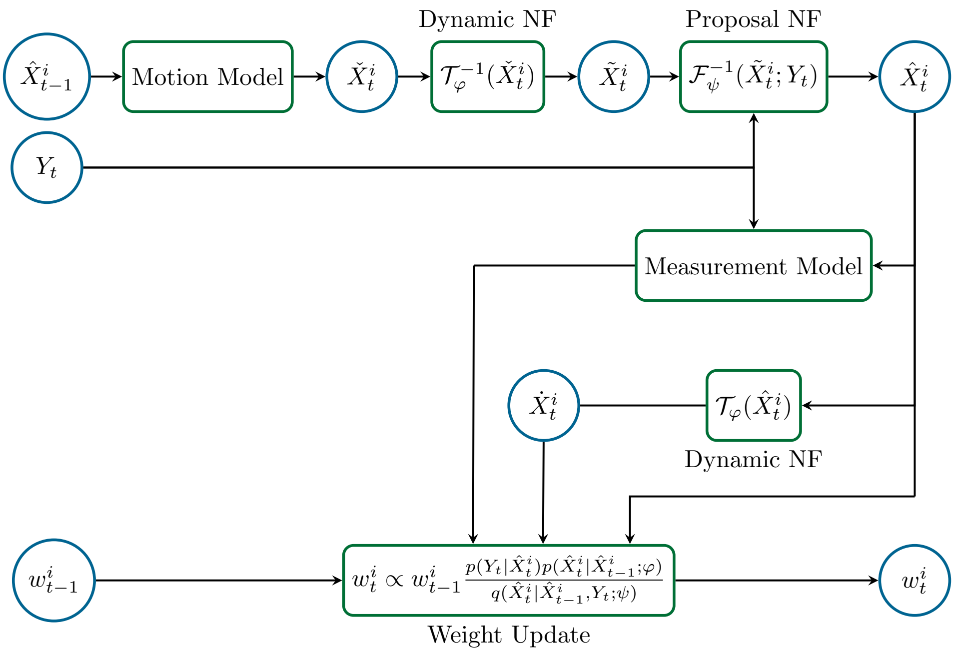
CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG



Prospects of Normalizing Flow-Based Differentiable Particle Filters in Target Modelling

Taking target modelling in complex and diverse target environments to the next level

Master's thesis in Complex Adaptive Systems

GUSTAV MOSSÉEN
SIMON NYGREN

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Prospects of Normalizing Flow-Based Differentiable Particle Filters in Target Modelling

Taking target modelling in complex and diverse target environments to the next level

GUSTAV MOSSÉEN
SIMON NYGREN

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Prospects of Normalizing Flow-Based Differentiable Particle Filters in Target Modelling
Taking target modelling in complex and diverse target environments to the next level
Gustav Mosséen
Simon Nygren

Supervisor: Mattias Hovgard, Saab AB
Examiner: Moritz Schauer, Mathematical Sciences

Master's Thesis 2025
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Illustration of one iteration of the Normalizing Flow-based Differentiable Particle Filter algorithm.

Typeset in LaTeX
Gothenburg, Sweden 2025

Prospects of Normalizing Flow-Based Differentiable Particle Filters in Target Modelling
Taking target modelling in complex and diverse target environments to the next level
Gustav Mosséen
Simon Nygren
Department of Mathematical Sciences
Chalmers University of Technology and University of Gothenburg

Abstract

Accurate target modelling is essential for modern air surveillance, particularly in the face of increasingly dynamic and unpredictable target behaviour, such as that exhibited by hypersonic missiles and UAVs. Traditional tracking methods, including Kalman Filters, Interacting Multiple Model (IMM) filters, and Particle Filters (PF), rely heavily on known target dynamics and are often limited in adaptability. This thesis explores the potential of Differentiable Particle Filters (DPFs), and in particular Normalizing Flow-based DPFs (NF-DPFs), as a data-driven alternative for modelling and tracking targets with unknown or highly non-linear dynamics.

To address these challenges, a NF-DPF was implemented for radar-based target tracking. The filter was designed as a fully differentiable architecture that learns both the proposal distribution and the system dynamics using conditional normalizing flows based on the RealNVP framework. It incorporates optimal transport-based differentiable resampling and was trained end-to-end using a combination of a blockwise Evidence Lower Bound (ELBO) loss and the RMSE in position, velocity, and acceleration. The training was conducted on target data generated from stochastic differential equations (SDEs) simulating a range of motion patterns, including turning and accelerating trajectories. The NF-DPF was benchmarked against a bootstrap PF and a IMM filter across three experiments with increasing data complexity and variations in filter state dimensionality.

Results showed that the NF-DPF provided competitive performance in low-dimensional settings and under specific dynamic behaviours, particularly turning trajectories that were not well captured by the IMM's filters. However, it struggled to outperform the IMM in more complex scenarios involving higher-dimensional state representations. This performance gap was largely attributed to the increased difficulty of learning high-dimensional transition and proposal distributions, combined with computational limitations such as a restricted number of particles.

The results highlight both the potential and current limitations of NF-DPFs for target modelling and contribute to the growing body of research exploring the integration of flexible, learning-based models into particle filtering frameworks for defence and surveillance applications.

Keywords: Differentiable Particle Filter, Target Tracking, Normalizing Flow

Acknowledgements

First and foremost, we would like to extend our gratitude towards Saab for providing us with the opportunity to conduct this thesis. We are especially grateful for the continuous support provided by our supervisor Mattias Hovgard. His insight into the field and guidance throughout the project has been invaluable. We would also like to thank Moritz Schauer for taking on this thesis. Further, we would like to thank the target tracking team at Saab for the warm welcome, and Oscar Schyum and Fredrik Nygren for making our days at Saab even more enjoyable.

As this thesis marks the end of our studies at Chalmers, we would also like to take a moment to express our appreciation to all the friends we made along the way. Our time at Chalmers, and our studies, would not have been the same without you. Lastly, to our families and loved ones, thank you for always remaining by our side, we are forever grateful.

Gustav Mosséen and Simon Nygren, Gothenburg, June 8, 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

CNF	Conditional Normalizing Flow
DPF	Differentiable Particle Filter
EKF	Extended Kalman Filter
ELBO	Evidence Lower Bound
IMM	Interacting Multiple Model
KF	Kalman Filter
NF	Normalizing Flow
NF-DPF	Normalizing Flow-Differentiable Particle Filter
PF	Particle Filter
RealNVP	Real-valued Non-Volume Preserving
RMSE	Root Mean Square Error
SDE	Stochastic Differential Equation
UKF	Unscented Kalman Filter

Nomenclature

Below is the nomenclature of indices, parameters, and variables used throughout this thesis.

Indices

i	Index for a particle
t	Discrete time step

Parameters

φ	Parameter set for the Proposal NF
ψ	Parameter set for the Dynamic NF
N	Number of particles
σ	Standard deviation of noise
C_D	Drag coefficient
A	Reference area
m_0	Initial mass of the target
$\dot{m}$	Mass burn rate
c	Thrust
ρ	Density of the surrounding fluid
ω	Frequency of normal acceleration oscillations (wobbling frequency)
τ	Amplitude of normal acceleration oscillations

Variables

X_t	Target state vector at time t, typically containing position, velocity (and possibly acceleration)

$\hat{X}_t$	Estimated state vector at time t
Y_t	Measurement vector at time t, e.g., $\left[r,\, \theta,\, v_r\right]$
dt	Time step for iteration
dt_{radar}	Time discretization step (measurement interval)
w_t^i	Weight/probability of the i^{th} particle at time t
x, y	Cartesian coordinates of the target's position
v_x, v_y	Components of the target's velocity in the x and y directions
a_x, a_y	Components of the target's acceleration in the x and y directions
a_n, a_t	Components of the target's acceleration in the normal and tangential directions
r	Range measurement
θ	Bearing angle
v_r	Radial velocity
$f(\cdot)$	Non-linear state-transition model
$h(\cdot)$	Measurement model
$q(\cdot)$	Proposal density used in the particle filtering algorithm
W_t	Process noise at time t
V_t	Measurement noise at time t

Contents

List of Figures

List of Tables

1
Introduction

Target tracking is the recursive prediction of a dynamical system's future state, or estimation of its current state based on measurements and predictions. It can describe a wide range of quantities, be it the exhaust temperature of a rocket booster, the value of a stock market share or the position of an aircraft. In the context of air surveillance the state X most often refers to the position, velocity and acceleration of an airborne target [1]. Knowing the state of a target is a necessity in reliable defence and surveillance applications, and is used for interception and awareness of potential threats.

The importance of accurate target modelling techniques has increased significantly due to the escalation of geopolitical tensions, which have been significantly intensified by the Russian invasion of Ukraine in early 2022 [2]. This conflict has seen an unprecedented reliance on aerial warfare, with missiles, drones and UAVs playing a central role [3], [4]. The widespread deployment of these systems has highlighted the need for target models capable of handling highly dynamic and evasive behaviours. Traditional tracking methods struggle to predict and adapt to the evolving tactics employed in modern air combat, underscoring the necessity of data-driven approaches that can learn and generalize across diverse and unknown target types.

Flexible target modelling would enhance tracking capabilities, leading to faster response times, quicker deployment of countermeasures, and a reduced window for potential threats. This reliable real-time awareness could have big societal impacts, contributing to the safekeeping of people and infrastructure. Even outside the context of defence applications, efficient target modelling is a necessity. Advancements in the field could enhance robotics by enabling more accurate tracking of people and objects to improve assistance and interaction. In autonomous vehicles, they could contribute to increased safety by enabling more precise tracking of other vehicles, obstacles, and pedestrians.

1.1 Problem Definition

In practical applications, even though the target's underlying state X_t at time t changes continuously in time, radar measurements are typically recorded at discrete intervals, leading us to consider the time-discretized version $X_t \in \mathbb{R}^f$, $t = 0, 1, 2, \dots,$ where f is the dimensionality of the state. At each t we therefore seek the estimate $\hat{X}_t \in \mathbb{R}^f$, which is to be as close as possible to X_t. In modern air surveillance, this estimate is based on techniques of combining likelihoods from *measurement models*, with predictions from *state-transition models*. These are constructed with the goal to accurately estimate the targets current state [5]. The state-transition model f describes how the target's state evolves over time, that is, its current state X_t based on its previous state X_{t-1},

$$X_t = f(X_{t-1}) + W_t, \tag{1.1}$$

where $W_t \in \mathbb{R}^f$ is some stochastic process noise drawn from a zero mean Gaussian noise distribution. The measurement model h transforms the state X_t into measurement coordinates $Y_t \in \mathbb{R}^m$,

$$Y_t = h(X_t) + V_t, \tag{1.2}$$

where $V_t \in \mathbb{R}^m$ is a zero mean Gaussian noise with known covariance and m denotes the number of measurement components. The complete dynamical system is referred to as the *state-space model* and includes the state-transition model in Eq. (1.1) and the measurement model in Eq. (1.2).

An accurate state-space model requires both a precise state-transition model and a reliable measurement model. Since the measurement model is typically well-defined, based on our equipment for which we have detailed specifications, the overall accuracy largely depends on the state-transition model. This usually requires extensive knowledge of the target's dynamics, as well as the goals of the controller. In cases where this state-space model is known, the problem of target tracking is simplified and can be addressed by standard methods [6].

1.2 Background

The simplest target tracking method is using a single state-transition model, which for example describes a constant velocity (CV) of the target with some small perturbations. The case of constant acceleration (CA), and constant turn (CT) can also be modelled by a single filter. These are sufficient in cases of a single, predictable trajectory. In the case of a linear state-transition and measurement model with additive Gaussian process noise and measurement noise, a closed form solution to the filtering problem exists and is given by the *Kalman Filter* (KF) [7].

The single filter approach works in cases where target dynamics exhibit just one distinct mode of operation. In cases where the target operates in multiple distinct modes and transitions between them unpredictably, such as a drone alternating between straight flight and hovering or a missile undergoing sudden acceleration, *single model filtering* may be insufficient. Such dynamics are better modelled by multiple filters, each describing one distinct dynamic mode. This modelling setup is referred to as *multiple model filtering* [8]. A widely used type of multiple model filter is the *Interacting Multiple Model filter* (IMM), which typically consist of some combination of CV-, CA-, and CT-Kalman filters. The final state prediction is then either a combination of the filters individual state predictions, or simply the prediction of the filter which describes the dynamics the best.

In scenarios where the state-transition and measurement models are non-linear, alternative methods have been proposed to address the filtering problem. Non-linear extensions to the Kalman filters such as the *Extended Kalman Filter* (EKF) and *Unscented Kalman Filter* (UKF) work by linearizing the dynamics and approximating the state distributions using a set of weighted particles respectively [9], [10].

The multiple-model filtering setup, even when including EKFs and UKFs, has inherent limitations. For instance, the number of filters is limited, and performance declines beyond a certain threshold [8]. Moreover, the filters rarely capture the true underlying dynamics and typically rely on approximate models that are only sufficiently accurate for specific scenarios. In scenarios where the non-linearity of the dynamics and measurements or the non-Gaussianity of the noise is not adequately addressed, *Particle Filters* (PF) have been introduced. Particle filters operate by maintaining a set of weighted hypothesis of the current true state. These hypothesis or *particles* are propagated through the state-transition model and are conditioned on the measurement, therefore putting no restrictions or assumptions on the shape of the noise [11]. Accurately representing the range of potential true states typically requires a large number of particles. This increased demand for computational resources is a well-known drawback of particle filters. Another shortcoming is particle degeneracy, where much of the weight is concentrated to only a few particles, thereby oversimplifying the distributions and losing important details [12].

The methods discussed so far rely on partially or completely known target dynamics. However, in most practical scenarios, particularly in target tracking for defence applications, many parameters of the state-space model are unknown. This is especially true when tracking adversarial targets, which are unlikely to move in a cooperative or predictable manner. But what fundamentally prevents an observer from knowing the target's dynamics? The first part of the challenge arises from the broad range of possible target types, including military and civilian aircraft, missiles, projectiles, as well as smaller objects such as birds and UAVs. Inferring the target class could provide valuable constraints on its state-transition model, yet classification itself is a non-trivial problem and does not fully resolve the ambiguity. Even within the same target type, motion can vary significantly with different modes of operation. This diversity in target dynamics necessitates modelling approaches that are both flexible and expressive, capable of adapting to highly non-linear, stochastic, and time-varying dynamics.

For this, data-driven approaches such as the *Differentiable Particle Filter* (DPF) offer a promising avenue. Differentiable particle filters are particle filters where some or all models of the system are trained. The models are generally parametrized by neural networks. These models can approximate the state-transition model, proposal dynamics, the measurement model, or all of them simultaneously. The key advantage of DPFs is their ability to learn representations directly from data, rather than relying on hand-tuned or assumed system models. This is particularly useful in scenarios where the true state-space dynamics are partially or completely unknown, or where the dynamics are highly non-linear and therefore difficult to formulate. By training on historical or simulated data, the learned models can generalize across a range of target behaviours and improve tracking performance [13], [14], [15].

The types of neural networks which has been historically employed in DPFs are vanilla artificial neural networks, meaning standard feed-forward architectures without explicit invertibility constraints. They are inherently non-invertible, preventing explicit density estimation. This limits their ability to model uncertainty and learn flexible proposal distributions, making them less effective in complex scenarios. Normalizing flows (NF) offer a solution by being explicitly invertible and allowing direct likelihood evaluation, enabling more expressive and adaptable proposal distributions [16].

The implementation of differentiable particle filters has been recently explored in a series of articles by Chen et al. [13], [16]. The second article focuses exclusively on Normalizing Flow-based Differentiable Particle Filter (NF-DPF), which are of particular interest for this thesis. It presents the general prospects of NF-DPFs by a series of experiments, with promising results. The experiment of highest relevance is a localization task of a visually obstructed disk. This experiment assumes some quantity of the system (in this case the velocity of the disk) is known, and the task is to estimate its position in every time step t.

In contrast, the problem of target modelling allows no obvious assumptions about the target's velocity, making the task significantly more complex. However, the adaptability of NF-DPF in learning structured distributions from data in localization tasks suggests that similar methodologies can be adapted to track objects under more uncertain conditions.

1.3 Aim

The aim of this Master's thesis is to evaluate the potential of differentiable particle filters in the target modelling context as a more efficient and adaptive alternative to multiple model filtering and non-differentiable particle filter approaches. Emphasis will be put on differentiable particle filters where the networks are Normalizing Flow (NF), since these have not been sufficiently explored by other researchers. This thesis will build upon the work by Chen et al., presented in [16], by extending the NF-DPF to more complex tracking scenarios, where target dynamics are partially unknown. Specifically, the goal is to approximate the posterior probability density function (pdf), defined as

$$p(X_t \mid Y_{0:t}), \tag{1.3}$$

where $Y_{0:t}$ denotes the sequence of measurements from time 0 to time t. In Bayesian filtering, this notation represents the conditional probability density of the state

X_t given all measurements up to time t. This is achieved by training a DPF on a diverse set of target trajectories, enabling it to learn approximations of both the state-transition model and the proposal dynamics.

To achieve this, a differentiable particle filter is proposed, where the neural networks are parametrized with normalizing flows. In addition, an interacting multiple model filter is developed, which serves as the primary algorithm for target modelling, and also a non-differentiable particle filter is implemented. The thesis aims to compare these three approaches using metrics that assess accuracy, uncertainty, and computational efficiency. Through these comparisons, the viability of normalizing flow based differentiable particle filters for complex target modelling contexts is to be determined.

1.4 Delimitations

To guarantee a reasonable and focused scope some delimitations have been made. First of all, all trajectories and filters are assumed to be constrained to a two dimensional plane, defined by some unknown and arbitrary elevation angle ϕ. This plane is spanned by the x- and y-axes. Further this constrains the possible state vector of the target to $X = (x, y, v_x, v_y, a_x, a_y)$ or similarly $X = (x, y, v_x, v_y, a_n, a_t)$ if acceleration is formulated in the normal-, and tangential direction relative to the target. Different variations and subsets of this state vector is considered in the different experimental setups. The measurement model is defined in polar coordinates, consisting of range r, bearing θ and radial velocity v_r. Therefore the proposed method is only applicable to the types of radars capable of providing this information.

Further, the employed framework assumes no missing radar measurements. The measurements are also assumed to be collected at a fixed and equally spaced interval of 1 second. Additionally, only the *online* filtering setup is considered, where state estimation is performed sequentially using only past and current measurements. That is, at each time step t, the state estimate is conditioned on the available measurements up to t: $p(X_t \mid Y_{0:t})$, rather than the full measurement sequence $Y_{0:T}$. This filtering approach enables real-time tracking but does not leverage future observations, which could otherwise refine state estimates in a smoothing framework.

The study does not take into account the problem of multiple-target tracking, although successful results in the limited single-target case would prove useful in the extended case as well. The setup with multiple targets becomes a trivial extension if the number of targets and their relation to the measurements are known.

The NF-DPF will be trained and evaluated on a limited set of dynamics, specifically on various realizations of Stochastic Differential Equations (SDEs) that simulate the behaviour of highly manoeuvrable hypersonic missiles across different operational modes, including acceleration and turning manoeuvrers. Furthermore, compatibility with Saab's current target-modelling pipelines will not be considered, and the measurement model is assumed to be known.

1.5 Report Guide

The remainder of the report follows the structure outlined below. Chapter 2 establishes the theoretical foundations of the research, covering target modelling and tracking, statistical concepts, and relevant machine learning contexts. Chapter 3 details the research methodology, including data generation, filter development, evaluation metrics and the conducted experiments. Chapter 4 provides a detailed description of the developed NF-DPF, focusing on an explanation of each component. Chapter 5 presents the research findings along with an analysis of the conducted experiments. Chapter 6 discusses the implications of these outcomes, and Chapter 7 concludes the report while outlining potential directions for future research.

2

Theory

This chapter begins by outlining the theoretical foundations of the research and reviewing existing methods for state-space modelling in the context of target tracking. It then provides an overview of the machine learning perspective relevant to the thesis. Finally, the chapter presents the statistical framework, used in the implementation and evaluation of the proposed filtering techniques.

2.1 State-Space Modelling

This section provides an overview of the modelling approaches currently used, or which are potentially applicable in target tracking. The first subsections will present single-filter techniques. The section will proceed by introducing Stochastic Differentiable Equations, often used for data generation is introduced.Finally a method that utilizes several filters for a more accurate prediction is introduced.

2.1.1 State-Space Preliminaries

In a Bayesian framework, the true latent state at time t is denoted by X_t, and its estimate inferred from observations is denoted by $\hat{X}_t$. The estimate is obtained from the *posterior probability density function* (pdf), which in the filtering context represents the marginal pdf of the current state X_t given all measurements up to time t. This posterior pdf is defined as:

$$p(X_t \mid Y_{0:t}), \tag{1.3}$$

where $Y_{0:t}$ is the sequence of measurements from time 0 to time t [5]. Each measurement can either be a scalar or a vector containing m components. The sequence of all measurements up to time t can then be written as an ordered set:

$$Y_{0:t} = [Y_0, \ldots, Y_t], \qquad Y_t \in \mathbb{R}^m, \tag{2.1}$$

where each Y_t is a measurement vector at time t. If the posterior pdf in Eq. (1.3) is known, an estimate of the state $\hat{X}_t$ at time t can be derived. Filtering aims to compute this posterior, or an approximation of it, from which state estimates are extracted.

What follows is the exact calculation of the pdf $p(X_t \mid Y_{0:t})$ which results in definitions of the *measurement model* and the *state-transition model* also called the *motion model* introduced in Eq. (1.1) and Eq. (1.2) respectively. By splitting the measurement term in Eq. (1.3) into two parts, the expression becomes:

$$p(X_t \mid Y_{0:t}) = p(X_t \mid Y_t, Y_{0:t-1}), \tag{2.2}$$

which with Bayes' theorem can be rewritten as [17]:

$$p(X_t \mid Y_{0:t}) = \frac{p(Y_t \mid X_t, Y_{0:t-1})p(X_t \mid Y_{0:t-1})}{p(Y_t \mid Y_{0:t-1})}, \tag{2.3}$$

which, given that the measurement density is conditionally independent of previous measurements when the state X_t is known, can be simplified to:

$$p(X_t \mid Y_{0:t}) = \frac{p(Y_t \mid X_t)p(X_t \mid Y_{0:t-1})}{p(Y_t \mid Y_{0:t-1})}. \tag{2.4}$$

As a result, the posterior pdf $p(X_t \mid Y_{0:t})$ is proportional to the product of a likelihood, $p(Y_t \mid X_t)$, referred to as the *measurement model*, and a prior, $p(X_t \mid Y_{0:t-1})$. This represents the combination of predictions from state-transition models with the likelihood from measurement models, as previously described. That is,

$$p(X_t \mid Y_{0:t}) \propto p(Y_t \mid X_t)\, p(X_t \mid Y_{0:t-1}). \tag{2.5}$$

This proportionality is sufficient because the posterior pdf is normalized by dividing by the marginal likelihood (evidence), $p(Y_t \mid Y_{0:t-1})$, which does not depend on the state X_t. As a result, for inference or sampling purposes, the unnormalized form captures the correct posterior shape, and the normalization constant can be computed separately if needed.

The prior $p(X_t \mid Y_{0:t-1})$ is found by marginalizing over the previous state X_{t-1}. By assuming that the target dynamics fulfil the Markov property [18], we get the Chapman-Kolmogorov equation [19],

$$p(X_t \mid Y_{0:t-1}) = \int p(X_t \mid X_{t-1})\, p(X_{t-1} \mid Y_{0:t-1})\, \mathrm{d}X_{t-1}, \tag{2.6}$$

the integral propagates the density $p(X_{t-1} \mid Y_{0:t-1})$ at time $t-1$ through the pdf $p(X_t \mid X_{t-1})$, which is our *state-transition model.* Using Eq. (2.4) and Eq. (2.6), we can express the posterior density $p(X_t \mid Y_{0:t})$, as a function of the previous posterior $p(X_{t-1} \mid Y_{0:t-1})$, the state-transition model $p(X_t \mid X_{t-1})$, and the measurement model $p(Y_t \mid X_t)$. If initialized with a prior density $p(X_0)$ we therefore have a recursive way of calculating the posterior pdf.

An alternate way of writing the state-transition and measurement model, which will be used frequently in this thesis, is given by Eqs. 1.1 and 1.2, introduced in

Section 1.1. The state-transition model f describes how the current state X_t evolves from the previous state X_{t-1},

$$X_t = f(X_{t-1}) + W_t, \tag{1.1}$$

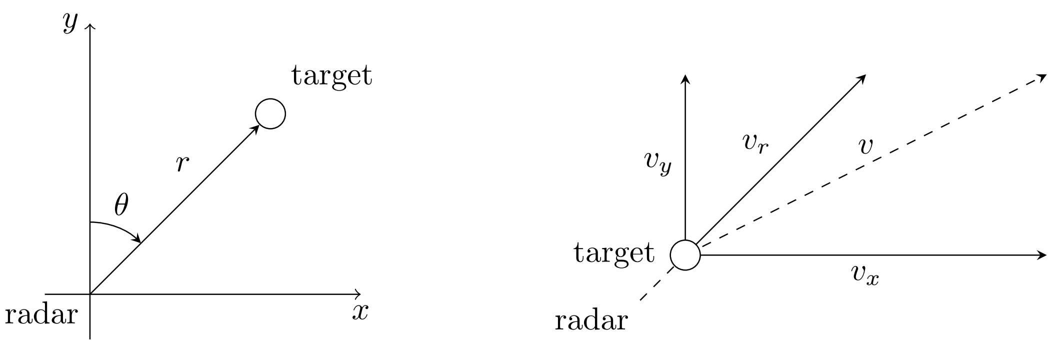
where W_t is some stochastic process noise drawn from a zero mean Gaussian noise distribution. $f(X_{t-1})$ typically represents everything deterministic, which can be modelled, while W_t represents everything which can not. The state is assumed to follow the Markov Property, that is the state at time t is deductible from the state at time $t-1$ alone [18]. The measurement model h transforms the state X_t into measurement coordinates Y_t,

$$Y_t = h(X_t) + V_t, \tag{1.2}$$

where V_i is a zero mean Gaussian noise with known covariance. Similar to the state-transition model, $h(X_t)$ represents everything deterministic which can be modelled, and V_t everything which can not. Together, f and h define the state-space model.

2.1.2 Measurement Definition

In the two dimensional spatial space, which will be considered in this thesis, the measurement vector at one time step t is defined as $Y_t = (r, \theta, v_r)$. The measurements are noisy with deviations in range (r), bearing (θ) and radial velocity (v_r). As illustrated in Figure 2.1, range r is the distance to the target, bearing θ refers to the relative angle in the Cartesian plane between the y-axis and the target and radial velocity v_r, also called *Doppler velocity,* is the component of the target's velocity along the line of sight of the radar.



(a) Horizontal x-y plane which the targets are constrained to, with definitions of r and θ.

(b) Definition of velocity vector v, and radial velocity v_r, along with components v_x and v_y.

Figure 2.1: Definition of introduced notations, (a) showing radar notations, and (b) velocity notations.

In the two-dimensional setting considered in this thesis, the measurement vector at time step t is given by $Y_t = (r, \theta, v_r)$. The non-linear measurement function h in Eq. (1.2) (with subscript t omitted for clarity) is defined as:

$$h(X) = \begin{bmatrix} r(X) \\ \theta(X) \\ v_r(X) \end{bmatrix} = \begin{bmatrix} \sqrt{x^2 + y^2} \\ \arctan(x, y) \\ (xv_x + yv_y)/r \end{bmatrix}, \tag{2.7}$$

where r and θ transform the state X into range r, and angle θ, respectively, and the radial velocity v_r is computed from the velocity components and range.

2.1.3 Data Generation

Experimental data for target modelling is traditionally generated by approximating solutions to *Stochastic Differentiable Equations* (SDEs), describing target trajectories. If available, measurements are generated by applying the measurement model in Eq. (1.2) at discrete time steps t.

Curvilinear-Motion Model
A curvilinear path describes the motion of a target moving along a curved trajectory. It can also represent straight-line motion by setting the normal acceleration to zero, $a_n = 0$, or close to straight-line motion by setting the mean normal acceleration to zero $\bar{a}_n = 0$. The standard curvilinear kinematics model presented by Rong is defined as [20]:

$$\dot{x}(t) = v(t) \cos \phi(t), \tag{2.8}$$

$$\dot{y}(t) = v(t) \sin \phi(t), \tag{2.9}$$

$$\dot{v}(t) = a_t(t), \tag{2.10}$$

$$\dot{\phi}(t) = \frac{a_n(t)}{v(t)}. \tag{2.11}$$

Here, (x, y) represent the target's position, with $\dot{x}(t), \dot{y}(t)$ being the change in x and y at time t. Further, v is the speed, and ϕ is the heading angle. The terms a_t and a_n correspond to the target's tangential and normal accelerations, respectively. The behaviour of a_t and a_n determines the model's dynamics, and is what defines different target trajectories.

Approximation of SDEs
Solutions to the Curvilinear-Motion Model SDEs are approximated using the Euler-Maruyama method, which extends the standard Euler method to stochastic differential equations by incorporating a noise term driven by the underlying Wiener process [21]. For a general SDE of the form:

$$dX_t = f(X_t, t)\, dt + g(X_t, t)\, dW_t,$$

where X_t represents the state at time t, $f(\cdot)$ is the drift term, $g(\cdot)$ is the diffusion term, and W_t is the Wiener process, the Euler-Maruyama discretization updates the state as:

$$X_{t+dt} = X_t + f(X_t, t)\, dt + g(X_t, t)\, \Delta W_t, \tag{2.12}$$

where $\Delta W_t \sim \mathcal{N}(0, dt)$ is the increment of the Wiener process. This numerical approximation is performed at discrete time steps of size dt over a total duration of T seconds. To ensure synchronization with the radar measurement interval $\Delta t_{\text{radar}} \in \mathbb{Z}^+$, T is chosen as a positive integer multiple of Δt_{radar}, i.e., $T \in \mathbb{Z}^+$.

Measurement Generation
Accurate and realistic measurement generation is essential for evaluating the performance of the filtering techniques in target tracking. Measurements are assumed to be generated from a known measurement model that relates the target state to the measurements.

As defined in Eq. (1.2), at each time step t, the measurement Y_t is given by $Y_t = h(X_t) + V_t$, where $h(\cdot)$ represents the non-linear measurement function in Eq. (2.7), and V_t is the measurement noise. To reflect real-world sensor uncertainty, measurement noise is modelled as an additive Gaussian process with $V_t \sim \mathcal{N}(0, R_t)$, where R_t is the covariance matrix describing the noise characteristics of the sensor at time t. For the measurement model considered, the noise, and therefore the covariance matrix R, is constant in time [22] and given by:

$$R = \begin{bmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_\theta^2 & 0 \\ 0 & 0 & \sigma_{v_r}^2 \end{bmatrix} = \begin{bmatrix} 10.0^2 & 0 & 0 \\ 0 & 1.0^2 & 0 \\ 0 & 0 & 5.0^2 \end{bmatrix}, \tag{2.13}$$

with variances in range r in meters, bearing θ in degrees and radial velocity v_r in m/s.

2.1.4 Kalman Filter

When the functions f and h are linear, and the noise terms W_t and V_t in the state-transition model Eq. (1.1) and the measurement model Eq. (1.2) are Gaussian, the optimal solution to the filtering problem, is given by the *Kalman filter* (KF). The Kalman filter is widely used for state estimation and signal processing [7]. It improves estimation accuracy by constructing joint probability distributions between predictions and observations [23].

The Kalman filter is a recursive estimator. The current state is approximated based only on the estimated state from the previous time step $\hat{X}_{t-1}$ and the current observation Y_t. Each iteration of the estimator consists of two phases, which are often considered separately: *Prediction* and *Measurement Update*. For a discrete time-series, the prediction step for the Kalman filter is defined as,

Prediction

$$\hat{X}_{t|t-1} = F_t \hat{X}_{t-1|t-1} + B_t U_t, \tag{2.14}$$

$$P_{t|t-1} = F_t P_{t-1|t-1} F_t^{\mathrm{T}} + Q_t. \tag{2.15}$$

The prediction begins with estimating the current state $\hat{X}_{t|t-1}$ by applying the state transition model F_t to the previously predicted state $\hat{X}_{t-1|t-1}$. When a control input

U_t is available, for example the acceleration of a car when estimating its position, it is added to current state together with the control input model B_t. The uncertainty of the estimated current state are predicted as the estimate covariance $P_{t|t-1}$. It gets updated according to Eq. (2.15), where Q_t is the process covariance.

The measurement update step refines the predicted state by incorporating the measurement Y_t, which in the car example could be an odometer reading, a measurement of the total distance the vehicle has travelled. This information is then used to correct the current estimated state $\hat{X}_{t|t-1}$.

Measurement update

$$\tilde{Y}_t = Y_t - H_t \hat{X}_{t|t-1}, \tag{2.16}$$

$$S_t = H_t P_{t|t-1} H_t^{\mathrm{T}} + R_t, \tag{2.17}$$

$$K_t = P_{t|t-1} H_t^{\mathrm{T}} S_t^{-1}, \tag{2.18}$$

$$\hat{X}_{t|t} = \hat{X}_{t|t-1} + K_t \tilde{Y}_t, \tag{2.19}$$

$$P_{t|t} = (I - K_t H_t) P_{t|t-1}. \tag{2.20}$$

The residual $\tilde{Y}_t$ quantifies the difference between the observed data and predicted measurement $H_t \hat{X}_{t|t-1}$, where H_t is the measurement model which maps the state to the measurement space. The measurement model is then also used for obtaining the measurement covariance S_t, where the observation covariance R_t are added. The Kalman gain K_t from Eq. (2.18) determines how much the predicted state should be corrected based on the new measurement. A high Kalman gain results in larger corrections, while a low gain leads to smaller ones. The updated state estimate $\hat{X}_{t|t}$ and its corresponding covariance $P_{t|t}$ are then computed using the Kalman gain, as shown in Eq. (2.19) and (2.20). The recursion is initialized by specifying an initial state estimate $\hat{X}_{0|0}$ and its covariance $P_{0|0}$.

2.1.5 Extended Kalman Filter

The Kalman Filter, is the optimal filtering technique only under certain conditions [9]. It relies on the assumptions of linear models and Gaussian distributions [7]. The *Extended Kalman Filter* (EKF) expands the regular Kalman filter by linearizing either one or both of the transition and the measurement model around the current state estimate. The EKF operates on the non-linear system:

$$X_t = f(X_{t-1}, U_t) + W_t,$$
$$Y_t = h(X_t) + V_t.$$

Where $f(X_{t-1}, U_t)$ is the non-linear transition model with added process noise $W_t \sim \mathcal{N}(0, Q_t)$, and $h(X_t)$ is the non-linear measurement model with added measurement noise $V_t \sim \mathcal{N}(0, R_t)$. Since the system equations are non-linear, the KF cannot be applied directly, as matrix operations such as multiplication and inversion are not valid for general non-linear functions. Instead, EKF linearizes the function by computing their first-order Taylor series approximation around the current state as follows [23]:

$$F_t = \left.\frac{\partial f}{\partial X}\right|_{\hat{X}_{t-1|t-1}, U_t}, \tag{2.21}$$

$$H_t = \left.\frac{\partial h}{\partial X}\right|_{\hat{X}_{t|t-1}}. \tag{2.22}$$

The EKF implementation therefore replaces F_t and H_t in Eq. (2.14)-(2.20) of the original KF with the linearizations in Eq. (2.21) and (2.22).

2.1.6 Unscented Kalman Filter

The Extended Kalman Filter has been widely adopted for non-linear state estimation due to its ability to handle non-linearities through first-order Taylor series expansion [10]. However, its reliance on local linearization introduces approximation errors, which can lead to divergence in highly non-linear systems [24]. To address these limitations, the *Unscented Kalman Filter* (UKF) was introduced by Julier and Uhlmann, providing a more accurate alternative through the Unscented Transform (UT) [24]. The UKF replaces the Jacobian-based linearization of the EKF in Eq. (2.21) and Eq. (2.22) with a sigma-point sampling approach, capturing the system's higher-order statistics more effectively. A set of $2n + 1$ sigma points $\{X_j\}_{j=0}^{2n}$ are deterministically generated around the current state mean $\hat{X}_{t-1|t-1}$ and covariance $P_{t-1|t-1}$ as follows:

$$X_0 = \hat{X}_{t-1|t-1}, \tag{2.23}$$

$$X_j = \hat{X}_{t-1|t-1} + \left(\sqrt{(n+\lambda)P_{t-1|t-1}}\right)_j, \quad j = 1, \ldots, n, \tag{2.24}$$

$$X_j = \hat{X}_{t-1|t-1} - \left(\sqrt{(n+\lambda)P_{t-1|t-1}}\right)_{j-n}, \quad j = n+1, \ldots, 2n, \tag{2.25}$$

where n is the dimension of the state vector, and λ is a scaling parameter controlling the spread of the sigma points. The square root of the covariance is typically computed using a *Cholesky decomposition* [24]. Each sigma point X_j is propagated through the non-linear transition function $f(\cdot)$, and the predicted mean and covariance are computed as:

$$\hat{X}_{t|t-1} = \sum_{j=0}^{2n} w_j^a f(X_j), \tag{2.26}$$

$$P_{t|t-1} = \sum_{j=0}^{2n} w_j^c \left(f(X_j) - \hat{X}_{t|t-1}\right)\left(f(X_j) - \hat{X}_{t|t-1}\right)^T + Q_t, \tag{2.27}$$

where w_j^a and w_j^c are the weights for the mean and covariance, respectively. The notation $\hat{X}_{t|t-1}$ and $P_{t|t-1}$ refers to the predicted (prior) mean and covariance at time t, before incorporating the measurement at time t. The measurement update will not be presented here, but it follows an analogous procedure: a new set of sigma points is constructed based on the predicted state, propagated through the measurement function h, and used to compute the predicted measurement distribution and Kalman gain. Interested readers are referred to the complete algorithm in [25]. The transformed set of sigma points is used to reconstruct a Gaussian approximation of the predicted distribution, preserving only the first two moments (mean and covariance), as visualized in Figure 2.2.

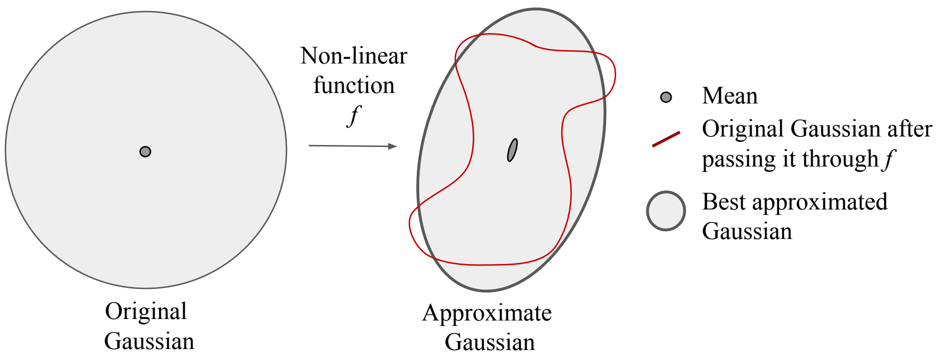


Figure 2.2: The UKF propagates a set of sigma points through the non-linear state-transition function f to estimate the transformed mean and covariance.

2.1.7 Interacting Multiple Model

When the target dynamics cannot be accurately captured by a single model, or when they switch unpredictably between different motion regimes, such as a drone flying straight ahead or hovering, or a fighter jet performing sudden turns, it may be necessary to employ multiple filters. A widely used approach for this purpose is the *Interacting Multiple Model* (IMM) filter.

The IMM filter includes several filters and actively switches between or combines the filters depending on the most optimal one at each time step [26]. The IMM filter is composed of four processes: *Interaction*, *Filtering*, *Model Probability Update*, and lastly *State Estimation*. A flowchart of the IMM described in this section is illustrated in Figure 2.3. The layout of the flowchart is inspired by the implementation described in [27]. Due to the scope of this project, only the method for IMM built with Kalman filter variations will be defined.

Interaction
The interaction process computes a weighted combination of the state estimates

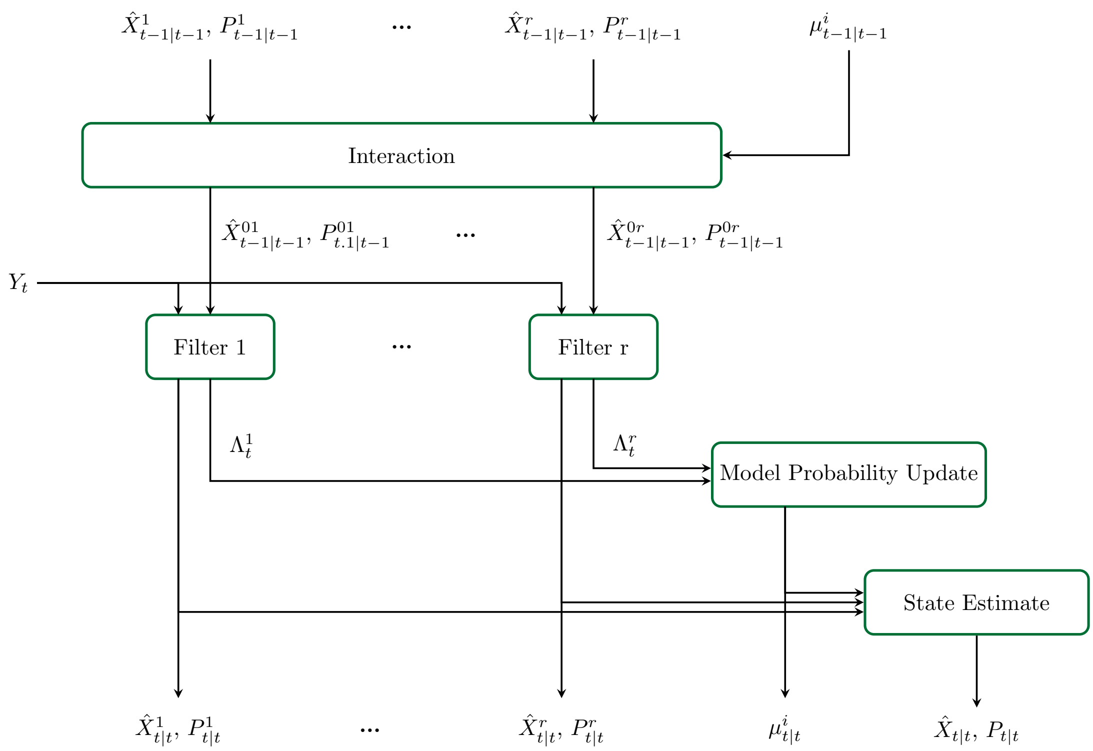


Figure 2.3: Flowchart of the Interacting Multiple Model (IMM) algorithm for Kalman Filters.

from the previous time step using the series of equations below.

$$\bar{c}_i = \sum_{j=1}^{r} \pi^{ij} \mu^j_{t-1|t-1}, \tag{2.28}$$

$$w_{ij} = \frac{\pi^{ij} \mu^j_{t-1|t-1}}{\bar{c}_i}, \tag{2.29}$$

$$\hat{X}^{0i}_{t-1|t-1} = \sum_{j=1}^{r} \hat{X}^j_{t-1|t-1} w_{ij}, \tag{2.30}$$

$$\Delta P^j = \left(\hat{X}^j_{t-1|t-1} - \hat{X}^{0i}_{t-1|t-1}\right)\left(\hat{X}^j_{t-1|t-1} - \hat{X}^{0i}_{t-1|t-1}\right)^T, \tag{2.31}$$

$$P^{0i}_{t-1|t-1} = \sum_{j=1}^{r} \left[P^j_{t-1|t-1} + \Delta P^j\right] w_{ij}, \tag{2.32}$$

where r is the number of filters, $\mu^i_{t-1|t-1}$ is the dynamical model probability and π^{ij} is the static predefined model transition probability, representing the probability of being in filter i at time $t-1$ and the probability of transitioning from model j to model i respectively. The interaction process ensures that the initial condition for each filter dynamically adapts to the previous most probable state.

Filtering

The filtering process passes $\hat{X}^{0i}_{t-1|t-1}$ from interaction into each filter that returns the state estimate $\hat{X}^i_{t|t}$. In the case of a IMM with Kalman filters, this corresponds to the state update in Eq. (2.19). In addition, the likelihood of each filter is calculated based on the measurement Y_t as follows:

$$\Lambda^i_t = p\left(Y_t \mid \hat{X}^i_{t|t}, P^i_{t|t}\right), \tag{2.33}$$

which in return can be calculated as the Gaussian probability density function:

$$\Lambda^i_t = \frac{1}{\sqrt{(2\pi)^d |S^i_t|}} \exp\left(-\frac{1}{2}\left(\tilde{Y}^i_t\right)^T \left(S^i_t\right)^{-1} \tilde{Y}^i_t\right). \tag{2.34}$$

A general definition of the multivariate Gaussian PDF is provided in Section 2.3.2.

Model Probability Update
Next step is to update the model probability $\mu^i_{t|t}$ according to the following:

$$\mu^i_{t|t} = \frac{\bar{c}_i \Lambda^i_t}{\sum_{j=1}^r \bar{c}_j \Lambda^j_t}, \tag{2.35}$$

which satisfies the property

$$\sum_i \mu^i_{t|t} = 1. \tag{2.36}$$

State estimation
The final step is the prediction of the current state and covariance, where either a combining or switching technique is used. In the combining approach, the predicted states are interpolated using the corresponding model probabilities, as follows:

$$\hat{X}_{t|t} = \sum_{j=1}^r \hat{X}^j_{t|t} \mu^j_{t|t}. \tag{2.37}$$

Simultaneously, the covariance matrix is calculated as follows:

$$P_{t|t} = \sum_{j=1}^r \mu^j_{t|t} \left[P^j_{t|t} + \left(\hat{X}^j_{t|t} - \hat{X}_{t|t}\right)\left(\hat{X}^j_{t|t} - \hat{X}_{t|t}\right)^T\right]. \tag{2.38}$$

If the model instead switches, the state and covariance matrix correspond to those of the model with the highest probability:

$$\hat{X}_{t|t} = \hat{X}^{j^*}_{t|t}, \quad P_{t|t} = P^{j^*}_{t|t}, \tag{2.39}$$

where:

$$j^* = \arg\max_j \mu^j_{t|t}. \tag{2.40}$$

This means that instead of blending the models, the IMM selects the state estimate and covariance from the model with the highest posterior probability $\mu^j_{t|t}$, assuming that this model best represents the current system dynamics.

2.1.8 Particle Filter

When tracking targets in complex environments, many real-world systems exhibit non-linear dynamics and non-Gaussian noise, which are jot adequately captured by the linearisation of the EKF and the sigma-point sampling of the UKF. In such cases, the estimation accuracy of these filters degrades, sometimes severely. To address this, Particle Filters (PFs) provide a more flexible alternative, able to handle non-linearities and non-Gaussian noise [28].

The Particle Filter is a recursive Bayesian filtering method that approximates the posterior distribution of a system's latent state using a set of weighted samples, or particles. Unlike the Kalman Filter and its extensions, which assume Gaussian distributions and rely on linearizations or moment approximations, the Particle Filter is non-parametric and can represent arbitrary distributions, making it well-suited for complex, multi-modal, or highly non-linear systems.

The PF approximates the posterior distribution $p(X_t \mid Y_{1:t})$ using a set of N weighted samples $\{\hat{X}_t^i, w_t^i\}_{i=1}^N$, where each particle $\hat{X}_t^i \in \mathbb{R}^d$ represents a candidate state at time t, and its associated weight $w_t^i \in \mathbb{R}$ reflects the likelihood of that state given the observations up to time t [29]. The algorithm consists of three key steps: *Propagation*, *Weight Update*, and *Resampling*. Additionally, a *State Estimation* step is performed. An overview of the particle filter procedure is visualized in Figure 2.4, and each step is described in detail below.

Propagation
Each particle is propagated forward using the system dynamics:

$$\hat{X}_t^i = f(\hat{X}_{t-1}^i, W_t), \tag{2.41}$$

where $f(\cdot)$ is the non-linear state transition model that models the prior distribution $f(\cdot) \sim p(\hat{X}_t^i \mid \hat{X}_{t-1}^i)$. The input W_t is the process noise. Note that, similarly to Kalman filters, the model can depend on a control input u_t so $f(\cdot) = f(\hat{X}_{t-1}^i, u_t, W_t)$, however this is omitted to simplify the notation.

Weight Update
Once a new measurement Y_t is received at time step t, the weights of the particles are updated according to their likelihood [16]:

$$w_t^i \propto w_{t-1}^i \frac{p(Y_t \mid \hat{X}_t^i) p(\hat{X}_t^i \mid \hat{X}_{t-1}^i)}{q(\hat{X}_t^i \mid \hat{X}_{t-1}^i, Y_t)}, \tag{2.42}$$

where $p(Y_t \mid \hat{X}_t^i)$ is the likelihood density derived from some measurement model $h(\cdot) \sim p(Y_t \mid \hat{X}_t^i)$ and $q(X_t^i \mid X_{t-1}^i, Y_t)$ is the proposal density. In general, this proposal distribution can differ from the *state-transition model* (prior density), $p(\hat{X}_t^i \mid \hat{X}_{t-1}^i)$. However, in the *Bootstrap Particle Filter*, these two densities are chosen to be the same, i.e., $q(\hat{X}_t^i \mid \hat{X}_{t-1}^i, Y_t) = p(\hat{X}_t^i \mid \hat{X}_{t-1}^i)$ [11], [30]. Under this assumption, the weight update simplifies to depend solely on the measurement

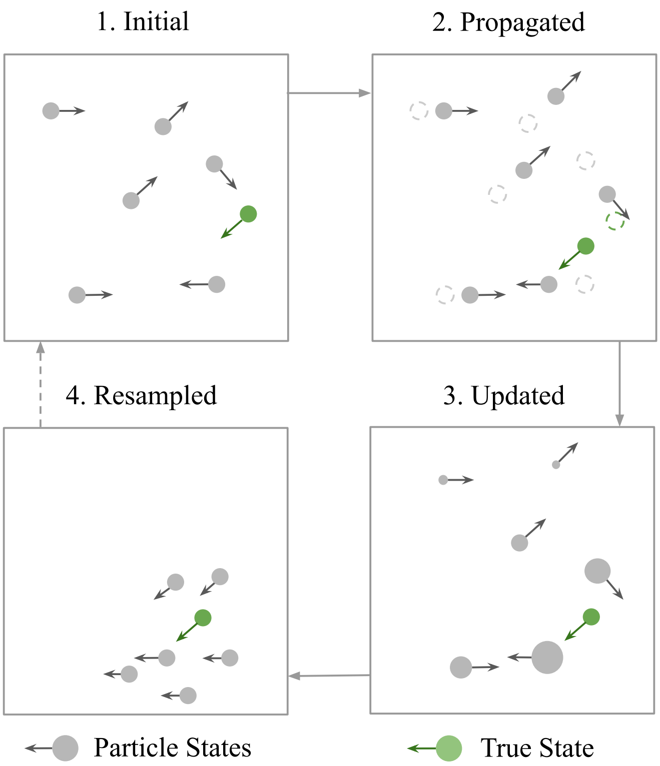


Figure 2.4: The particle filter algorithm from the initial distribution at time t and the three main steps. Step 1: The posterior distribution from the previous time step, $p(X_{t-1} \mid Y_{t-1})$, is represented by a set of weighted particles. Step 2: The particles are propagated through the motion model, yielding the prior $p(X_t \mid Y_{t-1})$. Step 3: The weights are updated based on the new observation Y_t, forming the posterior $p(X_t \mid Y_t)$. Step 4: If a significant weight imbalance is present, the posterior is resampled, producing particles for the next iteration. Larger dots indicate particles with higher weights.

likelihood density:

$$w_t^i \propto w_{t-1}^i \, p(Y_t \mid \hat{X}_t^i). \tag{2.43}$$

The final step of the weight update is normalization, such that $\sum_{i=1}^N w_t^i = 1$.

Estimation

The estimated state at time step t is computed as the weighted sum of the individual particle states, yielding the posterior mean of the state:

$$\hat{X}_t = \sum_{i=1}^N w_t^i \hat{X}_t^i, \tag{2.44}$$

where w_t^i are the normalized weights. This estimate coincides with the Bayesian posterior expectation $\mathbb{E}[X_t \mid Y_{0:t}]$. The covariance matrix $\hat{P}_t$ at time t can be approximated using the weighted sample covariance of the particle set:

$$\hat{P}_t = \sum_{i=1}^N w_t^i \left(\hat{X}_t^i - \hat{X}_t\right)\left(\hat{X}_t^i - \hat{X}_t\right)^T. \tag{2.45}$$

Resampling
A problem for particle filters is weight degeneracy, which is when the weight distribution becomes highly skewed with a few particles having large weights, while many have near-zero weights. When this occur, many particles contribute very little, reducing the efficiency of the filter. To mitigate this, a resampling step is often introduced. In this step, particles with higher weights are replicated, while those with lower weights are discarded or moved toward high-probability regions. The time step at which resampling can be triggered is determined based on a threshold applied to the Effective Sample Size (ESS) metric,

$$\mathrm{ESS}_t = \frac{1}{\sum_{i=1}^{N} w_t^i}. \tag{2.46}$$

Resampling is performed when ESS falls below a predefined threshold, often set to $\mathrm{ESS}_{\mathrm{min}} = \frac{N}{2}$ which can be interpreted as resampling being triggered when fewer than half the particles are effectively contributing.

For clarity, after a new measurement Y_t is introduced, the predicted particles in time step t become the input particles ($\hat{X}_t^i \rightarrow \hat{X}_{t-1}^i$) for the non-linear state transition model in Eq. (2.41) at time step $t+1$. The particles are continuously resampled, propagated, and assigned new probabilities, with the aim of converging toward the true state X_t.

2.2 Machine Learning Context

The following section introduces key concepts from the field of Machine Learning (ML) that are essential for understanding the methods and discussions presented in this thesis. It begins with a general overview of ML, with particular emphasis on model architecture and training procedures. Special focus is given to the Differentiable Particle Filter (DPF), a trainable extension of the traditional Particle Filter (PF), which plays a central role in this work. Finally, the specific types of neural networks used within the DPFs developed in this thesis are defined.

2.2.1 Machine Learning Preliminaries

Machine Learning is traditionally defined as the use of algorithms and statistical models to identify patterns in data without being explicitly programmed for specific tasks [31]. These models generalize to new, unseen data by applying the learned patterns to make predictions or draw conclusions. At the core of many machine learning models is the *artificial neural network*, a model composed of a series of interconnected layers parametrized by φ, where φ includes the *weights* w and *biases* b. Each layer contain small computational units called *neurons*, which are connected by the weights. The output of neuron i, denoted x_i, in layer $k+1$ is computed as a weighted sum of the outputs from the neurons connected to it, indexed by j, and the weights w_{ij}, as follows:

$$x_i(k+1) = g_\varphi\left(\sum_{j=1}^{N} w_{ij}\, x_j(k) + b_i\right). \tag{2.47}$$

Here g denotes the *activation function*, a non-linear transformation which allows the model to learn more complex patterns. The activation function g can take several forms, although the default is trending towards the Rectified Linear Unit (ReLU) [32]. The bias term b_i, together with the weights w_{ij}, constitutes the set of parameters that are updated during training to allow the artificial neural network to learn from data. By stacking several layers of neurons, artificial neural networks are able to capture complex patterns in data. These deep architectures, known as Deep Neural Networks (DNNs), form the foundation of the field of Deep Learning (DL) [33].

Training of Machine Learning Models

Training an artificial neural network typically begins by defining a loss function $\mathcal{L}$, which quantifies how well the model's output $\hat{y} = x(T)$ aligns with the learning objective. In supervised tasks, the loss often measures the difference between $\hat{y}$ and the true target y, while in more general contexts, such as variational inference, it can encode other criteria like entropy, divergence, or policy optimality. Secondly, the gradient ∇ of the loss function with respect to the model parameters φ is computed, that is $\partial\mathcal{L}/\partial\varphi_i$ for all parameters i in the network. An update step of magnitude η, where η is referred to as the learning rate, is performed in the direction of the negative gradient. This is the basis of the *optimizer* known as Stochastic Gradient Descent (SGD) [34]:

$$\varphi_{t+1} = \varphi - \eta\nabla\mathcal{L}(\varphi). \tag{2.48}$$

Another optimizer, which has become increasingly more popular in ML models is the Adaptive Moment Estimation Optimizer (Adam). Instead of using a single global learning rate for all parameters during training, the Adam optimizer adapts the learning rate individually for each parameter and at each update step. It does so by maintaining moving averages of both the gradients (first moment) and the squared gradients (second moment), which often leads to faster and more stable convergence [35].

The process of computing the loss with respect to the model parameters and updating the weights using an optimizer is known as *back-propagation*. Through successive iterations of back-propagation, the model refines its parameters, gradually converging toward a configuration that minimizes the loss function.

Challenges in Machine Learning

One common challenge in machine learning is that models often fail to generalize to new, unseen data, instead learning patterns specific only to the training set [32]. Techniques to encourage better generalization include training the models using batches, where a mean loss is computed over a set of data rather than a single instance of data [34]. This approach helps to prevent *overfitting*, reducing the risk that the model merely learns the training data rather than the underlying patterns of the dataset.

Additionally, the use of masking techniques can improve generalization. Masks randomly block parts of the data from being propagated through the model, forcing it to rely on smaller subsets of the input and thus promoting robustness. Another common technique is *dropout*, where random model parameters are temporarily set to zero during training, further preventing over-reliance on specific neurons or features and encouraging the model to develop more distributed representations [36].

Another challenge in training machine learning models, particularly deep neural networks, is the problem of exploding and vanishing gradients [37]. During back-propagation, gradients can either grow exponentially large or shrink toward zero as they are propagated through many layers. Exploding gradients can cause unstable updates to model parameters, leading to divergence during training. Vanishing gradients, on the other hand, slow down learning dramatically, often preventing the model from capturing long-range dependencies. To mitigate these issues, several strategies have been developed. Gradient clipping is commonly used to address exploding gradients by capping the magnitude of the gradients during back-propagation [37]. To counter vanishing gradients, specialized architectures such as Long Short-Term Memory (LSTM) networks [38] and Gated Recurrent Units (GRUs) [39] have been introduced, which are designed to maintain stable gradients over long sequences. Additionally, careful weight initialization methods and the use of activation functions such as ReLU can also help in maintaining healthy gradient flow during training [40].

2.2.2 Differentiable Particle Filter

The Differentiable Particle Filter (DPF) extends the Bootstrap Particle Filter described in Section 2.1.8 by using neural networks to learn the state-transition, and/or measurement models, as well as the proposal distribution, directly from data. To allow for back-propagation during training, the entire pipeline has to be differentiable. This approach improves flexibility and allows the filter to handle complex, non-linear, and partially unknown system dynamics. The following outlines how the steps differ from the Particle Filter described in Section 2.1.8.

Prediction
The non-linear state-transition model $f(\cdot)$ is replaced by the differentiable state-transition model $g_\varphi(\cdot, \cdot)$, a neural network parametrized by φ:

$$\hat{X}^i_{t|t-1} = g_\varphi(\hat{X}^i_{t-1|t-1}, W_t) \sim p(\hat{X}^i_t \mid \hat{X}^i_{t-1}; \varphi), \tag{2.49}$$

where W_t is a stochastic noise process used to capture process uncertainty. As previously noted in Section 2.1.8, the model may depend on a control input u_t, but this is omitted here for clarity. The measurement model $h(\cdot) \sim p(Y_t \mid \hat{X}^i_t)$ is also made differentiable, typically via function $l_\theta(\cdot)$ [16], parametrized by θ, resulting in the likelihood density:

$$p(Y_t \mid \hat{X}_t^i; \theta) \propto l_\theta(Y_t, \hat{X}_t^i). \tag{2.50}$$

In contrast to the Particle Filter, the proposal distribution is learned through a differentiable network $f_\psi(\cdot)$, parametrized by ψ, which proposes updated particles:

$$\hat{X}_{t|t}^i = f_\psi(\hat{X}_{t-1|t-1}^i, Y_t, \upsilon_t) \sim q(\hat{X}_t^i \mid \hat{X}_{t-1}^i, Y_t; \psi), \tag{2.51}$$

where υ_t is the sampling noise $\upsilon_t \sim p_{\upsilon_t}$.

Weight Update
The weight update follows the same structure as in the general Particle Filter formulation presented in Eq. (2.42). However, the densities are now dependent on the trained models, given by:

$$w_t^i \propto w_{t-1}^i \frac{p(Y_t \mid \hat{X}_t^i; \theta) p(\hat{X}_t^i \mid \hat{X}_{t-1}^i; \varphi)}{q(\hat{X}_t^i \mid \hat{X}_{t-1}^i, Y_t; \psi)}. \tag{2.52}$$

Estimation
The final state estimation is calculated identically for both the particle- and differentiable particle filter as the weighted sum of the individual particle states, in accordance to Eq. (2.44).

Differentiable Resampling
Similarly to the PF, a primary challenge in the Differentiable Particle Filter (DPF) is weight degeneracy, where a few particles dominate the distribution while others have negligible influence. Unlike the discrete and non-differentiable resampling step used in traditional PFs, the DPF requires a differentiable resampling mechanism that supports gradient-based optimization. To achieve this, the *Entropy-Regularized Optimal Transport* (OT-resampling) framework is employed [41], [42].

In the context of resampling, the current set of weighted particles is interpreted as a discrete source distribution:

$$\mu = \sum_{i=1}^{N} w_t^i \, \delta_{\hat{X}_t^i}, \tag{2.53}$$

where $\hat{X}_t^i \in \mathbb{R}^d$ is the i:th particle and $\delta_{\hat{X}_t^i}$ denotes the Dirac delta function centred at $\hat{X}_t^i$, representing a unit mass at that location. The target distribution is defined as:

$$\nu = \sum_{j=1}^{N} \frac{1}{N} \, \delta_{\tilde{X}_t^j}, \tag{2.54}$$

where $\tilde{X}_t^j \in \mathbb{R}^d$ is the j:th resampled (equally weighted) particle. These particles are initially treated as unknown interpolation targets and are computed through optimal transport. This setup interprets the weighted particles as a discrete source distribution (a posterior approximation) and generates a uniformly weighted target distribution, effectively redistributing particle mass while preserving the posterior shape.

This setup is cast as an optimal transport (OT) problem, which seeks a transport plan $\pi \in \mathbb{R}^{N\times N}$ that minimizes the cost of moving mass from μ to ν. The cost of transporting mass from $\hat{X}_t^i$ to $\tilde{X}_t^j$ is measured using the squared Euclidean distance, forming a cost matrix $C \in \mathbb{R}^{N\times N}$ where:

$$C_{ij} = \|\hat{X}_t^i - \tilde{X}_t^j\|^2. \tag{2.55}$$

The entropy-regularized OT problem is then formulated as:

$$\pi^* = \arg \min_{\pi\in\Pi(\mu,\nu)} \sum_{i,j} \pi_{ij} C_{ij} + \varepsilon \sum_{i,j} \pi_{ij} \log \pi_{ij}, \tag{2.56}$$

where $\varepsilon > 0$ is the entropic regularization coefficient, and $\Pi(\mu, \nu)$ denotes the set of all joint distributions with marginals μ and ν. The entropy term ensures strict convexity and enables efficient computation via the *Sinkhorn algorithm* [43], which iteratively updates scaling vectors $u, v \in \mathbb{R}^N$ using:

$$K = \exp\left(-C/\varepsilon\right), \qquad u^{(l+1)} = \frac{\mu}{Kv^{(l)}}, \qquad v^{(l+1)} = \frac{\nu}{K^\top u^{(l+1)}}. \tag{2.57}$$

After convergence, the optimal transport plan is given by:

$$\pi^* = \mathrm{diag}(u) \cdot K \cdot \mathrm{diag}(v). \tag{2.58}$$

This transport plan is then used to interpolate the resampled particle states:

$$\tilde{X}_t^j = \sum_{i=1}^{N} \pi_{ij} \hat{X}_t^i, \tag{2.59}$$

where each $\tilde{X}_t^j$ is a convex combination of the original particles $\hat{X}_t^i$, weighted by the corresponding entries in π^*. After resampling, the $\tilde{X}_t^j$ are reindexed as $\hat{X}_t^j$ to continue the filtering procedure with consistent notation.

Unlike discrete resampling schemes that duplicate or discard particles based on a multinomial draw, this differentiable resampling step continuously redistributes the states according to π^*, preserving smooth gradients throughout the pipeline. This makes it compatible with back-propagation and essential for training the DPF using gradient-based optimization. For a complete and in-depth mathematical treatment, readers are referred to the work by Corenflos et al. [42].

2.2.3 Normalizing Flows

Most Differentiable Particle Filters employ a standard neural network to predict the next state [44]. However, a limitation of these networks is that they do not allow for density estimation. Normalizing Flow (NF) are a sequence of invertible and differentiable mappings that allow the transformation of a simple probability distribution (e.g., Gaussian) into a more complex one [45]. The transformations in NF are called flows and range from simpler ones, such as *Radial* and *Planar* flows, to more complex models like *RealNVP*, *NeuralSDE* and *Masked Autoregressive Flows* [46].

Let $X \in \mathbb{R}^d$ be a latent variable drawn from the target distribution $p_X(X)$. Let $Z \in \mathbb{R}^d$ be a random variable drawn from a known base distribution $p_Z(Z)$. The transformation $Z = \mathcal{F}(X) = \mathcal{F}_K \circ \cdots \circ \mathcal{F}_2 \circ \mathcal{F}_1(X)$ denotes a composition of K invertible functions, where each $\mathcal{F}_i$ represents a single flow in the sequence. The composition operator $\circ$ indicates that the functions are applied from right to left: $\mathcal{F}_1$ is applied first to X, followed by $\mathcal{F}_2$, and so on, until the final transformation $\mathcal{F}_K$ is applied. The inverse transformation is denoted $\mathcal{F}^{-1}(Z) = (\mathcal{F}_K \circ \cdots \circ \mathcal{F}_2 \circ \mathcal{F}_1)^{-1} = \mathcal{F}_1^{-1} \circ \mathcal{F}_2^{-1} \circ \cdots \circ \mathcal{F}_K^{-1}(Z)$ where the individual inverse flows are applied in reverse order. This inverse transformation enables the transformed random variable X to be obtained from the latent variable Z, allowing samples to be generated from the target distribution $p_X(X)$ as:

$$X = \mathcal{F}^{-1}(Z).$$

The change in volume is determined by the product of the magnitudes of the determinants of the Jacobian matrices across all transformations [45]. The resulting density transformation becomes:

$$p_X(X) = p_Z(\mathcal{F}(X)) \left| \det \frac{\partial \mathcal{F}(X)}{\partial X^T} \right|, \tag{2.60}$$

where $\frac{\partial \mathcal{F}(X)}{\partial X^T}$ is the Jacobian of $\mathcal{F}$ at X and can be rewritten as $J_{\mathcal{F}}(X)$. Using the change-of-variable formula, $|\det J_{\mathcal{F}}(X)|$ can be rewritten as $|\det J_{\mathcal{F}^{-1}}(Z)|^{-1}$. The final density transformation can be formulated as:

$$p_X(X) = p_Z(Z) \left| \det J_{\mathcal{F}^{-1}}(Z) \right|^{-1}. \tag{2.61}$$

Conditional Normalizing Flow

The Conditional Normalizing Flow (CNF) extends the Normalizing Flow (NF) by making the transformation dependent on some external variable Y. This conditioning variable, which is either a scalar or a vector, can represent various information, such as measurements (e.g., radar observations) or statistical parameters (e.g., the mean and standard deviation of the current particle set). The transformation is then defined as:

$$X = \mathcal{F}^{-1}(Z; Y).$$

As previous, the transformation of density can be rewritten as:

$$p_X(X \mid Y) = p_Z(\mathcal{F}(X; Y)) \left| \det J_{\mathcal{F}}(X; Y) \right|. \tag{2.62}$$

The term $|\det J_{\mathcal{F}}(X;Y)|$ can be rewritten as $|\det J_{\mathcal{F}^{-1}}(Z;Y)|^{-1}$ using the change-of-variable formula. The final density transformation can be formulated as:

$$p_X(X \mid Y) = p_Z(Z \mid Y)\,|\det J_{\mathcal{F}^{-1}}(Z;Y)|^{-1}. \tag{2.63}$$

RealNVP

Real-valued Non-Volume Preserving (RealNVP) transformations provide a flexible and invertible mapping for density estimation and generative modelling [47]. Unlike alternative flows such as Radial and Planar, RealNVP utilizes affine coupling layers to ensure transformations while maintaining invertibility. The steps for a single inverse RealNVP pass are outlined below and illustrated in Figure 2.5.

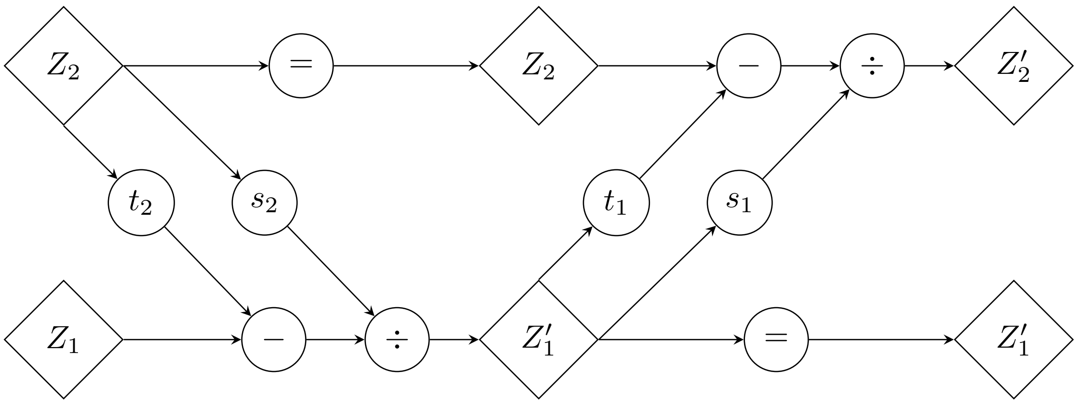


Figure 2.5: Flowchart illustrating a single inverse pass through a RealNVP affine coupling layer.

RealNVP Inverse Pass

1. The input variable Z is split into two parts: $Z = [Z_1, Z_2]$.

2. One part remains unchanged ($Z_2' = Z_2$), while the other is transformed using a differentiable scale s function and a differentiable shift t function:

$$Z_1' = (Z_1 - t_2(Z_2) \odot \exp(-s_2(Z_2)).$$

3. The transformation is then repeated with the roles of Z_1 and Z_2 reversed. This ensures full expressivity while maintaining invertibility. Each transformation step uses distinct scale and shift functions, resulting in four unique functions (two scale and two shift functions) used in a complete RealNVP mapping.

4. The change in probability density under the transformation is determined by the determinant of the Jacobian matrix. Since the Jacobian of the transformation is a diagonal matrix with:

$$J = \mathrm{diag}(\exp(-s(Z_i))),$$

the determinant simplifies to:

$$|\det J| = \sum_i \exp(-s(Z_i)).$$

This term reflects how the probability density changes under the transformation. A value above 1 expands space, reducing density, while a value below 1 compresses space, increasing density.

5. After applying the inverse transformations, the resulting variable Z is mapped back to the sample space as $X = Z$.

When RealNVP is applied as a flow in a Normalizing Flow model, the differentiable scale and shift functions are typically parametrized by some fully connected artificial neural networks. In addition, in Conditional Normalizing Flows (CNF), the conditional information Y is incorporated by concatenating it with the input Z_i before being passed into each scale and shift function, allowing the transformation to adapt to external information.

2.3 Statistical Context

This section presents the core statistical tools used for both training and evaluating the target tracking models developed in this thesis. It begins with the definition of the *Root Mean Squared Error* (RMSE), a standard metric for quantifying estimation accuracy. Next, the *Gaussian log-density* is outlined, which is essential for computing likelihoods in both Kalman and particle filters. The section then introduces the *Kullback-Leibler (KL) divergence*, which serves as a measure of discrepancy between probability distributions and is central to variational inference. Finally, the *Evidence Lower Bound* (ELBO) is described in the context of differentiable particle filters.

2.3.1 Root Mean Squared Error

The Root Mean Squared Error (RMSE) is a distance measure between a true sequence $X = [X_1, X_2, \ldots, X_n]$ and a predicted sequence $\hat{X} = [\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_n]$, where n is the number of discrete time steps [48]. The Euclidean RMSE is defined as:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \left\| X_i - \hat{X}_i \right\|_2^2}. \tag{2.64}$$

This metric computes the root of the mean squared Euclidean distances at each time step. The squaring penalizes large deviations, and a larger RMSE indicates a greater discrepancy between prediction and ground truth. To evaluate performance across multiple independent runs, let RMSE_i be the RMSE computed in run r, for $r = 1, \ldots, R$. The sample mean RMSE over R runs is:

$$\overline{\mathrm{RMSE}} = \frac{1}{R}\sum_{r=1}^{R} \mathrm{RMSE}_r, \tag{2.65}$$

and the sample standard deviation of the RMSE is:

$$\sigma_{\text{RMSE}} = \sqrt{\frac{1}{R-1}\sum_{r=1}^{R}\Big(\text{RMSE}_r - \overline{\text{RMSE}}\Big)^2}. \tag{2.66}$$

These statistics provide both a central measure of accuracy and its variability across runs.

2.3.2 Density Estimation of a Gaussian Distribution

As described in the standard *multivariate Gaussian* formulation [49], the probability density function (PDF) for a multivariate normal density in d dimensions can be stated as:

$$p(Y) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(Y-\mu)^T\Sigma^{-1}(Y-\mu)\right), \tag{2.67}$$

where μ is the mean, or location, of the distribution and Σ is the covariance matrix, or scale, of the distribution. Taking the logarithm and using logarithm properties will give the general log-density function for a Gaussian-distributed vector Y of length d,

$$\log p(Y) = -\frac{d}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(Y-\mu)^T\Sigma^{-1}(Y-\mu). \tag{2.68}$$

If one assumes no correlations between variables, meaning Σ is diagonal, one can reduce the equation to:

$$p(Y) = \frac{1}{(2\pi)^{d/2}\prod_{i=1}^{d}\sigma_i} \exp\left(-\frac{1}{2}\sum_{i=1}^{d}\frac{(Y_i-\mu_i)^2}{\sigma_i^2}\right), \tag{2.69}$$

where μ_i and σ_i is the mean and standard deviation in dimension i. Taking the logarithm and using logarithmic properties of the reduced equation will give the general log-density function for an independent Gaussian-distributed vector Y of length d,

$$\log p(Y) = -\frac{d}{2}\log(2\pi) - \sum_{i=1}^{d}\log\sigma_i - \frac{1}{2}\sum_{i=1}^{d}\frac{(Y_i-\mu_i)^2}{\sigma_i}. \tag{2.70}$$

In the context of a known measurement model defined over (r, θ, v_r), the parameters μ_i and σ_i correspond to the mean and standard deviations of the noise in each range, bearing, and radial velocity component. The vector Y in this case represents the residual between the measurement and the predicted measurement, obtained by transforming the state estimate $\hat{X}_t^i$ into measurement space using the non-linear measurement function $h(\cdot)$ defined in Eq. (2.7).

2.3.3 Kullback-Leibler Divergence

The *Kullback-Leibler* (KL) divergence is a measure from information theory that quantifies the discrepancy between two probability distributions. Given two continuous probability density functions $p(x)$ and $q(x)$, where p is typically the true distribution and q an approximation, the KL divergence from q to p is defined as [50]:

$$\mathrm{KL}(p\|q) = \int p(x) \log \left(\frac{p(x)}{q(x)} \right) dx. \tag{2.71}$$

For discrete probability distributions, where $p(i)$ and $q(i)$ denote the probabilities of outcome i, the KL divergence is given by the sum:

$$\mathrm{KL}(p\|q) = \sum_i p(i) \log \left(\frac{p(i)}{q(i)} \right). \tag{2.72}$$

The KL divergence is always non-negative and equals zero if and only if $p = q$ everywhere in the domain of definition. It is important to note that KL divergence is not symmetric; that is, $\mathrm{KL}(p\|q) \neq \mathrm{KL}(q\|p)$ in general.

2.3.4 Evidence Lower Bound in Particle Filters

In the context of particle filtering, the ELBO objective can be used to train components such as proposal distributions and transition models. By maximizing a variational lower bound on the marginal log-likelihood of observations, the model can be optimized end-to-end. A differentiable particle filter constructs a proposal distribution $q_\psi(X_{1:T} \mid Y_{1:T})$ and evaluates the ELBO:

$$\log p(Y_{1:T}) \geq \mathbb{E}_{q\psi} \left[\sum_{t=1}^{T} \log \frac{p(Y_t \mid X_t; \theta), p(X_t \mid X_{t-1}; \varphi)}{q(X_t \mid X_{t-1}, Y_t; \psi)} \right], \tag{2.73}$$

where $p(Y_t \mid X_t; \theta)$ is the observation likelihood, $p(X_t \mid X_{t-1}; \varphi)$ is the learned prior dynamics, and $q(X_t \mid X_{t-1}, Y_t; \psi)$ is the learned proposal distribution. The inner term,

$$\log p(Y_t \mid X_t; \theta) + \log p(X_t \mid X_{t-1}; \varphi) - \log q(X_t \mid X_{t-1}, Y_t; \psi), \tag{2.74}$$

can be interpreted as the log importance weight in a sequential Monte Carlo setting. This makes the ELBO particularly natural for differentiable particle filters (DPFs), which approximate the posterior using a weighted particle set.

Block-wise ELBO Computation
In practice, a block-wise ELBO is computed over short segments of the trajectory to avoid gradient instability. The KL divergence term defined in Eq. (2.72) between the learned proposal and prior model is for particle i estimated using:

$$\mathrm{KL}_t^i = \log q(\hat{X}_t^i \mid \hat{X}_{t-1}^i, Y_t; \psi) - \log p(\hat{X}_t^i \mid \hat{X}_{t-1}^i; \varphi), \tag{2.75}$$

and the expected log-likelihood under the particle distribution:

$$\mathbb{E}_q \left[\log p(Y_t \mid \hat{X}_t; \theta) \right] \approx \sum_{i=1}^{N} w_t^i \log p(Y_t \mid \hat{X}_t^i; \theta). \tag{2.76}$$

Combining Eq. (2.75) and (2.76) over a block of length K yields:

$$\mathrm{ELBO}_{\mathrm{block}} = \sum_{t=t_0}^{t_0+K-1} \sum_{i=1}^{N} w_t^i \left[\log p(Y_t \mid \hat{X}_t^i; \theta) - \mathrm{KL}_t^i \right], \tag{2.77}$$

which is averaged over multiple such blocks to compute the final training loss. This ELBO framework aligns with recent works on differentiable particle filters [16], [41] and [42].

3
Methodology

This chapter outlines the methodology used to achieve the objectives defined in Section 1.3 (Aim). It begins with an overview of the information collection process for both the state-space modelling and machine learning contexts covered in Chapter 2. Next, the target equations and data generation process are described, followed by the selection and development of the filtering techniques. The metrics used to evaluate the filters are then presented. Thereafter, the experimental setup is described. The chapter concludes with insights into the software and hardware used in development and evaluation.

3.1 Information Collection

The project began with an extensive information-gathering phase to develop a detailed understanding of the thesis objectives. Given its intersection between state-space modelling and machine learning, research from both fields was essential.

To deepen the understanding of state-space modelling and target tracking, the Target Tracking Team at Saab was consulted. Additionally, literature on interacting multiple model- and particle filters was reviewed to complement our knowledge. In parallel, relevant machine learning models were studied, particularly normalizing flows, due to their applicability to our problem.

A thorough analysis of the NF-DPF implementation by Chen et al., was also conducted [16]. Since this research is motivated by the promising results of their model, it was crucial to gain a deep understanding of their methodology and rationale. This involved dissecting their entire implementation, including experiments, datasets, and code.

3.2 Data Generation

All experimental data was generated by approximating solutions to Stochastic Differential Equations (SDEs), modelling missile dynamics, followed by the creation of synthetic measurements. The SDEs are defined as the normal and tangential acceleration terms in the curvilinear-motion model in Eqs. (2.8)-(2.11). This section presents the specific acceleration terms used in the curvilinear motion model SDE to represent different trajectory types, along with the corresponding parameters, and initial conditions. It also details the method for generating the synthetic radar measurements. The following three sections detail the models for the constant velocity-, the turning- and the accelerating cases.

This pipeline produces a dataset with nine dimensions per measurement time, consisting of the true state $[x, y, v_x, v_y, a_n, a_t]$ and the measurement $[r, \theta, v_r]$. To reiterate, the goal of the filtering problem is to approximate the true 6-dimensional state from the 3-dimensional measurement, given the observations available so far.

3.2.1 Constant Velocity Target

The first set of trajectories were modelled using acceleration terms that assumes a constant, yet noisy, velocity of the target. This was achieved by assuming that the thrust and drag of the target cancel each other out in every time step t.

In reality, drag would still be present, and maintaining a constant velocity would require some control input to counteract it. The tangential acceleration a_t is defined as (where the subscript t refers to the tangential direction, not time),

$$a_t(t) = \sigma_t \, dW_t, \tag{3.1}$$

where dW_t represents an increment of a Wiener process (standard Brownian motion), modelling continuous-time Gaussian noise. The acceleration is therefore a stochastic process with zero mean and standard deviation proportional to $\sigma_t\sqrt{dt}$, where dt is the time step. This formulation captures random fluctuations in tangential acceleration over time. For the constant velocity trajectory, the normal acceleration is given by:

$$a_n(t) = \tau \sin(\omega t) + \sigma_n dW_n(t). \tag{3.2}$$

This represents periodic oscillations, some kind of turbulence, with frequency w and amplitude τ, with a additional increment of a Wiener process, modelling continuous-time Gaussian noise with a mean of 0 and a standard deviation of $\sigma_n\sqrt{dt}$.

3.2.2 Turning Target

The turning target acceleration terms is similar to the constant velocity definition, but includes an additional term $u_n(t)$ in the normal acceleration. The additional term represents an external control input applied to the target, accounting for manoeuvring effect. The tangential acceleration is once more defined as:

$$a_t(t) = \sigma_t dW_t, \qquad (3.1)$$

and the normal acceleration is defined as,

$$a_n(t) = \tau \sin(\omega t) + \sigma_n dW_n(t) + u_n(t), \tag{3.3}$$

where $u_n(t)$ is 0 for all but some intervals starting at t_{turn} with duration T_{turn}, where it is set to the magnitude u_{mag}. This magnitude is uniformly drawn with a random sign to allow both left and right turns. This additional normal acceleration is non-zero during 0, 1, or 2 intervals per trajectory, defined by the probabilities p^1_{turn} and p^2_{turn}, to prevent the model from learning a fixed pattern.

3.2.3 Accelerating Target

The last set of trajectories is modelled using acceleration terms that accounts for thrust, opposing drag, and stochastic noise. In the accelerating case, the tangential acceleration is given by:

$$a_t(t) = \frac{c\dot{m}}{m(t)} - \frac{\rho v_t^2 A C_D}{2m(t)} + \sigma_t dW_t(t), \tag{3.4}$$

where each term reflects a force acting on the target. The first term represents *thrust*, which depends on the exhaust velocity c, the burn rate $\dot{m}$, and the current mass $m(t)$. The second term models *drag*, which is proportional to the square of the tangential velocity v_t, the reference area A, the drag coefficient C_D, and the fluid density ρ, all scaled by the inverse of twice the current mass, $1/(2m(t))$. Finally, the third term captures *stochastic noise*, modelled as a Wiener process dW_t with zero mean and standard deviation $\sigma_t\sqrt{dt}$. The mass of the system evolves as:

$$m(t) = m_0 - \dot{m}t, \tag{3.5}$$

where m_0 is the initial mass and $\dot{m}$ is the constant burn rate. The normal acceleration is once more given by:

$$a_n(t) = \tau \sin(\omega t) + \sigma_n dW_n. \tag{3.2}$$

3.2.4 Trajectory Generation

A trajectory is obtained by selecting the acceleration terms used in the curvilinear-motion model SDE, specifying a parameter set and initial conditions, and approximating its solution using a numerical method. The different acceleration terms have already been presented; this section details the remaining aspects.

Choice of Parameters

The parameters remained constant across all trajectory types and represent properties of the medium and target. The need for precise values were low since most parameters do not alter the trajectory's characteristics but rather affect the magnitudes of velocities and accelerations. The process noise in the normal and tangential direction, defined by the scale factors σ_n and σ_t, was defined as,

$$\sigma_t = \sigma_n \quad \Longrightarrow \quad \sigma_x = \sigma_y = \sigma_n = \sigma_t \equiv \sigma,$$

which reduced the covariance of the noise to isotropic form. This streamlined the implementation of the NF-DPF. All parameters were chosen to represent a highly manoeuvring hypersonic missile. The exact selected values are presented in Appendix A.

Choice of Initial Conditions
The solution to any SDE is undefined without a set of initial conditions. For the conducted experiments, the initial conditions were uniformly sampled within predefined ranges for x_0, y_0, v_0, and ϕ_0, where $x_0 = x(0)$, and similarly for the other variables. This ensures that the trajectory set does not develop a dominant class that the filter could learn naively. Instead, the filter must generalize across a diverse range of trajectories, each starting from a unique position. The ranges were varied for the different experiments conducted. For example, in the case of non-accelerating targets a higher initial velocity v_0 was used. the exact initial conditions used are presented in the upcoming experiments section.

Approximation of SDE
The respective SDEs were solved using the Euler-Maruyama method, as defined in Eq. (2.12), with a sufficiently small time step of $dt = 0.01$ to ensure numerical accuracy. The trajectories were simulated for 30 seconds, $T = 30$. The true states corresponding to the 30 time steps at which measurements were available were saved, the states at all other time steps were discarded.

3.2.5 Measurement Generation

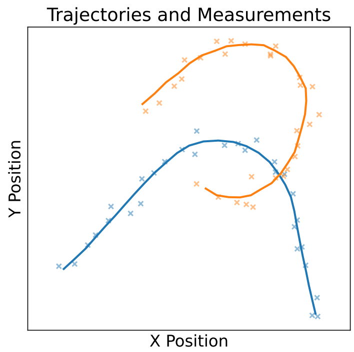
To generate synthetic measurement data, the true states were first collected at discrete time steps, corresponding to the assumed sensor sampling rate of $\Delta t_{\text{radar}} = 1/\text{s}$. Thereafter, the measurement function h in Eq. (2.7) was applied to transform the true states into measurement space and finally Gaussian noise, sampled from $\mathcal{N}(0, R)$, where R is the radar covariance defined in Eq. (2.13), was added to each measurement. Examples of the three sets of trajectories along with the generated measurements are presented in Figure 3.1.

3.3 Model Selection and Implementation

In the upcoming sections the three implemented filtering techniques, NF-DPF, IMM and PF, are introduced. In addition to detailing the implementation, the motivation behind each filter is stated. The chapter begins with a brief overview of the development process that guided the design. This is followed by a more detailed description of the development of each filter.

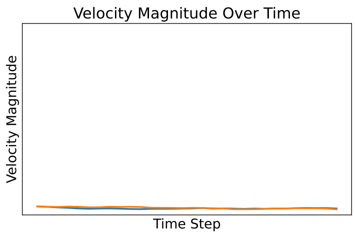
The development of all filters followed the same overarching principles. Initially, implementations which addresses similar problems techniques were researched. As introduced in Section 3.1, the NF-DPF proposed by Chen et al., heavily influenced the implementation of the NF-DPF model employed in this thesis [16]. In addition to the NF-DPF, the IMM-filter was developed as a benchmark model. Lastly, the bootstrap particle filter was constructed as an additional benchmark model by re-

(a) CV targets.

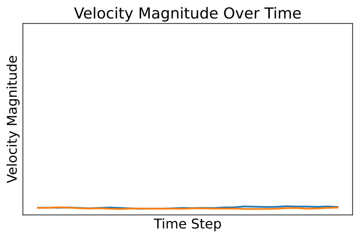


(b) Turning targets.

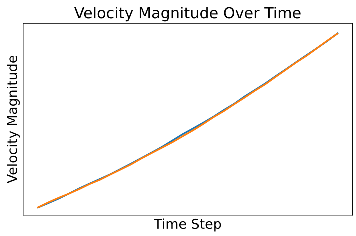
(c) Accelerating targets.



(d) CV targets, velocity magnitude over time.



(e) Turning targets, velocity magnitude over time.



(f) Accelerating targets, velocity magnitude over time.

Figure 3.1: Examples of target trajectories and their corresponding measurements. (a) displays two constant velocity targets generated by the SDE described in Section 3.2.1, along with their noisy radar observations. (b) illustrates two turning targets and their measurements, generated using the SDE in Section 3.2.2. (c) presents two accelerating targets and their measurements based on the SDE in Section 3.2.3. (d)-(f) illustrates the velocity magnitude, as reflected in the measurement spacing in (a)-(c).

moving the dynamic and proposal models from the NF-DPF. This PF served as a baseline, and was used to evaluate the effects of the dynamic and proposal models in the NF-DPF.

3.3.1 Normalizing Flow - Differentiable Particle Filter

As introduced in Section 2.2.2, a DPF operates by first *initializing particles*, (a set of hypothesis of the current true state). The particles are then propagated through consecutive state-transition and proposal models. In this NF-DPF implementation, the particles are moved through 3 models. This is visualized in Figure 3.2. The first model is a naive *Motion Update Model* which moves the particles based on their current state and spreads them based on some noise. This is followed by the normalizing flow models *Dynamic NF* and *Proposal NF*, in which the latter gets access to the current measurement. Thereafter the *Measurement Model* is used to calculate a likelihood for each particle. The particle weights are updated using the likelihood and the Jacobian determinants from both NFs. The final state prediction for the time step $\hat{X}_t$ is made as the weighted average of all particles. In the beginning of each time step the particles are potentially *resampled*, i.e. moved to high-probability areas. The Dynamic NF and Proposal NF are trained by evaluating a *loss function* and updating the model parameters. The following paragraphs outlines the design processes of some components of the model and justifies some of the final design choices. Chapter 4 provides an in-depth description of the developed NF-DPF model.

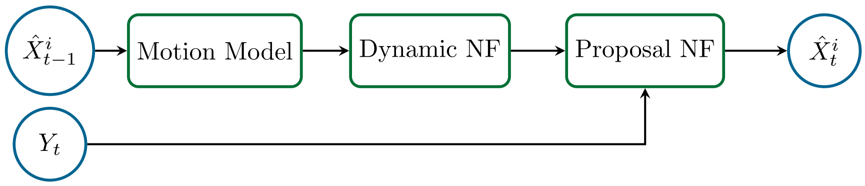


Figure 3.2: Flowchart of the three models that the particles $\hat{X}$ in the NF-DPF are propagated through in each time step t.

Particle Initialization

The implementation of the NF-DPF for this thesis was based on the NF-DPF implementation introduced by Chen et al. [16]. The model in the paper was (among a few other setups) used to solve a localization problem. Since this differs from the problem of target tracking, where the current velocity is unknown, substantial modifications had to be made. The data generation processes was changed to the setup described in the previous section. The particle were initialized as noisy variations of an initial state estimate. This initial estimate consists of approximating the position, velocity and acceleration. The initial positional estimate is simply the first radar measurement, converted from polar to Cartesian coordinates. The initial velocity is estimated as the change in position between the first and second measurement. The acceleration is assumed to be zero because estimating it from the first $n \geq 3$ measurements was inefficient. A small n provided unreliable estimates while a larger n delayed the first state estimation, so assuming zero acceleration enables estimation from the second measurement onward. This initial state estimate was then used as the mean around which the particles were uniformly distributed. This resulted in the initialized particle set.

Motion Update Model

The Motion Update Model works by first propagating each particle deterministically based on its current state, followed by the injection of zero-mean Gaussian noise. The full model is detailed in Section 4.2.2. This additive noise serves two primary purposes: it captures the stochastic accelerations arising from the Wiener-process terms in the target SDEs and encourages exploration to mitigate particle degeneracy and preserve diversity. In the ground-truth trajectories, independent Gaussian noise was added to the normal and tangential acceleration components, a_n and a_t. These perturbations propagate through the curvilinear equations of motion, Eqs. (2.8)–(2.11), affecting all state components $(x, y, v_x, v_y, a_n, a_t)$. To reflect this, the motion model noise must induce similar variations across the state.

To obtain the Motion Update noise, zero-mean Gaussian acceleration perturbations were injected at each integration step and propagated through the kinematic equations. Single integration yields velocity noise that grows as $\sqrt{M}\,dt$, where $M = \Delta t_{\text{radar}}/dt = 100$, while double integration provides the closed-form expression for the position noise [21]. In the data generation, each integration step of length $dt = 0.01$ adds an independent acceleration term $a_t^{\text{noise}} \sim \mathcal{N}(0, \sigma\sqrt{dt})$. Accumulating the noise over one measurement interval $\Delta t_{\text{radar}} = 1$ gives the equivalent process-noise standard deviations for the Motion Update Model:

$$\begin{aligned}
\sigma_{\text{acc}} &= \sigma\sqrt{dt}, \\
\sigma_{\text{vel}} &= \sigma\, dt\,\sqrt{M}, \\
\sigma_{\text{pos}} &= \sigma\, dt^{3/2}\sqrt{\frac{M(M+1)(2M+1)}{6}}.
\end{aligned}$$

The standard deviation of the process noise at the small, but finite integration time step dt was set to $\sigma = 20$, which resulted in the following:

$$(\sigma_{\text{pos}},\ \sigma_{\text{vel}},\ \sigma_{\text{acc}}) = (3.68,\ 2.0,\ 2.0).$$

The values above account only for process uncertainty and were found to produce insufficient spread in practice, as shown by a series of Monte-Carlo experiments, which models uncertainty by repeated experiments [51]. Instead, the noise standard deviations were determined through a hyper-parameter tuning procedure using *Grid Search* [52]. Each standard deviation $\sigma_{pos}, \sigma_{vel}, \sigma_n, \sigma_t$ was defined by:

$$\sigma = \alpha + \beta\|V\|. \tag{3.6}$$

This formulation balances a fixed noise level α with a velocity-dependent term $\beta\|V\|$, where $\|V\|$ is the norm of the velocity components in the state. This dependence ensures the particles are spread out more, when the target moves faster. The grid search was performed using the PF with the same number of particles as in the NF-DPF, with the objective of identifying motion update parameters that produce accurate and sufficiently diverse particle distributions for both filters. Since the normalizing flows in the NF-DPF are used to refine these distributions, selecting optimal parameters using the PF, without any learned components, provides a strong baseline. The search proceeded sequentially: first σ_{pos} was optimized, followed by σ_{vel} using the current optimal values of α_{pos} and β_{pos}. Finally σ_n and σ_t were tuned in the same manner. To further refine the values, the entire process was repeated using the best values found in the first iteration.

These velocity-dependent adaptive standard deviations were found to significantly improve the performance of the PF. However, in the case of non-accelerating targets, introducing velocity-dependent noise into the NF-DPF degraded performance. The added uncertainty from the β term introduced an additional challenge for the normalizing flows to learn, making generalization more difficult during training. As a result, for non-accelerating scenarios, the β term was set to zero, and the α term was tuned empirically, with the values found in the grid search as baseline.

For accelerating targets, the α and β values obtained from the grid search were used. In the accelerating setting, the velocity-dependent noise scaling aligned with the increased complexity and introduced more effective particle spread. The full list of noise parameters used for the NF-DPF can be found in Appendix D.2.

Proposal Logic

The method of passing observations into the Proposal NF was revisited multiple times during the development of the NF-DPF. Initially, the observations were provided to the Proposal NF as raw measurements in polar coordinates, i.e., $[r, \theta, v_r]$. However, it became apparent that the Proposal NF did not leverage the information contained in these measurements.

To improve this, the observations were converted to Cartesian coordinates before passed to the Proposal NF. This transformation was aimed at more directly relating the state with the observation, thus enabling a clearer interpretation (although the model should ideally be able to learn this conversion itself). However, even after converting the measurements to Cartesian coordinates, the Proposal NF still did not effectively utilize the information contained in the observations. This shortcoming indicated that simply providing a direct mapping from polar to Cartesian coordinates was insufficient for the Proposal NF to capture the underlying dynamics. As a result, an alternative approach, where the input to the Proposal NF was redefined as the distance between the particles and the current observation, was explored. This representation serves to simplify the corrective process by offering a clear, discrepancy vector, thereby encouraging the network to align the particles more closely with the measurement. Even though this method improves the correction process, it is acknowledged that the optimal method to incorporate the observations into the Proposal NF may not yet have been found.

Resampling Logic

Since the NF-DPF was trained end-to-end, all intermediate steps of the filter had to be fully differentiable. Therefore the fully differentiable resampling scheme Entropy-Regularized Optimal Transport resampling, as presented in Section 2.2.2 was implemented. Entropy regularization introduces a smoothing effect on the transport plan by adding an entropy term to the OT problem. This promotes numerical stability and ensures smoother gradients during back-propagation. In contrast to the OT-resampling implementation by Chen et al., the resampler is formulated without any *clamping operations*. Clamping refers to restricting the values of gradients or other quantities within a fixed range [37]. While this can improve numerical stability, it also suppresses finer variations in the gradients, potentially hindering learning. For this reason, clamping was omitted in this implementation.

In general, particle filters aim to minimize the number of resampling steps, as infrequent resampling indicates well-tuned state-transition and measurement models that successfully avoid weight degeneracy [29]. However, with a Motion Update Model such as the one used in this implementation, which deliberately spreads out the particles, weight degeneracy becomes inevitable. This is because many particles are intentionally propagated into low-probability regions, on the off-chance that one of them lands near the next true state. Moreover, unlike in many standard particle filter setups, the state of the particles in our implementation does not fully capture the trajectory over future time steps, due to the inherent stochasticity of the targets. In scenarios such as the target tracking presented in this thesis, a greater reliance on exploration becomes essential.

A major drawback of the OT-resampler is its high memory usage. Since all intermediate steps, especially the iterations within the Sinkhorn loop, must be temporarily stored to compute gradients during back-propagation, a significant amount of memory is required. This limits the maximum number of particles n, as the memory usage grows with $\mathcal{O}(n^2)$. Other differentiable resampling methods, such as soft-resampling [53], which introduces a trade-off between a uniform weight distribution and the previous distribution, help alleviate the memory issue. However, the OT-resampler still produced more promising results and was therefore used in the implementation.

Loss Function

The loss functions used in the training of DPFs are generally grouped into two categories, likelihood-based and task-specific. The likelihood-based loss function utilized in the training of the NF-DPF was the Evidence Lower Bound (ELBO) loss in, which was derived for the general DPF and tasked to learn the dynamic and proposal distributions [54], [55]. The ELBO loss is defined as the negative of the block ELBO from Eq. (2.77), meaning that minimizing the ELBO loss is equivalent to maximizing the block ELBO. Task-specific losses include the Root Mean Square Error (RMSE), a distance measure between the predicted and true state as defined in Eq. (2.64). Separate RMSE-losses were defined for position, velocity and acceleration. The complete loss function is for the general case, as suggested in similar setups, some combination of task-specific RMSE and likelihood-based ELBO-loss [42], [56],

$$\mathcal{L}(\varphi, \psi) = \alpha_{\text{pos}}\, \mathcal{L}_{\text{RMSE}}^{\text{pos}} + \alpha_{\text{vel}}\, \mathcal{L}_{\text{RMSE}}^{\text{vel}} + \alpha_{\text{acc}}\, \mathcal{L}_{\text{RMSE}}^{\text{acc}} + \alpha_{\text{ELBO}}\, \mathcal{L}_{\text{ELBO}}, \tag{3.7}$$

where α are the weighting factors of the respective terms. All terms depend on the parameter sets φ and ψ. The factor are constructed to satisfy the property,

$$\alpha_{\text{pos}} + \alpha_{\text{vel}} + \alpha_{\text{acc}} + \alpha_{\text{ELBO}} = 1.$$

The values of α_i were inspired both by the implementation in Chen et al.[16], and by empirical exploration. In designing these factors, it is essential that no single objective dominates the optimization process. If one term were to dominate, for instance by being minimized in isolation while causing the other objectives to worsen, the overall performance would be compromised. Achieving the right balance among the objectives is non-trivial because the loss terms operate on different scales. For example, ELBO values for accelerating trajectories were discovered to reach the order of hundreds, while acceleration RMSE values remained in the tens. Therefore, the final weighting factors were determined empirically by considering the relative importance of each objective and balancing their contributions.

The inclusion of all terms in the loss function were justified by an ablation study in which one term was systematically excluded at a time and the effects on convergence were examined. The results indicated that the combination of all terms yielded the best convergence. Further experiments explored dynamically weighting the loss terms by making the factors α_i a function of the epoch number. The hypothesis was that some terms or combinations of terms could guide the training toward a minimum, while other might be better suited for refining the parameters once the filter was near the minimum. Setups explored included the initial use of solely task-specific RMSE-losses with a later transition to incorporating the ELBO term, and a linear increase of the ELBO factor with the epoch number. Neither approach improved convergence. It was also observed that the combination of metrics improved the minimization of the ELBO component compared to solely training with the ELBO term (that is, with $\alpha_{\text{ELBO}} = 1$).

3.3.2 Particle Filter

The *Bootstrap Particle Filter* (see Section 2.1.8) was developed as a benchmark model to the NF-DPF, and used to study the effects of the normalizing flow. The PF is a stripped-down version of the NF-DPF, created by removing both the dynamic and Proposal NFs from the filter. The result was a PF that propagates particles solely using the Motion Update Model. This makes the filter even more reliant on exploration, that is, the deliberate spreading out of particles, and consequently, it requires a larger number of particles to maintain performance. The complete PF is not presented, as it constitutes a simplified version of the NF-DPF, however, its structure is illustrated in Figure 3.3.

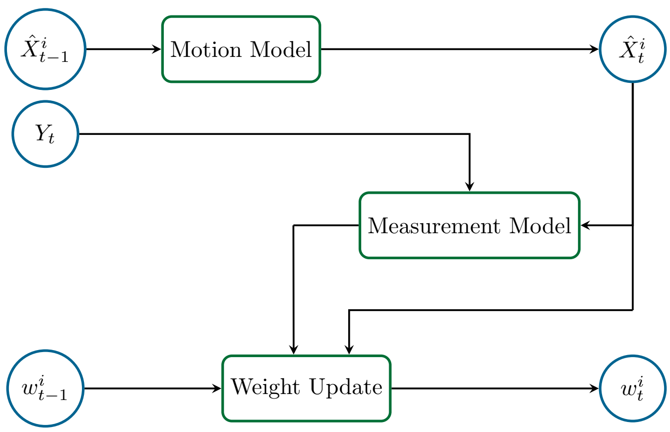


Figure 3.3: Flowchart of the implemented Particle Filter, including the Motion Update Model, the Measurement Model and the Weight Update.

3.3.3 Interacting Multiple Model Filter

As the IMM filter is a widely used method for target tracking [26], it serves as a suitable benchmark model for evaluating the NF-DPF. The IMM, implemented in this thesis, was constructed, according to Section 2.1.7, using a set of EKF filters, which are described in Section 2.1.5. Filter and parameter choices are problem-specific and were therefore tuned for the three experiments considered in this thesis.

Choice of Filters

Due to the non-linearity of the measurement function h in Eq. (2.7), standard Kalman Filters are inapplicable, as they require both the state-transition and measurement models to be linear. Furthermore, UKFs showed no significant improvement over EKFs, suggesting that the linearized measurement model was sufficient to capture the relevant dynamics. This experiment will not be presented. However, it is reasonable to expect, that for problems with stronger non-linearities, UKFs would be necessary. As a result, five EKFs were used in the IMM filter.

Filter Variations

The transition model F_t was implemented both as a Constant Velocity (CV) and a Constant Acceleration (CA) filter. The first set of filters assumed that the target travels at a constant velocity. In this case, the transition model F_t^{CV} disregards any influence of acceleration to the position and velocity, and was therefore defined as :

$$F_t^{\mathrm{CV}} = \begin{bmatrix} 1 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 1 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{3.8}$$

The other set of EKFs, assumed Constant Acceleration (CA) of the target [57]. When taking acceleration into consideration, the transition model F_t^{CA} was extended, and defined as:

$$F_t^{\mathrm{CA}} = \begin{bmatrix} 1 & 0 & \Delta t & 0 & 0.5\Delta t^2 & 0 \\ 0 & 1 & 0 & \Delta t & 0 & 0.5\Delta t^2 \\ 0 & 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.9}$$

No Constant Turn (CT) filters were implemented. CT filters typically model a constant rate of change in the heading angle ϕ, suitable for scenarios with steady circular motion. In this work, turning targets are described using the normal acceleration term in Eq. (3.3). In the curvilinear motion model defined by Eqs. (2.8)–(2.11), with $v(t)$ approximately constant, the normal acceleration $a_n(t)$ directly induces a constant change in the heading angle ϕ. This formulation captures the same effect that CT filters aim to represent. As a result, the CA filters already fully describe the turning dynamics, and implementing separate CT filters was not justified.

In addition to the transition matrices, the IMM filter also requires a process noise covariance matrix Q_i for each filter i, as well as a shared measurement noise covariance matrix R. This was implemented using multiple CV-EKFs and CA-EKFs, each assigned a distinct process noise covariance matrix to represent different levels of uncertainty. By combining these filters, the IMM could effectively capture a range of target dynamics, even when no single transition filter described the motion perfectly.

Filter Interaction
After selecting the filters and their corresponding covariance matrices, the transition probability matrix π^{ij} and the choice between the switch or combine method for the IMM were determined using the hyper-parameter optimization method grid search. Since the number of transition probabilities increases quadratically with the number of filters, simplifying the structure of this matrix is essential for practical implementation. As a result, the diagonal elements π^{ii} were assumed to be equal and treated as a hyper parameter, while the remaining elements were distributed uniformly. For each combination of interaction method and transition probability matrix, the IMM was evaluated on a set of trajectories to identify the configuration that yielded the lowest positional RMSE. The combine method, where the predicted state is an interpolation of multiple filters was chosen.

Final Model Overview
The architecture of the developed IMM-filter is presented in detail in Appendix B. This includes the utilized filters, transition probabilities, initialization of covariances and chosen switching/blending technique.

3.4 Evaluation Metrics

The following section covers the metrics used to evaluate the filters during the experiments. Initially some definitions are introduced. The true state, at time step t, is in the general case defined as $X_t = [x_t, y_t, v_{x,t}, v_{y,t}, a_{n,t}, a_{t,t}]$, where the elements in brackets are the true state vectors in the respective dimensions. The predicted state vector (with particle superscript i for the PF and NF-DPF), at time t is in the general case similarly defined as $\hat{X}_t^i = [\hat{x}_t^i, \hat{y}_t^i, \hat{v}_{x,t}^i, \hat{v}_{y,t}^i, \hat{a}_{n,t}^i, \hat{a}_{t,t}^i]$, where the first subscript of $a_{t,t}^{(i)}$ denotes *tangential* acceleration and the second subscript denotes time step t. For the particle-based PF and NF-DPF, the final state prediction P_t, at time step t, is given by the weighted average over the particles defined in Eq. (2.44). The final state prediction is therefore also 6-dimensional, at each time step. For the IMM filter, the final state prediction is just the current state $\hat{X}_t$.

The evaluation of the filtering techniques was performed by evaluating how close, the predicted state $\hat{X}_t$ generated by each filter, was to the true state X_t, for every time step t, and trajectory in the test dataset. This was achieved by some soon to be defined accuracy measure, computed over all experiments. Lastly, the computational viability of the three filters was examined. The evaluation metrics are defined explicitly in the following sections.

3.4.1 Accuracy Measure

The accuracy measure used to evaluate the difference between the final predicted state vector $\hat{X}$ and the true state vector X was the Euclidean Root Mean Square Error (RMSE), as defined in Eq. (2.64). The mean of the RMSE over the entire test data set was calculated according to Eq. (2.65). The RMSE was computed independently for position, velocity, and acceleration, effectively providing three distinct performance metrics. Additionally, the standard deviation of the RMSE across multiple runs was also calculated according to Eq. (2.66), to quantify the consistency of the results. For all models, the first 3 time steps were disregarded in the RMSE calculation, since filters, and corresponding state estimates, were assumed to be still converging.

It is worth noting that calculating the *Wasserstein distance* between the predicted posterior distribution $p(\hat{X} \mid Y)$ and the true posterior distribution $p(X \mid Y)$ would have be advantageous, as it directly quantifies the quality of the posterior approximation [58]. However, such a comparison is generally intractable and addressing this difficulty constitutes the core motivation of this project. While the distribution $p(X)$ is tractable, the distribution of interest is the conditional posterior $p(X \mid Y)$, which cannot be explicitly obtained [59]. Therefore, evaluating the models using RMSE was both a practical and justified choice, given the scope and constraints of this thesis.

3.4.2 Computational Viability Measure

The evaluation concluded by assessing the computational viability of the three filtering techniques. Since all filters were implemented as online methods, ensuring real-time operation on a reasonable computational budget was essential. This required both a short execution time and a manageable memory footprint. Execution time was defined as the time from receiving a new radar measurement to generating a prediction for a single target. It was considered acceptable if significantly faster than the 1-second interval between radar measurements in this setup. Additionally, the average time to execute an entire iteration, corresponding to one time step, was recorded for the filtering techniques.

The DPF implementation introduces additional latency: after receiving a measurement, particles are first (optionally) resampled and then propagated through all unconditional models before the measurement update is performed. To enable true online operation, it is reasonable to pre-compute all stages up to the conditional proposal normalizing flow, allowing it to trigger immediately upon measurement arrival. This reordering would reduce the delay and improved the model's suitability for real-time use.

When profiling the NF-DPF, no gradient information was saved, since the gradients are solely used for training. All computational metrics were recorded under settings tuned to minimize RMSE in position, velocity, and acceleration. However, in practice, it might have been beneficial to reduce the number of particles, allowing for slightly increased RMSE in exchange for lower runtime and memory demands.

3.5 Experiments

After the data had been generated, the filters defined, and the evaluation criteria formulated, a number of experiments was set up and performed. The experiments were formulated to evaluate the models performance, on the three datasets, which corresponds to evaluating the models ability to accurately track targets moving in a number of distinct ways. The first case was tracking constant velocity targets, the second, highly manoeuvring targets, with unexpected turns of random duration and magnitude, and the third, hyper-sonic and accelerating missiles. Before detailing each experiment, the parameters and settings that remained constant across cases is outlined.

Most hyper-parameters and filter architectures remained constant across all experiments, as to maintain uniformity and allow for efficient evaluation. There are a few exceptions, were filters have been tuned and configured to the current experiment, like varying the state dimension between them. This will be detailed further in the subsequent sections. The IMM filter maintained the same set of filters and configurations across all experiments, using a 6-dimensional state for both accelerating and non-accelerating data. This is reasonable, as the IMM combines CV and CA filters, which often ensures that at least one filter effectively captures the underlying dynamics. For all experiments, the initial conditions were varied between data samples to encourage the NF-models to learn a distribution that reflects real-world variability, where strict assumptions cannot be made.

For the three experiments, data was generated according to the descriptions in Sections 3.2.4 and 3.2.5, using the SDEs defined for the constant velocity, turning and accelerating cases. The ranges from which the initial conditions x_0, y_0, v_0, and ϕ_0 were drawn for each experiment are defined and presented in Tables C.1, C.2, and C.3.

An important aspect of the experiment methodology was to use a Out-Of-Sample (OOS) data pipeline [60]. This is the method of training the trainable models on one set of data (the *training data*), and evaluating how well it generalizes on a seperate set of data *the testing data*. This approach ensures the model doesn't learn specific patterns in the training samples, but rather learns the underlying general dynamics. In addition to the training and testing data, a third dataset is generated, referred to as the *validation data*. The model is never trained on this dataset but is continuously evaluated on it to guide hyper-parameter tuning and model configuration choices. For all three experiments, the final datasets contained 8000 trajectories each, split into 6400 for training, 800 for validation, and 800 for the final evaluation of the filters.

The IMM and PF filters, as they are not trainable, were only evaluated on the 800 trajectories in the test dataset, recording the metrics defined in Section 3.4.

3.5.1 Experiment 1 - Constant Velocity Trajectory

The first experiment was set up to evaluate how well the filters capture a target moving with nearly constant velocity, as described by the SDE in 3.2.1. It is worth mentioning that "constant velocity" trajectories do not imply a truly constant velocity. Instead, they are modelled as missiles with no thrust or drag, meaning that changes in velocity arise solely from the wobble and the stochastic Wiener processes in the a_n and a_t terms. Even when accelerations are small, the tracking problem is still far from trivial. The target experiences large fluctuations in position and velocity due to the stochastic noise terms. Furthermore, the uncertainties introduced by the measurement model remain substantial. Including this scenario, as one of the experiments, was therefore justified.

In Experiment 1, both the NF-DPF and PF approximate the target state using a reduced 4-dimensional state vector, instead of the general 6-dimensional state. This reduction omits acceleration terms and simplifies the learning task. This modelling choice is justified by the fact that the targets exhibit negligible average acceleration, with a mean acceleration close to zero throughout the trajectory. While this prior knowledge about the data may introduce a risk of data leakage, it was used here to isolate and evaluate model performance under simplified, lower-dynamic conditions.

Furthermore, the acceleration loss term $\mathcal{L}_{\text{RMSE}}^{\text{acc}}$ was not evaluated, and the loss function depended solely on $\mathcal{L}_{\text{RMSE}}^{\text{pos}}$, $\mathcal{L}_{\text{RMSE}}^{\text{vel}}$, and $\mathcal{L}_{\text{ELBO}}$. Consequently, the RMSE in acceleration was excluded from the final evaluation in Experiment 1.

3.5.2 Experiment 2 - Turning Trajectory

The second experiment was designed to evaluate the filters' performance in tracking targets which make unexpected turns, as modelled by the SDE in Section 3.2.2. This setup required the models to account for sudden changes in dynamics, effectively increasing the need to maintain a broader distribution, particularly for the particles in the NF-DPF and PF. Despite this, the added noise in the Motion Update Model for the NF-DPF and PF was kept the same as in Experiment 1. Although a larger spread could be beneficial in this scenario, the responsibility of adapting to these dynamics was left to the normalizing flows. The targets have small tangential accelerations and were therefore once more, with the same justification, modelled by a reduced state of 4 dimensions $\hat{X}_t = [\hat{x}, \hat{y}, \hat{v}_x, \hat{v}_y]$. Again, the term $\mathcal{L}_{\text{RMSE}}^{\text{acc}}$ was not evaluated, and the filters were not assessed based on RMSE in acceleration.

3.5.3 Experiment 3 - Accelerating Trajectory

The third experiment considers the full missile dynamics defined in Section 3.2.3, which simulates accelerating hypersonic missiles. In this final experiment, the full 6-dimensional state vector was used. This significantly increases the degrees of freedom, that is, the number of dimensions the model must approximate. As a result, a higher number of particles is required. The task becomes substantially more challenging because errors in acceleration estimations quickly propagate through the system. Even if the state of a particle accurately reflects the position and velocity of the target, an incorrect acceleration estimate can cause rapid divergence.

3.6 Software and Hardware

All filters were defined and evaluated using the *Python* programming language. The neural networks were implemented with the *PyTorch* framework. Result visualization was constructed using *Python's Matplotlib* library. The training was in its entirety carried out on a pair of NVIDIA L40S 48GB GPUs, and one NVIDIA A100 80GB GPU. The profiling was conducted using the *time.perf_counter()* function in *Python*. The CPU used for all experiments was an *Intel Xeon Processor (Icelake)* with 24 physical cores and AVX-512 support.

4

Normalizing Flow Differentiable Particle Filter

In contrast to previous chapters, this chapter will solely focus on the structure and implementation of the Normalizing Flow Differentiable Particle Filter (NF-DPF). The filter will be described in detail, as only a single prior work on NF-DPF models exists to date [16], limiting the availability of established references. The chapter will begin with an overview of the filter. Thereafter, each step of the filter will be presented in detail and lastly the training schematics will be defined.

4.1 NF-DPF Overview

The aim of the NF-DPF is to predict the true state X_t of the target at time t given noisy measurements Y_t. This is achieved by initializing a set of particles $\{\hat{X}_t^i, w_t^i\}_{i=1}^N$, where each particle $\hat{X}_t^i$, with associated weight w_t^i, represents a candidate state at time t. The particles are propagated through several models that adjust the state of each particle, thereafter the weights are updated correspondingly. This approach is consistent with the implementation by Chen et al. [16].

In the NF-DPF, it is essential that both the Dynamic and Proposal Normalizing Flows are not only fully differentiable but also invertible. Differentiability is a requirement of the broader class of Differentiable Particle Filters (DPFs), as it ensures that the mappings from inputs to outputs support end-to-end training through gradient descent [61]. This requirement applies to all components of the NF-DPF, including the Motion Update Model, Dynamic NF, Proposal NF, Measurement Model, Weight Update, and the Resampling.

The second requirement, invertibility, arises from the need to evaluate probability densities in both the dynamic and proposal models. In particle filtering, densities such as $p(\hat{X}_t \mid \hat{X}_{t-1})$, $q(\hat{X}_t \mid \hat{X}_{t-1}, Y_t)$, and $p(Y_t \mid \hat{X}_t)$ must be computed to perform proper particle weighting. While a standard neural network can generate samples $\hat{X}_t$ conditioned on $\hat{X}_{t-1}$ and Y_t, it does not provide a tractable expression for the probability density of those samples. Normalizing Flows (NFs) address this limitation by being both invertible and differentiable, with a tractable Jacobian that enables exact density computation via the change of variables formula. By transforming samples from a known base distribution (e.g., Gaussian) into complex distributions, NFs allow the required densities for both the dynamic and proposal models to be computed explicitly, making the weight update both probabilistically valid and differentiable.

Figure 4.1 presents an extended flowchart of the NF-DPF, building on the simplified version shown in Figure 3.2. It illustrates one iteration of the NF-DPF for a particle i when a new radar measurement Y_t becomes available at time t, including the propagation through the Motion Model, the Dynamic NF, the Proposal NF, the Measurement Model, and the Weight Update step. Additional steps not shown in the flowchart in Figure 4.1 include *Particle Initialization*, *State Prediction*, *Resampling*, and *Training*. The complete procedure is outlined in the pseudo-code in Algorithm 1. While Figure 4.1 illustrates a single filtering step, the algorithm describes the full NF-DPF pipeline over an entire trajectory, highlighting that training is performed at the end of each sequence.

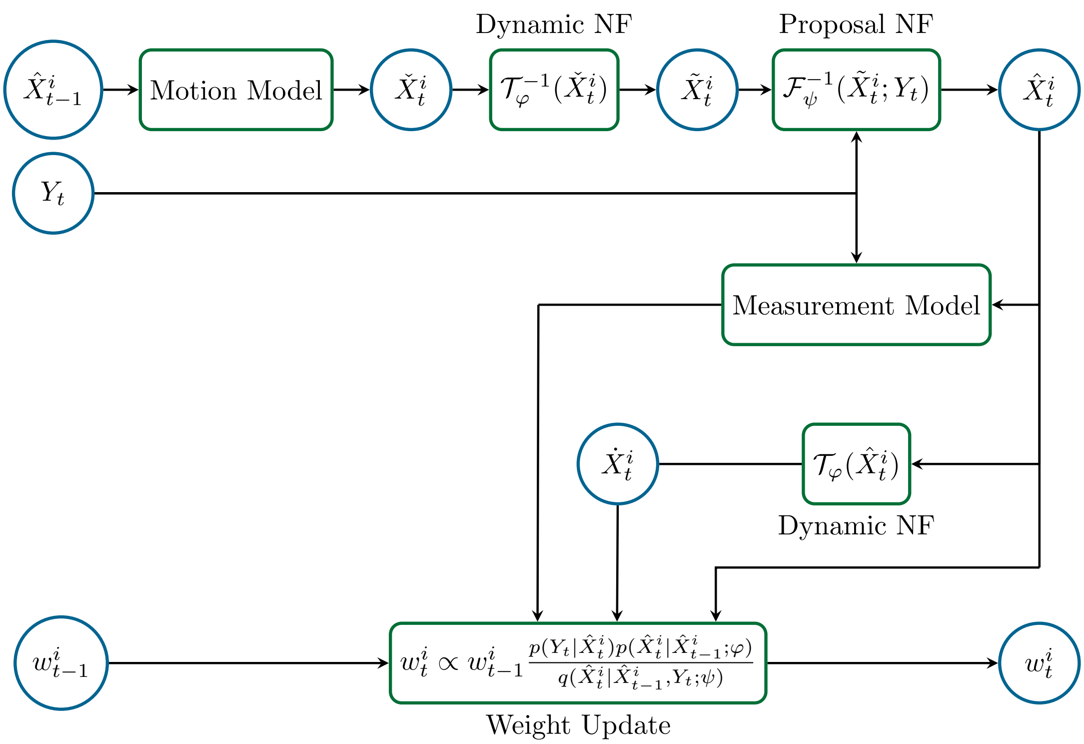


Figure 4.1: The flowchart illustrates one iteration in the NF-DPF. Particles X^i_{t-1} are first updated via the Motion Update Model with added stochastic noise to model process uncertainty and encourage exploration, yielding $\check{X}^i_t$. These are refined by the Dynamic Normalizing Flow to produce $\tilde{X}^i_t$, and finally updated by the Proposal Normalizing Flow using the measurement, resulting in $\hat{X}^i_t$. Each particle is then assigned a weight w^i_t based on the measurement likelihood from the Measurement Model and the transition probability, and the weights are normalized to approximate the posterior distribution.

Algorithm 1 Normalizing Flow Differentiable Particle Filter

for trajectory in batch **do**
  Initialize particles: $\{\hat{X}_0^i\}_{i=1}^N$
  Initialize weights: $\{w_0^i\}_{i=1}^N = \frac{1}{N}$
  for $t = 1$ to T **do**
    Update particles through motion update: $\check{X}_t^i \leftarrow \hat{X}_{t-1}^i$
    Update through Dynamic NF: $\tilde{X}_t^i = \mathcal{T}_\varphi^{-1}(\check{X}_t^i)$
    Update through Proposal NF: $X_t^i = \mathcal{F}_\psi^{-1}(\tilde{X}_t^i; Y_t)$
    Calculate measurement likelihood: $p(Y_t \mid \hat{X}_t^i)$
    Calculate prior probability with $\mathcal{T}_\varphi(\hat{X}_t^i)$: $p(\hat{X}_t^i \mid \hat{X}_{t-1}^i, \varphi)$
    Compute weights: $w_t^i \propto w_{t-1}^i \frac{p(Y_t|\hat{X}_t^i)p(X_t^i|\hat{X}_{t-1}^i;\varphi)}{q(\hat{X}_t^i|\hat{X}_{t-1}^i,Y_t;\psi)}$
    Normalize weights: $\tilde{w}_t^i \propto w_t^i$
    Predicted state: $\hat{X}_t = \sum_{i=1}^N \hat{X}_t^i w_t^i$
    Compute effective sampling size:
    $\text{ESS} = \frac{1}{\sum_{i=1}^N (w_t^i)^2}$
    if ESS < $\text{ESS}_{\min}$ **then**
      $\{\hat{X}_t^i\}_{i=1}^N \leftarrow \mathcal{R}\left(\{\hat{X}_t^i\}_{i=1}^N, \{w_t^i\}_{i=1}^N\right)$
      $\{w_t^i\}_{i=1}^N \leftarrow \frac{1}{N}$
    end if
  end for
end for
Compute loss $\mathcal{L}(\varphi, \psi)$ over batch.
Update φ and ψ through back-propagation.

4.2 NF-DPF Step Description

The following sections presents an in-depth description of each step in the NF-DPF model. The steps will be presented one by one in the order of one of the iterations presented in Algorithm 1. Note that the following descriptions assumes a 6-dimensional particle state vector, which includes acceleration. In cases where the state vector is 4-dimensional and does not include acceleration, any steps involving acceleration-related components are omitted accordingly.

4.2.1 Particle Initialization

At the start of each trajectory, the particle states and weights are initialized. This is performed by first constructing a guess of the initial state and then spreading out the particles uniformly within a bounded interval centred around this guess. The first two measurements Y_1 and Y_2, in polar coordinates (r, θ, v_r), are transformed into Cartesian positional coordinates using the measurement model function h in (2.7), as detailed in subsequent steps. These positions are then used to approximate the initial velocity by computing the difference between the first two positions. The acceleration are set up as terms of normal a_n and tangential a_t acceleration. Due to the lack of a reliable method of approximating the accelerations using few measurements, the initial acceleration guess is zero in both a_n and a_t. Combining the position from the first measurement with the approximated initial velocity and the acceleration yields the initial state estimate.

The initial particle set is constructed by uniformly perturbing the initial state estimate of each particle within predefined bounds. The velocity perturbations are applied in polar coordinates and allow for a directional spread of up to 180 degrees. This procedure yields N particles with state vectors $\{\hat{X}_0^i\}_{i=1}^N = (\hat{x}_0^i, \hat{y}_0^i, \hat{v}_{x,0}^i, \hat{v}_{y,0}^i, \hat{a}_{n,0}^i, \hat{a}_{t,0}^i)$ and corresponding uniform weights $\{w_0^i\}_{i=1}^N = 1/N$.

4.2.2 Motion Update Model

The Motion Update Model deterministically evolves each particle state based on its previous state and adds stochastic noise as $\hat{X}_t^i = \hat{X}_{t-1}^i + W$. Specifically, the position, velocity and acceleration components are updated with Gaussian noise as follows:

$$\begin{bmatrix}\check{x}\\ \check{y}\end{bmatrix} = \begin{bmatrix}\hat{x}\\ \hat{y}\end{bmatrix} + \begin{bmatrix}\hat{v}_x\\ \hat{v}_y\end{bmatrix} + \frac{1}{2}\begin{bmatrix}\hat{a}_n\\ \hat{a}_t\end{bmatrix} + W_{\text{pos}}, \quad W_{\text{pos}} \sim \mathcal{N}\left(0, \sigma_{\text{pos}}^2\right), \tag{4.1}$$

$$\begin{bmatrix}\check{v}_x\\ \check{v}_y\end{bmatrix} = \begin{bmatrix}\hat{v}_x\\ \hat{v}_y\end{bmatrix} + \begin{bmatrix}\hat{a}_n\\ \hat{a}_t\end{bmatrix} + W_{\text{vel}}, \quad W_{\text{vel}} \sim \mathcal{N}\left(0, \sigma_{\text{vel}}^2\right), \tag{4.2}$$

$$\check{a}_n = \hat{a}_n + w_n, \quad w_n \sim \mathcal{N}\left(0, \sigma_n^2\right), \tag{4.3}$$

$$\check{a}_t = \hat{a}_t + w_t, \quad w_t \sim \mathcal{N}\left(0, \sigma_t^2\right). \tag{4.4}$$

The standard deviation σ controls the amount of process noise added to the state during the motion update and reflects uncertainty in how the target moves over time. When the target has a higher velocity, the uncertainty in its future state may increase due to more abrupt or less predictable dynamics. To capture this, σ is made dependent on the current velocity of the target. In this implementation, the noise terms are scaled proportionally to the target's speed, using the velocity magnitude $\|\hat{V}\| = \sqrt{\hat{v}_x^2 + \hat{v}_y^2}$ as follows:

$$\sigma = \alpha + \beta\|V\|, \tag{3.6}$$

where α represents a minimum baseline noise level, and β controls how much the noise increases with speed. This formulation ensures that the noise remains non-zero even at low velocities, while allowing for increased stochasticity during high-speed motion. The impact of this velocity dependent scaling may vary depending on the scenario, and in some experiments β is set to zero. The noise levels used for the three experiments are presented in Appendix D. The Motion Update Model ultimately returns the updated particle state vector, the applied noise, and the corresponding standard deviations:

$$\check{X}_t^i = \begin{bmatrix} \check{x} \\ \check{y} \\ \check{v}_x \\ \check{v}_y \\ \check{a}_n \\ \check{a}_t \end{bmatrix}, \quad W = \begin{bmatrix} W_{\text{pos}} \\ W_{\text{vel}} \\ w_{\text{n}} \\ w_{\text{t}} \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \sigma_{\text{pos}} \\ \sigma_{\text{vel}} \\ \sigma_{\text{n}} \\ \sigma_{\text{t}} \end{bmatrix}, \tag{4.5}$$

where subscript t and superscript i are dropped in the $\check{X}$ vector for clarity.

4.2.3 Dynamic Normalizing Flow

The Dynamic Normalizing Flow, $\mathcal{T}_\varphi$, appears two times in the proposed NF-DPF model. The flow is a conditional NF with 4 consecutively stacked RealNVP flows, each flow parametrized by the set φ in the scale and shift neural networks, both consisting of 3 hidden layers with 32 neurons each. Both passes of the Dynamic NF are presented in the order of execution.

a) Inverse pass - $\mathcal{T}_\varphi^{-1}(\check{X}_t^i)$
The first Dynamic Normalizing Flow is the inverse transformation $\mathcal{T}_\varphi^{-1}$, which maps a sample from a simple base distribution into a more complex learned distribution. The flow takes the updated particle state vector $\check{X}_t^i$ from the Motion Update Model as input and conditions them on the mean and standard deviation of the particle set, $(\mu_{\check{X}_t}, \sigma_{\check{X}_t})$. This transformation can be expressed as follows:

$$\tilde{X}_t^i = \mathcal{T}_\varphi^{-1}(\check{X}_t^i; (\mu_{\check{X}_t}, \sigma_{\check{X}_t})), \tag{4.6}$$

where the output $\tilde{X}_t^i$ is the transformed particles. Using Eq. (2.60) and (2.61), the density transformation can be stated as:

$$p(\tilde{X}_t^i \mid \hat{X}_{t-1}^i) = p(\check{X}_t^i \mid \hat{X}_{t-1}^i)\Big|\det J_{\mathcal{T}_\varphi^{-1}}(\check{X}_t^i)\Big|^{-1}. \tag{4.7}$$

Here, $p(\check{X}_t^i \mid \hat{X}_{t-1}^i)$ corresponds to the process noise distribution and can be written as $p(W_t^i)$, where p denotes the Gaussian density defined in Section 2.3.2. The transformation in Eq. (4.7) can therefore be rewritten as:

$$p(\tilde{X}_t^i \mid \hat{X}_{t-1}^i) = p(W_t^i)\Big|\det J_{\mathcal{T}_\varphi^{-1}}(\check{X}_t^i)\Big|^{-1}. \tag{4.8}$$

This density is used as part of the proposal distribution $q(\hat{X}_t^i \mid \hat{X}_{t-1}^i, Y_t; \psi)$, discussed in Section 4.2.4.

b) Forward pass - $\mathcal{T}_\varphi(\hat{X}_t^i)$
The second Dynamic NF is applied after the Proposal NF $\mathcal{F}_\psi^{-1}(\tilde{X}_t^i; Y_t)$ as a forward transformation $\mathcal{T}_\varphi$, which maps the proposed particle $\hat{X}_t^i$ from the complex learned proposal distribution back into the simple base distribution. Unlike the inverse Dynamic NF pass, this transformation does not yield the final particle state but is

instead used to evaluate the prior density $p(\hat{X}_t^i \mid \hat{X}_{t-1}^i; \varphi)$ as part of the importance weighting step. This is achieved by transforming $\hat{X}_t^i$ into latent space. The transformation can be expressed as:

$$\dot{X}_t^i = \mathcal{T}_\varphi(\hat{X}_t^i; (\mu_{\check{X}_t}, \sigma_{\check{X}_t})), \tag{4.9}$$

where the output $\dot{X}_t^i$ are particles drawn from the simple distribution $p(\cdot \mid \hat{X}_{t-1}^i; \varphi)$. The particles $\hat{X}_t^i$ are assumed to be drawn from the prior distribution $p(\hat{X}_t^i \mid \hat{X}_{t-1}^i; \varphi)$. Through the transformation $\mathcal{T}_\varphi$ and its change in density, we can express the prior distribution as:

$$p(\hat{X}_t^i \mid \hat{X}_{t-1}^i; \varphi) = p(\dot{X}_t^i \mid \hat{X}_{t-1}^i)\Big|\det J_{\mathcal{T}_\varphi}(\hat{X}_t^i)\Big|. \tag{4.10}$$

Here, $p(\dot{X}_t^i \mid \hat{X}_{t-1}^i)$ can be rewritten as $p(\dot{X}_t^i - (\check{X}_t^i - W_t^i))$ where p denotes the Gaussian density defined in Section 2.3.2. The prior density in Eq. (4.10) can therefore be rewritten as:

$$p(\hat{X}_t^i \mid \hat{X}_{t-1}^i; \varphi) = p(\dot{X}_t^i - (\check{X}_t^i - W_t^i))\Big|\det J_{\mathcal{T}_\varphi}(\hat{X}_t^i)\Big|. \tag{4.11}$$

4.2.4 Proposal Normalizing Flow

The Proposal Normalizing Flow, $\mathcal{F}_\psi$, is also constructed as a conditional RealNVP-based normalizing flow, as defined in Section 2.2.3. The model uses 4 stacked flows, were each flow is parametrized by the set ψ in the scale and shift neural networks, both consisting of 3 layers with 32 neurons each.

This flow operates in inverse mode, $\mathcal{F}_\psi^{-1}$ where it transforms the particles $\tilde{X}_t^i$ from an intermediate base distribution into the final proposed state $\hat{X}_t^i$. The key characteristic of the Proposal Normalizing Flow is that it is conditioned, not only on the mean and standard deviation of the particle set, but also on the radar measurement Y_t. Rather than using the positional coordinates directly, the conditioning features are derived from measurement-particle distances, as $(\Delta x, \Delta y, v_r)$, where v_r is left unchanged. This feature transformation was found to improve the NF-DPF's ability to predict the target distribution. The transformation is expressed as:

$$\hat{X}_t^i = \mathcal{F}_\psi^{-1}(\tilde{X}_t^i; (Y_t, \mu_{\tilde{X}_t}, \sigma_{\tilde{X}_t})), \tag{4.12}$$

where the output $\hat{X}_t^i$ are the proposed particles, which after reweighing represent the posterior pdf, and are used for the final state estimation. The density transformation can be stated as:

$$p(\hat{X}_t^i \mid \tilde{X}_t^i, Y_t) = p(\tilde{X}_t^i \mid \check{X}_t^i, Y_t)\Big|\det J_{\mathcal{F}_\psi^{-1}}(\tilde{X}_t^i; Y_t)\Big|^{-1}. \tag{4.13}$$

Combining this result with the density transformation in Eq. (4.8), the proposal distribution can be derived as:

$$q(\hat{X}_t^i \mid \hat{X}_{t-1}^i, Y_t; \psi) = p(W_t^i)\Big|\det J_{\mathcal{T}_\varphi^{-1}}(\check{X}_t^i)\Big|^{-1}\Big|\det J_{\mathcal{F}_\psi^{-1}}(\tilde{X}_t^i; Y_t)\Big|^{-1}. \tag{4.14}$$

4.2.5 Measurement Model

The objective of the measurement model is to represent the relationship between the observation Y_t and the state $\hat{X}_t^i$ through a likelihood function that models the distribution $p(Y_t \mid \hat{X}_t^i)$. This distribution describes how likely a particular observation is, given a specific state. In contrast to the approach presented by Chen et al. [16], where this relationship is learned using a conditional normalizing flow, the measurement model used in this work is known and explicitly defined as a Gaussian distribution in polar coordinates.

More specifically, given particle states expressed in Cartesian coordinates, the measurement model transforms the particle states into polar coordinates r, θ and v_r as in Eq. (2.7), and computes the measurement likelihood under an additive Gaussian noise assumption. The covariance of the measurement noise is predefined with respect to the noise characteristics of the sensor, as defined in Eq. (2.13), and the likelihood is computed using the probability of the corresponding multivariate Gaussian. Using the derived equation Eq. (2.69) for a multivariate Gaussian, and applying it for polar coordinates one gets the following:

$$
\begin{aligned}
p(Y_t \mid \hat{X}_t^i) &= \frac{1}{(2\pi)^{3/2}\prod_{j=1}^{3}\sigma_j} \exp\left(-\frac{1}{2}\sum_{j=1}^{3}\frac{(Y_{t,j} - h(\hat{X}_{t,j}^i))^2}{\sigma_j^2}\right) &\quad (4.15)\\
&= \frac{1}{(2\pi)^{3/2}\sigma_r\sigma_\theta\sigma_{v_r}} \exp\left(-\frac{1}{2}\left[\frac{(r-\hat{r})^2}{\sigma_r^2} + \frac{(\theta-\hat{\theta})^2}{\sigma_\theta^2} + \frac{(v_r-\hat{v}_r)^2}{\sigma_{v_r}^2}\right]\right). &\quad (4.16)
\end{aligned}
$$

The known measurement model h does not require learning from data, and therefore the likelihood is defined as $p(Y_t \mid \hat{X}_t^i)$, in contrast to the parametrized form $p(Y_t \mid \hat{X}_t^i; \theta)$ in Eq. (2.50), which depends on the parameter set θ.

4.2.6 Weight Update

The weights are updated to reflect the relative importance of each particle under the target distribution, which is given by:

$$
w_t^i \propto w_{t-1}^i \frac{p(Y_t \mid \hat{X}_t^i)\, p(\hat{X}_t^i \mid \hat{X}_{t-1}^i; \varphi)}{q(\hat{X}_t^i \mid \hat{X}_{t-1}^i, Y_t; \psi)}. \quad (4.17)
$$

Using the derived expressions for the probabilities from Eq. (4.11) and (4.14), and keeping the measurement probability Eq. (4.15) on its original form, the weight update can be rewritten as:

$$
w_t^i \propto w_{t-1}^i \frac{p(Y_t|\hat{X}_t^i) p(\dot{X}_t^i - \check{X}_t^i + W_t^i)\left|\det J_{\mathcal{F}_\psi^{-1}}(\tilde{X}_t^i; Y_t)\right|\left|\det J_{\mathcal{T}_\varphi^{-1}}(\check{X}_t^i)\right|\left|\det J_{\mathcal{T}_\varphi}(\hat{X}_t^i)\right|}{p(W_t^i)}. \quad (4.18)
$$

The proportionality symbol indicates that normalization across all weights is necessary. The normalized weights $\tilde{w}_t^i$ are calculated as:

$$\tilde{w}_t^i = \frac{w_t^i}{\sum_{k=1}^N w_t^i}. \tag{4.19}$$

However, for numerical stability, the weights are computed in logarithmic form ($w_t^i = \log w_t^i$). Thus, the normalized weights are calculated using the following numerically stable expression:

$$\tilde{w}_t^i = \frac{\exp\left(w_t^i - \max_j w_t^j\right)}{\sum_{k=1}^N \exp\left(w_t^k - \max_j w_t^j\right)}. \tag{4.20}$$

4.2.7 State Prediction

The final predicted state, used for estimating the target's state in the tracking context, follows the same principle as in the particle filter and is given by Eq. (2.44). It is computed as the weighted average of the particles:

$$\hat{X}_t = \sum_{i=1}^N w_t^i \hat{X}_t^i. \tag{2.44}$$

4.2.8 Resampling

Since NF-DPF requires a fully differentiable resampling technique to enable backpropagation, Entropy-Regularized Optimal Transport Resampling, defined in Section 2.2.2, is used. This resampling method avoids weight degeneracy by spreading the states continuously according to the target distribution. Importantly, resampling is performed only for those trajectories in the batch where the effective sample size (ESS) falls below the predefined threshold, $\text{ESS}_{\min} = \frac{N}{2}$.

4.3 NF-DPF Training Process

The NF-DPF, including the Dynamic NF and Proposal NF which constitute the trainable components, was trained on 6,400 target trajectories as described in Section 3.5, for 100 epochs or until convergence. The loss over epochs for the training of the NF-DPF for the 3 Experiments is reported in Appendix E.

4.3.1 Loss Function

The loss function $\mathcal{L}(\varphi, \psi)$ used to train the NF-DPF is based on the weighted combination of ELBO and RMSE for position, velocity, and acceleration, as previously defined in Eq. (3.7):

$$\mathcal{L}(\varphi, \psi) = \alpha_{\text{pos}}\mathcal{L}_{\text{pos}}^{\text{RMSE}} + \alpha_{\text{vel}}\mathcal{L}_{\text{vel}}^{\text{RMSE}} + \alpha_{\text{acc}}\mathcal{L}_{\text{acc}}^{\text{RMSE}} + \alpha_{\text{ELBO}}\mathcal{L}_{\text{ELBO}}. \tag{3.7}$$

4.3.2 Optimization

The parameters (φ, ψ) in the Normalizing Flows were continuously updated, as described in Section 2.2.1, by using the Adam optimizer, predefined in *PyTorch*. The initial learning rate was set to 1e−4 and was scheduled to be updated by a factor of 0.8 every 10 epochs.

Training was conducted over batches of 32 trajectories. As described in Section 3.2, the full set of trajectories was divided into training, validation, and test datasets. During training, batches were randomly sampled from the training set to ensure variation and improve generalization. Overfitting was mitigated by monitoring the training and validation performance throughout training, allowing early detection in cases where the training loss began to drop below the validation loss. This did not occur, which is further discussed in Appendix E. The model that achieved the best combined loss $\mathcal{L}(\varphi, \psi)$ on the validation data during training was saved as the final model.

5

Results

This chapter presents the results obtained from the experiments described in Sections 3.5.1 - 3.5.3. To reiterate, for all experiments, the state of the target, is estimated using an online filtering setup based on $p(X_t \mid Y_{0:t})$, as opposed to the more trivial offline setup, which conditions on the full observation sequence $p(X_t \mid Y_{0:T})$. The results are grouped into the *quantitative* results where the final performance is reported in terms of Euclidean RMSE in position, velocity, and acceleration (when available) for all filters, and boxplots that visualize performance across the test dataset. Secondly, for each experiment, the *qualitative* results are presented, including a few representative runs containing the true target positions, the radar measurements, and the filters positional estimates. For the NF-DPF, contour plots are included for the intermediate particle states at three key steps: after the Motion Update Model ($\check{X}$), after the Dynamic NF ($\tilde{X}$), and after the Proposal NF ($\hat{X}$). The chapter concludes with an assessment of the computational viability of each filter, including execution time and and mathematical complexity.

5.1 Experiment 1 – Constant Velocity Target

As described in Section 3.5.1, Experiment 1 evaluates the filtering techniques based on how accurately they estimate the true 4-dimensional state of targets moving at approximately constant velocity. The filters are assessed both quantitatively and qualitatively using 100 particles in the PF and NF-DPF. Section 5.1.3 provides a summary of the experimental results.

5.1.1 Experiment 1 – Quantitative Results

The IMM, PF and the trained NF-DPF were evaluated on the 800 trajectories in the test dataset generated from Eq. (3.1)-(3.2). The Euclidean RMSE, as defined in Eq. (2.64) was recorded. The positional and velocity Euclidean RMSE, along with the combined RMSE, which was computed with the same weighting as the loss presented in Table D.4, are presented in Table 5.1. The positional Euclidean RMSE between the radar measurements Y_t, converted to Cartesian coordinates, and the true target positions is also presented in Table 5.1. This value represents the lowest baseline performance, where simply using the measurements as positional estimates is a trivial solution that any viable filter should outperform.

Table 5.1: Experiment 1 – RMSE results for the three filters, evaluated on the test dataset. The $\pm$ values represent the standard deviation across trajectories in the set. The combined RMSE is computed as 70% RMSE position and 30% RMSE velocity, matching the weighting used in the training loss for NF-DPF, see Table D.4. Boldface marks the best result in each category.

	RMSE Position	RMSE Velocity	RMSE Combined
Measurement	28.42 ± 3.47	–	–
IMM	$\mathbf{17.44 \pm 3.79}$	8.66 ± 2.80	$\mathbf{14.81 \pm 3.49}$
PF	24.62 ± 3.71	19.72 ± 13.06	23.15 ± 6.51
NF-DPF	18.42 ± 3.77	$\mathbf{8.55 \pm 2.12}$	15.46 ± 3.28

The PF resulted in the highest RMSE for both position and, more notably, velocity, leading to the poorest overall performance. The IMM filter slightly outperforms the NF-DPF in positional accuracy and in the combined RMSE, while the NF-DPF shows a marginal advantage in velocity estimation. All filtering techniques achieve lower positional RMSE than the radar measurements. The standard deviations are similar across filters in the positional case, whereas in velocity, the NF-DPF demonstrates the lowest variability, followed by the IMM, and with the PF performing the worst. Overall, the IMM and NF-DPF deliver comparable results in this first experiment, while the PF was clearly less accurate.

The Euclidean RMSE over the test dataset is also illustrated using boxplots for each filtering method and individual dimension. Each box represents the interquartile range (IQR) (from the 1st to the 3rd quartile), with whiskers indicating the data range within 1.5 times the IQR, excluding outliers. The horizontal line within each box denotes the median. The boxplots provide a summary of the distribution and variability of the RMSE across test trajectories. Once more the RMSE between the radar measurement and the true position is included. The boxplots for the positional RMSEs are found in Figure 5.1, while the boxplots for the velocity RMSEs are found in Figure 5.2.

The IMM and NF-DPF demonstrate comparable accuracy, as also reflected in the mean performance values reported in Table 5.1. The NF-DPF displays a wider spread within the whiskers, indicating greater variability in standard cases. However, the NF-DPFs lower standard deviation from Table 5.1 suggests that the IMM may suffer from more frequent or more severe outliers. This implies that the NF-DPF is at least comparable in terms of robustness, if not more resilient to extreme errors.

Once more the IMM and NF-DPF exhibit comparable accuracies. Notable is that the PF has the lowest accuracy, both in terms of median, and when considering all quartiles. This is explained by the fact that the parameters of the PF were tuned by evaluating the filter on positional accuracy and not accuracy in velocity.

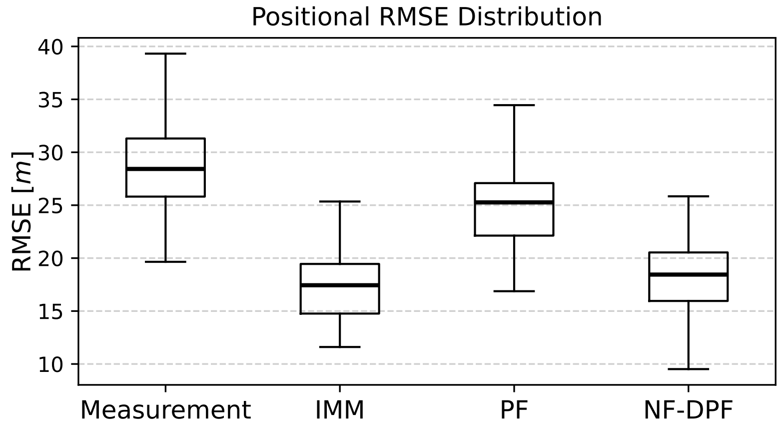


Figure 5.1: Boxplot of the positional RMSE distribution for the measurements, IMM, PF and NF-DPF for Experiment 1.

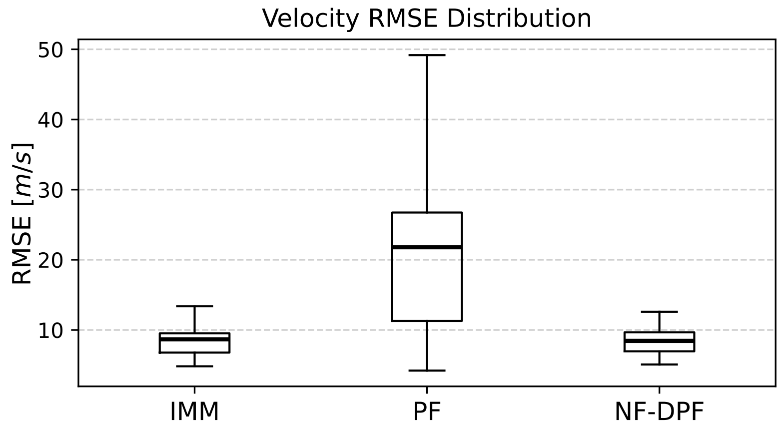
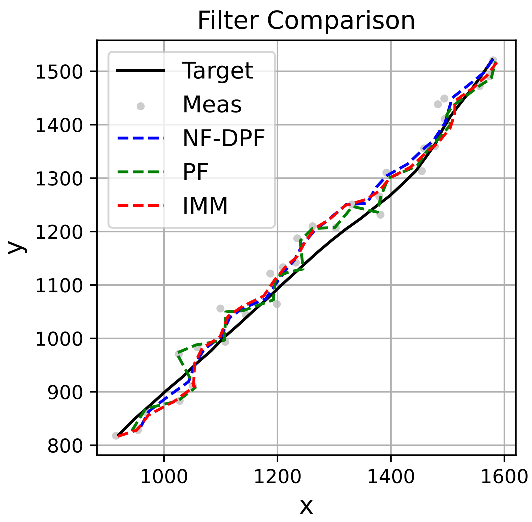


Figure 5.2: Boxplot of the velocity RMSE distribution for the IMM, PF and NF-DPF for Experiment 1.

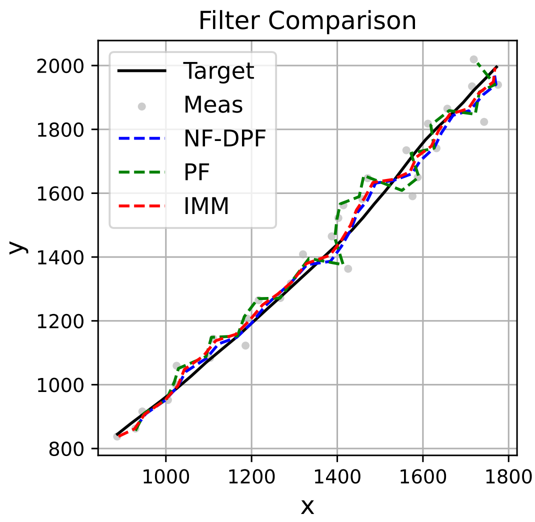
5.1.2 Experiment 1 – Qualitative Results

Figure 5.3 illustrates the application of the three filtering techniques on two representative trajectories from the test dataset. The plots visualize the reconstructed paths of the target by each filtering method, allowing for a qualitative comparison of tracking behaviour. The true trajectory is plotted together with the radar measurements and the filtered estimates from the NF-DPF, PF, and IMM. This type of visualization helps to assess how closely the filters align with the underlying ground truth, especially in terms of smoothness, bias, and responsiveness to measurement noise.

Figure 5.3 shows that the predicted trajectories from the NF-DPF and IMM exhibit highly similar dynamics, with only minor deviations in isolated parts of the trajectory. This observation aligns well with the positional RMSE values reported in Table 5.1, where both models demonstrate comparable error and variance. In contrast, the PF displays a greater bias and is more sensitive to measurement noise.



(a) One random trajectory from dataset.



(b) One random trajectory from the dataset.

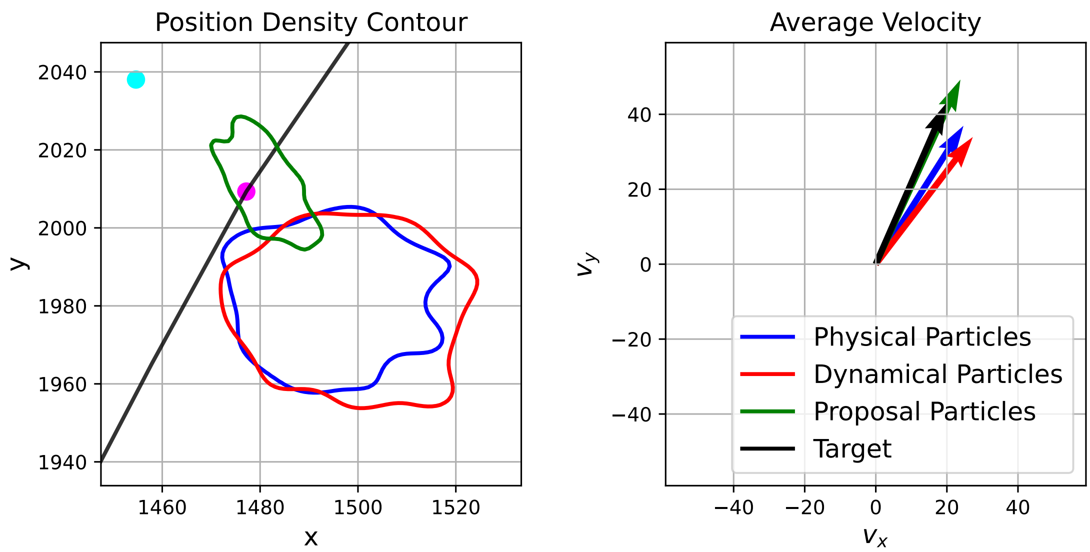
Figure 5.3: The filters applied on the measurements from two different trajectories in the Constant Velocity Target test dataset used in Experiment 1.

This is particularly noticeable in the latter part of the trajectory in Subfigure 5.3b, where the PF overreacts to individual measurements, leading to abrupt deviations that do not align with the expected target dynamics.

Figure 5.4 illustrates the effect of each NF on the particle distribution after a single measurement update. It shows the particle states after the Motion Update model, Dynamic NF, and Proposal NF, represented by their respective particle sets. Subfigure 5.4a displays a two-dimensional contour that encloses 85% of the particles, visualizing the region where the filter believes the target is most likely located. This contour reflects the spatial density and uncertainty of the NF-DPF's belief after the respective update steps. Subfigure 5.4b shows the corresponding average velocity of the particle sets using arrows. This highlights how each step influences the velocity component of the estimated state.

As shown in Subfigure 5.4a, the Proposal NF performs the most substantial transformation of the particle distribution. It both contracts the spatial spread of particles and shifts their positions towards the radar measurement, without ending up at the measurement, which usually indicating overfitting to the measurements. Instead, the transformed distribution encloses the true target state, yielding a good positional prediction. It is worth noting that this does not always occur. In contrast, the Dynamic NF produces a minor transformation of the initial spread of the particles.

Subfigure 5.4b shows a similar transformation for the velocity states. The Proposal NF increases the magnitude and adjusts the direction of the average velocity vector to better align with the target, whereas the Dynamic NF produces only a minor



(a) The positional contour from all particles after Motion Update, Dynamic NF and Proposal NF.

(b) The average velocity from all particles after Motion Update, Dynamic NF and Proposal NF.

Figure 5.4: Positional contour and average velocity of the particle cloud after the Motion Update (blue), Dynamic NF (red), and Proposal NF (green), shown at a representative time step from a trajectory in Experiment 1 – Constant Velocity Target. In both subfigures, the true target position (black) is indicated. For the positional contour (left), the radar measurement (cyan) and the ground-truth position (magenta) are also included. The legend is shared across both subfigures.

update. While this time step was selected for illustrative purposes, similar patterns occur throughout the sequence with a small broadening from the Dynamic NF, followed by a shift and contraction from the Proposal NF, depending on measurement.

5.1.3 Experiment 1 – Summary

In Experiment 1, the NF-DPF shows comparable results to the IMM. As shown in the tables, the IMM slightly outperforms the NF-DPF in position and combined RMSE, while the NF-DPF shows marginally better results in velocity. In contrast, the PF underperforms in both positional- and velocity accuracy. Both filters outperform the noisy radar measurements in positional accuracy. The PF just slightly outperforms the measurements in positional estimates. The qualitative analysis confirms these trends, with both IMM and NF-DPF closely tracking the target trajectory, while the PF is less stable and overreacts to each measurement. The contour plots show how the NF-DPF's Proposal NF scales and shifts the cloud of particles toward the measurement and adjusts the velocity vector, while the Dynamic NF performs a slight transformation of the particle set. Overall, the NF-DPF demonstrates robust behaviour and competitive accuracy for low-manoeuvrable targets.

5.2 Experiment 2 – Turning Target

As described in Section 3.5.2, Experiment 2 evaluates the filtering techniques based on their ability to estimate the true 4-dimensional state of targets moving at approximately constant velocity, but subject to a varying number of turns with unknown magnitude, duration, and start and end times. The filters are once more assessed both quantitatively and qualitatively using 100 particles in the PF and NF-DPF. Section 5.2.3 provides a summary of the experimental results.

5.2.1 Experiment 2 – Quantitative Results

The IMM, PF and the trained NF-DPF were evaluated on the 800 trajectories in the test dataset generated from Eq. (3.1) and (3.3). The Euclidean RMSE, as defined in Eq. (2.64) was recorded. Positional and velocity RMSE values, along with the combined RMSE, are presented in Table 5.2. The positional Euclidean RMSE between the radar measurements Y_t Cartesian coordinates and the true target positions x, y is also presented in the same table.

Table 5.2: Experiment 2 – RMSE on the test dataset. The ± values represent the standard deviation across trajectories. The combined RMSE is computed as 70% RMSE position and 30% RMSE velocity, matching the weighting used in the training loss for NF-DPF (see Table D.4). Boldface marks the best result in each category.

	RMSE Position	RMSE Velocity	RMSE Combined
Measurement	26.57 ± 4.08	–	–
IMM	19.50 ± 4.06	11.82 ± 3.51	17.20 ± 3.90
PF	28.05 ± 6.23	34.94 ± 19.69	30.12 ± 10.27
NF-DPF	$\mathbf{18.91 \pm 3.77}$	$\mathbf{10.72 \pm 2.93}$	$\mathbf{16.45 \pm 3.52}$

In the turning target experiment, the PF no longer outperforms the naive approach of using the measurements directly as positional estimates, as the latter yields a lower positional RMSE. In contrast, the NF-DPF achieves the best performance across all categories, with both lower means and lower standard deviations in position, velocity, and combined RMSE. The NF-DPF results are consistent with those reported in Table 5.1, while the IMM performs notably worse. This suggests that the target dynamics of Experiment 2 is less suitable for the IMM filter, likely because the motion characteristics in the dataset are not well represented by its filters. These results highlight the NF-DPF's flexibility across diverse target types.

As in Experiment 1, the Euclidean RMSE over the test dataset is illustrated using boxplots for each filtering method and dimension. The boxplots for positional RMSE, including those of the measurements and all filtering techniques, are shown in Figure 5.5, while the corresponding boxplots for velocity RMSE are presented in Figure 5.6.

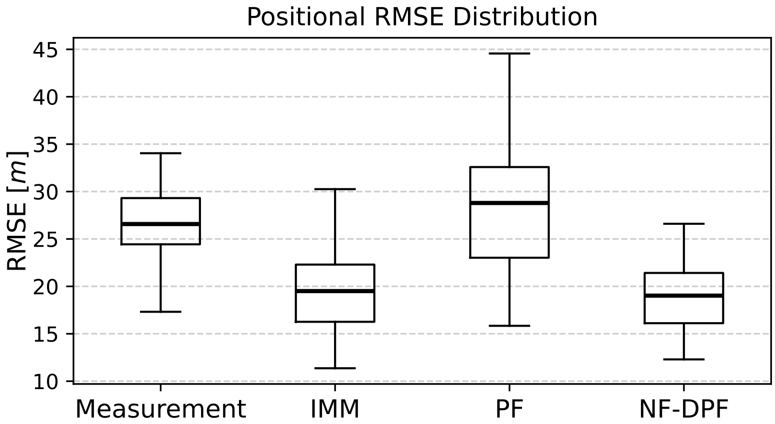


Figure 5.5: Boxplot of the positional RMSE distribution for the measurements, IMM, PF and NF-DPF for Experiment 2.

Figure 5.5 supports the same conclusions as those drawn from the results in Table 5.2: the NF-DPF achieves the lowest positional RMSE and the smallest spread among the filtering techniques. The IMM shows comparable performance to the NF-DPF, while the PF is clearly outperformed by both.

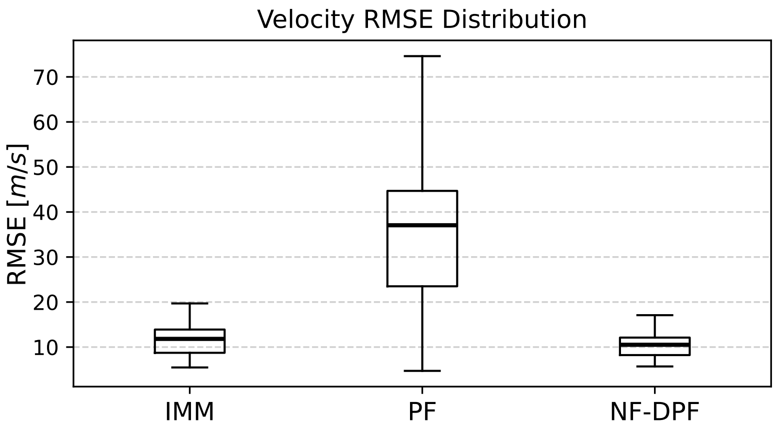


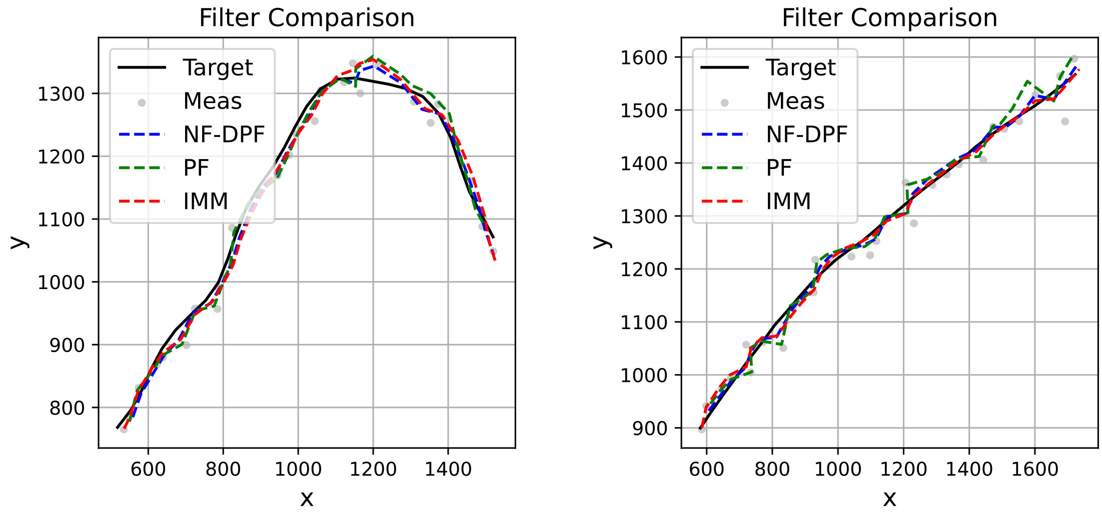
Figure 5.6: Boxplot of the velocity RMSE distribution for the IMM, PF and NF-DPF for Experiment 2.

As noted from the results in Table 5.2, the PF is unable to approximate the velocity, achieving both a high median, and a large spread. The NF-DPF and IMM exhibit comparable performance, with the NF-DPF performing slightly better.

5.2.2 Experiment 2 – Qualitative Results

As in Experiment 1, a few representative trajectories from the test dataset are, together with the positional predictions from the three filtering techniques; IMM, PF and NF-DPF, illustrated in Figure 5.7. The radar measurements, converted to

Cartesian coordinates are also included.



(a) A representative turning target from the turning trajectory dataset. All filters capture the turn, although with some delay.

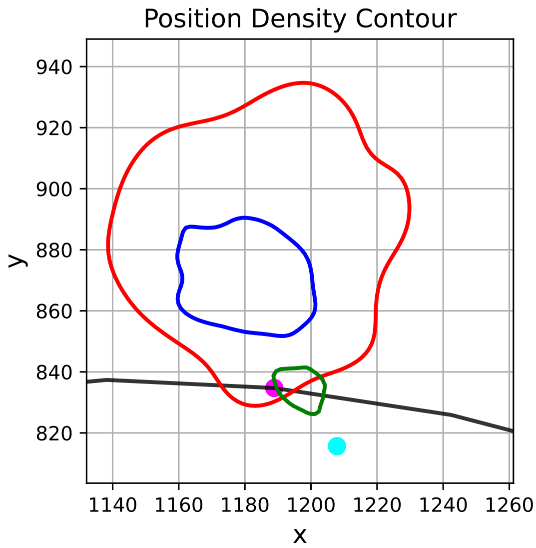
(b) A representative nineon-turning target from the same dataset, illustrating the adaptability of the filtering techniques.

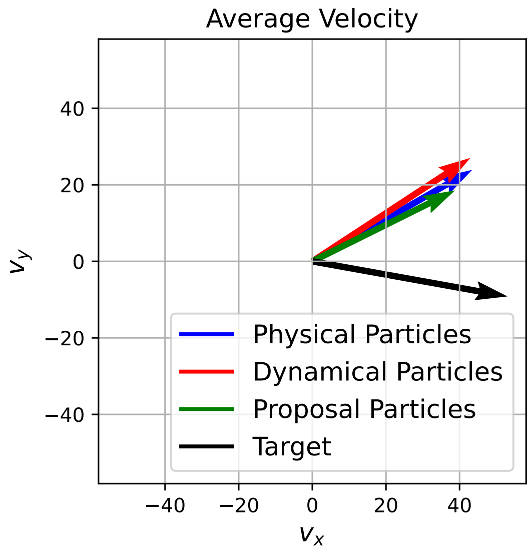
Figure 5.7: Two representative trajectories from the turning target dataset used in Experiment 2: one where the target turns once (a) and one where it does not turn (b). Positional predictions from all filtering techniques are shown.

The IMM and NF-DPF exhibit similar positional predictions, but the NF-DPF anticipates the turn shown in Subfigure 5.7a more effectively. This is evident from its alignment with the measurements at the onset of the turn, rather than continuing straight ahead as seen with the IMM. All models successfully recover their tracking after the turn. In Subfigure 5.7b, the PF is the most reactive, with its positional predictions closely overlapping with the radar measurements. This leads to trajectory estimates that are not physically plausible, in contrast to the smoother and more consistent predictions produced by the IMM and NF-DPF.

Figure 5.8 illustrates the effects of each model in the NF-DPF. As before, the particle distribution (represented by a contour enclosing 85% of the particles) is shown after the Motion Update Model, the Dynamic NF, and the Proposal NF. Since the second dataset contains turning trajectories, a single time step where the target initiates a right turn is selected for visualization. The current radar measurement (cyan) and the true target position (magenta) are also included.

Although the example in Figure 5.8 was selected to illustrate a representative turning trajectory from Experiment 2, it highlights several behaviours that reoccur across different trajectories and time steps. Most notably, the Dynamic NF tends to spread out the particles, likely as a precautionary measure to handle potential abrupt motion change such as turns. This dispersion strategy is particularly useful given the





(a) The Dynamic NF spreads out the particles (red contour), preparing for a possible turn, while the Proposal NF moves the particles toward the measurement (green contour), successfully relocating them close to the true position (purple dot).

(b) The average velocity from all particles after Motion Update Model, Dynamic NF and Proposal NF. The models attempt to correct the particle velocities to reflect the turn. The black arrow is the true target velocity vector.

Figure 5.8: The positional contour, and average velocity from all particles after Motion Update Model (blue), Dynamic NF (red) and Proposal NF (green), illustrated at a single time step from a representative turning target trajectory. The visualized moment occurs during a turn, where the velocity vectors have not yet aligned with the change in direction. The legend applies to both subfigures.

dataset's structure, where turn events occur with random durations and start times, and one-third of the trajectories contain no turns at all. Because the filter cannot assume a turn is occurring, it maintains flexibility to handle potential changes in motion.

The Proposal NF consistently shifts particles toward the current radar measurement. During turns, it is also observed that the velocity magnitudes decrease, an effect likely due to reduced radial velocities, which the Proposal NF interprets as a drop in speed. While the velocity vectors may not immediately align with the true direction of motion, the model compensates over the course of several time steps. Much of this correction occurs during the resampling step, which helps the system recover even if the dynamics are not perfectly modelled by the Motion Update Model and NFs.

These behaviours, particle dispersion by the Dynamic NF, measurement-driven adjustment by the Proposal NF, and gradual correction of velocities, combine to yield rather effective posterior estimates.

5.2.3 Experiment 2 – Summary

Experiment 2 suggests that the NF-DPF is well-suited to handling turning trajectories with uncertain dynamics. It shows competitive performance across position and velocity estimation tasks, achieving the lowest RMSE scores among the evaluated filters. While the IMM performs comparably in some cases, it appears less capable of anticipating sudden changes in motion, as this is not captured by the filters in the IMM. The PF, although highly responsive, often produces less physically plausible predictions. Overall, the results indicate that the learned components of the NF-DPF contribute to more consistent tracking performance across a range of motion patterns.

5.3 Experiment 3 – Accelerating Target

As described in Section 3.5.3, Experiment 3 evaluates the filtering techniques based on how accurately they estimate the true 6-dimensional state of accelerating targets. The filters are assessed both quantitatively and qualitatively. 400 particles were used in both the PF and NF-DPF, which was the computational limit. Finally, Section 5.3.3 provides a summary of the experimental results.

5.3.1 Experiment 3 – Quantitative Results

The IMM, PF and trained NF-DPF were evaluated on the 800 trajectories in the test dataset generated from the SDEs in Eq. (3.2) and Eq. (3.4). The Euclidean RMSE, as defined in Eq. (2.64) was recorded. The position, velocity and acceleration Euclidean RMSE, along with the combined RMSE, are presented in Table 5.3. The positional Euclidean RMSE between the radar measurements Y_t Cartesian coordinates and the true target positions x, y is also presented.

Table 5.3: Experiment 3 – RMSE on the test dataset. The ± values represent the standard deviation across trajectories. The combined RMSE is omitted, but is clearly the lowest for the IMM filter. Boldface marks the best results in each category.

	RMSE Position	RMSE Velocity	RMSE Acceleration
Measurement	226.32 ± 32.95	–	–
IMM	**152.04 ± 46.11**	**69.49 ± 34.47**	**30.22 ± 10.01**
PF	379.66 ± 337.77	90.35 ± 49.49	37.86 ± 8.47
NF-DPF	199.78 ± 60.81	175.39 ± 97.07	42.91 ± 9.12

In the accelerating case, the IMM and NF-DPF are able to produce more accurate positional predictions, then naively using the radar measurements, while the PF does not. All models, but specifically the NF-DPF struggle in predicting the velocity. All filtering techniques exhibit large standard deviations between runs.

As in Experiment 1 and 2, the Euclidean RMSE over the test dataset is illustrated using boxplots for each filtering technique and dimension. The boxplots for the positional RMSEs, including the measurements and all filtering techniques, are shown in Figure 5.9. The corresponding boxplots for the velocity RMSEs and acceleration RMSEs are shown in Figure 5.10 and Figure 5.11, respectively.

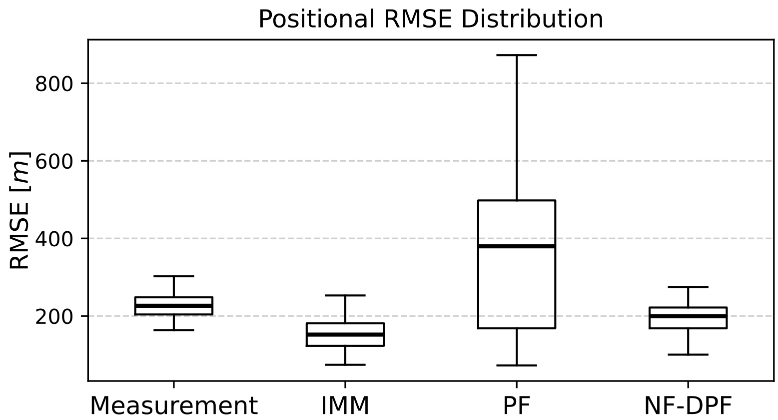


Figure 5.9: Boxplot of the positional RMSE distribution for the measurements, IMM, PF and NF-DPF for Experiment 3.

The PF has a notably large spread and high median in positional RMSE. The NF-DPF and IMM perform comparable, with slightly lower RMSEs than the measurement.

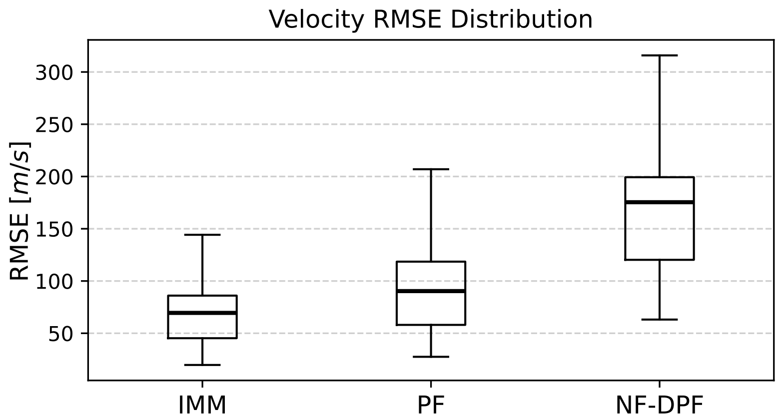


Figure 5.10: Boxplot of the velocity RMSE distribution for the IMM, PF and NF-DPF for Experiment 3.

When evaluated on velocity, the NF-DPF has both the largest spread and highest median. The PF performs slighlty better than the NF-DPF, exhibiting a narrower spread but a similarly elevated error. The IMM performs the very best, but it is still concluded that all filtering techniques have a hard time estimating the velocity of the targets in Experiment 3.

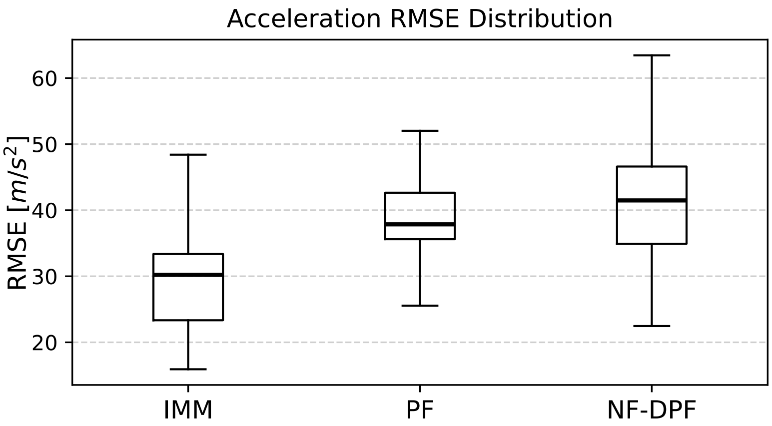


Figure 5.11: Boxplot of the acceleration RMSE distribution for the IMM, PF and NF-DPF for Experiment 3.

Acceleration estimation proves challenging across all filtering techniques. The NF-DPF exhibits the highest median and largest spread, indicating significant difficulty in capturing second-order dynamics. It appears that the NF-DPF prioritizes achieving acceptable positional estimates over accurate velocity and acceleration estimation. While this may partially reflect the weighting used in the training loss, it is more likely a consequence of the hard-to-learn acceleration dynamics of the trajectories in the dataset. This will be further discussed in Section 5.3.3.

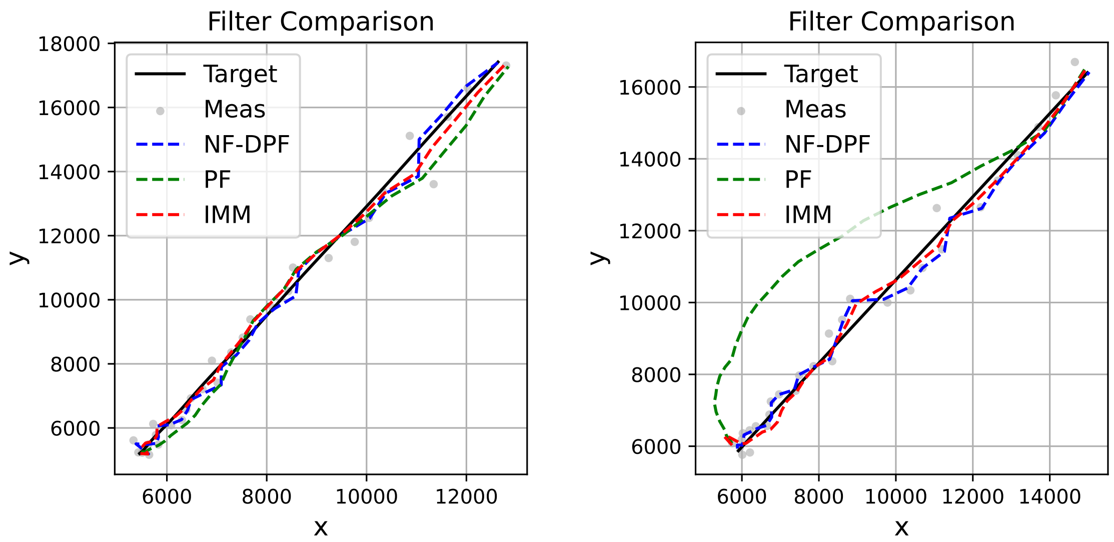
5.3.2 Experiment 3 – Qualitative Results

As in Experiment 1 and 2, a few representative trajectories from the test dataset are, together with the positional predictions from the three filtering techniques; IMM, PF and NF-DPF, illustrated in Figure 5.12. The radar measurements, converted to Cartesian coordinates are also included.

The IMM and NF-DPF exhibit similar positional predictions, although the NF-DPF appears more reactive. As shown in Subfigure 5.12b from Experiment 3, the NF-DPF becomes highly measurement-driven, relying on the Proposal NF to shift particles toward the radar measurements and on the resampling step to redistribute them around these observations. This behaviour suggests that the NF-DPF does not learn deeper motion patterns beyond directly reacting to the measurements, as opposed to Experiment 1 and 2 where the predicted trajectories were more smooth.

As shown in Subfigure 5.12b, the PF exhibits a prolonged period of poor predictions. The most likely cause of this failure is the orientation of the first two radar measurements, which suggest that the target is moving straight toward the top-left of the figure. In contrast, both the IMM and NF-DPF recover from this more quickly.

Figure 5.13 shows the effect of the Proposal NF and Dynamic NF, as before, through a contour line that encapsulates 85% of the particles, along with the average velocity of the particles. The figure illustrates a single time step that represents the general



(a) A representative accelerating target from the accelerating target dataset.

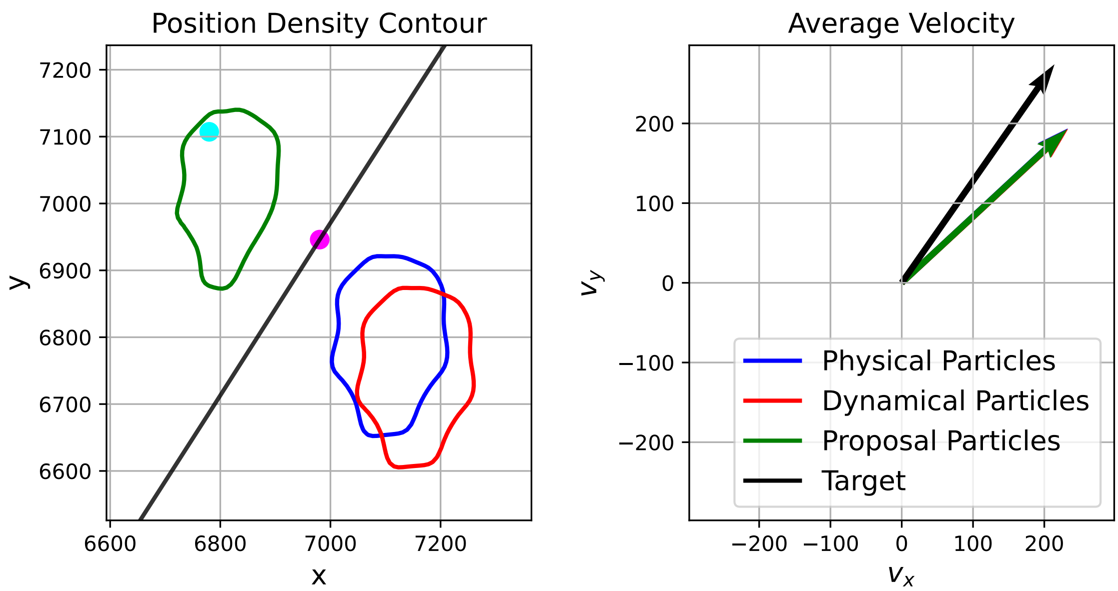
(b) A representative accelerating target from the accelerating target dataset.

Figure 5.12: Two representative trajectories from the accelerating target dataset used in Experiment 3: One where all filtering techniques produce reasonable trajectories (a) and one where the PF fails (b) probably due to some hard-to-recover-from measurements in the beginning of the trajectory.

pattern, but not necessarily what occurs at every measurement. Even though the NF-DPF in Experiment 3 includes additional acceleration terms in the state vector, the corresponding contour plot has been omitted for consistency with Figures 5.4 and 5.8, and also because, similar to the average velocity, the transformations had no observable effect on the acceleration vector.

From Subfigure 5.13a, it is clear that neither of the NFs performs any major transformation of the distribution, aside from shifting the position of the entire particle cloud. The Dynamic NF applies the smallest transformation, solely translating the particles. The Proposal NF, however, introduces a slight change in density and then strongly shifts the particles toward the measurement. In contrast to the trained NF-DPFs in Experiment 1 and Experiment 2, the Proposal NF, and in return, the entire NF-DPF, appears to over-fit to the measurements, resulting in the reactive trajectories observed in Figure 5.7.

Subfigure 5.13b illustrates the negligible effect the normalizing flows have on the velocity vector. As briefly discussed in Section 5.3.1, this could be due to the NF-DPF prioritizing positional accuracy over velocity when the position predictions are uncertain.



(a) The Dynamic NF are shifting the particles slightly (red contour). The Proposal NF moves the particles towards the measurement (green contour), accidentality over-fits to it and misses the true position (purple dot).

(b) The average velocity from all particles after Motion Update, Dynamic NF and Proposal NF. The NFs have no effect on the velocity vector. The black arrow is the true target velocity vector.

Figure 5.13: The positional contour, and average velocity from all particles after Motion Update Model (blue), Dynamic NF (red) and Proposal NF (green), illustrated at a single time step from a representative accelerating target trajectory.

5.3.3 Experiment 3 – Summary

Experiment 3 demonstrates that the NF-DPF struggles with high-dimensional, accelerating targets. Although it achieves better positional accuracy than the radar measurements and outperforms the PF, it falls short of the IMM across all metrics. The PF shows the worst and most unstable performance, particularly in position. Qualitative analysis reveals that the Proposal NF tends to over-fit to noisy measurements, resulting in reactive trajectories, while both flows fail to effectively correct the velocity. This suggests that the NF-DPF prioritizes positional fitting under uncertainty, possibly at the expense of higher-order motion estimation.

The performance drop in Experiment 3 is likely due to the increased complexity of the 6-dimensional state vector. With a limited number of particles, the NF-DPF cannot effectively capture the additional degrees of freedom, resulting in reduced generalization. As a consequence, the model compensates by reactively fitting to the position measurements, without leveraging the acceleration states. This suggests that, under the current configuration, the problem is too complex for the NF-DPF to solve accurately given the amounts of particles used in the model.

5.4 Cross-Dataset Evaluation of NF-DPF

To evaluate the generalization ability of the NF-DPF, each model was evaluated not only on the dataset it was trained on, but also on the two remaining datasets representing other target dynamics. The goal of this cross-evaluation is to investigate whether a model trained under one type of target motion (e.g., constant velocity) can perform well on unseen regimes, such as turning or accelerating trajectories. Table 5.4 presents the positional RMSEs obtained by each model across all datasets.

Table 5.4: Positional RMSE of NF-DPF models evaluated across datasets. No model generalizes well across all target types.

	CV Data	Turning Data	Accelerating Data
CV Model	**17.44 ± 3.79**	27.52 ± 14.50	210.26 ± 30.18
Turning Model	19.01 ± 3.43	**18.91 ± 3.77**	265.47 ± 58.76
Accelerating Model	66.27 ± 219.68	50.55 ± 49.77	**199.78 ± 60.81**

As expected, each model performs best on the dataset it was trained on. The model trained on turning targets shows some ability to generalize to constant velocity data, achieving performance close to the CV model. This is likely because the turning dataset also includes non-turning trajectories, allowing the model to learn both types of behaviour. In contrast, the models trained on CV and accelerating data perform poorly outside their respective domains. Notably, the accelerating model generalizes the worst, with significantly higher RMSEs when evaluated on the CV and turning datasets. This supports the findings in Experiment 3, where the NF-DPF struggled with the added complexity of the 6-dimensional state vector.

5.5 Computational Viability

The following section includes the results obtained from the computational viability measurements presented in Section 3.4.2. With the same number of particles across the PF and NF-DPF, the PF would run faster since it is a stripped-down version of the NF-DPF. However, to ensure a fair comparison, the particle count in the PF was adjusted to represent a well-tuned filter: 400 particles for Experiments 1 and 2, and 1600 particles for Experiment 3. Even with these adjustments, the NF-DPF still outperformed the PF across all experiments. The NF-DPF used 100, 100 and 400 particles, respectively, with the 400 being the computational limit due to the resampler. This resampler memory bottleneck is revisited in the discussion. The computational viability was analysed by considering execution time and complexity of the filtering techniques across the experiments.

5.5.1 Execution Time

To reiterate, the execution time of a target tracking algorithm refers to the time required to compute the posterior state estimate, assuming the prior has already been calculated. For the filtering techniques implemented in this thesis, this corresponds to the time from when a radar measurement is made available, to the time where the state prediction is made. This processing time must be significantly shorter than the interval between measurements, $\Delta t_{\text{radar}} = 1$ second. The execution times in milliseconds for all three filtering techniques, across all three models, are presented in Table 5.5, evaluated on both an L40S GPU and a Intel Xeon CPU.

Table 5.5: Mean, standard deviation, and peak time in milliseconds (ms), from measurement to state prediction. Evaluated over the 800 trajectories in the test dataset.

	Experiment 1	Experiment 2	Experiment 3
		GPU	
IMM	**0.17 ± 0.00 (0.19)**	**0.18 ± 0.00 (0.21)**	**0.17 ± 0.00 (0.19)**
PF	0.35 ± 0.01 (0.41)	0.36 ± 0.01 (0.43)	0.36 ± 0.01 (0.39)
NF-DPF	2.22 ± 0.08 (2.90)	2.34 ± 0.07 (2.61)	2.24 ± 0.05 (2.55)
		CPU	
IMM	**0.22 ± 0.07 (0.87)**	**0.21 ± 0.01 (0.25)**	**0.21 ± 0.00 (0.21)**
PF	0.48 ± 0.20 (2.35)	0.47 ± 0.10 (1.13)	0.89 ± 0.07 (0.95)
NF-DPF	2.01 ± 0.09 (2.62)	2.01 ± 0.13 (2.77)	1325.53 ± 1.67 (1354.20)

The IMM is considerably faster than all other filtering techniques. In contrast, particle-based methods such as the PF and NF-DPF are inherently more computationally demanding, as they require maintaining and updating a large set of particles at each time step. This overhead is further amplified in the NF-DPF, where each particle must be propagated through two separate normalizing flow models.

While the NF-DPF performs efficiently on the GPU, with processing times consistently below 3 milliseconds, it exhibits a severe performance bottleneck on the CPU in Experiment 3. This is due to the increased state dimensionality, which necessitates a larger particle set, and the model's poor tracking performance in this setting, which likely triggers resampling at every time step.

The overall execution time of the NF-DPF on the GPU is comparable to that on the CPU for smaller particle sets, like those in the first and second experiment. This is because the benefit of GPU acceleration is only realized when the particle number, or network sizes are sufficiently large. For smaller setups, the overhead associated with data transfer, and synchronization can outweigh the computational gains from the GPU.

In addition to the execution time from measurement availability to state prediction, the time required for a full iteration, that is one complete filtering step, was also recorded. This duration must also remain below the radar update interval, Δt_{radar}. The results for all filtering techniques, across all experiments, on both CPU and GPU, are presented in Table 5.6.

Table 5.6: Mean, standard deviation, and peak time (ms) for a full filter iteration (prediction and update). Evaluated over 800 trajectories.

	Experiment 1	Experiment 2	Experiment 3
		GPU	
IMM	**0.37 ± 0.01 (0.40)**	**0.38 ± 0.01 (0.44)**	**0.36 ± 0.01 (0.41)**
PF	7.15 ± 1.71 (14.81)	7.56 ± 1.17 (14.20)	57.85 ± 13.80 (103.05)
NF-DPF	6.83 ± 0.50 (8.41)	8.48 ± 0.56 (10.25)	11.84 ± 1.69 (17.76)
		CPU	
IMM	**0.45 ± 0.07 (1.12)**	**0.44 ± 0.01 (0.47)**	**0.44 ± 0.01 (0.44)**
PF	42.43 ± 45.66 (369.10)	90.49 ± 89.99 (949.94)	6123.98 ± 123.81 (6859.79)
NF-DPF	5.41 ± 0.41 (7.44)	6.28 ± 0.57 (8.99)	6148.88 ± 12.10 (6214.98)

The PF is consistently slow, with particularly high execution times on the CPU. In Experiment 3, its runtime exceeds 6 seconds per iteration, making it unsuitable for real-time applications. The NF-DPF demonstrates competitive performance on the GPU but suffers from the same CPU bottlenecks in Experiment 3 due to the high particle count and frequent resampling. Only the IMM meets real-time constraints, achieving sub-millisecond execution times with practically no variation across experiments. This is expected, as the IMM relies on closed-form Kalman updates and does not require iterative procedures, such as the Entropy-Regularized Optimal Transport Resampling used in the PF and NF-DPF.

5.5.2 Mathematical Complexity

In this section, the time and memory complexity of the three filtering techniques is outlined. Let n denote the number of particles, h the hidden size of the normalizing flow networks, L the number of flow layers, and r the number of models in the IMM filter.

For the PF, the primary computational cost arises in the optimal transport (OT) resampling step. This introduces a per-step time complexity of $\mathcal{O}(n^2)$ due to pairwise cost computations and iterations in the Sinkhorn algorithm. The memory complexity is dominated by the $\mathcal{O}(n^2)$ cost of storing the OT-related matrices (e.g., cost matrices, transport plans). The memory required for storing the particles themselves is only $\mathcal{O}(n)$.

For the NF-DPF, two additional sources of cost are introduced: the proposal and dynamics models, both implemented as conditional normalizing flows. Evaluating these models adds a cost of $\mathcal{O}(nhL)$ per time step. Combined with OT resampling, the total per-step time complexity at inference becomes $\mathcal{O}(n^2 + nhL)$. During training, memory complexity scales as $\mathcal{O}(n^2 \times \text{batch size} \times \text{number of Sinkhorn iterations} \times \text{number of time steps})$.

For the IMM filter, each time step involves r parallel Kalman filter updates and model probability mixing. Each Kalman update has time complexity $\mathcal{O}(1)$ due to matrix inversion and multiplication, resulting in a total per-step complexity of $\mathcal{O}(r)$. The memory complexity is $\mathcal{O}(r)$, corresponding to the storage of r state vectors and covariance matrices. With a small number of models and moderate state dimension, the IMM is significantly more efficient than both the PF and NF-DPF.

These theoretical complexities are well reflected in the execution time measurements reported in Section 5.5.1. Worth emphasizing is that the complexity of the NF-DPF is largely due to the OT-resampler.

5.6 Results Summary

Through Experiments 1-3, it is clear that IMM exhibits the most robust performance. The NF-DPF shows promising results with the 4-dimensional state vector and most notably with the turning targets of Experiment 2, where it outperforms the IMM. The limits of the NF-DPF becomes apparent in Experiemnt 3. When introduced to both accelerating targets and a 6-dimensional state vector, tracking performance degrades significantly. The PF performs the worst overall, often producing volatile and physically implausible trajectories.

The NF-DPF shows strong potential when the state space is low-dimensional and the number of particles is sufficient. However, in the 6-dimensional setting, when the degrees of freedom increase, the filter struggles due to the limited particle number, a restriction placed by the computational budget. It also tends to place to much confidence in the radar measurements when the target dynamics are harder to learn. This is especially clear in Experiment 3, where the Dynamic and Proposal NFs fails to update the velocity and acceleration components in any meaningful way.

6

Discussion

In this Chapter, a discussion is carried out regarding the outcome of the NF-DPF modelling, and the experimental results, in relation to the aim presented in Section 1.3. The goal was to compare the three filtering techniques introduced in the thesis, in relation to their accuracy, uncertainty, and computational efficiency. Through these comparisons, the viability of normalizing flow-based differentiable particle filters in complex target modelling contexts is to be determined. Further, several areas that warrant additional consideration are discussed, including why the NF-DPF does not generalize well to accelerating targets, the modelling choices made in the NF-DPF, and the metrics used to evaluate the filters. Special attention is given to how performance is affected as the state dimension increases, particularly when acceleration components are added.

6.1 Filter Performance

When comparing the performance of the NF-DPF to the PF and IMM, the results presented in Chapter 5 highlight the potential of the NF-DPF, particularly in Experiment 2, where the target dynamics are poorly captured by the IMM filter. By observing the representative trajectories with the positional predictions from the filtering techniques in Experiment 2, it becomes clear that the NF-DPF produces plausible trajectories. The NF-DPF both anticipates and responds to turns slightly faster than the IMM and PF. This adaptability is further observed in the contour plots, where the Dynamic NF spreads out the particles at each time step to remain prepared for potential changes in motion. Further, by considering the Euclidean RMSE in position and velocity for Experiment 2, the NF-DPF outperforms all other models, in all dimensions and with the lowest standard deviations. This indicates a more accurate and stable algorithm, for low-dimensional target modelling when, compared to the IMM, when evaluated on the chosen measurements and experiments.

Whether the NF-DPF's marginal performance over the IMM can be fully attributed to the NF-DPF is open for discussion. One could argue that, since the IMM exhibits a higher positional RMSE in Experiment 2 compared to Experiment 1, the apparent advantage of the NF-DPF may primarily stem from the IMM's degraded performance, rather than from a substantial improvement in the NF-DPF itself. This interpretation is supported by the fact that the NF-DPF also shows increased positional RMSE in Experiment 2. However, an important distinction must be made between the two models: the IMM is limited to a fixed set of predefined filters, in contrast, the NF-DPFs learnable components allow it to adapt to the turning targets in the training data and therefore generalize to a broader range of target behaviours. This adaptability likely enables the NF-DPF to better cope with dynamic variations, thereby maintaining more consistent performance even when absolute RMSE increases across experiments.

Although the results for Experiment 2 are promising when considered in isolation, additional factors must be taken into account before a final conclusion can be drawn. Firstly, when the two additional experiments are examined, particularly Experiment 3, it becomes evident that the NF-DPF is unable to outperform the IMM. Both the PF and NF-DPF perform significantly worse than the IMM in terms of RMSE across all dimensions. This is further observed in the representative trajectories presented under Experiment 3, where the IMM creates more physically plausible target estimates. Although this might be attributed to how suitable the data is for the IMM to model, as compared to the data in Experiment 2, the IMM still has to be given some credit. The fact that the IMM filter remains fixed across all experiments, while the NF-DPF is effectively retuned for each dataset, strengthens the case in favour of the IMM.

6.2 Computational Viability

No matter how much effort and consideration was put into the tuning of the NF-DPF it was still not able to outperform the IMM with a 6-dimensional state. The computational limitations of the NF-DPF, evident in the restriction to 400 particles, made it too difficult for the filter to accurately approximate targets that utilizes 6 dimensions. This particle limitation warrants further discussion.

In practice, it is widely accepted that significantly more than 400 particles are needed for 6-dimensional state spaces in particle filters. However, this restriction arose from the choice of the OT-resampling scheme, which is computationally demanding. Specifically, this is due to the need to save gradients of the optimal transport plan matrix π in Eq. (2.56), solved by the Sinkhorn algorithm. Consequently, a $n \times n$ matrix of gradients must be stored for up to 20 iterations at each time step and for every trajectory in the batch. With a particle count of 400, this equates to gradients amounting to gigabytes of memory. It is important to note that this only applies during training since no gradients need to be saved during inference, where only a single matrix must be defined, which is then continuously overwritten. This memory constraint calls into question the practicality of iterative resamplers, particularly those relying on a $n \times n$ transport plan. Although other resamplers, such as the Soft resampler, were explored, they did not yield results as accurate as the OT-resampler in low-dimensional cases. The question of the number of dimensions that can be effectively estimated using a max of 400 particles has been a recurring theme throughout this project, and is therefore discussed in detail in Section 6.3.

The execution times for all filters, when run on the GPU, were sufficiently fast in all experiments to meet the requirements for on-line execution. Although the implementations were developed in Python, which is not known for high-performance computing, the observed runtimes did not justify reimplementation in a more performance-oriented language. Nevertheless, for real-time or embedded applications, it may be beneficial to optimize key components or translate the code to languages such as *C++* or *Rust*.

6.3 Accelerating Targets

The NF-DPF was found to generalize poorly to accelerating targets, where acceleration is explicitly included in the state vector. The problem of estimating acceleration in tracking is well known and stems from several challenges. Most notably, the measurement model in this study provides no direct observation of acceleration, as the radar measurement vector $Y_t = (r, \theta, v_r)$ lacks any explicit acceleration terms. However, this limitation alone does not fully account for the NF-DPF's underperformance in Experiment 3. Despite relying on the same 6-dimensional state vector that includes acceleration components, the IMM was still able to produce more reliable state estimates.

This suggests that the NF-DPF struggles not solely due to missing acceleration information, but rather due to how it handles the underlying state components. Unlike the IMM, which adjusts the acceleration terms based on the spread of previous positions, the NF-DPF estimates the state through a weighted combination of particles. These weights are computed using Eq. (4.17) and were found to be primarily influenced by the measurement likelihood, as its magnitude was larger than that of the other contributing terms.

Additionally, in the Motion Update Model, the stochastic noise added to explore the acceleration components a_n and a_t does not immediately affect the position or velocity. Since acceleration only influences the target dynamics over time, its effect on the observable parts of the state is delayed. This breaks the connection between the updated acceleration values and the likelihood term $p(Y_t \mid X_t)$ used in the weight update. As a result, particles with very different acceleration states but similar positions and velocities are assigned nearly identical weights. When resampling is performed directly after the weight update, the added acceleration noise has no measurable effect and is essentially ignored. This makes it difficult for the NF-DPF to preserve or optimize the acceleration components, leading to poor estimation in state dimensions that are not directly constrained by the measurement model.

One way to address this would be to rethink how exploration is introduced during the Motion Update Model. The noise in this model has two distinct purposes: it should capture the process noise of the target and also allow for exploration in the NF-DPF. Preferably, this exploration should be done in a more realistic way, by first adding noise to the state and then updating it through the naive motion model. For example, with this setup, particles that receive large velocity noise will naturally move further, which is not necessarily the case if the state is updated before the noise is added. This would also allow the acceleration noise to be properly incorporated, as it would have time to influence the trajectory. This approach, however, is not possible in the current setup. The log-density of the noise in Eq. (2.70), which is needed for the weight update, only has an analytical solution when the noise is independent across state dimensions. This restriction limits what the Motion Update Model can represent, since realistic exploration would require dependent noise. These types of structured perturbations cannot be used, which in turn limits the filter's ability to explore high-dimensional states in a physically meaningful way.

6.3.1 Markov Property

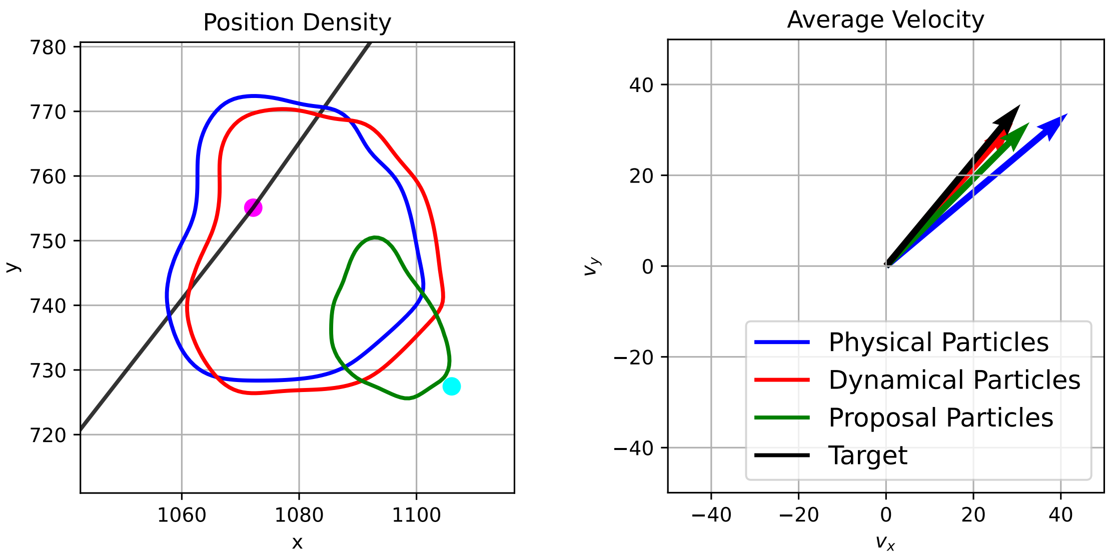
Another contributing factor to the increased difficulty of modelling the accelerating targets in Experiment 3 may lie in the structure of the underlying target dynamics themselves. In particular, the trajectories used in this experiment do not strictly follow a Markov process, as the time-varying mass $m(t)$, which directly influences the system dynamics through thrust and drag in Eq. (3.4), is not included in the state vector. Neither is the time t. As a result, the evolution of the state cannot be fully determined from the previous state alone, thereby violating the Markov property.

Since the filtering techniques employed in this thesis rely on the assumption of Markovian dynamics, an assumption introduced by the state-transition model through the derivation of the Chapman-Kolmogorov equation in Eq. (2.6). This structural mismatch may impair the filters' ability to accurately model and predict the system's behaviour. This limitation may have effected the performance under the 6-dimensional conditions of Experiment 3, where modelling errors were more pronounced. This may partly explain why the data in Experiment 3 is, in contrast to the first two experiments, more difficult to model effectively.

6.3.2 State Dimensionality

To investigate the role of state dimensionality, another experiment was made with extending the state vector of the NF-DPF to 6 dimensions and applying it on the datasets from Experiment 1 and Experiment 2. The quantitative results, in terms of RMSE, remained largely unchanged, however the qualitative results revealed structural behaviour in the transformations of the particles.

In Figure 6.1, the contour plots for the NF-DPF with a 6-dimensional state vector trained on the Experiment 1 – Constant Velocity Target dataset are presented. Subfigure 6.1a notably shows that the Dynamic NF primarily shifts the entire particle cloud rather than altering its internal structure. These results are consistent with the pattern observed in Experiment 3 – Accelerating Target, where the Dynamic



(a) The positional contour from all particles after Motion Update, Dynamic NF and Proposal NF

(b) The average velocity from all particles after Motion Update, Dynamic NF and Proposal NF.

Figure 6.1: Positional contour and average velocity of the particle cloud after the Motion Update (blue), Dynamic NF (red), and Proposal NF (green), shown at a representative time step from a trajectory in Experiment 1 – Constant Velocity Target when running NF-DPF with a 6-dimensional state vector. In both subfigures, the true target position (black) is indicated. For the positional contour (left), the radar measurement (cyan) and the ground-truth position (magenta) are also shown. The legend is shared across both subfigures. The physical and dynamical are under the proposal and not the true one.

NF fails to significantly reshape the particle density. Additionally, the Proposal NF frequently moves the particles closer to the measurement at each time step. This behavior closely resembles that of a bootstrap Particle Filter (PF), particularly in Experiment 1, where particles are repeatedly pulled toward observations, leading to abrupt deviations that do not align with the expected target dynamics. In effect, the Proposal NF behaves more like a measurement-driven corrector than a learned probabilistic model of the posterior. However, in contrast to Experiment 3, Subfigure 6.1b shows that the NF-DPF successfully transforms the higher-order state components, as reflected by noticeable changes in average velocity across successive normalizing flow layers. This suggests that the model is capable of capturing complex dynamics, even in a 6-dimensional state space.

This experiment, together with Experiments 1-3, point toward the conclusion that the NF-DPF is more suitable for problems with lower state dimensionality, and therefore fewer degrees of freedom. The NF-DPF implementation by Chen et al. [16], which served as a key inspiration for this thesis, strengthens this conclusion. The NF-DPF in their disk localization experiment operates on a very low-dimensional state. In their case, the state is 2-dimensional and only represents the position of the disk. Furthermore, their motion update model differs significantly from the approach presented here. The disks velocity vector is assumed to be known and remain within a constant range throughout the trajectory, resulting in a process noise that is constant and straightforward to tune. This enables the motion update model alone to yield accurate approximations of the prior distribution, reducing the reliance on the dynamic and proposal normalizing flows. The combination of low dimensionality and known velocities makes the problem substantially easier. With a low dimensionality, the issue of excessive resampling is also avoided, which have been prominent in this implementation, due to the need to spread particles more broadly in higher-dimensional state spaces. As this thesis builds partially on the success of Chen et al., it is important to note that the results are not directly comparable, given the substantial difference in degrees of freedom.

The mechanism described above, together with the Dynamic NF inability to reshape the density estimation, is the likely reasons behind the NF-DPF's underperformance in Experiment 3. The filter does not manage to learn the dynamics and instead relies on the Motion Update Model, together with excessive resampling, effectively running as a bootstrap PF.

6.4 Generalization and Limitations of Setup

While discussing filtering accuracy, it is also worth concluding that the current implementations and tunings are far from optimal. All techniques are defined by a wide range of design choices and parameters which has to be tuned. For example, the performance of the NF-DPF depends on the noise modelled in the Motion Update Model, training parameters like the learning rate, optimizers, dataset sizes, and batch sizes, architectural choices like the depth, width, type and number of flows, and all parameters which govern the resampling. As with the training of any machine learning-based model, some design choices have to be assumed, as the entirety of the solution space cannot be searched. This results in a close to guaranteed sub-optimal solution. The argument also holds for the IMM and PF, as these are likewise governed by a wide range of parameters. All of this must be considered while reviewing the results.

It is also worth discussing the implications of some of the methodological choices in Chapter 3, and how they may have effected the results. Firstly, for the evaluation of the filtering techniques, only one instance of each filter was used. Optimally, a greater sample of individual model instances, trained on different instances of the same dataset, would provide a better ability to draw general conclusions on the performance on the models. This was not possible, due to the computational limitations of the thesis, where training was time-consuming.

Furthermore, all data used in this thesis was synthetically generated. This raises several important questions. The first is whether the filtering techniques will generalize effectively to real-world target data. This question is highly dependent on the specific application context, as the data in this thesis was designed to represent highly-manoeuvring hypersonic missiles. The second consideration concerns the risk of implicit data leakage. Since the characteristics of the synthetic data are known and may influence model development, there is a possibility that the filters have been over-fitted to these specific dynamics. This further complicates the question of generalizability, as it remains uncertain how the filters would perform on unseen, real-world scenarios that deviate from the simulated conditions.

These limitations tie directly into how the NF-DPFs were trained and evaluated. Specifically, three separate NF-DPF models were trained on different datasets corresponding to distinct motion regimes: constant velocity, turning, and accelerating targets. This setup was selected primarily as a proof of concept, allowing the model to specialize in each type of motion. However, it restricts the ability to draw conclusions about the feasibility of a general-purpose NF-DPF model trained across a broader range of target dynamics. The results cannot directly inform whether a single model could handle transitions between regimes or adapt to unseen dynamics during inference. Additionally, differences in the Motion Update noise, state dimensionality, and loss function design between the experiments further limit comparability across models and hinder generalization. These differences, though practically motivated, mean that the NF-DPFs trained in this study are tightly coupled to their respective data regimes.

Despite this, some evidence of generalization can still be observed, as presented in Table 5.4. Notably, the NF-DPF trained on the turning-target dataset performed well, even though a third of the included trajectories did not involve any turning and shared the same underlying SDE as the constant-velocity case. This suggests that the model was able to handle a certain degree of internal variation within the dataset. This was further explored in the cross-dataset evaluation in which the NF-DPF trained on the turning-target dataset was evaluated on test trajectories from the constant-velocity dataset. The resulting RMSEs were comparable to those obtained in Experiment 1, indicating that some degree of generalization is possible, even when the filter is trained on a more complex motion regime. This result is encouraging, but limited, and underscores the importance of future work on unified, generalizable NF-DPF architectures.

6.5 Evaluation Metrics

A reasonable point of discussion is the primary use of Euclidean RMSE as the evaluation criterion for the three filtering techniques (along with some qualitative analysis), and how this choice may have influenced the results. By only comparing the filters to each other, one can determine which filter performs the best in relative terms, but not how close that performance is to an optimal solution. To do that, one of two things would have been necessary.

First, the RMSE could have been compared to a theoretical lower bound derived from the data, such as the Posterior Cramér-Rao Lower Bound (PCRLB), available for some state-space models. However, this was not possible due to the non-linear dependence of the measurement noise on the state, which makes an analytical computation of the PCRLB intractable.

The other option would have been to use data with a known and tractable posterior distribution, enabling direct comparison between each filter's estimate and the true distribution. However, as noted in Section 3.4, this is not possible for the current datasets due to the non-Gaussian nature of the measurement model. As a result, evaluating how closely each filter approximates the true posterior is not feasible. A linear and Gaussian system, where the posterior is analytically tractable, would allow for such an evaluation and offer a clearer validation of the NF-DPF's performance. This was not pursued due to the limited scope of the thesis but remains a valuable direction for future work.

Further, the evaluations reliance on RMSE could be lowered by introducing additional evaluation metrics. One option that was considered was the Log-Likelihood (LL) of the measurements under the predicted state distribution. This could have added more nuance to the analysis and strengthened the conclusions. However, it was omitted due to challenges in fairly defining predictive likelihoods across all three filters. Other metrics that would allow for a consistent and fair comparison across the IMM, PF, and NF-DPF were not identified. Since the main objective of the thesis was to evaluate the viability of the NF-DPF relative to the PF and IMM, RMSE was deemed sufficient for that comparison.

7
Conclusion and Future Work

In this chapter, the final conclusions of the thesis are presented in relation to the original objectives. Additionally, several directions for future work are proposed, with particular emphasis on the continued evaluation and potential extension of the NF-DPF within the field of radar-based target modelling. Due to the limited scope of this thesis, these were left unexplored.

7.1 Conclusion

This thesis investigated the use of Normalizing Flow-based Differentiable Particle Filters (NF-DPFs) for radar-based target tracking in complex and diverse target environments, and compared their performance to that of traditional filtering techniques, namely Particle Filters (PF) and Interacting Multiple Model (IMM) filters. Across three experiments, covering constant velocity, turning, and accelerating targets, the filters were evaluated in terms of accuracy, reliability, and computational viability.

The results show that the NF-DPF offers promising performance for low-dimensional state spaces, particularly in Experiment 2, where it outperformed both the PF and IMM filter in tracking turning targets. This suggests that the NF-DPF can learn to capture non-linear dynamics when the system complexity is manageable. In contrast, the NF-DPF struggled in Experiment 3, where acceleration terms increased the state dimensionality to six. The filter was unable to effectively incorporate acceleration information, leading to degraded performance relative to the IMM. Several factors contributed to this decline, including the NF-DPF's reliance on independent noise in the Motion Update Model, the delayed influence of acceleration on the likelihood term, and the practical constraints imposed by limited particle count. From a computational standpoint, the NF-DPF is viable on GPUs, although its CPU performance degrades sharply with increased dimensionality and particle count. These limitations may be addressed through optimized implementations or alternative resampling strategies.

In conclusion, the NF-DPF is a flexible and learnable filtering method that shows strong potential for radar-based target tracking in moderately complex settings. However, in its current form, it faces clear limitations when applied to higher dimensional problems.

7.2 Future Work

As discussed in previous sections, the performance of the NF-DPF depends heavily on a wide range of design choices, including hyper-parameters for training, the structure of the normalizing flows, the noise in the Motion Update Model, and resampling parameters. A more systematic approach to hyper-parameter tuning could potentially improve performance. However, such an undertaking could prove to be very time-consuming, even proving to be futile.

Another promising direction is to train the NF-DPF on datasets containing a wider variety of target behaviours. As seen in Section 6.4 and Table 5.4, the model trained on turning targets generalized reasonably well to non-turning trajectories. Expanding this approach could improve robustness across diverse motion regimes. Additionally, by introducing more types of targets, additional cases where the NF-DPF outperforms the IMM could be identified. In particular, targets with abrupt velocity changes or sudden starts and stops may not be well captured by the constant velocity (CV) and constant acceleration (CA) filters used in the IMM, but could be better approximated by a flexible, learned filter such as the NF-DPF.

It would also be worthwhile to retrain the NF-DPF on the accelerating targets from Experiment 3, but with the current mass $m(t)$ included in the state vector. Although this would introduce additional degrees of freedom, which the filters have previously struggled to generalize to, any errors in the added mass component would likely remain confined to that variable rather than affecting the other states, as was the case with acceleration. Most importantly, this inclusion would make the system Markovian, which is assumed by the filtering frameworks.

The NF-DPF architecture itself could be revisited. The the use of RealNVP as the flow architecture has been assumed throughout, without comparison to more recent alternatives such as Masked Autoregressive Flows or Neural SDEs [62], [63]. Additionally, in cases where the predicted state diverges significantly from the observations over multiple time steps, it may be beneficial to introduce a reset mechanism. For instance, particles could be reinitialized around recent measurements if the positional RMSE exceeds three standard deviations for three consecutive time steps. Such a mechanism could also benefit standard PF implementations. Most importantly, the use of Entropy-Regularized Optimal Transport Resampling must be reevaluated against alternative differentiable resampling schemes. The particle restriction resulting from this computationally demanding resampler is the main reason for the degrading performance of the NF-DPF in 6-dimensional state spaces. Reducing the number of Sinkhorn iterations could provide some relief, but the core issue lies in the iterative nature and the $n \times n$ transport plan matrix. Consequently, an alternative resampling scheme must be employed.

Finally, a key recommendation is to retrain the NF-DPF on data generated from a system with a known and tractable posterior distribution. This would allow a direct comparison between the learned posterior and the true one, offering clearer insights into how well the NF-DPF, and the other filters, approximate the filtering distribution. While this would require simplifying the dynamics to a linear-Gaussian system, the insights gained could inform both filter development and evaluation in more complex settings.

Bibliography

[1] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation with Applications to Tracking and Navigation.* Wiley-Interscience, 2001.

[2] T. X. Hammes, "Lessons from the Ukraine Conflict: Modern Warfare in the Age of Autonomy, Information, and Resilience," *Center for Strategic and International Studies*, 2025. [Online]. Available: https://www.csis.org/analysis/lessons-ukraine-conflict-modern-warfare-age-autonomy-information-and-resilience

[3] "Artificial intelligence is going to make drone wars much more deadly. It's already started," *Business Insider*, 2025. [Online]. Available: https://www.businessinsider.com/ukraines-smart-drones-more-likely-hit-targets-2025-3

[4] Reuters, "How drone combat in Ukraine is changing warfare," *Reuters Graphics*, 2023. [Online]. Available: https://www.reuters.com/graphics/UKRAINE-CRISIS/DRONES/dwpkeyjwkpm/

[5] D. Svensson, *Target tracking in complex scenarios.* Chalmers Tekniska Hogskola (Sweden), 2010.

[6] S. S. Blackman and R. Popoli, *Design and Analysis of Modern Tracking Systems.* Boston, MA: Artech House, 1999.

[7] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[8] J. B. B. Gomes, "An Overview on Target Tracking Using Multiple Model Methods," Master's thesis, Instituto Superior Técnico, Lisbon, Portugal, 2008. [Online]. Available: https://www.dem.ist.utl.pt/poliveira/Ensino/MSc_2008b.pdf

[9] P. S. Maybeck, *Stochastic Models, Estimation, and Control*, ser. Mathematics in Science and Engineering. New York, NY: Academic Press, 1979, vol. 141.

[10] E. A. Wan and R. van der Merwe, "The Unscented Kalman Filter for Nonlinear Estimation," in *Proceedings of the IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium (AS-SPCC).* Lake Louise, Alberta, Canada: IEEE, October 2000, pp. 153–158.

[11] F. Gustafsson, "Particle filter theory and practice with positioning applications," *IEEE Aerospace and Electronic Systems Magazine*, vol. 25, no. 7, pp. 53–82, 2010.

[12] H. Sidenbladh, "Multi-Target Particle Filtering for the Probability Hypothesis Density," *International Conference on Information Fusion*, pp. 800–806, 2003.

[13] X. Chen and Y. Li, "An overview of differentiable particle filters for data-adaptive sequential Bayesian inference," 2023. [Online]. Available: https://arxiv.org/abs/2302.09639

[14] J. Li, X. Chen, and Y. Li, "Learning Differentiable Particle Filter on the Fly," *arXiv preprint arXiv:2312.05955*, 2023.

[15] R. Jonschkowski, D. Rastogi, and O. Brock, "Differentiable Particle Filters: End-to-End Learning with Algorithmic Priors," in *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

[16] X. Chen and Y. Li, "Normalizing Flow-based Differentiable Particle Filters," 2025. [Online]. Available: https://arxiv.org/abs/2403.01499

[17] T. Bayes, "An Essay towards solving a Problem in the Doctrine of Chances," *Philosophical Transactions of the Royal Society of London*, vol. 53, pp. 370–418, 1763, published posthumously by Richard Price.

[18] A. F. Filippov, "The Markov Property Revisited," *Mathematical Notes*, vol. 58, no. 1-2, pp. 77–83, 1995.

[19] A. N. Kolmogorov, "Über die analytischen Methoden in der Wahrscheinlichkeitsrechnung," *Mathematische Annalen*, vol. 104, no. 1, pp. 415–458, 1931.

[20] X. R. Li and V. Jilkov, "Survey of maneuvering target tracking. part i. dynamic models," *IEEE Transactions on Aerospace and Electronic Systems 39*, no. 4, p. 1333–1364, 2003.

[21] D. J. Higham, "An Algorithmic Introduction to Numerical Simulation of Stochastic Differential Equations," *SIAM Review*, vol. 43, no. 3, pp. 525–546, 2001.

[22] O. Spjuth, "Large-Scale Transformer-Based Multi-Target Tracking: Extending Transformer-Based MTT Methods to an Air-Surveillance Context," Master's thesis, Chalmers University of Technology, Gothenburg, Sweden, 2024, department of Mathematical Sciences, Division of Applied Mathematics and Statistics. [Online]. Available: https://www.chalmers.se

[23] D. Simon, *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches.* Wiley-Interscience, 2006.

[24] S. J. Julier and J. K. Uhlmann, "A New Extension of the Kalman Filter to Nonlinear Systems," in *Proceedings of AeroSense: The 11th International Symposium on Aerospace/Defense Sensing, Simulation, and Controls.* Orlando, FL, USA: SPIE, April 1997.

[25] S. Särkkä, "On Unscented Kalman Filtering for State Estimation of Continuous-Time Nonlinear Systems," *IEEE Transactions on Automatic Control*, vol. 52, no. 9, pp. 1631–1641, Sep. 2007.

[26] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation with Applications to Tracking and Navigation: Theory Algorithms and Software.* Wiley-Interscience, 2001.

[27] D.-J. Jwo, F. Chung, and K. Yu, "GPS/INS Integration Accuracy Enhancement Using the Interacting Multiple Model Nonlinear Filters," *Journal of applied research and technology*, vol. 11, pp. 496–509, 08 2013.

[28] A. Doucet, N. de Freitas, and N. Gordon, *Sequential Monte Carlo methods in practice.* Springer, 2001.

[29] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, 2002.

[30] J. S. Liu and R. Chen, "Sequential Monte Carlo methods for dynamic systems," *Journal of the American Statistical Association*, vol. 93, no. 443, pp. 1032–1044, 1998.

[31] N. Kühl, M. Goutier, R. Hirt, and G. Satzger, "Machine Learning in Artificial Intelligence: Towards a Common Understanding," *arXiv preprint arXiv:2004.04686*, 2020.

[32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016. [Online]. Available: https://www.deeplearningbook.org/

[33] ScienceDirect Topics, "Deep Neural Network - an overview | ScienceDirect Topics," https://www.sciencedirect.com/topics/computer-science/deep-neural-network, 2025, accessed: 2025-04-25.

[34] F. E. C. Léon Bottou and J. Nocedal, *Optimization Methods for Large-Scale Machine Learning*. Society for Industrial and Applied Mathematics, 2018, vol. 60, no. 2.

[35] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *International Conference on Learning Representations (ICLR)*, 2015. [Online]. Available: https://arxiv.org/abs/1412.6980

[36] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[37] R. Pascanu, T. Mikolov, and Y. Bengio, "On the Difficulty of Training Recurrent Neural Networks," *Proceedings of the 30th International Conference on Machine Learning (ICML)*, vol. 28, no. 3, pp. 1310–1318, 2013.

[38] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[39] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, 2014.

[40] X. Glorot and Y. Bengio, "Understanding the Difficulty of Training Deep Feedforward Neural Networks," *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 9, pp. 249–256, 2010.

[41] M. Zhu, K. Murphy, and R. Jonschkowski, "Towards Differentiable Resampling," *CoRR*, vol. abs/2004.11938, 2020. [Online]. Available: https://arxiv.org/abs/2004.11938

[42] A. Corenflos, J. Thornton, G. Deligiannidis, and A. Doucet, "Differentiable Particle Filtering via Entropy-Regularized Optimal Transport," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 2100–2111. [Online]. Available: https://proceedings.mlr.press/v139/corenflos21a.html

[43] J. Feydy, T. Séjourné, F.-X. Vialard, S. ichi Amari, A. Trouvé, and G. Peyré, "Interpolating between Optimal Transport and MMD using Sinkhorn Divergences," 2018. [Online]. Available: https://arxiv.org/abs/1810.08278

[44] A. Corenflos, J. Thornton, G. Deligiannidis, and A. Doucet, "Differentiable Particle Filtering via Entropy-Regularized Optimal Transport," 2021. [Online]. Available: https://arxiv.org/abs/2102.07850

[45] I. Kobyzev, S. J. Prince, and M. A. Brubaker, "Normalizing Flows: An Introduction and Review of Current Methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 11, pp. 3964–3979, 2021. [Online]. Available: https://arxiv.org/abs/1908.09257

[46] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, "Normalizing Flows for Probabilistic Modeling and Inference," *Journal of Machine Learning Research*, vol. 22, no. 57, pp. 1–64, 2021.

[47] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density Estimation using Real NVP," in *International Conference on Learning Representations (ICLR)*, 2017. [Online]. Available: https://arxiv.org/abs/1605.08803

[48] R. L. Burden and J. D. Faires, *Numerical Analysis*, 9th ed. Cengage Learning, 2010.

[49] C. M. Bishop, *Pattern Recognition and Machine Learning.* New York: Springer, 2006.

[50] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 1st ed. New York: Wiley-Interscience, 1999.

[51] N. Metropolis and S. Ulam, "The Monte Carlo Method," *Journal of the American Statistical Association*, vol. 44, no. 247, pp. 335–341, 1949.

[52] P. Liashchynskyi and P. Liashchynskyi, "Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS," 2019. [Online]. Available: https://arxiv.org/abs/1912.06059

[53] P. Karkus, D. Hsu, and W. S. Lee, "Particle Filter Networks with Application to Visual Localization," 2018. [Online]. Available: https://arxiv.org/abs/1805.08975

[54] T. A. Le, M. Igl, T. Rainforth, T. Jin, and F. Wood, "Auto-Encoding Sequential Monte Carlo," *arXiv preprint arXiv:1705.10306*, 2018.

[55] C. A. Naesseth, S. W. Linderman, R. Ranganath, and D. M. Blei, "Variational Sequential Monte Carlo," *arXiv preprint arXiv:1705.11140*, 2018.

[56] X. Chen, H. Wen, and Y. Li, "Differentiable Particle Filters through Conditional Normalizing Flow," 2021. [Online]. Available: https://arxiv.org/abs/2107.00488

[57] X. R. Li and V. P. Jilkov, "Survey of Maneuvering Target Tracking. Part I: Dynamic Models," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 39, no. 4, pp. 1333–1364, 2003.

[58] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *Proceedings of the 34th International Conference on Machine Learning (ICML)*, vol. 70, p. 214–223, 2017.

[59] A. Doucet, S. Godsill, and C. Andrieu, "On Sequential Monte Carlo Sampling Methods for Bayesian Filtering," *Statistics and Computing*, vol. 10, no. 3, pp. 197–208, 2000.

[60] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. Springer, 2009.

[61] R. Jonschkowski, D. Rastogi, O. Groth, M. R. Oswald, and I. Posner, "Differentiable Particle Filters: End-to-End Learning with Algorithmic Priors," *Journal of Machine Learning Research*, vol. 21, no. 70, pp. 1–35, 2020.

[62] G. Papamakarios, T. Pavlakou, and I. Murray, "Masked Autoregressive Flow for Density Estimation," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 2335–2344. [Online]. Available: https://papers.nips.cc/paper/6828-masked-autoregressive-flow-for-density-estimation

[63] B. Tzen and M. Raginsky, "Neural Stochastic Differential Equations: Deep Latent Gaussian Models in the Diffusion Limit," *arXiv preprint arXiv:1905.09883*, 2019. [Online]. Available: https://arxiv.org/abs/1905.09883

[64] J. O. Nichols, "Analysis and Compilation of Missile Aerodynamic Data - Volume 1 Data Presentation and Analysis," *NASA*, 1977.

[65] S. Hughes, "Atmospheric Flight Mechanics," NASA, Tech. Rep. NASA/SP-2013-217370, 2013, available at https://ntrs.nasa.gov/api/citations/20130003336/downloads/20130003336.pdf.

[66] International Organization for Standardization, "Standard Atmosphere," *ISO 2533:1975*, 1975.

A

Stochastic Differential Equation Parameters

The parameters used in the three SDEs to generate the trajectories in the datasets, and which are used consistently across all experiments, are presented in Table A.1. It is worth emphasizing that finding perfectly realistic values was not critical, as these parameters mainly determined the scale of the target state, which the models were expected to handle across a wide range of magnitudes. The values are presented for the purpose of experimental reproducibility.

Table A.1: SDE parameters presented by name, notation, chosen value and some notes.

Parameter and Notation	**Value**	**Notes / Source**
Drag Coefficient, C_D	0.2	In reality depends on velocity [64].
Reference Area, A	1 m^2	Relatively large estimate [65]
Initial Mass, m_0	4000 kg	Varies between missile types.
Burn Rate, $\dot{m}$	100 kg/s	Set to achieve acceleration of >10g.
Thrust, c	1000 m/s	Set to achieve acceleration of >10g.
Density of Fluid, ρ	0.5 kg/m^3	Air density at 5–6 km altitude [66].
Tangential Noise Amp., σ_t	200 m/s^2	Set to achieve visible wobbling.
Normal Noise Amp., σ_n	200 m/s^2	Set to achieve visible wobbling.
Wobble Frequency, ω	$X \sim \mathcal{U}(0.5, 5)$	Uniformly distributed.
Wobbling Amp., τ	$X \sim \mathcal{U}(5, 10)$	Uniformly distributed.

B

Interacting Multiple Model Filter Architecture

This section provides a detailed overview of the developed IMM filter, including the selected subfilters, the linearized measurement model, the transition probability matrix π, the blending technique, initialization procedure, and the evaluation setup. The IMM used the full 6-dimensional state vector $X = [x, y, v_x, v_y, a_x, a_y]$ and a 2-dimensional measurement vector $Y = [r, \theta]$. The model remained constant across experiments.

Chosen Filters

The IMM consists of five Extended Kalman Filters (EKFs), comprising of two Constant Velocity (CV) filters that use the state-transition matrix defined in Eq. 3.8, and three Constant Acceleration (CA) filters that use the matrix defined in Eq. 3.9. Each filter is uniquely defined by a scaling factor Q^i_{factor} for filter $i \in \{1, \dots, 5\}$, which multiplies the base process noise covariance matrix Q. This allows the IMM to include filters with varying levels of process noise and, consequently, varying levels of confidence in the dynamics. The scaled process noise matrix Q^i for filter i is given by:

$$Q^i = Q^i_{\text{factor}} \times Q = Q^i_{\text{factor}} \times \begin{bmatrix} \frac{dt^4}{4} & 0 & \frac{dt^3}{2} & 0 & \frac{dt^2}{2} & 0 \\ 0 & \frac{dt^4}{4} & 0 & \frac{dt^3}{2} & 0 & \frac{dt^2}{2} \\ \frac{dt^3}{2} & 0 & dt^2 & 0 & dt & 0 \\ 0 & \frac{dt^3}{2} & 0 & dt^2 & 0 & dt \\ \frac{dt^2}{2} & 0 & dt & 0 & 1 & 0 \\ 0 & \frac{dt^2}{2} & 0 & dt & 0 & 1 \end{bmatrix}. \tag{B.1}$$

Here, $dt = \Delta t_{\text{radar}}$ and denotes the time interval between filter updates, equal to the radar measurement time. The Q^i_{factor} values used for each filter are listed in Table B.1.

Table B.1: Filter configurations used in the IMM, showing the filter type and corresponding Q factor. Note how the Q factor increases quadratically to reflect growing process noise.

Filter i	1	2	3	4	5
Filter Type	CV	CV	CA	CA	CA
Q Factor	1	4	1	4	16

Linearized Measurement Model

The linearized measurement model H, used in all EKFs in the IMM, is defined as the Jacobian of the nonlinear measurement function $h(X)$ evaluated at the predicted state estimate $\hat{X}_{t|t-1}$:

$$H = \left.\frac{\partial h}{\partial X}\right|_{\hat{X}_{t|t-1}} = \begin{pmatrix} \frac{\hat{x}}{\sqrt{\hat{x}^2+\hat{y}^2}} & \frac{\hat{y}}{\sqrt{\hat{x}^2+\hat{y}^2}} & 0 & 0 & 0 & 0 \\ \frac{180}{\pi}\frac{\hat{y}}{\hat{x}^2+\hat{y}^2} & -\frac{180}{\pi}\frac{\hat{x}}{\hat{x}^2+\hat{y}^2} & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{B.2}$$

Here, the first row corresponds to the range measurement and the second to the bearing measurement (in degrees). The subscript t denoting the time step has been omitted for readability. This Jacobian uses the first two dimensions of the measurement vector, range and bearing:

$$h(X) = \begin{bmatrix} \sqrt{x^2+y^2} \\ \arctan(x,y) \end{bmatrix}.$$

If the radial velocity (range rate) v_r was used, the Jacobian H would need to be extended with an additional row derived from:

$$v_r = \frac{xv_x + yv_y}{\sqrt{x^2+y^2}}$$

Transition Probabilities

The model transition probabilities define the likelihood of switching between different filters in the IMM. To determine an appropriate transition probability matrix π, the diagonal elements were constrained to be equal, and a parameter search was performed. The optimal value was selected based on the lowest Euclidean RMSE in position. The best-performing configuration set the self-transition probabilities π^{ii} to 0.9, with all off-diagonal probabilities equally distributed. The final transition matrix is defined as:

$$\pi = \begin{bmatrix} 0.9 & 0.025 & 0.025 & 0.025 & 0.025 \\ 0.025 & 0.9 & 0.025 & 0.025 & 0.025 \\ 0.025 & 0.025 & 0.9 & 0.025 & 0.025 \\ 0.025 & 0.025 & 0.025 & 0.9 & 0.025 \\ 0.025 & 0.025 & 0.025 & 0.025 & 0.9 \end{bmatrix} \tag{B.3}$$

Blending

In the IMM framework, the final state estimate can be obtained either by selecting the most probable filter (switching) or by computing a probabilistic interpolation of all filters (blending). In this work, blending was used, as it consistently produced smoother and more accurate estimates. The blending approach combines the individual filter estimates based on their mode probabilities. As defined in Eq. (2.37), the overall state estimate is a weighted average of the filter-specific posterior states. The corresponding covariance, defined in Eq. (2.38), accounts for both the uncertainty within each filter and the dispersion between filters. Empirically, the blending method outperformed switching by providing more stable predictions.

Initialization

Each filter in the IMM was initialized using the first available measurement to estimate the position component of the state vector, $\hat{X}_{0|0}$. The velocity is initialized with a velocity of 100 m/s in the x-direction and zero velocity in the y-direction, i.e., $v_x = 100\,\mathrm{m/s}$ and $v_y = 0\,\mathrm{m/s}$. The acceleration components are initialized to 0. The initial state covariance matrix $P^i_{0|0}$ was chosen to be the same for all filters $i \in \{1, 2, \ldots, 5\}$, and is defined as:

$$P^i_{0|0} = \mathrm{diag}\Big(30^2,\ 30^2,\ 30^2,\ 30^2,\ 30^2,\ 30^2\Big) \tag{B.4}$$

This corresponds to a standard deviation of 30 in each state dimension (position, velocity, and acceleration). In practice, the covariance matrix converges quickly within a few time steps, which reduces the sensitivity of the filter to the exact initialization values.

Evaluation

The evaluation followed the description provided in Section 3.4, where the Euclidean RMSE in position, velocity and acceleration is calculated. The first 3 predictions from the IMM were omitted from the evaluation, as the filter was still converging.

C

Stochastic Differential Equation Initial Conditions

The initial conditions used to generate the trajectories for the three datasets are listed in Tables C.1-C.3, respectively. Although the bounds from which the initial conditions are uniformly drawn may appear narrow, the resulting data distribution is complex and challenging to model. The values are presented for the purpose of experimental reproducibility. For all cases, the accelerations a_t and a_n are initialized as 0 in the data generation process.

Table C.1: Initial conditions for the data generated from the SDE in 3.2.1, used for Experiment 1. The bound denotes the interval, from which the initial conditions are uniformly drawn. The initial velocity v_0 is noticeably higher than in the accelerating case, and is specifically chosen to prevent the targets from accidentally reaching a velocity of zero at any time step.

Parameter	Lower Bound	Upper Bound
x_0	500	1000
y_0	500	1000
v_0	40	50
ϕ_0	$\pi/5$	$\pi/4$

Table C.2: Initial conditions for data generated from the SDE in 3.2.2, used for Experiment 2. The bound denotes the interval, from which the initial conditions are uniformly drawn. The same reasoning as in Experiment 1 for the higher initial velocity v_0 applies for Experiment 2 as well.

Parameter	Lower Bound	Upper Bound
x_0	500	1000
y_0	500	1000
v_0	40	50
ϕ_0	$\pi/5$	$\pi/4$
T_{turn}	10.0	15.0
t_{turn}	3.0	18.0
u_{mag}	5	7.5

The turns are characterized not only by the start time t_{turn}, duration T_{turn}, and magnitude u_{mag}, but also by the probabilities p^1_{turn} for a first turn and p^2_{turn} for a second turn, conditional on the first turn occurring. The probabilities were set to $p^1_{\text{turn}} = 0.66$ and $p^2_{\text{turn}} = 0.5$, as to get the same amount of trajectories which turns 0, 1 and 2 times in the dataset.

Table C.3: Initial conditions for data generated from the SDE in 3.2.3, used for Experiment 3. The bound denotes the interval, from which the initial conditions are uniformly drawn. The range from which the initial position is drawn has been shifted to a higher magnitude to reduce the impact of target movement on the measurement noise, which would otherwise vary significantly in magnitude between the beginning and end of the trajectory.

Parameter	Lower Bound	Upper Bound
x_0	5000	6000
y_0	5000	6000
v_0	10	50
ϕ_0	$\pi/6$	$\pi/3$

D

Model Architecture and Training Configuration

This appendix provides detailed information about the model architectures and training configurations used in the experiments. It includes parameter settings for both the Particle Filter and the proposed Normalizing Flow-based Differentiable Particle Filter (NF-DPF).

D.1 Particle Filter

The Motion Update Model propagates the particles deterministically and adds some stochastic noise defined by a drawing a noise sample from a 0-mean Gaussian distribution with some covariance. This covariance is defined as a constant parameter α and some velocity-magnitude-dependant parameter β as $\sigma = \alpha + \beta\|V\|$, where $\|V\|$ denotes the magnitude of the particle's velocity. The parameters α and β for each noise component were selected via grid search to minimize the positional Euclidean RMSE. The optimal values are reported in Table D.1. The number of particles used is presented in Table D.2.

Table D.1: Best (α, β) values for the noises in the bootstrap PF that yields the highest positional RMSE with 100 particles for Experiment 1 and 2, and 400 particles for Experiment 3.

	Experiment 1	Experiment 2	Experiment 3
Position	$(5, 1)$	$(10, 1)$	$(15, 0.75)$
Velocity	$(1, 0.01)$	$(1, 0.01)$	$(10, 0.1)$
Normal	-	-	$(0.5, 0.005)$
Tangential	-	-	$(5, 0.5)$

Table D.2: Number of particles used in the PF for the three experiments.

	Experiment 1	Experiment 2	Experiment 3
Number of Particles	100	100	400

D.2 Normalizing Flow - Differentiable Particle Filter

The architectural details of the NF-DPF used in the experiments are provided in Table D.3. These values are included to ensure experimental reproducibility. The number of particles is increased in Experiment 3 to account for the higher state dimensionality, which includes acceleration in addition to position and velocity. The observation dimensionality refers to the dimensionality of the measurement vector, where the 3-dimensional case corresponds to the full measurement vector $[r, \theta, v_r]$. The Motion Update Model noise terms presented in Table D.5 were also used in the NF-DPF experiments.

Table D.3: NF-DPF Architecture Parameters.

	Experiment 1	Experiment 2	Experiment 3
Hidden Size	32	32	32
Hidden Depth	3	3	3
Number of Flows	2	2	2
State Dimensionality	4	4	6
Observation Dimensionality	3	3	3
Number of Particles	100	100	400

The number of epochs which the models in the three experiments are trained for along with the weights α_i in the loss-function are presented in Table D.4.

Table D.4: NF-DPF Training Parameters

	Experiment 1	Experiment 2	Experiment 3
Number of Epochs	100	65	100
α_{pos}	0.63	0.63	0.57
α_{vel}	0.27	0.27	0.24
α_{acc}	0.00	0.00	0.09
α_{ELBO}	0.10	0.10	0.10

The α and β values used for training and testing the NF-DPF arepresented in Table D.5.

Table D.5: The (α, β) values for the noise components used for the NF-DPF.

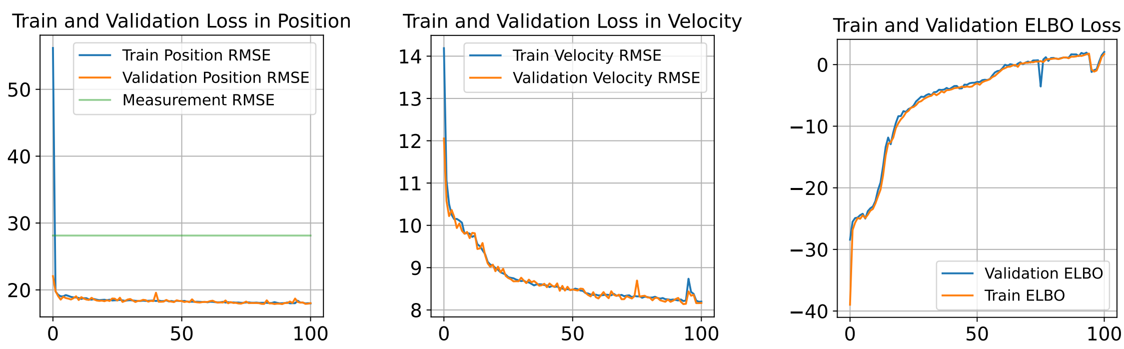
Noise Component	Experiment 1	Experiment 2	Experiment 3
Position	$(10, 0)$	$(10, 0)$	(15, 0.10)
Velocity	$(10, 0)$	$(10, 0)$	(5, 0.15)
Tangential	–	–	(0.05, 0.005)
Normal	–	–	(0.025, 0.005)

E

Training Dynamics

E.1 Experiment 1

The loss terms as a function of epoch number is presented for the NF-DPF model in Figure E.1. In Subfigure E.1a the Euclidean RMSE between the measurement and the true state is added as reference.



(a) Positional loss. **(b)** Velocity loss. **(c)** ELBO loss.

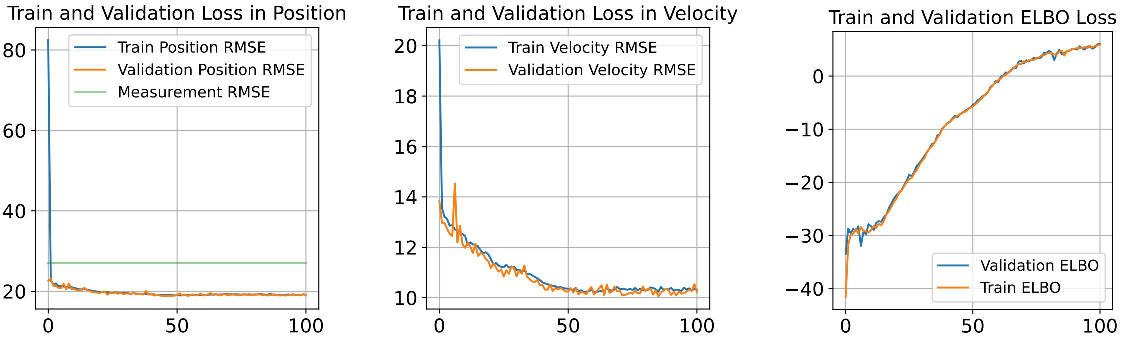
Figure E.1: Experiment 1 - training and validation losses over the 100 training epochs.

The NF-DPF shows clear sign of convergence after about 10 epochs in position and 60 in velocity. The model show no signs of overfittning to the training data, as the validation loss is consistently similar to the training loss.

E.2 Experiment 2

The loss terms as a function of epoch number is presented for the NF-DPF model in Figure E.2. In Subfigure E.2a the Euclidean RMSE between the measurement and the true state is added as reference.

The NF-DPF training in Experiment 2, shows clear similarities to that in Experiment 1, with clear convergence and no signs of overfitting to the training data.

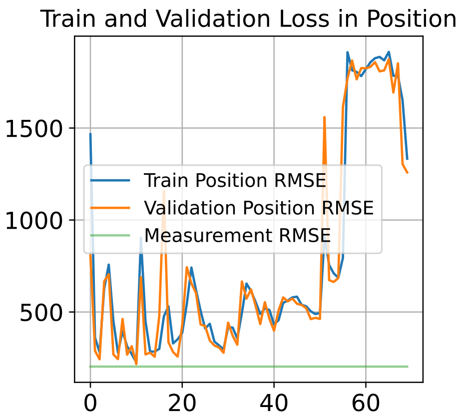


(a) Positional loss. **(b)** Velocity loss. **(c)** ELBO loss.

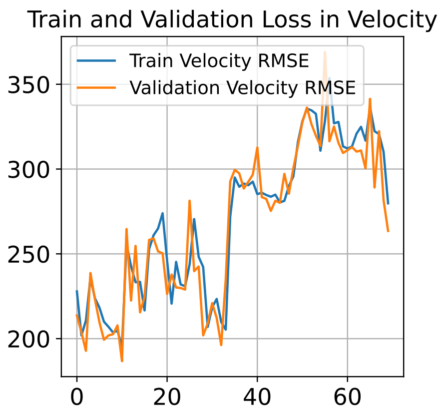
Figure E.2: Experiment 2 - training and validation losses over 100 training epochs.

E.3 Experiment 3

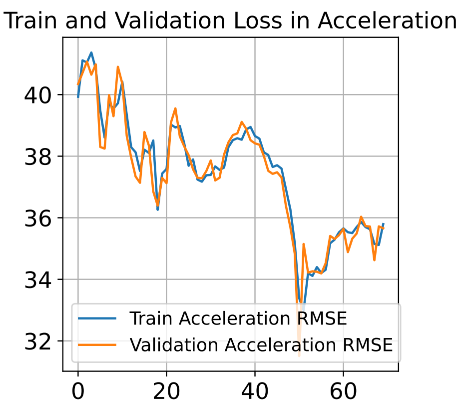
The loss terms as a function of epoch number is presented for the NF-DPF model in Figure E.3. In Subfigure E.3a the Euclidean RMSE between the measurement and the true state is added as reference.



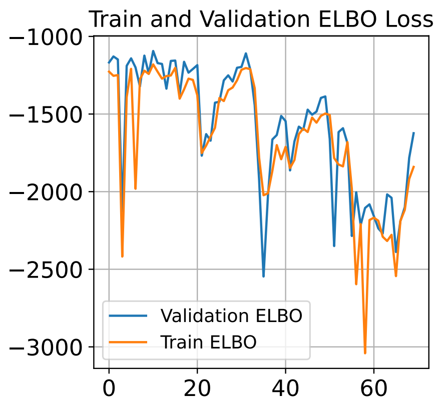
(a) Positional loss.



(b) Velocity loss.



(c) Acceleration loss.



(d) ELBO loss.

Figure E.3: Experiment 3 - training and validation losses over 65 training epochs.

In Experiment 3, the NF-DPF once more shows no signs of overfitting to the training data, but neither no sign of learning from the data, as no general trends can be seen. This is once more attributed to the hard-to-model data under the computational restrictions of 400 particles.

The difficulty of modelling the data is emphasized by the observation that the NF-DPF appears unable to overfit the training data, as demonstrated in all figures of this Appendix. While this may partly be due to overfitting prevention mechanisms such as masking and batch-based training, it is more likely that the model struggles to fully learn the underlying structure of the data.

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden