, rotation and scaling are both linear transformations that can be expressed as matrix multiplications, while translation is viewed as a vector addition.

Let $\mathrm{y} \in \mathcal{Y}$ be a vector that results from an affine transformation of $\mathrm{x} \in \mathcal{X}$, then formally this can be expressed as

$$\mathrm{y} = f(\mathrm{x}) = A\mathrm{x} + \mathrm{d} \tag{6.15}$$

where $f : \mathcal{X} \to \mathcal{Y}$ is the affine map, A is a linear map representing scaling and rotation, while d is the translation vector. Alternatively, the above relation can be expressed using augmented matrices as

$$\begin{bmatrix} \mathrm{y} \\ 1 \end{bmatrix} = \mathcal{A} \begin{bmatrix} \mathrm{x} \\ 1 \end{bmatrix} = \left[\begin{array}{ccc|c} & A & & \mathrm{d} \\ 0 & \dots & 0 & 1 \end{array} \right] \begin{bmatrix} \mathrm{x} \\ 1 \end{bmatrix} \tag{6.16}$$

where $\mathcal{A}$ is called the *affinity matrix.*

Now, let us consider two consecutive frames (e.g. Figure 6.7) I_k and I_{k-1}, of equal size (I_{width}, I_{height}), where I_{k-1} corresponds to the frame at time t_{k-1} and I_k at time t_k. Each frame can then be viewed as a 2-D matrix, where $I_k(x, y)$ returns the value of the corresponding pixel located at frame coordinates (x, y).

Continuing, suppose that the track estimates in Figure 6.7 can be defined in terms of a vector x_k^i, containing the pixel coordinates of the vertices defining the bounding box containing the i-th track, as follows:

$$\mathrm{x}_k^i = \left[x_{min}^i, y_{min}^i, x_{max}^i, y_{max}^i\right]^T \tag{6.17}$$

where x_{min}^i, x_{max}^i denote the x coordinates of the left and right box vertices, while y_{min}^i, y_{max}^i denote the y coordinates of the top and bottom vertices. Note that (6.17) can be obtained from (6.1) by setting $x_{min}^i = x_k^i$, $x_{max}^i = x_k^i + w_k^i$, $y_{min}^i = y_k^i$ and $y_{max}^i = y_k^i + h_k^i$.

Making the assumption that the camera follows a pinhole camera model[1], and that the observed scene remains mostly static between t_k and t_{k-1}, it is possible to deduce that I_k is the product of an affine transformation applied on

[1]A pinhole camera model ensures that no distortions are applied to the observed image as a result of the camera lens.

I_{k-1}. This should become more apparent by considering that, subject to linearity assumptions, a pan motion of the camera results in a horizontal translation of the image, tilt motion results in a vertical translation, while zooming introduces scaling.

Let $\mathcal{F}_{k-1} = \left[\mathrm{f}_{k-1}^1, \ldots, \mathrm{f}_{k-1}^n\right]^T$ denote a set of feature points extracted from I_{k-1}, where f_{k-1}^i contains some information that distinguishes the point from all other points. Now, assume that a set $\mathcal{F}_k = [\mathrm{f}_k^1, \ldots, \mathrm{f}_k^n]^T$ of matching points is identified in I_k, such that $\mathrm{f}_k^i = \mathrm{f}_{k-1}^i$, for all $i \in [1, n]$. Also, let $\mathcal{P} : \mathcal{F} \rightarrow \mathbb{R}^2$ be a function that maps feature points to their respective (x, y) coordinates on the image plane, e.g.:

$$\mathcal{P}(\mathrm{f}_k^i) = [x_k^i, y_k^i]^T \tag{6.18}$$

Considering the above, and making use of the affinity relations of (6.15), we can define a relation between the coordinates $\mathcal{P}(\mathcal{F}_k)$ of a given set of features in I_k and the corresponding coordinates $\mathcal{P}(\mathcal{F}_{k-1})$ of the matching features in I_{k-1} as

$$\mathcal{P}(\mathcal{F}_k) = A\mathcal{P}(\mathcal{F}_{k-1}) + \mathrm{d} \tag{6.19}$$

Equivalently, by noting that the state of (6.17) can be viewed as the (x, y) coordinates of the corresponding frame pixels that lie underneath each vertex, we can generalise the previous statement to deduce that the state x_k^i of a given track i can be viewed as an affine transformation of the state x_{k-1}^i, giving rise to the following state correction equation:

$$\mathrm{x}_k^i = A\mathrm{x}_{k-1}^i + \mathrm{d} \tag{6.20}$$

Therefore, if the affine transformation between two consecutive frames can be determined, it should be possible to correct for the errors induced by the motion of the camera on the track states by utilising (6.20), while circumventing the necessity of considering the camera feedback. A solution to this problem can be obtained by attempting to solve (6.19), given a pair of matching feature sets $\mathcal{F}_k$ and $\mathcal{F}_{k-1}$, extracted from two consecutive frames. In Section 6.4.1 we discussed how the set of features $\mathcal{F}_{k-1}$ can be extracted from frame I_{k-1}, while Section 6.4.2 showed discussed how to approximate the locations of a set of matching features $\mathcal{F}_k$ in I_k. However, due to the involved approximations, $\mathcal{F}_k$ may not be directly related to $\mathcal{F}_{k-1}$ as per (6.19), and thus it is not possible to deduce the affinity coefficients directly. Instead, an approximation to the coefficients can be obtained using robust model fitting methods, such as RANSAC [177].

6.4.4 Correcting the Track Estimates

Having obtained an expression for the state of a track following the transformation through (6.17), it is also necessary to derive an expression for correcting the posterior distribution from the previous timestep. It is important to note that the correction process is executed upon receival of a new frame at time t_k, but before the tracks are processed by the tracker for estimation purposes. This necessitates the introduction of new notation, to avoid confusion with the notation used in previous chapters. Thus, let $p(\mathrm{x}_{k-1|k-1}|Y_{k-1})$ denote the target posterior distribution produced at t_{k-1}, prior to any transformation, with $p(\mathrm{x}_{k-1}|Y_{k-1})$ denoting, as per previous notation, the transformed posterior that is forwarded to the tracker. For the purposes of this discussion, the track index i is dropped for notational simplicity.

If a Particle Filter is used to perform filtering, then, as discussed in Section 3.2.3, the posterior from t_{k-1} is approximated using a set of weighted particles, as follows:

$$p(\mathrm{x}_{k-1|k-1}|\mathrm{Y}_{k-1}) \approx \sum_{i=1}^{N_p} \mathrm{w}^i_{k-1|k-1}\delta(\mathrm{x}_{k-1|k-1} - \mathrm{x}^i_{k-1|k-1}) \tag{6.21}$$

Then, the transformed posterior $p(\mathrm{x}_{k-1}|Y_{k-1})$ can be obtained by applying (6.20) to each particle, while maintaining the same weights, i.e.,

$$p(\mathrm{x}_{k-1}|\mathrm{Y}_{k-1}) \approx \sum_{i=1}^{N_p} \mathrm{w}^i_{k-1}\delta(\mathrm{x}_{k-1} - \mathrm{x}^i_{k-1}) \tag{6.22}$$

where

$$\begin{aligned} \mathrm{x}^i_{k-1} &= A\mathrm{x}^i_{k-1|k-1} + \mathrm{d} \\ \mathrm{w}^i_{k-1} &= \mathrm{w}^i_{k-1|k-1} \end{aligned} \tag{6.23}$$

In the case of Kalman Filters, we know from Section 3.2.2 that the state distribution at any given point is approximated as a Gaussian distribution. Thus, the posterior generated at t_{k-1} is a pdf of the form

$$p(\mathrm{x}_{k-1|k-1}|\mathrm{Y}_{k-1}) = \mathcal{N}(\mathrm{x}_{k-1|k-1}; \mu_{k-1|k-1}, \Sigma_{k-1|k-1}) \tag{6.24}$$

where $\mu_{k-1|k-1}$, $\Sigma_{k-1|k-1}$ represent the mean and covariance of the pdf, respectively. The mean $\mu_{k-1|k-1}$ can be transformed in an identical manner to the particles in the case of the Particle Filter, however the covariance $\Sigma_{k-1|k-1}$ must be scaled to reflect the effects of the affine transform on the uncertainty of the state. Thus, the transformed posterior is computed as follows:

$$p(\mathrm{x}_{k-1}|\mathrm{Y}_{k-1}) = \mathcal{N}(\mathrm{x}_{k-1}; \mu_{k-1}, \Sigma_{k-1}) \tag{6.25}$$

where

$$\begin{aligned} \mu_{k-1} &= A\mu_{k-1|k-1} + \mathrm{d} \\ \Sigma_{k-1} &= A\Sigma_{k-1|k-1}A^T \end{aligned} \tag{6.26}$$

6.4.5 Results

To benchmark the proposed algorithm, a real commercial Pan-Tilt-Zoom camera unit was used to generate video streams while the camera is manually steered to follow vessels cruising near the sensor. Once the streams are extracted, a collection of suitable frames are identified and manual annotation is performed to form the ground-truth bounding boxes, denoting where targets should appear within each frame. Following the above approach, a collection of 100 image pairs is used as the basis of evaluation. An illustration of the feature matching process for 3 example image pairs is shown in Figure 6.8. Furthermore, an illustration of the corrected track estimates for the same 3 examples is shown in Figure 6.9.

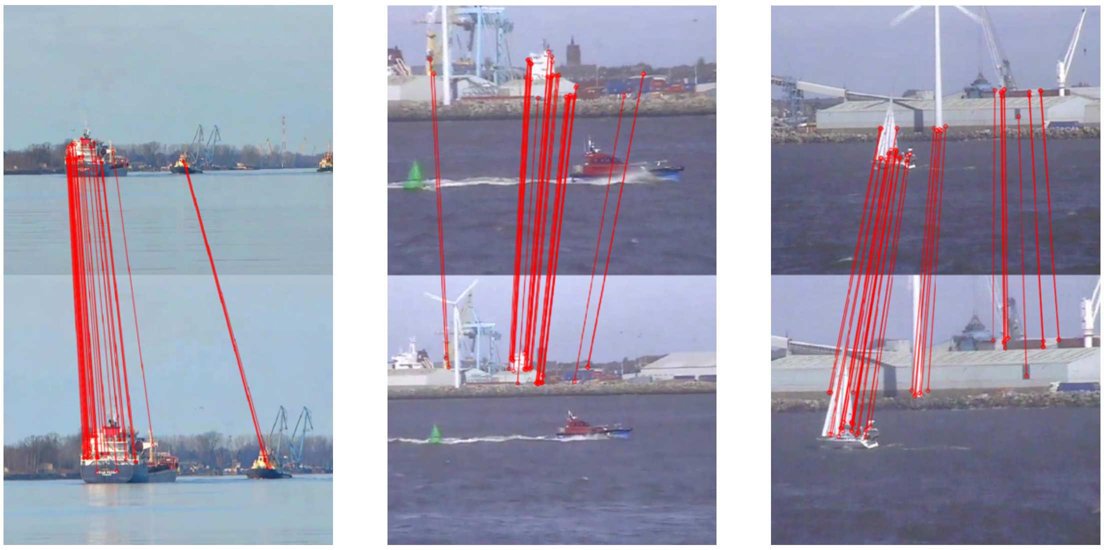


Figure 6.8: Features matched between two frames using the proposed approach

Once the ground-truth data has been extracted, the performance is evaluated by applying the algorithm on the pairs on images and computing the localisation

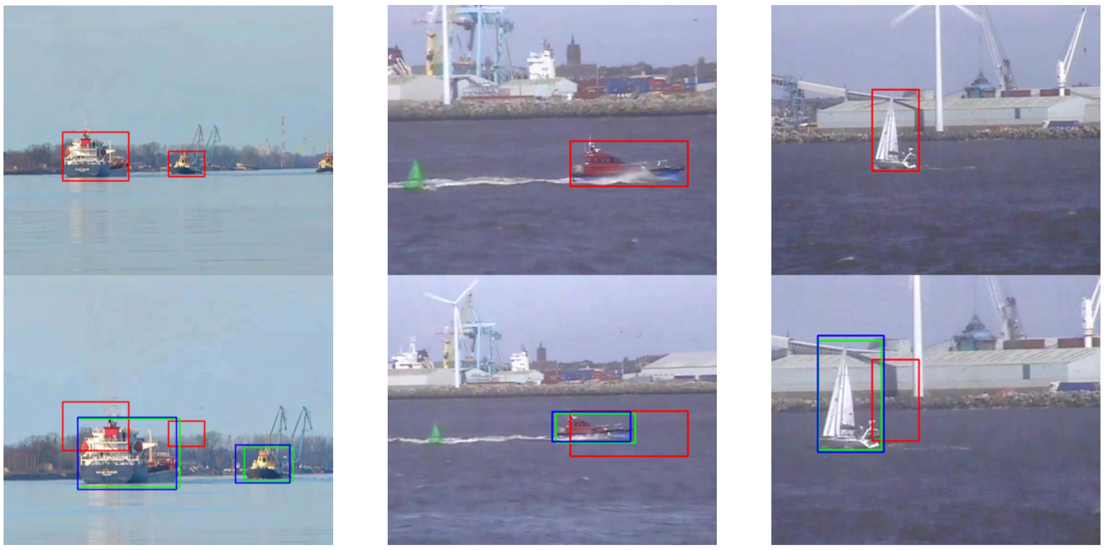


Figure 6.9: Examples of corrected track estimates using the proposed approach. The top row shows the first frame, while the second shown the second frame. Green colour is used to depict the ground-truth track estimates, blue colour is used for the corrected estimates and red is used for the non-corrected estimates.

component of the OSPA [106] metric. The OSPA metric is configured with parameters $p = 2$ and $c = 500$, while a euclidian distance is employed as the base metric. The computed metric values are then compared against the metric values computed when the method is not applied (NA). In addition, a comparison is made against the proposed method (OF), where feature detection and matching is performed using Optical Flow, and other well known feature matching techniques that utilise ORB [178], SIFT [179] and SURF [180] features respectively.

The results presented in Figure 6.10 showcase the performance achieved by the proposed method. As expected, all examined approaches (OF, ORB, SIFT, SURF) can be seen to reduce significantly the OSPA error when compared to the case where no correction method is applied (NA). Therefore, it can be concluded that estimating the inter-frame affine transformation, based on features extracted across frames, is an adequate approach for correcting the errors introduced by camera motion. Continuing, for the specific evaluation dataset, the results show that the proposed approach (OF) performs marginally better than all other methods. Although this cannot (and should not) be used to make any strong claims about the employed feature matching method, it does serve as an indication that the proposed method is expected to perform comparatively well for the given application. Nevertheless, the results relating to efficiency are certainly conclusive. The OF algorithm provides at least a tenfold reduction in computation

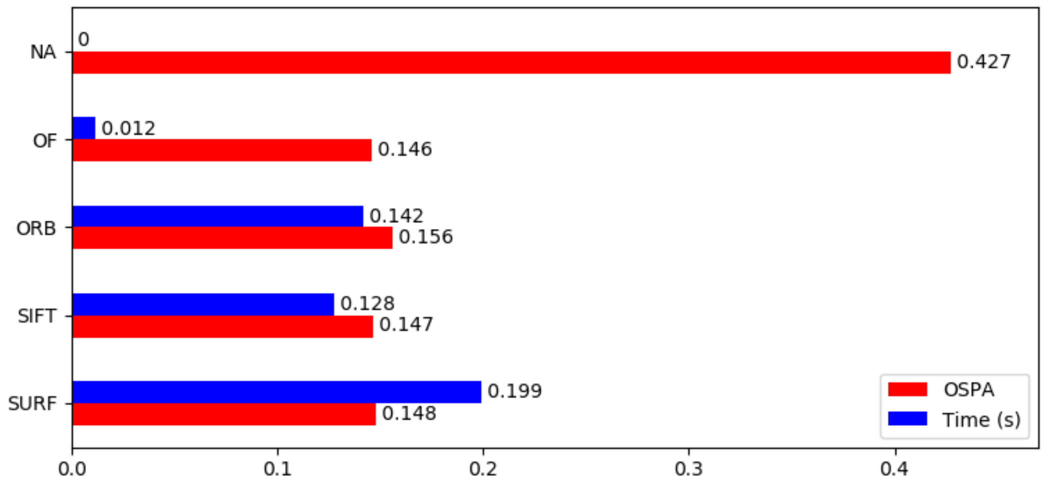


Figure 6.10: Plot of OSPA vs computation time. The proposed method is denoted by OF and NA denotes the case where no correction is applied. The acronyms SURF, SIFT and ORB denote the methods where feature matching is performed using the respective algorithm.

time, thus allowing the proposed method to be applied in real-time, without having a significant effect on the computational requirements of the overall tracking process.

6.5 Conclusion

This chapter summarised the work performed by the author in the domain of commercial Maritime Video Detection & Tracking. A review of existing EO ship detection approaches was presented followed by an introduction to the state-of-the-art detectors that base their operation on Convolutional Neural Networks (CNNs). Under the same section, a robust CNN-based ship detector was introduced, that utilises a pre-trained CNN model that is subsequently trained on a custom dataset of images to provide an operational advantage. Finally, an effective method was introduced for estimating and accounting for the tracking errors induced by camera motion in real-time, in the absence of camera feedback.

CHAPTER 7

Modular Frameworks for Tracking and State Estimation

7.1 Introduction

Tracking and state estimation methods are widely researched and applied in a variety of domains that include astronomy, air surveillance, maritime situational awareness, economics and biology. As follows, there exist a multitude of tracking and state estimation methods that have been developed over the years, each with its own strengths and deficiencies in terms of accuracy, scalability, computational complexity and other factors. What is more, novel algorithms are continuously being developed and claims are often made that variants of existing algorithms perform better than previous iterations. As newly developed algorithms are becoming increasingly complex, researchers and skilled practitioners are faced with the challenge of implementing and systematically evaluating these state-of-the-art algorithms on a daily basis. In order for academics and skilled practitioners to assess the performance of such algorithms, it is paramount for one to produce robust implementations of the algorithms being compared, while oftentimes this would necessitate studying and implementing unfamiliar and complex algorithms to achieve this goal. However, the process of evaluating and identifying the algorithm that is best for each particular application is becoming increasingly difficult, while it is near impossible for those with little experience to make this determination.

As a direct consequence of the above, there is an evergrowing necessity for the development of unified frameworks for tracking and state estimation, that shall allow academics and practitioners to swiftly develop and evaluate the performance of different algorithms. Only a small number of such frameworks are known to exist, which generally suffer greatly from strict modularity and flexibility constraints, making them hard to learn and use. As an additional hurdle, most existing frameworks are released under closed-source proprietary licences, meaning that there are high costs associated to using these frameworks, while it is nigh impossible to look under-the-hood such as to extend their functionality. Also, industrial-scale dataset owners, who might not be experts in the mathematics of tracking, need to be able to run their data through multiple algorithms quickly in order to compare the performance of the algorithms and select the best algorithm for their application without being influenced or biased by personal favourites.

This chapter shall focus on presenting work done by the author on developing and contributing towards the development of open-source frameworks for tracking and state estimation. Section 7.2 introduces an open-source object-oriented MATLAB toolbox, developed by the author over the course of the PhD project and comprised of efficiently coded implementations of target tracking algorithms, with the aim of assisting and accelerating future research within the field. Section 7.3 discusses Stone Soup [12, 13, 14], an open-source Python framework that is currently under development, stemming from combined international efforts led by the Defence Science and Technology Laboratory (UK) in direct collaboration with the Defence Research and Development Canada (Canada), the Air Force Research Laboratory (US), and the University of Liverpool (UK), to which the author has been an major contributor.

7.2 TrackingX

There exist a few MATLAB toolkits that offer tracking capabilities. The Tracker Component Library (TCL) [181] is an open-source library of common routines that go into target-tracking algorithms written in MATLAB, with some functions written in C and C++ with MATLAB interfaces. Indeed, TCL incorporate a collection of highly efficient tracking routines, that span across various fields, including astronomy, magnetism, navigation and others, however it does not offer a unified interface of types and classes that allow for straight-forward component interchangeability. Continuing, the MATLAB Sensor Fusion and Tracking Toolbox (SFTT) [182] includes multi-object trackers, sensor fusion filters, motion and sensor models, and data association algorithms that let users evaluate fusion architectures using real and synthetic data. While SFTT offers a lot of useful functionality, it still suffers from the same issues of component interchangeability, while it also requires a substantial purchase fee.

TrackingX is a unified, re-usable and highly-modular and open-source object-oriented MATLAB toolbox, comprised of efficient implementations of state-of-the-art target tracking and state estimation algorithms. The toolbox has been developed by the author, over the course his PhD project, with the objective to enable researchers and practitioners to swiftly prototype and evaluate tracking algorithms. The great advantage of TrackingX is that it is completely open-source and free to use [1], while its class architecture and interfaces of are being designed so that components of the same class provide a common interface to all classes that utilise them, thus allowing for a virtual plug & play functionality.

7.2.1 Class Architecture

The philosophy behind TrackingX is to develop an open-source framework with modular, interchangeable components to enable the rapid prototyping and testing of tracking and state estimation algorithms. The toolkit is designed with the intention that users can easily develop their own components, or re-use existing ones, such as to swiftly prototype and evaluate tracking algorithms, with the added ability to mix and match components. In support of this idea, the TrackingX architecture has been built on the principles of modularity and uniformity of external interfaces.

TrackingX is object-oriented and makes use of encapsulation, abstraction and inheritance, where tracking algorithms are built as hierarchical objects. Abstraction is achieved by defining an abstract class for each type of component which specifies the external interface for that class; that is, the parameters and functions an object of that class must make available to the rest of the framework. Continuing, new component classes can be derived as extensions of existing (abstract) classes, thus inheriting the specified properties and functions of their parent/extended class. This approach ensures that encapsulation is achieved, meaning that derived objects are interchangeable (within certain limits), and that other framework components can utilise them, without the necessity of knowing the details of their implementation. In support of the aforementioned modularity and common external interfaces, TrackingX objects are built based on a class inheritance hierarchy.

[1]Subject to obtaining a valid MATLAB license.

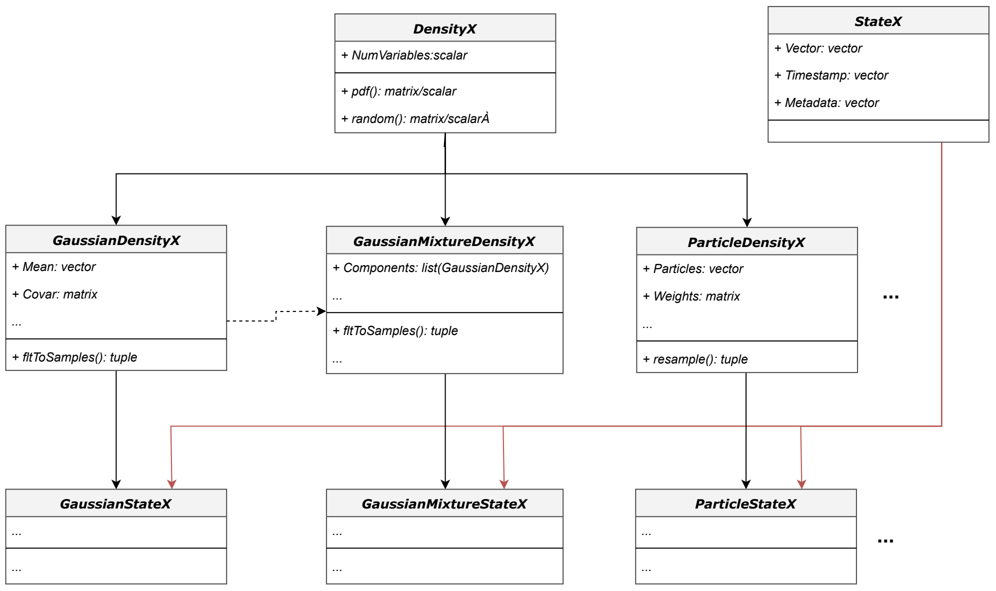


Figure 7.1: Class structure of DensityX and StateX components

7.2.2 Data Types

7.2.2.1 Distributions and States

TrackingX defines a ***DensityX*** class that serves as the abstract class for all densities. The ***DensityX*** interface defines that a valid implementation of the class must define the property *NumVariables*, which represents the number of random variables which the density incorporates, and the methods *random()* and *pdf()* that provide an interface for sampling from and evaluating the density, respectively.

A ***StateX*** abstract class is defined in TrackingX that forms the basis for representing target states. A valid ***StateX*** object must define an *Vector* property, which returns an estimate for the state in the form of an n-dimensional vector. In addition, an optional *Timestamp* can be attached to the ***StateX*** object, such that it can be used when performing time-series analysis. Finally, the ***StateX*** class is defined as a subclass of the MATLAB *dynamicprops* class, which means that it is possible to add custom properties to an instance depending on the needs of each application. The class architecture (see Figure 7.1) is defined such that target states are defined as sub-classes of ***StateX*** and a given subclass of ***DensityX***.

A first subclass example of ***DensityX*** is ***GaussianDensityX***, that represents a Gaussian density, thus requiring the definition of *Mean* and *Covar* properties, relating to the mean and covariance of the distribution. An optional *Weight* property can also be specified, whose default value is equal to 1. The *Weight* property can be used as an intensity indicator, where any value different that 1 would indicate that the object is not a probability density. A second example, that showcases component re-usability, is the ***GaussianMixtureDensityX*** class, that represents a Gaussian Mixture density. A ***GaussianMixtureDensityX*** object is essentially a combination of appropriately weighted ***GaussianDensityX*** objects. Finally, a ***ParticleDensityX*** class is defined that incorporates the notion of a density that is approximated via the use a set of appropriately weighted set of particles (samples). An important feature of TrackingX is that is allows for different densities to be instantiated on the basis of other densities. For example, it is possible to instantiate a ***ParticleDensityX*** object, let us call it B, by specifying a target object A, that is a subclass of ***DensityX***, together with the desired number of particles. Then, the constructor of the ***ParticleDensityX*** class, will utilise the *random()* method of A to sample a set of equally weighted number of particles, thus giving rise to the new ***ParticleDensityX*** object B.

Continuing, examples of ***StateX*** subclasses include the ***GaussianStateX***, ***GaussianMixtureStateX*** and ***ParticleStateX*** classes, derived from ***GaussianDensityX***, ***GaussianMixtureDensityX*** and ***ParticleDensityX***, respectively. By doing so, the above defined state objects can inherit the properties and functions of the underlying ***DensityX*** constructs. Thus, a ***GaussianStateX*** object can be queried for its mean and covariance, while a ***ParticleStateX*** can be queried for its particles and weights. Furthermore, this architecture provides the same functionality as outlined in the previous paragraph, whereby different ***StateX*** objects can be instantiated from other types of ***StateX*** object. Following the same example as before, a ***ParticleStateX*** object can be constructed from a ***GaussianStateX*** object, and vice-versa. This is an important feature that adds to the interchangeability nature of TrackingX, which will be discussed in more detail in Section 7.2.4.

7.2.2.2 Measurements

Measurements in TrackingX are implemented as a concrete sub-class of the ***StateX*** type, via the ***MeasurementX*** class. By doing so, ***MeasurementX*** objects inherrit the *Vector* property, which is used to store the measurement vector, as well as the optional *Timestamp* and *Metadata* properties. The class also includes an optional *Model* property, that is of type ***StateSpaceModelX*** and contains information relevant to the model used to generate the measurement. This is particularly useful in a multi-sensor scenario, where the noise characteristics, or even the entire measurement model, may be different for each particular sensor.

Furthermore, an optional *Tag* property allows for a unique tag to be attached to each measurement, if such a functionality is necessary. Finally, ***MeasurementX*** objects are also sub-classes of the MATLAB *dynamicprops* class, meaning that it is possible to attach dynamic properties as and when necessary.

7.2.2.3 Tracks

TrackingX provides a construct for Tracks via the ***TrackX*** data type. ***TrackX*** objects define 4 main properties, of which only the first is required: i) *States*; ii) *Filter*; iii) *Score*; iv) *Tag*. The *States* property is a list of ***StateX*** objects, that is used to store the state history (trajectory) of a given track. Continuing, ***TrackX*** objects can be optionally assigned a ***FilterX*** object, stored in the Filter property, that is used to perform filtering of the track state throughout its lifetime. This is in accordance to the way that ***FilterX*** objects are defined in TrackingX (see Section 7.2.4), whereby each ***FilterX*** object is considered as a Finite-state-Machine, with memory. An optional *Score* property is used by tracks to store a quality indicator (e.g. an existance probability, or log-likelihood ratio). If unique identification of tracks is necessary, the *Tag* property allows for such an identifier to be maintained by each ***TrackX*** object. Finally, the ***TrackX*** is derived from the dynamicprops MATLAB class, meaning that new class properties can be added to any TrackX instance as and when needed.

7.2.3 State-Space Models

TrackingX makes use of the concept of state-space models, similar in a sense to how they were described in Chapter 2. The base class for all models is the ***ModelX*** abstract class. The derivatives of the ***ModelX*** class are all abstract classes that can be split in two main categories: i) model-types; and ii) interfaces. The model-type derivatives of the ***ModelX*** class are then further split into five types, where each type relates to a different class of models that, when aggregated, formulate the state-space model, represented by the container class ***StateSpaceModelX***, on which a given tracking algorithm operates. On the other hand, the interface derivatives of ***ModelX*** are defined with the aim of enforcing a common interface to the sub-classes of the model-type derivatives. The above architecture is summarised in Figure 7.2.

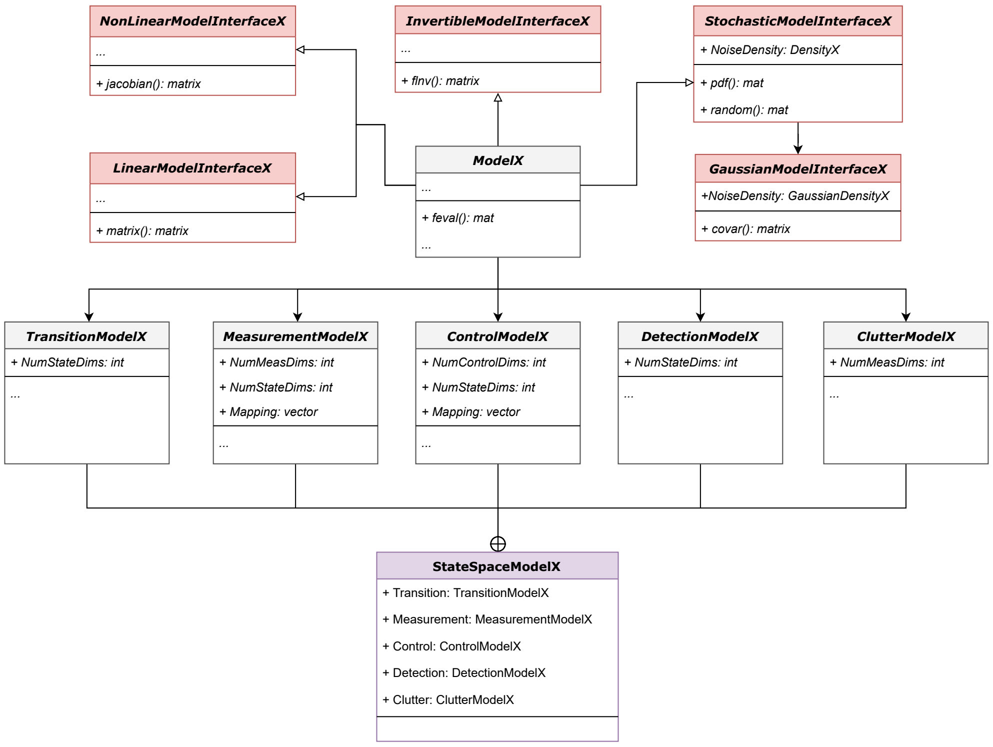


Figure 7.2: Class structure of State-Space Model components

7.2.3.1 Model Interfaces

The interface derivatives of ***ModelX*** are used to enforce the interface of the different models. Using a crude generalisation, models can typically be conceptualised as a function $f : \mathcal{X} \rightarrow \mathcal{Y}$, which we call the *model function*, that applies some form of transformation to an input vector $\mathrm{x} \in \mathcal{X}$ to generate the output $\mathrm{y} \in \mathcal{Y}$, i.e.:

$$\mathrm{y} = f(\mathrm{x}, \xi) \tag{7.1}$$

where ξ is used to indicate that the model function can accept other inputs. Thus, the ***ModelX*** class requires that all concrete sub-classes provide in implementation of the function *feval()*, which essentially performs the operation described by the model function. Expanding on the same concept, linear models are a class of models where the model function involves the multiplication of a matrix, called the *model matrix*, with the input vector, e.g.:

$$\mathrm{y} = f(\mathrm{x}, \xi) = A(\xi)\mathrm{x} \ldots \tag{7.2}$$

To accommodate for this, the ***LinearModelInterfaceX*** class enforces that a linear model class implements a *matrix()* function, which when evaluated returns the model matrix. In cases where the model function is non-linear, the model becomes non-linear and thus ***NonLinearModelInterfaceX*** requires that such models implement the *jacobian()* function that returns the first-order Taylor approximation to the model matrix.

In cases where the model function is invertible, then the function $f^{-1} : \mathcal{Y} \rightarrow \mathcal{X}$, called the *model inverse function*, can be defined such that

$$\mathrm{x} = f^{-1}(\mathrm{y}, \xi) \tag{7.3}$$

Therefore, for such invertible models the ***InvertibleModelInterfaceX*** enforces that the function *finv()* is implemented, which performs the operation described by the model inverse function.

It is also possible that a given model may be stochastic; as a matter of fact this is almost always the case. In this case, random perturbation/noise is applied when a given input is propagated through the model function, where the noise characteristics can be described by some density. For the time being TrackingX only supports stochastic models with additive noise, where the model function is given as:

$$\mathrm{y} = f(\mathrm{x}, \xi) = f'(\mathrm{x}, \xi) + noise \tag{7.4}$$

Therefore, the ***StochasticModelInterfaceX*** interface enforces that stochastic models have a *NoiseDensity* property, that is of type *DensityX*, and implement the *random()* and *pdf()* functions, which allow for the generation of random noise samples from the *NoiseDensity*, and the evaluation of the density pertaining to the *NoiseDensity*, respectively. By default, the ***StochasticModelInterfaceX*** provides an implementation for *random()* and *pdf()* by calling the *NoiseDensity.random()* and *NoiseDensity.pdf()*, although it is possible to override them. Finally, the ***GaussianModelInterfaceX*** interface is a subclass of ***StochasticModelInterfaceX***, where the *NoiseDensity* is expected to be of type ***GaussianDensityX***, and a default implementation of the method *covar()* is provided, which again can be overridden.

7.2.3.2 Model Types

TrackingX considers five different model types: i) Transition/Dynamic; ii) Measurement; iii) Control; iv) Detection; v) Clutter. An in-depth discussion about the different types of models can be found in Section 2.

Dynamic models

Dynamic models in TrackingX are represented via the ***TransitionModelX*** class. In this context, the function *feval()* implements the relation of (2.1). Examples of implemented dynamic models in TrackingX include ***ConstantVelocityX*** (see Section 2.2.2), ***OrnsteinUhlenbeckX*** (see Section 2.2.3) and ***ConstantHeadingX*** (see Section 2.2.4). The ***TransitionModelX*** class requires that all subclasses include the property *NumStateDims*, which relate to the number of state dimensions that the model considers.

Measurement models

Measurement models are represented via the ***MeasurementModelX*** class, where the *feval()* function implements the relation of (2.19). Examples of implemented measurement models include ***LinearGaussianX*** (see Section 2.3.1) and ***GaussianAzimuthRangeX*** (see Section 2.3.2). Both presented examples are also invertible and thus provide an implementation for the *finv()* function, which implements the relation of (2.21). The ***MeasurementModelX*** requires that all sub-classes include the following properties: i) *NumMeasDims*: The dimensionality of the considered measurement vector; ii) *NumStateDims*: The dimensionality of the state vector transformed by the model; iii) *Mapping*: A mapping that relates indices of measurement vector to indices of the state vector.

Detection models

Detection models are represented via the ***DetectionModelX*** class. In this class of models, the *feval()* function implements the relation of (2.28). The detection models implemented thus far in TrackingX are ***ConstantDetectionRateX*** (see Section 2.4.1) and ***StateDependentDetectionRateX*** (see Section 2.4.2).

Clutter models

Clutter models are represented via the ***ClutterModelX*** class. Due to the inherent stochastic nature of clutter[2], ***ClutterModelX*** is defined such that it directly implements a ***StochasticModelX*** interface. As a matter of fact, following the discussions of Section 2.5, ***ClutterModelX*** are expected to exhibit stochastic behaviour in both the spatial distribution of clutter, as well as the clutter intensity (i.e. number of expected clutter points). Thus the *random()* and *pdf()* functions for this class include an additional expected input to specify the distribution to which each function relates. More details can be found in the documentation of each class. The only existing TrackingX clutter model is the ***PoissonRateUniformPositionX*** (see Section 2.5.1).

7.2.4 Filters

TrackingX includes a comprehensive collection of state-of-the-art Bayesian Filtering algorithms. In the TrackingX architecture, the base class for all filters is the ***FilterX*** class. The structure of ***FilterX*** enforces the following: i) All filters have a *Model* property, that is of type ***StateSpaceModelX*** and contains all information pertaining to the state-space model; ii) All filters must provide implementations of *initialise()*, *predict()* and *update()*; iii) All filters have state related properties *StatePrior*, *StatePrediction*, *MeasurementPrediction* and *StatePosterior*, all of which must be sub-classes of the ***StateX*** class. An explanation for the various state properties is provided below:

- *StatePrior*: The *StatePrior* property aims to encapsulate all information pertaining to the distribution $p(\mathrm{x}_{k-1}|\mathrm{y}_{1:k-1})$[3]; that is the posterior distribution from time t_{k-1}, that serves as a prior at time t_k. The *StatePrior* property is initialised upon object construction, or a call to the *initialise()*

[2]So far the author has not come across s a scenario where the clutter is deterministic

[3]The single-target, single-measurement prior notation $p(\mathrm{x}_{k-1}|\mathrm{y}_{1:k-1})$ is used for simplicity, but depending on the type of filter it should be considered as equivalent to $p(\mathrm{x}_{k-1}|Y_{1:k-1})$ or $p(X_{k-1}|Y_{1:k-1})$. The same applies for all other distributions.

method, and its value is updated at the start of each call to the *predict()* method.

- *StatePrediction*: The *StatePrediction* property aims to encapsulate all information pertaining to the distribution $p(\mathrm{x}_k|\mathrm{y}_{1:k-1})$, that is the predicted distribution at time t_k. The *StatePrediction* is updated after each call to the *predict()* method.

- *MeasurementPrediction*: The *MeasurementPrediction* property aims to encapsulate all information pertaining to the distribution $p(\mathrm{y}_k|\mathrm{y}_{1:k-1})$, that is the expected distribution of measurements at time t_k. The *Mesurement-Prediction* is updated after each call to the *predict()* method.

- *StatePosterior*: The *StatePrediction* property aims to encapsulate all information pertaining to the distribution $p(\mathrm{x}_k|\mathrm{y}_{1:k})$; that is the posterior state distribution at time t_k. The *StatePosterior* is updated after each call to the *update()* method.

By default, ***FilterX*** objects are designed with the intention to be utilised as (Markov-Chain) State Estimators that operate on a given problem as deterministic Finite State Machines (FSMs), with self-contained memory. In other words, it is expected that for most problems, a ***FilterX*** object will go through any heavy parameterisation once, after which it shall be used to recursively execute a prediction-update state estimation loop. ***FilterX*** objects aim to preserve all the inter-state information they require such that they can execute the next recursion step, without being provided with information they already know. For example, although it is possible to do so, it is not necessary to provide a prior each time the *predict()* method is used, as the filter will utilise its *StatePosterior* property (generated from the last call to the *update()* method) as its prior, thus storing it in the *StatePrior* property. That said, it is possible to provide a prior as an input argument to a *predict()* call, in which case the filter object will overwrite its memory with the provided information. The same stands for the *StatePrediction* and *MeasurementPrediction* properties when it comes to the *update()* method.

The full list of filters implemented in TrackingX, along with annotated relations between them, is depicted in Figure 7.3. As mentioned previously, ***FilterX*** is the base Abstract class for all filter objects. Next, three different abstract subclasses are defined: i) ***KalmanFilterX***; ii) ***ParticleFilterX***; and iii) ***PHD-FilterX***. All classes that are direct sub-classes of the ***KalmanFilterX*** and ***ParticleFilterX*** base classes (with the exclusion of the ***BernoulliParticle-Filter***) are interchangeable. A summary of the functionality provided by each type of filter is provided below.

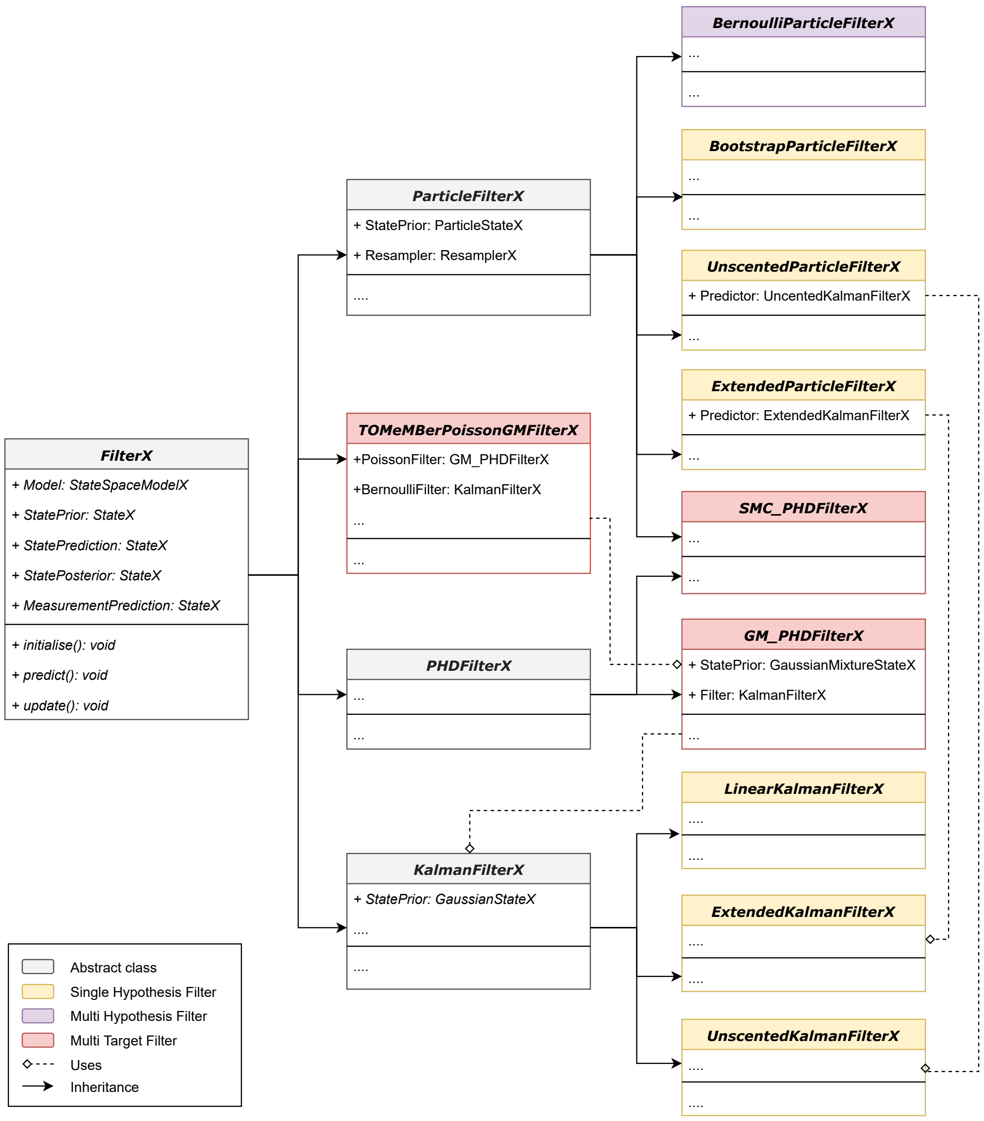


Figure 7.3: Class structure of TrackingX filter components

The ***KalmanFilterX*** class is the base class for all Kalman Filters, thus enforcing that all state related information (*StatePrior*, *StatePrediction*, *MeasurementPrediction*, *StatePosterior*) is stored in the form of ***GaussianStateX*** objects. The class makes use of MATLAB's Property Access Methods to ensure that, should a ***KalmanFilterX*** sub-class be provided with state information of a type different than ***GaussianStateX***, an attempt is made to convert it into such a type (using the state interchangeability functions discussed in Section 7.2.2.1), returning an error if it fails. Furthermore, the ***KalmanFilterX*** class provides an implementation for the *update()* method which, given a *StatePrediction* and a *MeasurementPrediction*, is equivalent for all sub-classes. Then, the ***LinearKalmanFilterX*** (see Algorithm 3.2.2), ***ExtendedKalmanFilterX*** (see Algorithm 3.2.3) and ***UnscentedKalmanFilterX*** (see Algorithm 3.2.4) classes simply implement the *predict()* method which is different between the 3 algorithms.

The ***ParticleFilterX*** class is the base class for all Particle Filters, thus enforcing that all state related information is stored in the form of ***ParticleStateX*** objects. The class makes use of MATLAB's Property Access Methods to ensure that, should a ***ParticleFilterX*** sub-class be provided with state information of a type different than ***ParticleStateX***, an attempt is made to convert it into such a type (using the state interchangeability functions discussed in Section 7.2.2.1), returning an error if it fails. The ***BootstrapParticleFilterX*** is an implementation of Algorithm 3.2.5, where the importance density is set to $p(\mathrm{x}_k|\mathrm{x}_{k-1})$. The ***ExtendedParticleFilterX*** and ***UnscentedParticleFilterX*** sub-classes utilise an ***ExtendedKalmanFilterX*** and ***UnscentedKalmanFilterX***, respectively, to get an approximation to the optimal proposal $p(\mathrm{x}_k|\mathrm{x}_{0:k-1}, y_{1:k})$. Finally, the ***BernoulliParticleFilterX*** is an implementation of a single-target Bernoulli Particle Filter, as documented in Algorithm 2 of [183].

The ***PHDFilterX*** class is the base class for all Probability Hypothesis Density (PHD) filter implementations in TrackingX. Unlike the base classes discussed so far, the ***PHDFilterX*** class does not enforce any restictions in terms of the state information, but defines properties such as the *BirthIntensityFunction* and *SurvivalProbability* (see γ_k and p_S in Algorithm 4.5.1, respectively) that are required by the algorithm. The ***SMC_PHDFilterX*** is a Particle-based implementation of the PHD filter (thus it inherits its state related information from ***ParticleFilterX***), whose algorithm is a direct implementation of the one documented in Section 3.4 of [68]. The ***GM_PHDFilterX*** class is a GaussianMixture implementation of the PHD Filter and therefore enforces that all state related information are of type ***GaussianMixtureStateX***. An additional functionality of the ***GM_PHDFilterX*** is that it is able to utilise any of the ***KalmanFilterX*** sub-classes to perform prediction and update operations on the individual ***GaussianStateX*** components of its state. The algorithmic implementation of the ***GM_PHDFilterX*** is extracted from Section 3.2 of [67].

Finally, a GaussianMixture implementation of the Track-Oriented Marginal MeMBer-Poisson Filter proposed in [83], is provided via the ***TOMeMBerPoissonGMFilterX***. Although fully functional, the specific class is still in a development stage and thus it has not been fully incorporated in the TrackingX architecture. The class essentially maintains 2 sets of state related information, each represented using a ***GaussianMixtureStateX***. The first is related to the density of undetected targets, which is tracked using a ***GM_PHDFilterX***, and the second is related to the set of "confirmed" (or detected) targets, where each component is predicted and updated using a ***KalmanFilterX*** subclass.

7.2.5 Data Associators

TrackingX includes implementations of a number of Data Association algorithms. In the TrackingX architecture, the base class for all data associators is the ***DataAssociatorX*** class and the main requirement enforced by the base class is that all sub-classes implement a method called *associate()*. This method should expect 2 required arguments: i) *TrackList*: A list of ***TrackX*** objects, ii) *MeasurementList*: A list of ***MeasurementX*** objects. The output of the *associate()* method is a list of association hypotheses that can be used to update each track in the *TrackList*.

At the time of writing this thesis, TrackinX includes implementations of 5 Data Association algorithms (see Figure 7.4, all of which have been thoroughly documented in this thesis. The first algorithm is the ***NearestNeighbourDataAssocX***, that implements the Nearest Neighbour approach discussed in Section 3.3.2. Continuing the ***GlobalNearestNeighbourDataAssocX*** class is derived from the ***NearestNeighbourDataAssocX*** class and provides an implementation of the Global Nearest Neighbour algorithm discussed in Section 4.3.3.2. Similarly, an implementation of the Probabilistic Data Association algorithm discussed in Section 3.3.3 is provided by the ***ProbabilisticDataAssocX*** class, which also forms the base class for ***JointProbabilisticDataAssoc*** that implements the Joint Probabilistic Data Association algorithm of Section 4.3.3.3. Finally, and based on the notes presented in Section 4.4, the Joint Integrated Probabilistic Data Association algorithm is implemented via the ***JointIntegratedProbabilisticDataAssocX*** class that extends ***JointProbabilisticDataAssocX***. All of the above classes can be seamlessly interchanged within all TrackingX components that utilise them.

7.2.6 Track Managers

Track Management is handled in TrackingX through appropriately defined ***TrackInitiatorX*** and ***TrackDeletorX*** sub-classes. As discussed in Section 4.3.2, the approaches to track management can vary greatly. The role of ***TrackInitiatorX*** is then to utilise unassociated detections, such as to initiate and confirm tracks though the expected *initialise()* method. Then ***TrackDeletorX*** can be used to

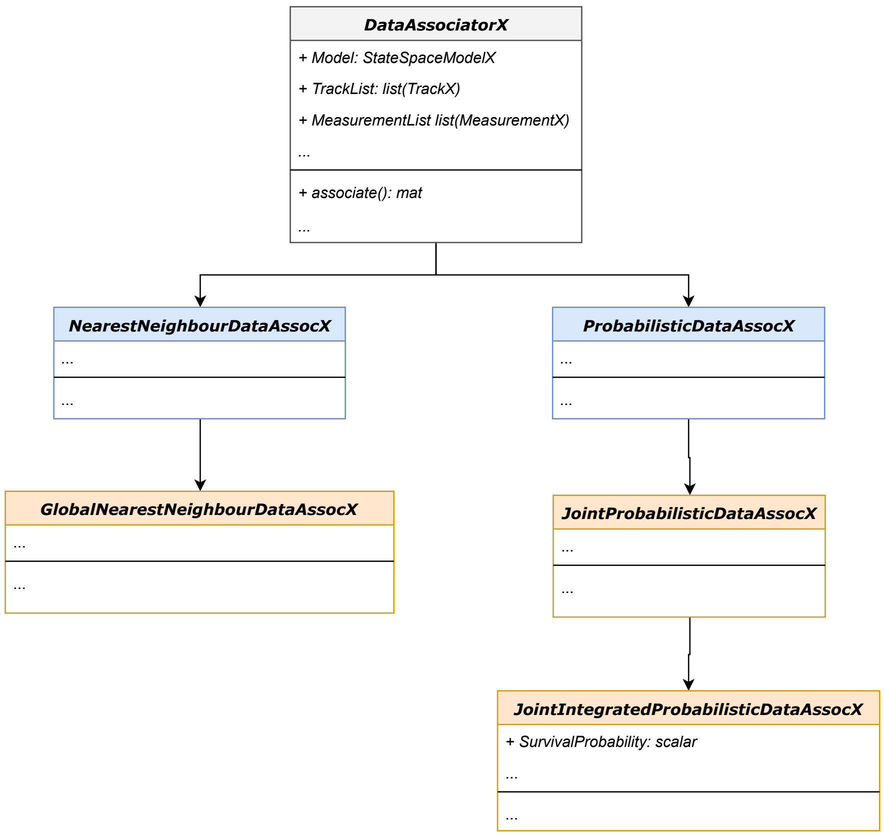

DataAssociatorX
+ Model: StateSpaceModelX
+ TrackList: list(TrackX)
+ MeasurementList list(MeasurementX)
...
+ associate(): mat
...
NearestNeighbourDataAssocX
...
...
ProbabilisticDataAssocX
...
...
GlobalNearestNeighbourDataAssocX
...
...
JointProbabilisticDataAssocX
...
...
JointIntegratedProbabilisticDataAssocX
+ SurvivalProbability: scalar
...
...

Figure 7.4: Class structure of TrackingX data association components

delete confirmed tracks by calling its *delete()* method.

TrackingX includes implementations of 3 track initiators. The first is the ***MofNTrackInitiatorX***, which is based on the *M-out-of-N* approach discussed in Section 4.3.2. The ***MofNTrackInitiatorX*** makes use of the ability to append dynamic properties to ***TrackX*** objects by defining a new property *DetectionHistory*, which is a binary vector of length M, that indicates whether a given track of successfully detected. The second track initiator is the ***LLRTrackInitiatorX***, which implements the *Log Likelihood Ratio Test* approach discussed in Section 4.3.2. On each call, the ***LLRTrackInitiatorX*** computes the Log Likelihood Ratio (LLR) for each track, and makes updates the *Score* property of ***TrackX*** objects. Both the ***MofNTrackInitiatorX*** and ***LLRTrackInitiatorX*** classes have been designed such as to allow seamless interchangeability of the ***DataAssociatorX*** and ***FilterX*** classes used to perform data association and filtering of tentative tracks. Finally, the third track initiator is the ***PhdTrackInitiatorX*** that implements the PHD track initiation scheme discussed in Section 4.6, whereby a ***PHDFilterX*** object is used to filter the density of tentative tracks. The ***PhdTrackInitiatorX*** allows users to specify the type of ***FilterX*** that should be configured and attached to the *Filter* property of initiated ***TrackX*** objects.

TrackingX also includes 2 implementations of ***TrackDeletorX*** classes. The first is a ***PropertyBasedDeletorX***, which allows users to specify a condition for deletion which is applied to a selected property of ***TrackX*** objects. The nature of this track deleter provides users with a lot of flexibility and can also be used as the base class for defining more specific ***TrackDeletorX*** implementations. For example, the ***PropertyBasedDeletorX*** can be configured to delete tracks whose *Score* property falls below a given threshold, which could also be formalised into a ***ScoreBasedTrackDeletorX*** class. The second ***TrackDeletorX*** implementation is the ***TimeBasedDeletorX***. Once provided with a given duration, the ***TimeBasedDeletorX*** identifies the last time a given ***TrackX*** object was successfully updated with a measurement and deletes the object if the elapsed time is higher that the specified duration.

7.2.7 Metric Generators

TrackingX also includes provisioning for metric generators that can be used to evaluate performance during simulations. This functionality is provided though the base class ***MetricGeneratorX***. The definition of this class specifies that each subclass must provide an implementation for an *evaluate()* methods, which when provided with two lists of ***TrackX*** objects, one relating to ground-truth tracks and on to estimated tracks, outputs a metric that indicates the estimation accuracy.

At the time of writing, TrackingX provides two solid implementations of the ***MetricGeneratorX*** class. The first is the ***OSPAMetricGeneratorX***, which implements the Optimal Sub-Pattern Assignment (OSPA) metric [106]. The second implementation is the ***GOSPAMetricGeneratorX***, that implements the Generalised OSPA (GOSPA) [184], using an adapted version of the MATLAB implementation provided by the supplementary materials of [185].

7.2.8 Simulators

TrackingX enables users to simulate both ground-truth data, as well as measurements based on such ground-truth data. This is achieved by the ***GroundTruthSimulatorX*** and ***MeasurementSimulatorX*** classes respectively.

The ***GroundTruthSimulatorX*** class defines the base class for simulators that generate ground-truth data related to one or more targets moving through the detection area. The objective of for ground-truth simulators is to generate position data at defined intervals, based on the dynamic behaviour specified through a given ***TransitionModelX***. TrackingX includes an implementation of a ***SingleTargetGroundTruthSimulatorX*** and a ***MultiTargetGroundTruthSimulatorX*** which can be used to generate ground-truth for a single and multiple targets, respectively. The ***SingleTargetGroundTruthSimulatorX*** accepts a single ***TrackX*** object, that should contain an initial state, and outputs the same object, whose trajectory has been simulates and stored in the *States* property of the object. The ***MultiTargetGroundTruthSimulatorX*** performs the same operation, but for a list of ***TrackX*** objects. In both cases, the lifetime of tracks can either be specified deterministically, via the inclusion of a *TimeOfDeath* property for each track, or can be allowed to be determined by the simulator by specifying a *SurvivalProbability.*

The ***MeasurementSimulatorX*** class defines the base class for simulators that simulate detections by a sensor, whose measurement model characteristics are specified through a given ***MeasurementModelX***. The ***SingleTargetMeasurementSimulatorX*** class is a solid implementation of the ***MeasurementSimulatorX***, that aims to generate detections for a single ground-truth target and does not consider the existence of clutter and missed detections. The ***MultiTargetMeasurementSimulatorX*** extends the ***SingleTargetMeasurementSimulatorX*** to a multi-target scenario in the presence of clutter and missed-detections, whose properties are specified by user defined ***ClutterModelX*** and ***DetectionModelX*** implementations, respectively. Both simulators accept ground truth data as inputs in the form of a ***TrackX*** object or a list of ***TrackX*** object respectively, and output measurements in the form of a list of ***MeasurementX*** objects for each discrete time interval.

7.2.9 Simulation Example

A simple use-case example is demonstrated here to showcase the usage of TrackingX. The overall structure of the different TrackingX components is shown in Figure 7.5. A ***MultiTargetGroundTruthSimulatorX*** is used to generate ground-truth data for 10 targets, which is then forwarded to a ***MultiTargetMeasurementSimulator*** that generates detections based on the ground-truth data. Continuing, a Tracker loop is defined that makes use of a ***PHDTrackInitiatorX***, that initiates confirmed tracks, each parameterised with a ***ParticleFilterX***, from the intensity of a ***SMC_PHDFilterX***, a ***PropertyBasedDeleter***, that deletes tracks based on their *Score* property, and a ***JointIntegratedProbabilisticDataAssocX*** to perform Data Association. Finally, the tracks generated from the Tracker loop, together with the ground-truth tracks, are forwarded to a ***GOSPAMetricGeneratorX***, that evaluates the accuracy of the tracker and returns a GOSPA metric. Figure 7.6 shows an example plot output from the tracking process, while Figure 7.7 shown a plot of the GOSPA metric evaluated over the course of the simulation.

7.3 Stone Soup

Stone Soup [12, 13, 14] is an open-source Python framework that aims to provide researchers and practitioners with the ability to develop and implement new and existing tracking and state estimation algorithms for ease of comparison. The content presented herein is derived from and incorporates text published in [14], licensed under the Open Government License [186], that announces the beta release of Stone Soup, to which the author has been a major contributor and co-author[4].

Development efforts for Stone Soup are led by the UK Defence Science and Technology Laboratory (Dstl) and have thus far included contributions from the Defence Research and Development Canada (DRDC), the US Air Force and Research Laboratory (AFRL) and the University of Liverpool (UoL). These efforts have been part of the first Stone Soup development phase; the Consortium phase.

[4]As Stone Soup is still under development at the time of writing this thesis, the structural composition of Stone Soup may deviate from what is reported herein.

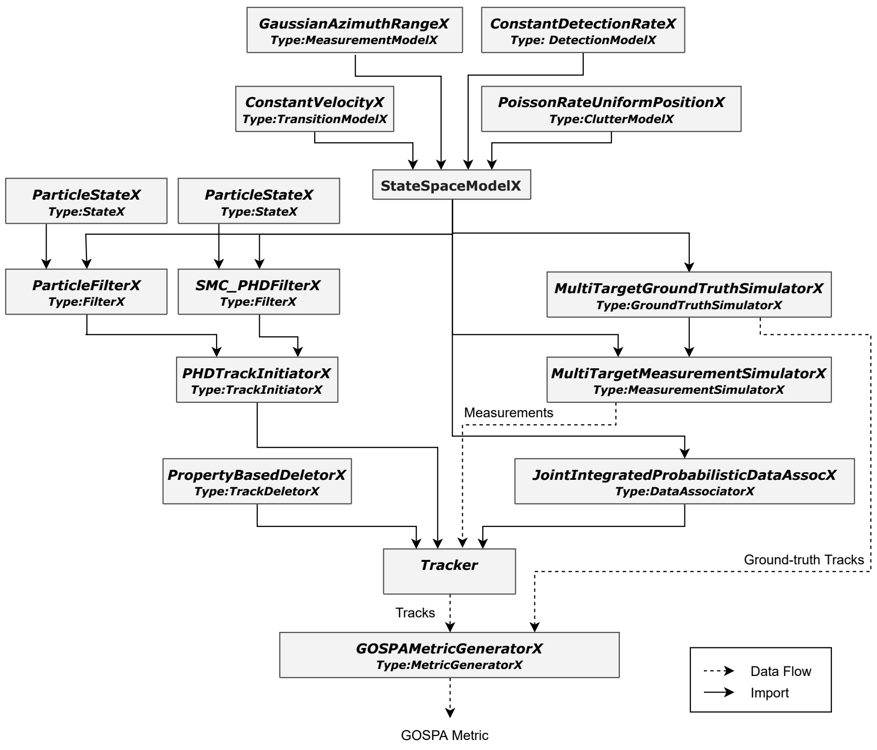
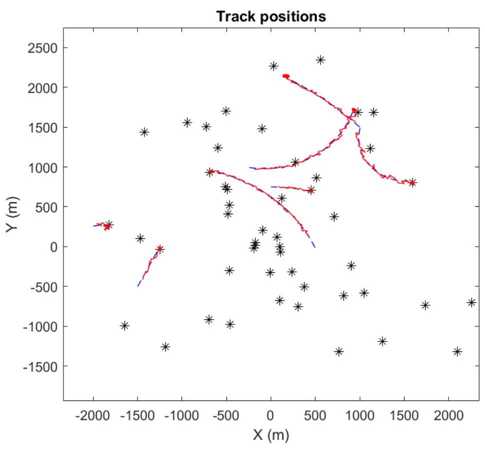
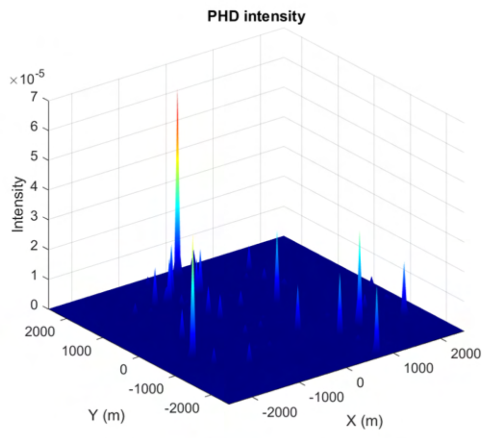


Figure 7.5: Architecture of TrackingX Use Case



(a) Positions of estimated tracks



(b) Intensity of ***PhdTrackInitiatorX***

Figure 7.6: Plot output of TrackingX simulation example

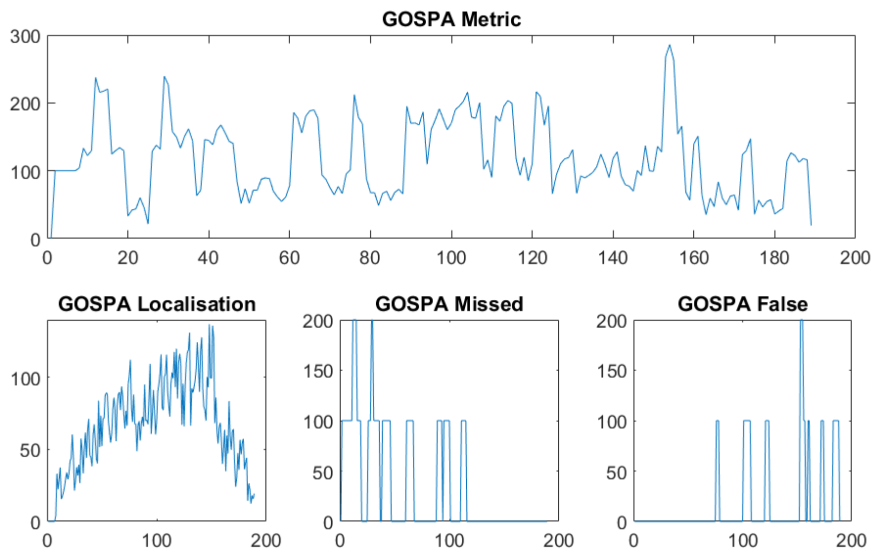


Figure 7.7: GOSPA metric output for the example simulation

This phase extended from the Stone Soup program commencement in 2017 until the open source release date in April 2019. In this phase, the consortium of contributors came together to define the requirements and develop the Stone Soup framework. Work in this stage was focused on building the framework and the components for a few exemplar tracking scenarios; work on additional components was left primarily until the second phase.

The second phase (ongoing) of the Stone Soup project is the Open phase, which began in April 2019 when Stone Soup was open-sourced to the tracking community as the beta version (V0.1b1). During this phase, is is expected that the main contributors to the project will be academic researchers, who contribute components to the project in order to receive academic recognition. These contributions will be reviewed by the Stone Soup development committee (consisting of members from Dstl and DRDC) for merging into the main Stone Soup branch; however, it is possible for individual researchers to maintain and develop custom Stone Soup code repositories for their own projects.

Stone Soup is initially targeted at two different groups of users. The first is academics conducting research into tracking and state estimation. Academics will be able to implement new tracking algorithms in Stone Soup, evaluate their performance against built-in metrics, and compare the results with those of other tracking algorithms. This will lend credence to claims of improved performance in published research, and facilitate rapid replication by others in the academic community.

The second group is the users, owners or processors of real world datasets or sensor systems. Members of this group may have data that they wish to exploit, but they do not necessarily have deep expertise in multiple tracking methods. Stone Soup allows this group to construct the appropriate experiment to test many algorithms against their data to determine which algorithm best suits their application. Stone Soup will provide a quick-start tracker builder that will assist these users in creating experiments without requiring a deep knowledge of the tracker components being used.

For instance, users could test a new tracking algorithm against the body of existing algorithms, or they could run real-world or simulated sensor data through multiple tracking algorithms to determine which is best for their application. Although the project is open-source, it can be downloaded and used to implement proprietary algorithms, or to evaluate proprietary/classified data. Stone Soup is designed with modular components, which allows users to rapidly assemble different combinations of components (including in new and unexpected configurations) to build the best tracker for their application.

7.3.1 Framework architecture

The philosophy behind Stone Soup is very similar to that of the TrackingX framework discussed in Section 7.2. It aims to enable users to test, verify, and benchmark a variety of multi-sensor and multi-object estimation algorithms. The framework is built in Python, with the intent of providing support for integration of tracking algorithms in other high-level languages, such as MATLAB, C and C++. In order to achieve these goals, Stone Soup is comprised of a framework with modules related to Data, Algorithms, Metrics, Simulators, and Sensor Models (Figure 7.8). The central theme of the Stone Soup design philosophy is interchangeability. The framework is designed with the idea that users can easily insert their own components (e.g. predictors, updaters, associators, data readers, etc.) into existing Tracker constructs, with the added ability to mix and match components in new and unexpected ways. In support of this goal, the Stone Soup code architecture has been built on the principles of modularity and uniformity of external interfaces.

Stone Soup is object-oriented and makes use of encapsulation, abstraction and inheritance, where trackers are built as hierarchical objects. For example, a ***Tracker*** object may contain ***Track Initiator***, ***Track Deleter***, ***Detector***, ***DataAssociator***, and ***Updater*** objects (Figure 7.10). An abstract class is defined for each object which specifies the external interface for that class; that is,

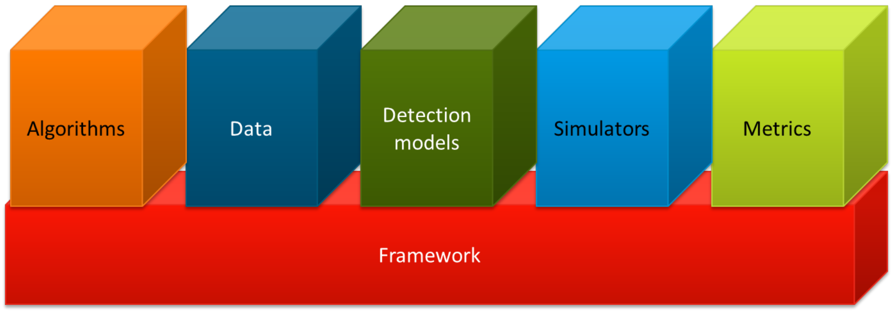


Figure 7.8: Stone Soup framework architecture. [14]

the parameters and functions an object of that class must make available to the outside world (abstraction). As an example, the ***Updater*** abstract class specifies that any derived object must have a property of type ***MeasurementModel***, and that it must have a function *update()* that returns a ***State*** object. Therefore, all implementations of ***Updater*** objects in Stone Soup (e.g. ***KalmanUpdater***, ***ExtendedKalmanUpdater***, ***ParticleUpdater***, etc.) must have the specified properties and functions (inheritance). This approach ensures that different ***Updater*** objects are interchangeable (within limits), and that the ***Tracker*** can utilise them without knowing the details of their implementation (encapsulation).

In order to support this modularity and common external interfaces, Stone Soup objects are built based on a class inheritance hierarchy (Figure 7.9). Stone Soup contains a ***Base*** abstract class, which is an empty class, and all other Stone Soup classes are descended from this. All implementations of a given abstract class must be children of that abstract class, which defines the common external interface for that abstract class. Whenever a new object is instantiated, Stone Soup will verify that the instantiation call includes all of the properties that are required by that class and all ancestors in the hierarchy; this check helps enforce the interchangeability of the Stone Soup modules.

It should be noted the partial inheritance hierarchy in Figure 7.9 has been designed to be simple and easy to understand for new users, and it is not definitive; contributors to Stone Soup are free to suggest changes and develop alternative abstract classes and inheritance hierarchies.

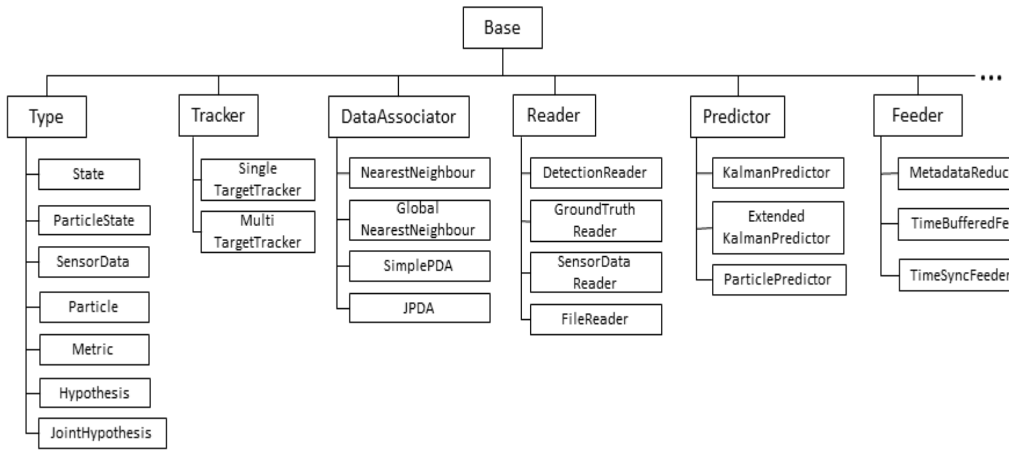


Figure 7.9: An incomplete view of the Stone Soup class inheritance hierarchy. [14]

7.3.2 Components

7.3.2.1 Data

The basis of most tracking and state estimation algorithms is sensor detection data (with the notable exception of Track-Before-Detect methods). Stone Soup is capable of both generating simulated target and detection data, as well as ingesting real-life data from arbitrary external sources. Both options offer advantages for different reasons. Simulated data is useful for the rapid implementation and testing of new tracking algorithms. Such simulated data gives access to both the ground-truth target information (e.g. position, velocity, etc.), as well as the sensor detections. This is useful because the majority of metrics require ground-truth information in order to evaluate the accuracy of the tracking algorithms. Real-life data is useful for testing tracking algorithms against real-world situations (as opposed to laboratory situations) in order to evaluate which algorithm performs the best in a particular situation. However, as is often the case, real data typically come without the ground-truth data required by certain performance metrics.

Stone Soup contains several models that can be used to simulate target (e.g. airplane, ship) movement in order to generate simulated datasets. This simulated data represents the ground-truth data, which is later detected by sensor detection models; these models add uncertainty to the measurements representing the error associated with real sensor measurements. This simulated motion is generated by a number of 1-dimensional transition models, which can be combined into higher-dimensional transition models that describe the targets movement. The process for generating simulated data is described in more detail in Section 7.3.2.4.

Stone Soup provides options for writing simulated detection data to an output file, and reading this data back into Stone Soup. This is useful because the simulated data is generated using a Python generator, which generates data just-in-time rather than pre-generating it. This approach saves computer memory space for large simulated datasets, but it also means that, by default, the dataset does not persist. For users who wish to save the simulated data (e.g. to run the same simulated dataset against multiple tracking algorithms), these file writing/reading options provide persistent access to the data. The class ***YAMLWriter*** writes simulated data to an output file in the YAML format, and the class ***YAMLReader*** reads this data back into Stone Soup format.

Stone Soup has been developed to support data from many types of real-world sensors. Working with this data typically requires a ***Reader*** class that reads sensor data from an input stream. It also often requires a ***Feeder*** class that translates data from the sensors measurement space into Stone Soups state space, orders out-of-sequence data chronologically, and removes duplicate data. Stone Soup provides several reader and feeder classes, but users of Stone Soup can easily write their own to serve their specific applications.

7.3.2.2 Tracking Algorithms

The core of Stone Soup is the tracking functionality. A ***Tracker*** is the top-level algorithm that utilises several other algorithmic components to generate Tracks from Detections. Stone Soup has been designed to make these components as modular as possible, allowing users to interchange them to evaluate new components and test different combinations. Figure 7.10 shows the types of these components and the dependency relationships between them. The base ***Tracker*** object requires ***Initiator***, ***Deleter***, ***Detector***, ***DataAssociator***, and ***Updater*** components. Different Trackers can handle a single or multiple targets moving through the space, and can identify and discard clutter.

The ***Initiator*** initialises new ***Track*** objects from a provided set of ***Detection*** objects, that are determined not to be part of another ***Track***; it sets up the initial parameters that allow the ***Track*** position to be predicted into the future. The ***Deleter*** kills a ***Track*** after certain conditions have been met which indicate that the ***Track*** has been lost. For example, a ***Track*** may be deleted if it has not been detected for a certain number of time steps, or the uncertainty associated with its prediction is very large for the same reason.

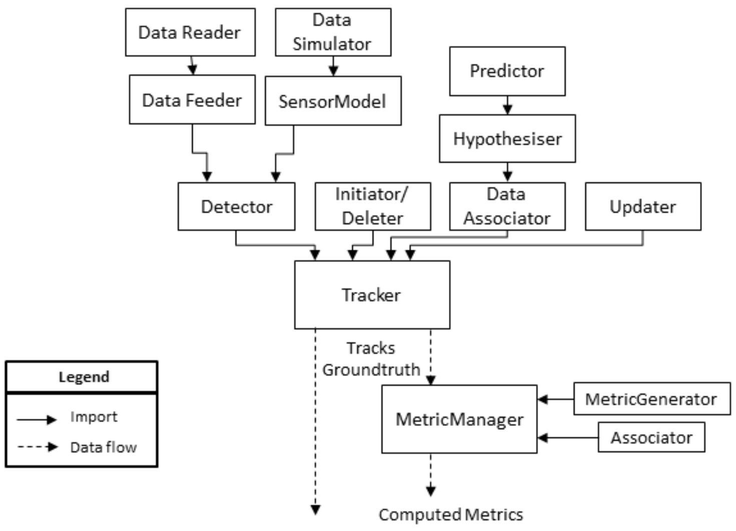


Figure 7.10: Stone Soup module structure and information flow. [14]

The ***Detector*** provides a stream of ***Detection***s to the ***Tracker***. The data source for the ***Detector*** is a detection simulator (for simulated data), real data read from a file or a data stream, or any other data input methods that users would like to develop. In order to use simulated data, the user may employ a ground truth data simulator that utilises a transition model to model the motion of the targets. The simulated ground-truth generated by this simulator is then processed by detection simulator, which uses a measurement model that approximates how a sensor detects the ground-truth data (with uncertainty of detection location, missed detections, etc.). As discussed in Section 7.3.2.3, both the ground-truth and the detections generated by the measurement model are used as inputs to the various Stone Soup metrics.

The ***Predictor*** and ***Updater*** are the foundation of any tracking scheme; they implement the functionality of the underlying basic filter. While Stone Soup currently contains implementations of a few basic filters, this is an area with rich opportunities for development by the academic/user community. Typically, a ***Predictor*** and ***Updater*** with matching approaches must be paired (e.g. both Kalman, both Particle).

The data association function of a tracker with which the tracking community is likely to be familiar is encapsultated in Stone Soup using two different components: the ***Hypothesiser*** and the ***DataAssociator***. Roughly speaking, the data association process associates Detections with Tracks (Targets). In Stone Soup, the ***Hypothesiser*** calculates the measures (e.g. distance, probability) between the Detections and predicted Track states on which this decision is based, and the ***DataAssociator*** makes this decision. We realize that different classes of tracking algorithms calculate and represent the state of the system in different ways. As befits a project that is under continuous development, accomodations for this fact are ongoing. Readers should refer to Section 7.3.4 for future plans in this area. Users who are implementing new tracking algorithms in Stone Soup are recommended to look at existing implementations to help them decide the best approach to implementing their own algorithm.

7.3.2.3 Metrics

One of the main goals of Stone Soup is to compare the performance of different tracking algorithms. Currently, Stone Soup provides support for metrics that evaluate the accuracy of tracking algorithms. These metrics accept the output of a Tracker operating over a dataset and evaluate how well the Tracks (what we think happened) match the ground truth (what actually happened).

There are three major components related to Stone Soup metrics (Figure 7.10). The ***MetricManager*** parses and stores the Tracks and ground truth produced by the Tracker (and any other outputs produced by future Tracker implementations), and calls the metrics that have been assigned to it. Several of the metrics for Tracker performance measure how well the identified Tracks match the ground truth. In order to measure this, Stone Soup uses an ***Associator*** to associate Tracks and ground truth. The metrics themselves are calculated by ***MetricGenerators***. These classes accept the Tracker outputs and return the calculated metrics. Stone Soup provides implementations of several industry-standard metrics; users are also free to implement other metric suites and add them to the Stone Soup code base.

7.3.2.4 Data Simulators

In the case where Stone Soup users do not have real datasets available to test tracking algorithms, Stone Soup provides simulators that can generate realistic Detection datasets. Simulating Detections is a 2-step process: 1) generate ground truth data based on transition models, and 2) model the sensor state and behaviour as it detects the ground truth Targets.

The Stone Soup ***GroundTruthSimulators*** generate data related to one or more Targets moving through the detection area. They generate position data at defined time steps, and the Target motion is based on user-selected transition models. The ***DetectionSimulators*** simulate detections by a sensor platform whose characteristics are specified by the user-specified measurement model. The simulator accepts ground truth data as inputs, and outputs Detections as well as Clutter measurements based on the state of the measurement model. Detections can be generated using a generic sensor model or one of the sensor models discussed in Section 7.3.2.5.

Simulated data is important for fully exploring the capabilities of different tracking algorithms because ground truth data is needed for several of the the Stone Soup metrics and real sensor data often does not include ground truth data.

7.3.2.5 Sensor Models

Stones Soup contains a SensorPlatform class that represents a mobile platform that can move in 2D or 3D space to represent the motion of an aircraft, ship, or other mobile sensor. The platform motion can consist of any of the motion models that have been previously described. A platform can contain multiple sensors to model a variety of real-world platforms; e.g. a platform containing both an EO/IR sensor and a radar sensor. The sensors can be mounted at arbitrary positions on the platform.

Stone Soup contains a generic sensor model as well as several sensor models that simulate the physics of sensor detections in greater detail. These sensor models replicate the characteristics, strengths, and weaknesses of distinctive real-world sensors so users can analyse the performance of different tracking algorithms against sensor-specific datasets.

Each sensor on a platform contains an instance of a measurement model. The measurement model is responsible for transforming the models state vector into a measurement vector. The measurement models also specify the noise properties of the measurement; e.g. noise covariance, so that these can be used in the tracking filter to update the state estimates. The measurement model can also be used to generate a set of noisy measurements drawn from a Gaussian distribution; these measurments can be used in the particle filter implementations.

7.3.2.6 Stone Soup Component Implementations

The objective of the Stone Soup development team prior to the April 2019 open source release date was to develop a framework to which the tracking community can contribute their own implementations of different algorithms. However, as part of the initial development, the Stone Soup development team implemented a few tracking algorithms along with their component parts as a starting point. We expect that these implementations will serve as a base case that other contributors can build upon and make their own contributions to the project. A listing of the beta release components can be found in Table 7.1.

Simulator	**Deleter**
SingleTargetGroundTruthSimulator	CovarianceBasedDeleter
MultiTargetGroundTruthSimulator	UpdateTimeStepsDeleter
SimpleDetectionSimulator	UpdateTimeDeleter
File Reader	**Predictor**
JSON_AISDetectionReader	KalmanPredictor
CSVDetectionReader	ExtendedKalmanPredictor
YAMLReader	UnscentedKalmanPredictor
	ParticlePredictor
File Writer	
YAMLWriter	**Updater**
	KalmanUpdater
Feeder	ExtendedKalmanUpdater
MetadataReducer	UnscentedKalmanUpdater
TimeBufferedFeeder	ParticleUpdater
TimeSyncFeeder	
	Hypothesiser
Transition Models	DistanceHypothesiser
CombinedLinearGaussianTransitionModel	FilteredDetectionsHypothesiser
LinearGaussianTimeInvariantTransitionModel	PDAHypothesiser
ConstantVelocity	
ConstantAcceleration	**Data Associator (tracker)**
Singer	NearestNeighbour
SingerApproximate	GlobalNearestNeighbour
ConstantTurn	SimplePDA
	JPDA
Measurement Models	
LinearGaussian	**Tracker**
RangeBearingElevationGaussianToCartesian	SingleTargetTracker
RangeBearingGaussianToCartesian	MultiTargetTracker
BearingElevationGaussianToCartesian	MultiTargetMixtureTracker
Sensor models	**Metric Manager**
RadarRangeBearing	SimpleManager
RadarRotatingRangeBearing	
	Metric Generator

(mobile) Platform models	BasicMetrics
SensorPlatform	GOSPAMetric [184]
	OSPAMetric [106]
Initiator	TwoDPlotter
SinglePointInitiator	SIAPMetrics [187]
LinearMeasurementInitiator	
GaussianParticleInitiator	**Measure**
	Euclidean
Data Associator (metrics)	EuclideanWeighted
EuclideanTrackToTrack	Mahalanobis
EuclideanTrackToTruth	SquaredGaussianHellinger
	GaussianHellinger

Table 7.1: Component implementations in Stone Soup (April 2019).

7.3.3 Using Stone Soup

7.3.3.1 Run Manager

The process for building a Stone Soup Tracker to run an experiment can be complex and time-consuming if performed manually (see supplemental materials associated with Section 7.3.3.3). We expect that members of the Stone Soup user community will want to run large blocks of experiments. They may wish to run experiments on a given Tracker with certain components swapped or parameters varied over a range in order to find the optimal combinations.

Alternatively, they may wish to run a Monte Carlo simulation that runs the same experiment many times with different input data in order to see how the Tracker performs over a wide variety of situations. In order to reduce the workload for building and running these types of experiments, Stone Soup provides a Run Manager (available a few weeks after the initial beta release) that can automatically build, run, and collect the results from a large block of experiments.

The Tracker and Metrics configurations to be used in an experiment are encoded into an experiment configuration file, along with the conditions for running the experiment multiple times or varying one or more of the components or model parameters. This experiment configuration file is then fed into the Run Manager, which can build and run all of the Trackers specified. The Run Manager then collects the results and metrics from the experiments and then exports them to an output file for further processing and analysis.

7.3.3.2 Jupyter Notebooks

Stone Soup comes with extensive documentation regarding its component parts and how they fit together to build Tracker experiments. However, sometimes the best way to learn is through example and demonstration. The Stone Soup development team makes extensive use of Jupyter Notebooks [188] which combine Python code with supplemental explanations into a format that can be executed using a web browser as a graphical frontend.

The Stone Soup beta release provides a library of Jupyter Notebooks that demonstrate different use cases and show how Stone Soup components fit together to build Tracker experiments. Members of the Stone Soup user community can experiment with these Notebooks in order to learn how Stone Soup works, and they can modify the code in order to integrate their own components developed for use in the Stone Soup framework, or to import their own datasets.

7.3.3.3 Example Use Case

The Jupyter Notebook One of the Jupyter Notebooks in the Stone Soup library [189] entitled "Stone Soup SPIE Use Case.ipynb" demonstrates a simple use case for the Stone Soup framework. This use case showcases the different components used to build a Stone Soup Tracker and how they fit together. In this example (Figure 7.11), a Tracker operates over a simulated dataset, and metrics are calculated indicating how well the Tracker performed.

The Tracker used in this scenario is a Kalman Filter-based Tracker. It uses basic Stone Soup implementations such as the Nearest Neighbour data associator, Mahalanobis Distance hypothesiser, track initiator, and data simulator, as well as the Kalman Predictor and Updater. The data simulator generates ground truth Target location data and detections from the simulated sensor; the Tracker consumes this data and then outputs the Tracks calculated by the Tracker. The results are processed by a MetricManager; it uses several MetricGenerators to calculate metrics over the calculated Tracks, as well as a Track to Truth data associator that measures how well the calculated Tracks match the ground truth.

Figure 7.12 shows the graphical representation of the Tracks produced by a particular run of this Use Case. Table 7.2 shows the numerical results of the Basic Metrics and the SIAP Metrics calculated by this run, and Figure 7.13 shows the OSPA distance calculated by the OSPA Metrics at each time step over the course of the experiment.

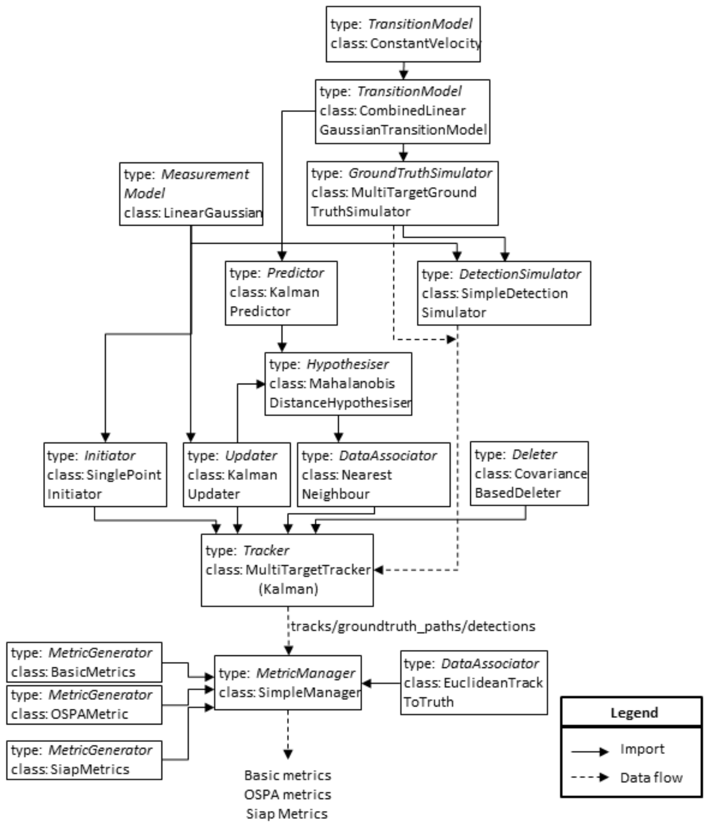

type: TransitionModel class: ConstantVelocity
type: TransitionModel class: CombinedLinear GaussianTransitionModel
type: Measurement Model class: LinearGaussian
type: GroundTruthSimulator class: MultiTargetGround TruthSimulator
type: Predictor class: Kalman Predictor
type: DetectionSimulator class: SimpleDetection Simulator
type: Hypothesiser class: Mahalanobis DistanceHypothesiser
type: Initiator class: SinglePoint Initiator
type: Updater class: Kalman Updater
type: DataAssociator class: Nearest Neighbour
type: Deleter class: Covariance BasedDeleter
type: Tracker class: MultiTargetTracker (Kalman)
tracks/groundtruth_paths/detections
type: MetricGenerator class: BasicMetrics
type: MetricGenerator class: OSPAMetric
type: MetricGenerator class: SiapMetrics
type: MetricManager class: SimpleManager
type: DataAssociator class: EuclideanTrack ToTruth
Basic metrics
OSPA metrics
Siap Metrics
Legend
Import
Data flow

Figure 7.11: Architecture of Stone Soup Use Case.

Metric	Use Case Value
Number of Tracks	63
Number of Targets	55
Track-to-Target ratio	1.145456
SIAP A	1.0
SIAP C	0.963925
SIAP LS	0.898119
SIAP LT	1527.50
SIAP S	0.118734

Table 7.2: Metrics for a particular run of the Use Case.

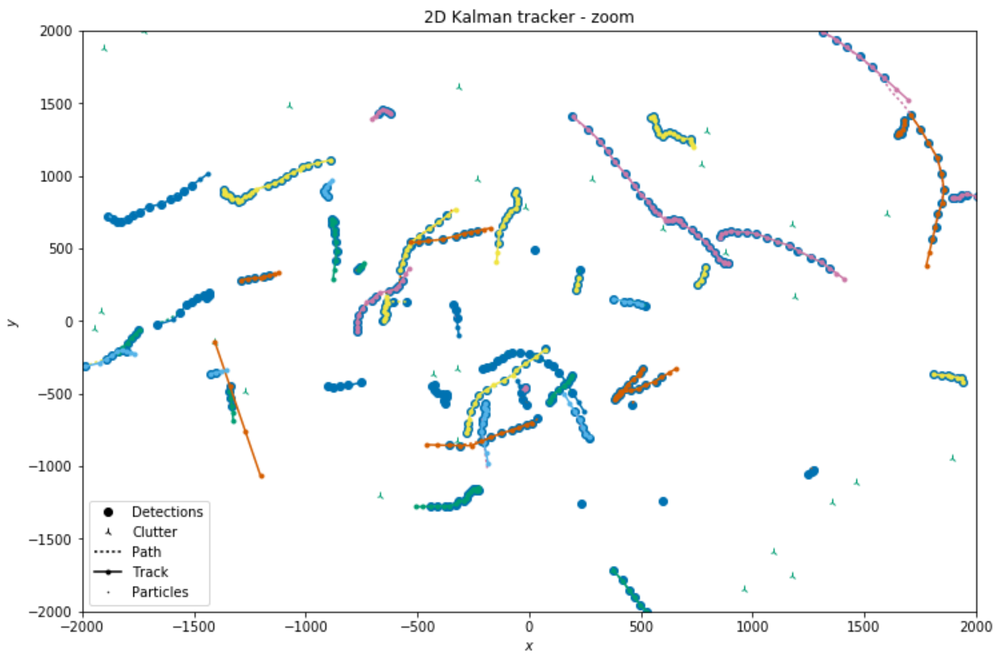


Figure 7.12: Graphical output of Kalman MultiTargetTracker.

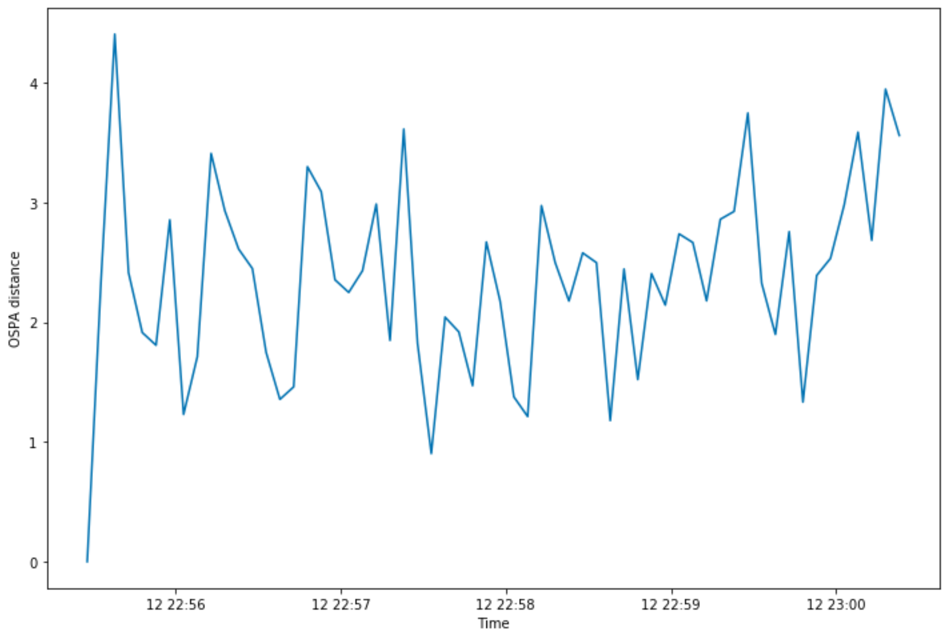


Figure 7.13: OSPA distance metric for a particular run of the Use Case over the time span of the experiment.

7.3.4 Future Work

Following the Stone Soup open-source release in April, 2019, it is anticipated that members of the tracking community will begin to contribute implementations of trackers and various components to the project. One focused mechanism for this will be the working group established under the International Society for Information Fusion (ISIF), called the Open Source Tracking and Estimation Working Group (OSTEWG) (https://isif-ostewg.org/), of which the author is an active member. This group aims to form a community focused on development and use of algorithms in Stone Soup for an annual use/data-case and to present the results at a Special Session at each ISIF Fusion Conference.

In addition to this, the current Stone Soup development team has plans for a few critical additions. Currently, the primary means of building and running Stone Soup experiments is through Jupyter Notebooks. It is expected that this will be a natural and comfortable interface for academic researchers; however, this interface will not meet the needs of all potential Stone Soup users. Members of the defence or security community who bring large sensor datasets to Stone Soup may not have a background in tracking or software development, so they will need a simpler way to build and run experiments.

A major planned addition to Stone Soup moving forward will be the User Interface (UI). This will be developed as a web browser-based Graphical UI that can be used to interact with Stone Soup. The UI will have three main components. The Experiment Builder will allow users to select Stone Soup components to build Trackers and experiments. The Builder will also provide a wizard to guide users with minimal experience in tracking through the process of building an experiment. The Run Manager section of the UI will interact with the Stone Soup RunManager to run experiments and collect the output Tracks and metrics. The Results Viewer will allow users to view and interact with the results of the experiments. It will show plots of the Detections/Tracks generated by the experiment, it will display metrics results, and it will contain a Playback Viewer that will allow the user to step through the experiment by timestep. This UI will be one of the major efforts of the Stone Soup development team moving forward.

Most of the tracking components currently implemented in Stone Soup deal with radar-type sensor detection data. Another area in the tracking space that has not yet been addressed is Wide-Angle Motion Imagery (WAMI). WAMI systems contain multiple high-pixel cameras that perform surveillance on a large area (tens or hundreds of square kilometres) and then stitch the images together in order to track and analyse the behaviour of a large number of individuals, vehicles, etc. WAMI systems provide much richer intelligence than single camera platforms with a narrower field of view. Moving forward, we plan to add support for WAMI tracking into Stone Soup.

The Stone Soup architecture is currently oriented towards tracking algorithms that represent the state of the system as a set of Tracks, where new Detections are associated to a single Track. However, some tracking algorithms do not maintain distinct Tracks but instead represent the state of the system as a Gaussian Mixture (e.g. the Gaussian Mixture Probabiliistic Hypothesis Density (GMPHD) filter). Thus, the Stone Soup architecture is currently under review, so as to better accommodate these types of algorithms, and it is expected that these changes will be merged into the Stone Soup code base in the near future. Once these modifications have been made, an implementation of the GMPHD filter will also be made available.

The tracking academic community has developed many variants of the main tracking algorithms. There are tweaks to algorithms to improve their accuracy in certain situations, and there are upgrades to improve the running time of the algorithms. As an example, one of the tracking algorithms currently implemented in Stone Soup is the Joint Probabilistic Data Association Filter (JPDAF). As part of its operation, JPDAF calculates the probabilities related to every permutation of associations between Detections and Tracks. In a Target-dense environment, an enumerative approach to this calculation will soon fall prey to combinatorial explosion. Different researchers have developed approaches to performing these calculations in less than exponential time, an example of which is the EHM algorithm discusse in Section 4.3.3.3. Part of the future work on Stone Soup will be to implement select variants of tracking algorithms to demonstrate these types of gains in processing speed.

7.4 Conclusion

This chapter presented work done by the author on developing and contributing towards the development of open-source frameworks for tracking and state estimation. Section 7.3 discussed Stone Soup [12, 13, 14], an open-source Python framework for tracking and state estimation, to which the author has been an major contributor. Section 7.2 introduced an open-source object-oriented MATLAB toolbox, developed by the author over the course of the PhD project and comprised of efficiently coded implementations of target tracking algorithms, with the aim of assisting and accelerating future research within the field.

CHAPTER 8

Summary, Conclusions and Future Work

The engineering challenge tackled by the PhD project reported in this thesis was to study and experiment with models that are well placed to capitalise on the abilities of Particle Filters and to develop solutions that make use of such models to deliver a direct operational advantage in real applications within the commercial maritime domain. This was accomplished by:

1. Presenting an in-depth background and review of existing algorithms and methodologies that pertain to the problem of single-target tracking. (Sections 3.2-3.3)
2. Experimenting with non-linear, non-Gaussian models and demonstrating that an operational advantage can be achieved with the use of such models in a Particle Filtering context. (Section 3.4)
3. Presenting an in-depth background to multi-target tracking methods and drawing relations between conventional approaches and the state-of-the-art. (Chapter 4)
4. Presenting new applications of state-of-the-art algorithms, with particular focus on applications relating to maritime radar and electro-optical sensors. Demonstrating the good performance of such algorithms for these applications using a mixture of real and simulated data. (Sections 4.6, 6.3 and 6.4)

5. Actively engaging with the target tracking community to develop solutions that improve the ability of future researchers to develop and evaluate the performance of new target tracking solutions. (Chapter 7)

8.1 Summary and Conclusions

Chapter 2 presented the set of dynamic, measurement, detection and clutter models that form the basis of the algorithms and applications considered in the thesis. This was achieved by formulating the fundamental framework of state-space model equations as considered in this thesis, as well as present readers to the general notation that was utilised.

Chapter 3 delved into the specifics of the Bayesian Filtering and Data Association problems in the context of Single Target Tracking, building on the framework of state-space models presented in Chapter 2. An introduction was provided to the fundamental concepts of the standard Bayesian Filtering framework, which were then utilised to formulate the Kalman and Particle Filtering algorithms, while discussing the various advantages and pitfalls of each approach. Furthermore, the Data Association problem was considered in the context of Single-Target Tracking. This discussion was followed by a brief discussion on the Nearest Neighbour and Probabilistic Data Association algorithms, while in the latter case a clear distinction was made between the application of the algorithm in Kalman and Particle Filtering context. Finally, performance evaluations were performed that demonstrated the performance benefits of combining Particle Filters with non-linear, non-Gaussian dynamic and measurement models, while in both cases an emphasis was made on the amplification of the achieved enhancement when the problem is extended to consider clutter and missed detections.

Chapter 4 extended the discussion from Chapter 3 to the case of Multi-Target Tracking. An introduction to the Multi-Target Bayes Filter was presented, while drawing relation to the Standard Bayes Filter. A structural overview of conventional Multi-Target trackers was presented, with special focus to the utilised Data Association and Track Management methods. Along the same discussion, the author presented drew a relation between the Joint Probabilistic Data Association (JPDA) and Joint Integrated PDA algorithms, highlighting how the latter can be performed using the same constructs. Furthermore, a formulation of the Probability Hypothesis Density (PHD) filter as an approximation to the Multi-target Bayes Filter was briefly discussed. Finally, the author demonstrated the application of a state-of-the-art radar track initiation technique which utilises a PHD filter to model the density of uninitiated targets and consecutively propose tracks for initiation on the basis of target existence probabilities. Preliminary simulation results were presented to showcase the performance benefits of the PHD track initiator compared to other mainstay approaches using synthetic data and a case study was performed on real data collected from a commercial radar, whereby a more thorough qualitative analysis was presented on a pair of challenging scenarios, with the aim of demonstrating a real operational advantage.

Chapter 5 demonstrated the applicability of the Expectation Maximisation to parameter estimation for LDSs with control inputs. A derivation of the complete set of equations required to perform the estimation process was presented, as well as a straightforward formulation of the relevant algorithmic steps. Also, an intuitive way of generating initial parameter estimates was introduced, given that certain assumption hold, and the performance implications of such initial estimates was discussed, given that no parameters are constrained throughout the learning process. The performance of the presented algorithm has been evaluated, based on a case study performed on a real segway system. The presented results show that the proposed technique achieves acceptable results, where the estimation performance can be greatly improved, if one (or more) of the parameters are known a priori. The content presented in this chapter was derived from and incorporated text from a paper that has been prepared for submission to the IEEE Signal Processing Letters, for which the author is the main author.

Chapter 6 summarised the work performed by the author in the domain of Maritime Video Detection & Tracking. Section 6.2 initiated the discussions by introducing the reader to the basic concepts and notation. Section 6.3 was then focused on the engineering challenges relating to development of accurate real-time ship detectors with the use of active Electro-Optical sensors (cameras). A review of existing approaches was presented, followed by an introduction to the state-of-the-art detectors that base their operation on Convolutional Neural Networks (CNNs). Under the same section, a robust CNN-based ship detector was introduced, that utilises a pre-trained CNN model that is subsequently trained on a custom dataset of images to provide an operational advantage. Continuing, Section 6.4 demonstrated an effective method for estimating and accounting for the errors induced by the camera motion in real-time. The use of an active (Pan-Tilt-Zoom) camera, introduces errors due to the inherent ability of the camera to exhibit motion. Thus, in addition to the multi-target tracking complexities discussed in Chapter 4, further measures must be employed in order to ensure that the positions of targets are estimated accurately, even after the camera has changed its orientation. The presented method formulates the problem of frame-to-frame variation as an affine transformation problem. First, a feature detector is applied to extract a set of “good” features, along with their respective locations, from the previous frame. Then, Optical Flow is employed to track and locate the detected feature locations in the new frame, following which the two sets of matching features are utilised in order to estimate the affine transformation between the two frames. Finally, the estimated affinity coefficients are used to correct the posterior state distributions of known tracks, such as to compensate for the errors introduced by the movement of the camera. The performance of the algorithm was compared against other well known feature matching techniques that utilise ORB [178], SIFT [179] and SURF [180] features respectively.

Finally, Chapter 7 presented work done by the author on developing and contributing towards the development of open-source frameworks for tracking and state estimation. Section 7.3 discussed Stone Soup [12, 13, 14], an open-source Python framework for tracking and state estimation, to which the author has been an major contributor. The content presented in relation to Stone Soup was derived from and incorporated text published in [14], licensed under the Open Government License [186], to which the author has been a major contributor and co-author. An overview of the various components and structures that make up the Stone Soup architecture was provided, followed by an example use-case scenario that demonstrated its functionality. Continuing, section 7.2 introduced an open-source object-oriented MATLAB toolbox, developed by the author over the course of the PhD project reported in this thesis, and comprised of efficiently coded implementations of target tracking algorithms, with the aim of assisting and accelerating future research within the field. An in-depth analysis of the TrackingX architecture was performed, with particular focus on the interface that enables the various components to be swiftly interchanged such as to allow for effortless prototyping and evaluation. Finally, an example use-case was presented to demonstrate the functionality of the developed framework.

8.2 Future work

- *Track-Before-Detect solutions and the (Poisson) Multi-Bernoulli Mixture Filter*: Following more experimentation with the track initiation technique of Section 4.6, it was found that in certain scenarios of dense clutter, tracks would be initiated prematurely. This problem can be attributed to the "short memory" effect of the PHD filter, as discussed in [190, 191]. An avenue of research that took place in the late stages of the PhD was to utilise state-of-the-art algorithms, such as the Poisson Multi-Bernoulli Mixture Filter [192], or the Trajectory PHD filter [193], with the aim of delaying the initiation of tracks that have a poor history of measurement associations. Although a substantial part of the work was performed, time limitations prevented this ambitious plan from fully materialising. Upon submission of the thesis, the author will remain with the university, as a post-doctoral researcher, and continue to work closely with the primary supervisor (Prof. Simon Maskell) and Denbridge Marine.

- *Automated Maritime Image Capturing and Surveillance System*: The work presented in Chapter 6 was inspired by an identified commercial requirement for Denbridge Marine, for the development of an automated system that can detect the periods at which a camera is unused and proceed to automate its control, with the aim of extracting high quality snapshots of vessels, from various viewing angles. The development of such a system will provide the means for constructing a database of images, which can be further utilised for future identity verification of vessels and even made available to the relevant maritime bodies and authorities. Once again, a substantial amount of work was performed on this cohort, however progress was halted during the last year of the PhD due to technical difficulties with the equipment for experimentation. Thus, although an experimental system has partly been developed that is able to track and focus on targets of interest, its performance has not reached the desired performance levels, so the author decided not to include it in the main body of the thesis. Nevertheless, as mentioned before, it is the author's intention to continue to pursue this goal, even after the submission of the thesis.

- *A Track-Before-Detect Method for Qualitative Detection of Small Targets using Radar and PTZ Cameras*: Having the ability to accurately detect and track vessels in images shall provide the benefit of adding an additional (and otherwise unused) sensor, to the overall tracking system, whose detection data can be fused with data coming from Radar and AIS, to provide an improvement in terms of both detection and tracking performance. One important application would be in the case of low SNR radar detections (e.g. small vessels). In such cases, where ambiguity exists as to whether a target is present in a certain region, the camera could be used for qualitative verification. Additionally, once developed, such a system can be utilised as a further means of generating positional ground truth data for vessels, in cases where AIS data are not available. This is another potential avenue for research and experimentation in the months to come.

- *Paper publications*: The author recognises that the amount of publications that have stemmed from the PhD project at the time of writing the thesis is not rich. As the thesis should suggest, this is not due to lack of development or innovations, but it is a direct consequence of the broadness of topics investigated by the author, which necessitated for a longer time to be devoted on gaining a firm understanding of the broader concepts that relate to each field. Nevertheless, there are plans for publications once the thesis is submitted: a paper shall be prepared and published that outlines the MATLAB toolbox (TrackingX) developed by the author; both of the afore-mentioned avenues for future work (previous two bullet points) are not far from materialisation and it is both the author's, as well as the supervisor's (Prof. Simon Maskell), intention to continue to pursue these, with the ultimate aim of generating publications.

References

[1] S. Blackman, R. Popoli, *Design and Analysis of Modern Tracking Systems.* Norwood, MA 02062: Artech House, 1999.

[2] J. N. Briggs, *Target Detection by Marine Radar.* Radar, Sonar & Navigation, Institution of Engineering and Technology, 2004.

[3] L. P. Perera, P. Oliveira, and C. Guedes Soares, "Maritime traffic monitoring based on vessel detection, tracking, state estimation, and trajectory prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 3, pp. 1188–1200, 2012.

[4] P. Braca, R. Grasso, M. Vespe, S. Maresca, and J. Horstmann, "Application of the jpda-ukf to hfsw radars for maritime situational awareness," in *2012 15th International Conference on Information Fusion*, pp. 2585–2592, 2012.

[5] K. Laws, J. Vesecky, and J. Paduan, "Monitoring coastal vessels for environmental applications: Application of kalman filtering," in *2011 IEEE/OES 10th Current, Waves and Turbulence Measurements (CWTM)*, pp. 39–46, 2011.

[6] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking.," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174 – 188, n.d.

[7] A. Doucet and A. Johansen, "A tutorial on particle filtering and smoothing: Fifteen years later," *Handbook of Nonlinear Filtering*, vol. 12, 01 2009.

[8] M. G. Rutten, N. J. Gordon, and S. Maskell, "Recursive track-before-detect with target amplitude fluctuations," *IEE Proceedings - Radar, Sonar and Navigation*, vol. 152, no. 5, pp. 345–352, 2005.

[9] Y. Boers and J. N. Driessen, "Multitarget particle filter track before detect application," *IEE Proceedings - Radar, Sonar and Navigation*, vol. 151, no. 6, pp. 351–357, 2004.

[10] F. Heymann, J. Hoth, P. Banys, and G. Siegert, "Validation of radar image tracking algorithms with simulated data," *TransNav, the International Journal on Marine Navigation and Safety of Sea Transportation*, vol. 11, pp. 511–518, 09 2017.

[11] P. A. Kountouriotis and S. Maskell, "Maneuvering target tracking using an unbiased nearly constant heading model.," *2012 15th International Conference on Information Fusion*, p. 2249, 2012.

[12] P. Thomas, J. Barr, B. Balaji, and K. White, "An open source framework for tracking and state estimation ('Stone Soup')," *Signal Processing, Sensor/Information Fusion, and Target Recognition XXVI*, vol. 10200, 2017.

[13] P. Thomas, J. Barr, S. Hiscocks, C. England, S. Maskell, B. Balaji, and J. Williams, "Stone Soup: An Open-Source Framework for Tracking and State Estimation," *International Society for Information Fusion (ISIF) Perspectives*, vol. 1, no. 2, pp. 1–38, 2019.

[14] D. Last, S. Hiscocks, J. Barr, D. Kirkland, M. Rashid, S. B. Li, L. Vladimirov, and P. Thomas, "Stone soup: announcement of beta release of an open-source framework for tracking and state estimation," April 2019.

[15] S. Särkkä, *Bayesian Filtering and Smoothing.* Cambridge University Press, 2013.

[16] L. Stone, R. Streit, T. Corwin, and K. Bell, *Bayesian Multiple Target Tracking, Second Edition.* Radar/Remote Sensing, Artech House, 2013.

[17] A. Markov, "Extension of the Limit Theorems of Probability Theory to a Sum of Variables Connected in a Chain," *The Notes of the Imperial Academy of Sciences of St. Petersburg*, Dec. 1907.

[18] C. Graham, *Markov Chains: Analytic and Monte-Carlo Computations.* John Wiley & Sons Ltd., 2014.

[19] S. Maskell, *Sequentially Structured Bayesian Solutions.* University of Cambridge, 2004.

[20] D. Lerro and Y. Bar-Shalom, “Tracking with debiased consistent converted measurements versus ekf,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 29, pp. 1015–1022, July 1993.

[21] Mo Longbin, Song Xiaoquan, Zhou Yiyu, Sun Zhong Kang, and Y. Bar-Shalom, “Unbiased converted measurements for tracking,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 34, pp. 1023–1027, July 1998.

[22] P. Horridge, “Tracking with inter-visibility variables,” *IET Conference Proceedings*, pp. 59–68(9), January 2008.

[23] P. Horridge and S. Maskell, “Using a probabilistic hypothesis density filter to confirm tracks in a multi-target environment,” in *Proc. 6th Workshop on Sensor Data Fusion, Berlin, Germany*, Citeseer, 2011.

[24] Y Bar-Shalom, X. Rong Li, T. Kirubarajan, *Estimation with Applications to Tracking and Navigation: Theory Algorithms and Software.* Hoboken, New Jersey: John Wiley & Sons Ltd., 2004.

[25] S. Maskell, “Statistical Methods for Target Tracking,” *Wiley Encyclopedia of Computer Science and Engineering*, Mar. 2009.

[26] Y. Bar-Shalom and E. Tse, “Tracking in a cluttered environment with probabilistic data association,” *Automatica*, vol. 11, no. 5, pp. 451 – 460, 1975.

[27] Y. Bar-Shalom and T. E. Fortmann, *Tracking and data association. [electronic resource].* Mathematics in science and engineering: v. 179, Boston : Academic Press, c1988., 1988.

[28] A. Marrs, S. Maskell, and Y. Bar-shalom, “Expected likelihood for tracking in clutter with particle filters,” in *in Proceedings of SPIE Signal and Data Processing of Small Targets*, pp. 230–239, 2002.

[29] A. J. Haug, *Bayesian Estimation and Tracking: A Practical Guide.* John Wiley & Sons Ltd., 2012.

[30] J. F. C. Kingman, “On the chapman-kolmogorov equation,” *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 276, no. 1259, 1974.

[31] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME Journal of Basic Engineering*, vol. 82, 1960.

[32] R. E. Kalman, R. S. Bucy, "New Results in Linear Filtering and Prediction Theory," *Transactions of the ASME Journal of Basic Engineering*, vol. 83, 1961.

[33] G. L. Smith, S. F. Schmidt, L. A. McGee, "Application of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle," National Aeronautics and Space Administration, Washington, D.C., 1962.

[34] J. B. Pearson and E. B. Stear, "Kalman filter applications in airborne radar tracking," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-10, pp. 319–329, May 1974.

[35] B. Uhrmeister, "Kalman filters for a missile with radar and/or imaging sensor.," *Journal of Guidance, Control, and Dynamics*, vol. 17, no. 6, pp. 1339–1344, 1994.

[36] T., Mitchell, "1st-order Taylor-Series Approximations and Cost-Functions," *Scandinavian Journal Of Economics*, vol. 92, no. 3, pp. 513 – 524, 1990.

[37] A. H. Jazwinski, *Stochastic processes and filtering theory [by] Andrew H. Jazwinski.* Academic Press New York, 1970.

[38] F. Gustafsson and G. Hendeby, "Some relations between extended and unscented kalman filters.," *IEEE Transactions on Signal Processing*, p. 545, 2012.

[39] G. Klanar, L. Tesli, and I. krjanc, "Mobile-robot pose estimation and environment mapping using an extended kalman filter," *International Journal of Systems Science*, vol. 45, no. 12, pp. 2603–2618, 2014.

[40] J. R. Hervas, M. Reyhanoglut, and T. Hui, "Nonlinear automatic landing control of unmanned aerial vehicles on moving platforms via a 3d laser radar.," *AIP Conference Proceedings*, vol. 1637, no. 1, pp. 907 – 917, 2014.

[41] C. G. Prevost, A. Desbiens, and E. Gagnon, "Extended kalman filter for state estimation and trajectory prediction of a moving object detected by an unmanned aerial vehicle," in *2007 American Control Conference*, pp. 1805–1810, July 2007.

[42] S. J. Julier, J. K. Uhlmann, and H. F. Durrant-Whyte, “A new approach for filtering nonlinear systems,” in *American Control Conference, Proceedings of the 1995*, vol. 3, pp. 1628–1632 vol.3, Jun 1995.

[43] S. J. Julier and J. K. Uhlmann, “A new extension of the kalman filter to nonlinear systems,” pp. 182–193, 1997.

[44] S. Julier, J. Uhlmann, and H. F. Durrant-Whyte, “A new method for the nonlinear transformation of means and covariances in filters and estimators,” *IEEE Transactions on Automatic Control*, vol. 45, no. 3, pp. 477–482, 2000.

[45] T. Lefebvre, H. Bruyninckx, and J. De Schuller, “Comment on ”a new method for the nonlinear transformation of means and covariances in filters and estimators” [with authors’ reply],” *IEEE Transactions on Automatic Control*, vol. 47, pp. 1406–1409, Aug 2002.

[46] S. J. Julier and J. K. Uhlmann, “Unscented filtering and nonlinear estimation.,” *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401 – 422, 2004.

[47] E. A. Wan and R. V. D. Merwe, “The unscented kalman filter for nonlinear estimation,” in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pp. 153–158, 2000.

[48] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, “Novel approach to nonlinear/non-gaussian bayesian state estimation,” *IEE Proceedings F - Radar and Signal Processing*, vol. 140, pp. 107–113, April 1993.

[49] A. Doucet, S. Godsill, and C. Andrieu, “On sequential monte carlo sampling methods for bayesian filtering.,” *STATISTICS AND COMPUTING*, vol. 10, no. 3, pp. 197 – 208, n.d.

[50] T. Li, M. Bolic, and P. M. Djuric, “Resampling methods for particle filtering: Classification, implementation, and strategies,” *IEEE Signal Processing Magazine*, vol. 32, pp. 70–86, May 2015.

[51] J. D. Hol, T. B. Schon, and F. Gustafsson, “On resampling algorithms for particle filters,” in *2006 IEEE Nonlinear Statistical Signal Processing Workshop*, pp. 79–82, Sept 2006.

[52] J. V. Candy, “Bootstrap particle filtering,” *IEEE Signal Processing Magazine*, vol. 24, no. 4, pp. 73–85, 2007.

[53] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P. . Nordlund, "Particle filters for positioning, navigation, and tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 425–437, 2002.

[54] S. Godsill, "Particle filtering: the first 25 years and beyond," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 7760–7764, 2019.

[55] H. Griffiths, "Sea clutter: Scattering, the k distribution and radar performance (ward, k.d., et al.; 2006) [book review]," *IEEE Aerospace and Electronic Systems Magazine*, vol. 22, pp. 28–28, Jan 2007.

[56] X. Rong Li and Y. Bar-Shalom, "Tracking in clutter with nearest neighbor filters: analysis and performance," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 32, pp. 995–1010, July 1996.

[57] I. J. Cox, "A review of statistical data association techniques for motion correspondence," *International Journal of Computer Vision*, vol. 10, no. 1, pp. 53–66, 1993.

[58] Y. Bar-Shalom, F. Daum, and J. Huang, "The probabilistic data association filter," *IEEE Control Systems*, vol. 29, pp. 82–100, Dec 2009.

[59] D. J. Salmond, "Mixture reduction algorithms for target tracking in clutter," vol. 1305, 1990.

[60] D. F. Crouse, P. Willett, K. Pattipati, and L. Svensson, "A look at gaussian mixture reduction algorithms," in *14th International Conference on Information Fusion*, pp. 1–8, July 2011.

[61] J. Vermaak, S. J. Godsill, and P. Perez, "Monte carlo filtering for multi target tracking and data association," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 41, pp. 309–332, Jan 2005.

[62] A. F. Tchango, V. Thomas, O. Buffet, A. Dutech, and F. Flacher, "Tracking multiple interacting targets using a joint probabilistic data association filter," in *17th International Conference on Information Fusion (FUSION)*, pp. 1–8, July 2014.

[63] T. Fortmann, Y. bar shalom, and M. Scheffe, "Sonar tracking of multiple targets using joint probabilistic data association. ieee journal of oceanic engineering, oe-8, 173-184," *Oceanic Engineering, IEEE Journal of*, vol. 8, pp. 173 – 184, 08 1983.

[64] T. E. Fortmann, Y. Bar-Shalom, and M. Scheffe, “Multi-target tracking using joint probabilistic data association,” in *1980 19th IEEE Conference on Decision and Control including the Symposium on Adaptive Processes*, pp. 807–812, Dec 1980.

[65] M. Schuster, J. Reuter, and G. Wanielik, “Probabilistic data association for tracking extended targets under clutter using random matrices,” in *2015 18th International Conference on Information Fusion (Fusion)*, pp. 961–968, July 2015.

[66] F. Folster and H. Rohling, “Data association and tracking for automotive radar networks,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 6, pp. 370–377, Dec 2005.

[67] B. N. Vo and W. K. Ma, “A closed-form solution for the probability hypothesis density filter,” in *2005 7th International Conference on Information Fusion*, vol. 2, pp. 8 pp.–, July 2005.

[68] B. ngu Vo and S. Singh, “Sequential monte carlo implementation of the phd filter for multi-target tracking,” in *In Proceedings of the Sixth International Conference on Information Fusion*, pp. 792–799, 2003.

[69] R. P. S. Mahler, “Multitarget bayes filtering via first-order multitarget moments,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 39, pp. 1152–1178, Oct 2003.

[70] R. Mahler, “A Theoretical Foundation for the Stein-Winter Probability Hypothesis Density (PhD) Multi-Target Tracking Approach,” in *Proceedings of the 2000 MSS National Symposium on Sensor and Data Fusion*, 2002.

[71] R. P. S. Mahler, *Statistical Multisource-Multitarget Information Fusion.* Norwood, MA, USA: Artech House, Inc., 2007.

[72] B. Ristic and B.-N. Vo, “Sensor control for multi-object state-space estimation using random finite sets,” *Automatica*, vol. 46, no. 11, pp. 1812–1818, 2010.

[73] M. Vihola, “Rao-blackwellised particle filtering in random set multitarget tracking,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 43, pp. 689–705, April 2007.

[74] V. D. Nguyen and T. Claussen, “Reducing computational complexity of gating procedures using sorting algorithms,” in *Proceedings of the 16th International Conference on Information Fusion*, pp. 1707–1713, July 2013.

[75] T. Cheng, X. R. Li, and Z. He, "Comparison of gating techniques for maneuvering target tracking in clutter," in *17th International Conference on Information Fusion (FUSION)*, pp. 1–8, July 2014.

[76] X. Wang, S. Challa, and R. Evans, "Gating techniques for maneuvering target tracking in clutter," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 38, pp. 1087–1097, Jul 2002.

[77] H. Roufarshbaf and J. K. Nelson, "A bayesian tree-search track initiation algorithm for dim targets," in *2013 47th Annual Conference on Information Sciences and Systems (CISS)*, pp. 1–5, March 2013.

[78] D. Svensson, F. Govaers, M. Ulmke, and W. Koch, "Target existence probability in the distributed kalman filter," in *2013 Workshop on Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, pp. 1–5, Oct 2013.

[79] D. Musicki, "Track score and target existence," in *2006 9th International Conference on Information Fusion*, pp. 1–7, July 2006.

[80] P. Horridge and S. Maskell, "Searching for, initiating and tracking multiple targets using existence probabilities," in *2009 12th International Conference on Information Fusion*, pp. 611–617, July 2009.

[81] J. Vermaak, S. Maskell, and M. Briers, "A unifying framework for multi-target tracking and existence," in *2005 7th International Conference on Information Fusion*, vol. 1, pp. 9 pp.–, July 2005.

[82] P. Horridge and S. Maskell, "Using a probabilistic hypothesis density filter to confirm tracks in a multi-target environment," in *2011 Jahrestagung der Gesellschaft fr Informatik*, October 2011.

[83] J. L. Williams, "Marginal multi-bernoulli filters: Rfs derivation of mht, jipda, and association-based member," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 51, pp. 1664–1687, July 2015.

[84] H. W. Kuhn, "The hungarian method for the assignment problem," in *50 Years of Integer Programming*, 2010.

[85] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[86] D. P. Bertsekas and D. A. Castanon, "The auction algorithm for the transportation problem," *Annals of Operations Research*, vol. 20, no. 1, pp. 67–96, 1989.

[87] D. B. Malkoff, “Evaluation of the jonker-volgenant-castanon (jvc) assignment algorithm for track association,” *Proc. SPIE*, vol. 3068, pp. 228–239, 1997.

[88] R. J. Fitzgerald, “Development of practical pda logic for multitarget tracking by microprocessor,” in *American Control Conference, 1986*, pp. 889–898, June 1986.

[89] Z. Hu, H. Leung, and M. Blanchette, “Evaluation of data association techniques in a real multitarget radar tracking environment,” *Proc. SPIE*, vol. 2561, pp. 509–518, 1995.

[90] S. Maskell, M. Briers, and R. Wright, “Fast mutual exclusion,” *Proc. SPIE*, vol. 5428, pp. 526–536, 2004.

[91] P. R. Horridge and S. Maskell, “Real-time tracking of hundreds of targets with efficient exact jpdaf implementation.,” in *FUSION*, pp. 1–8, IEEE, 2006.

[92] D. Musicki and R. Evans, “Integrated probabilistic data association in clutter with finite resolution sensor,” in *Proceedings of 32nd IEEE Conference on Decision and Control*, pp. 912–917 vol.1, Dec 1993.

[93] D. Musicki, R. Evans, and S. Stankovic, “Integrated probabilistic data association,” *IEEE Transactions on Automatic Control*, vol. 39, pp. 1237–1241, June 1994.

[94] D. Musicki and R. Evans, “Joint integrated probabilistic data association: Jipda,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 40, pp. 1093–1099, July 2004.

[95] D. Musicki and R. Evans, “Clutter map information for data association and track initialization,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 40, pp. 387–398, April 2004.

[96] R. P. S. Mahler, “Random-set approach to data fusion,” *Proc. SPIE*, vol. 2234, pp. 287–295, 1994.

[97] L. Lin, Y. Bar-Shalom, and T. Kirubarajan, “Track labeling and phd filter for multitarget tracking,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 42, pp. 778–795, July 2006.

[98] B. N. Vo and W. K. Ma, “The gaussian mixture probability hypothesis density filter,” *IEEE Transactions on Signal Processing*, vol. 54, pp. 4091–4104, Nov 2006.

[99] B. N. Vo, S. Singh, and A. Doucet, "Sequential monte carlo methods for multitarget filtering with random finite sets," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 41, pp. 1224–1245, Oct 2005.

[100] T. Zajic, R. B. Ravichandran, R. P. S. Mahler, R. K. Mehra, and M. J. Noviskey, "Joint tracking and identification with robustness against unmodeled targets," *Proc. SPIE*, vol. 5096, pp. 279–290, 2003.

[101] M. Tobias and A. D. Lanterman, "Probability hypothesis density-based multitarget tracking with bistatic range and doppler observations," *IEE Proceedings - Radar, Sonar and Navigation*, vol. 152, pp. 195–205, June 2005.

[102] D. Clark, B.-N. Vo, and J. Bell, "Gm-phd filter multitarget tracking in sonar images," *Proc. SPIE*, vol. 6235, pp. 62350R–62350R–8, 2006.

[103] D. E. Clark and J. Bell, "Bayesian multiple target tracking in forward scan sonar images using the phd filter," *IEE Proceedings - Radar, Sonar and Navigation*, vol. 152, pp. 327–334, October 2005.

[104] N. T. Pham, W. Huang, and S. H. Ong, "Tracking multiple objects using probability hypothesis density filter and color measurements," in *2007 IEEE International Conference on Multimedia and Expo*, pp. 1511–1514, July 2007.

[105] E. Maggio, E. Piccardo, C. Regazzoni, and A. Cavallaro, "Particle phd filtering for multi-target visual tracking," in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, vol. 1, pp. I–1101–I–1104, April 2007.

[106] D. Schuhmacher, B. T. Vo, and B. N. Vo, "A consistent metric for performance evaluation of multi-object filters," *IEEE Transactions on Signal Processing*, vol. 56, pp. 3447–3457, Aug 2008.

[107] S. Roweis and Z. Ghahramani, "A unifying review of linear gaussian models," *Neural Computation*, vol. 11, no. 2, pp. 305–345, 1999.

[108] Z. Ghahramani and G. E. Hinton, "Parameter Estimation for Linear Dynamical Systems," tech. rep., University of Toronto, 1996.

[109] V. Digalakis, J. R. Rohlicek, and M. Ostendorf, "ML estimation of a stochastic linear system with the EM algorithm and its application to speech recognition," *IEEE Transactions on Speech and Audio Processing*, vol. 1, pp. 431–442, Oct 1993.

[110] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.

[111] E. Holmes, "An EM algorithm for maximum likelihood estimation given corrupted observations," tech. rep., University of Washington & National Marine Fisheries Service, 2006.

[112] S. Gibson and B. Ninness, "Robust maximum-likelihood estimation of multivariable dynamic systems," *Automatica*, vol. 41, no. 10, pp. 1667 – 1682, 2005.

[113] A. W. Blocker, "An EM algorithm for the estimation of a ne state-space systems with or without known inputs," 2008.

[114] T. B. Schön, "An explanation of the expectation maximization algorithm, report no. lith-isy-r-2915," tech. rep., 2009.

[115] H. E. Rauch, C. T. Striebel, and F. Tung, "Maximum Likelihood Estimates of Linear Dynamic Systems," *Journal of the American Institute of Aeronautics and Astronautics*, vol. 3, pp. 1445–1450, Aug. 1965.

[116] C. F. J. Wu, "On the convergence properties of the em algorithm," *The Annals of Statistics*, vol. 11, no. 1, pp. 95–103, 1983.

[117] D. BLOISI and L. IOCCHI, "Argos a video surveillance system for boat traffic monitoring in venice," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 23, no. 07, pp. 1477–1502, 2009.

[118] S. Fefilatyev, D. Goldgof, M. Shreve, and C. Lembke, "Detection and tracking of ships in open sea with rapidly moving buoy-mounted camera system," *Ocean Engineering*, vol. 54, pp. 1 – 12, 2012.

[119] W. Krger and Z. Orlov, "Robust layer-based boat detection and multi-target-tracking in maritime environments," in *2010 International WaterSide Security Conference*, pp. 1–7, Nov 2010.

[120] F. Fusier, V. Valentin, F. Brémond, M. Thonnat, M. Borg, D. Thirde, and J. Ferryman, "Video understanding for complex activity recognition," *Machine Vision and Applications*, vol. 18, no. 3, pp. 167–188, 2007.

[121] K. M. Gupta, D. W. Aha, R. Hartley, and P. G. Moore, "Adaptive maritime video surveillance," *Proc. SPIE*, vol. 7346, pp. 734609–734609–12, 2009.

[122] A. Brkle and B. Essendorfer, "Maritime surveillance with integrated systems," in *2010 International WaterSide Security Conference*, pp. 1–8, Nov 2010.

[123] A. Samama, "Innovative video analytics for maritime surveillance," in *2010 International WaterSide Security Conference*, pp. 1–8, Nov 2010.

[124] H. Wei, H. Nguyen, P. Ramu, C. Raju, X. Liu, and J. Yadegar, "Automated intelligent video surveillance system for ships," *Proc. SPIE*, vol. 7306, pp. 73061N–73061N–12, 2009.

[125] B. J. Rhodes, N. A. Bomberger, M. Seibert, and A. M. Waxman, "Seecoast: Automated port scene understanding facilitated by normalcy learning," in *MILCOM 2006 - 2006 IEEE Military Communications conference*, pp. 1–7, Oct 2006.

[126] S. Fefilatyev and D. Goldgof, "Detection and tracking of marine vehicles in video," in *ICPR 2008 19th International Conference on Pattern Recognition(ICPR)*, vol. 00, pp. 1–4, Dec. 2009.

[127] S. Fefilatyev, D. B. Goldgof, and C. Lembke, "Tracking ships from fast moving camera through image registration," *2010 20th International Conference on Pattern Recognition*, pp. 3500–3503, 2010.

[128] S. Fefilatyev, D. Goldgof, M. Shreve, and C. Lembke, "Detection and tracking of ships in open sea with rapidly moving buoy-mounted camera system," *Ocean Engineering*, vol. 54, pp. 1 – 12, 2012.

[129] R. Wijnhoven, K. van Rens, E. G. T. Jaspers, and P. H. N. de With, "Online Learning for Ship Detection in Maritime Surveillance," in *Thirty-first Symposium on Information Theory in the Benelux*, pp. 73–80, May 2010.

[130] M. J. H. Loomans, P. H. N. de With, and R. G. J. Wijnhoven, "Robust automatic ship tracking in harbours using active cameras," *2013 IEEE International Conference on Image Processing*, pp. 4117–4121, 2013.

[131] D. Bloisi, L. Iocchi, M. Fiorini, and G. Graziano, "Camera based target recognition for maritime awareness," in *2012 15th International Conference on Information Fusion*, pp. 1982–1987, July 2012.

[132] D. Bloisi, L. Iocchi, M. Fiorini, and G. Graziano, "Automatic maritime surveillance with visual target detection," in *Proceedings of the International Defense and Homeland Security Simulation Workshop (DHSS)*, (Rome, Italy), pp. 141–145, 2011.

[133] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, pp. I–511–I–518 vol.1, 2001.

[134] P. Viola and M. J. Jones, “Robust real-time face detection,” *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[135] S. Zhang, C. Bauckhage, and A. B. Cremers, “Informed haar-like features improve pedestrian detection,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[136] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS’12, (USA), pp. 1097–1105, Curran Associates Inc., 2012.

[137] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, June 2009.

[138] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (voc) challenge,” *International Journal of Computer Vision*, vol. 88, pp. 303–338, Jun 2010.

[139] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580–587, June 2014.

[140] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, “Overfeat: Integrated recognition, localization and detection using convolutional networks,” *CoRR*, vol. abs/1312.6229, 2013.

[141] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016.

[142] J. Redmon and A. Farhadi, “Yolo9000: Better, faster, stronger,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6517–6525, 2017.

[143] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” *Lecture Notes in Computer Science*, p. 2137, 2016.

[144] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollr, and C. L. Zitnick, "Microsoft coco: Common objects in context," *Lecture Notes in Computer Science*, p. 740755, 2014.

[145] R. Girshick, "Fast r-cnn," *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec 2015.

[146] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, p. 11371149, Jun 2017.

[147] Z. Zhao, P. Zheng, S. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[148] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and et al., "Speed/accuracy trade-offs for modern convolutional object detectors," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017.

[149] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[150] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[151] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[152] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[153] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, June 2016.

[154] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," *CoRR*, vol. abs/1411.1792, 2014.

[155] I. Krasin, T. Duerig, N. Alldrin, V. Ferrari, S. Abu-El-Haija, A. Kuznetsova, H. Rom, J. Uijlings, S. Popov, S. Kamali, M. Malloci, J. Pont-Tuset, A. Veit, S. Belongie, V. Gomes, A. Gupta, C. Sun, G. Chechik, D. Cai, Z. Feng, D. Narayanan, and K. Murphy, "Openimages: A public dataset for large-scale multi-label and multi-class image classification.," *Dataset available from https://storage.googleapis.com/openimages/web/index.html*, 2017.

[156] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[157] TensorFlow, "Tensorflow detection model zoo." `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md`, Jul 2019. Accessed: 2019-08-27.

[158] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, (USA), pp. 379–387, Curran Associates Inc., 2016.

[159] E. Zhang and Y. Zhang, *Average Precision*, pp. 192–193. Boston, MA: Springer US, 2009.

[160] L. LIU and M. T. ÖZSU, eds., *Mean Average Precision*, pp. 1703–1703. Boston, MA: Springer US, 2009.

[161] Z. Kim, "Real time object tracking based on dynamic feature grouping with background subtraction," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, IEEE, 2008.

[162] A. A. Argyros, K. E. Bekris, and S. C. Orphanoudakis, "Robot homing based on corner tracking in a sequence of panoramic images," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 2, pp. II–II, IEEE, 2001.

[163] R. K. C. Billones, A. A. Bandala, E. Sybingco, L. A. G. Lim, A. D. Fillone, and E. P. Dadios, "Vehicle detection and tracking using corner feature

points and artificial neural networks for a vision-based contactless apprehension system," in *2017 Computing Conference*, pp. 688–691, July 2017.

[164] C. Harris and M. Stephens, "A combined corner and edge detector," in *Procedings of the Alvey Vision Conference 1988*, Alvey Vision Club, 1988.

[165] Jianbo Shi and Tomasi, "Good features to track," in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 593–600, June 1994.

[166] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, no. 1, pp. 185 – 203, 1981.

[167] J. Heikkonen, "Recovering 3-d motion parameters from optical flow field using randomized hough transform," *Pattern Recognition Letters*, vol. 16, no. 9, pp. 971 – 978, 1995.

[168] H. C. Longuet-Higgins and K. Prazdny, "The interpretation of a moving retinal image.," *Proceedings of the Royal Society of London. Series B, Biological sciences*, vol. 208 1173, pp. 385–97, 1980.

[169] Y. Mae, Y. Shirai, J. Miura, and Y. Kuno, "Object tracking in cluttered background based on optical flow and edges," in *Proceedings of 13th International Conference on Pattern Recognition*, vol. 1, pp. 196–200 vol.1, Aug 1996.

[170] M. Lucena, J. M. Fuertes, and N. P. de la Blanca, "Using optical flow for tracking," in *Progress in Pattern Recognition, Speech and Image Analysis* (A. Sanfeliu and J. Ruiz-Shulcloper, eds.), (Berlin, Heidelberg), pp. 87–94, Springer Berlin Heidelberg, 2003.

[171] A. K. Chauhan and P. Krishan, "Moving object tracking using gaussian mixture model and optical flow," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 4, 2013.

[172] D. D. Doyle, A. L. Jennings, and J. T. Black, "Optical flow background estimation for real-time pan/tilt camera object tracking," *Measurement*, vol. 48, pp. 195 – 207, 2014.

[173] R. S. Rakibe and B. D. Patil, "Background subtraction algorithm based human motion detection," *International Journal of scientific and research publications*, vol. 3, no. 5, pp. 2250–3153, 2013.

[174] S. Liu, L. Yuan, P. Tan, and J. Sun, “Steadyflow: Spatially smooth optical flow for video stabilization,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4209–4216, June 2014.

[175] A. Lim, B. Ramesh, Y. Yang, C. Xiang, Z. Gao, and F. Lin, “Real-time optical flow-based video stabilization for unmanned aerial vehicles,” *Journal of Real-Time Image Processing*, Jun 2017.

[176] B. D. Lucas and T. Kanade, “An iterative image registration technique with an application to stereo vision,” in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI’81, (San Francisco, CA, USA), pp. 674–679, Morgan Kaufmann Publishers Inc., 1981.

[177] M. A. Fischler and R. C. Bolles, “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,” *Commun. ACM*, vol. 24, pp. 381–395, June 1981.

[178] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “Orb: an efficient alternative to sift or surf,” pp. 2564–2571, 11 2011.

[179] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, vol. 60, pp. 91–110, Nov 2004.

[180] H. Bay, T. Tuytelaars, and L. Van Gool, “Surf: Speeded up robust features,” vol. 3951, pp. 404–417, 07 2006.

[181] D. F. Crouse, “The tracker component library: free routines for rapid prototyping,” *IEEE Aerospace and Electronic Systems Magazine*, vol. 32, pp. 18–27, May 2017.

[182] MATLAB, “Sensor fusion and tracking toolbox.” `https://uk.mathworks.com/help/pdf_doc/fusion/fusion_ug.pdf`. Accessed: 2019-08-31.

[183] B. Risti, B.-N. Vo, B.-N. Vo, and A. Farina, “A tutorial on bernoulli filters: Theory, implementation and applications,” *IEEE Transactions on Signal Processing*, vol. 61, 07 2013.

[184] A. S. Rahmathullah, . F. Garca-Fernndez, and L. Svensson, “Generalized optimal sub-pattern assignment metric,” in *2017 20th International Conference on Information Fusion (Fusion)*, pp. 1–8, July 2017.

[185] Á. F. García-Fernández and L. Svensson, “Spooky effect in optimal ospa estimation and how gospa solves it,” *arXiv preprint arXiv:1908.08815*, 2019.

[186] “Open government licence.”

[187] H. D. S. J. Karoly, J. W. Wilson and J. W. Maluda, “Single Integrated Air Picture (SIAP) Attributes Version 2.0,” Tech. Rep. Technical report ref. SIAP SE TF-TR-2003-029, Joint Single Integrated Air Picture (SIAP) System Engineering Task Force – Arlington, VA 22203, August 2003.

[188] Project Jupyter, “Jupyter.” `https://jupyter.org/`. Accessed: 2019-03-14.

[189] Dstl, “Stone soup jupyter notebooks repository.” `https://github.com/dstl/Stone-Soup-Notebooks`. Accessed: 2019-09-02.

[190] G. Davidson, *Random finite sets for multitarget tracking with applications.* PhD thesis, University of Oxford, 2011.

[191] O. Erdinc, P. Willett, and Y. Bar-Shalom, “Probability hypothesis density filter for multitarget multisensor tracking,” in *2005 7th International Conference on Information Fusion*, vol. 1, pp. 8 pp.–, July 2005.

[192] Á. F. García-Fernández, J. L. Williams, K. Granstrm, and L. Svensson, “Poisson multi-bernoulli mixture filter: Direct derivation and implementation,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 54, pp. 1883–1901, Aug 2018.

[193] Á. F. García-Fernández and L. Svensson, “Trajectory probability hypothesis density filter,” *2018 21st International Conference on Information Fusion (FUSION)*, pp. 1430–1437, 2018.