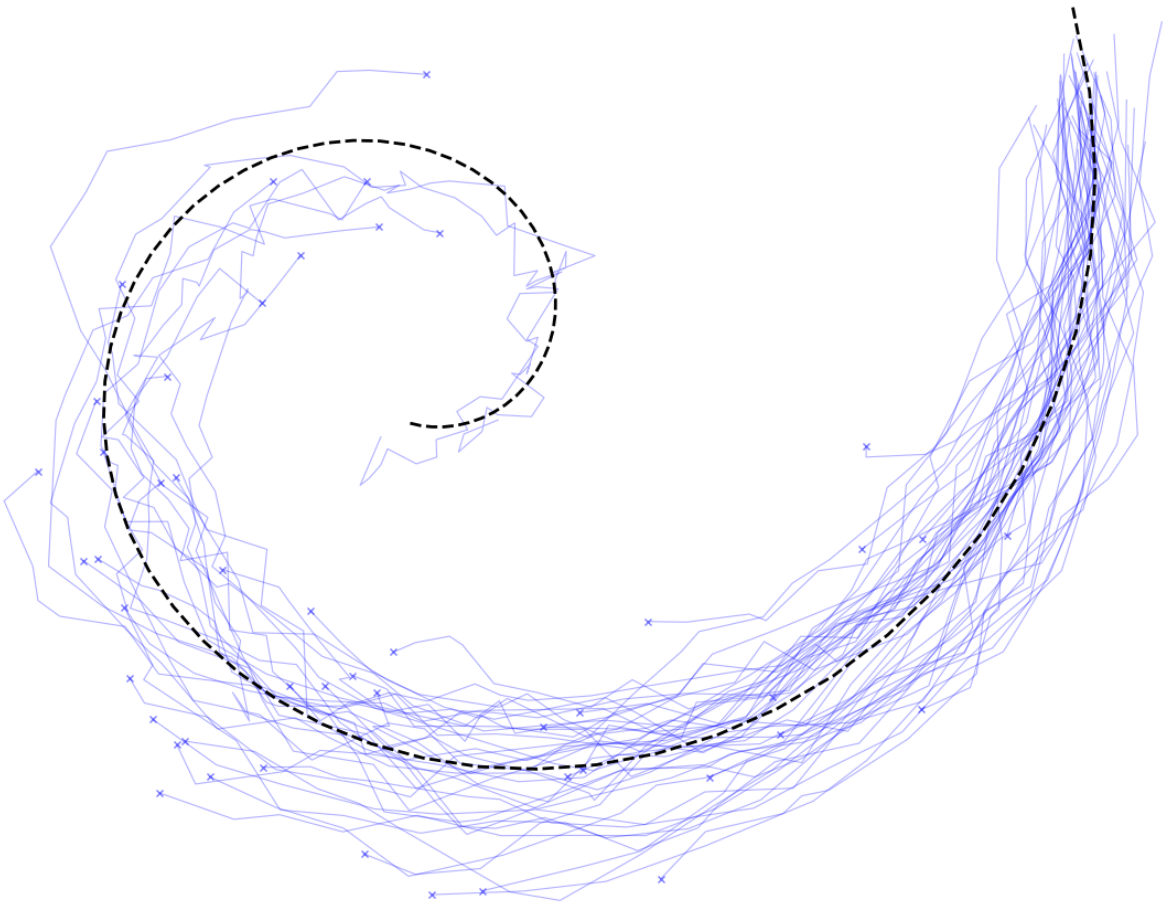


Schrödinger bridges for Bayesian filtering

Enabling cost-effective filtering in nonlinear state space models

Master's thesis in Engineering Mathematics and Computational Sciences

ELLEN CARLSSON, JOAKIM COLPIER

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Master's thesis 2026

Schrödinger bridges for Bayesian filtering

Enabling cost-effective particle filtering in nonlinear state space models

ELLEN CARLSSON, JOAKIM COLPIER

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2026

Schrödinger bridges for Bayesian filtering
Enabling cost-effective particle filtering nonlinear state space models
ELLEN CARLSSON, JOAKIM COLPIER

Supervisor: Karl Hammar, Saab
Supervisor: Benjamin Svedung Wettervik, Saab
Examiner: Moritz Schauer, Department of Mathematical Science, Chalmers

Master's Thesis 2026
Department of Mathematical Science
Division of Applied mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Weight degeneracy for particle filter usage in filtering problem.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Schrödinger bridges for Bayesian filtering
Enabling cost-effective particle filtering in nonlinear state space models
ELLEN CARLSSON, JOAKIM COLPIER
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

While particle filters are able to approximate the filtering distribution in nonlinear and non-Gaussian scenarios, they suffer from an inherent flaw known as weight degeneracy. This leads to rough state estimates and wasted computational resources. This thesis seeks to mitigate weight degeneracy in particle filters by formulating the proposal as an optimal transport problem – the Schrödinger bridge. Theoretically, this allows for exact sampling from the target distribution, entirely bypassing the traditional weight update. Because the exact numerical solution to the Schrödinger bridge is computationally heavy, we approximate the optimal transport dynamics as a neural network using amortized learning. Our results demonstrate that using the exact Schrödinger bridge proposal completely eliminates the weight degeneracy, although the computations are slow and scale poorly. We also find that the neural networks currently follow the dynamic processes too poorly to bypass the weight updates entirely. However, when used as a proposal distribution to a marginal particle filter, the network provides a viable state estimation. In fact, it competes or bests commonly used filtering methods such as the bootstrap particle filter and the unscented Kalman filter in the nonlinear case, both with respect to the mean squared error to the true position and weight variance. These findings indicate the potential of using approximate Schrödinger bridges in particle filters. With further refinement to the network training, this integration has the potential to give rise to a highly efficient particle filter, free from weight degeneracy.

Keywords: Amortized learning, Marginal particle filter, Optimal transport, Particle filter, Schrödinger bridge, Sinkhorn, State estimation, Weight degeneration.

Acknowledgements

First and foremost, we would like to express our most sincere gratitude to our supervisors at Saab, Karl Hammar and Benjamin Svedung Wettervik, for their their invaluable support throughout the thesis. Thank you for lighting up our path during this journey, we could not have made it without you. We would also like to thank our examinator Moritz Schauer at Chalmers, who provided much-needed insight into our problems. We never knew what we would get from a meeting with you, only that it would be precisely what we needed. A big thank you also to our opponent, Fredrik Boman, and all other people we have met at Saab, to have made this period so enjoyable.

Last but not least, we would like to dedicate a few words to our respective true states, Alexander Karlsson and Erika Molnár. Without you, we would just be a bunch of noisy observations.

Ellen Carlsson and Joakim Colpier, Gothenburg, May 2026

List of Acronyms

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

AMPF	Auxiliary marginal particle filter
APF	Auxiliary particle filter
DSS	Discrete state Schrödinger proposal
ESS	Effective sample size
LO	Locally optimal proposal
MLP	Multilayer perceptron
MPF	Marginal particle filter
NF	Normalizing flow
PF	Regular particle filter
RNN	Recurrent neural network
SBP	Schrödinger bridge problem
UKF	Unscented Kalman filter

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

$\mathcal{U}(D)$	Uniform distribution of domain D
$\mathcal{N}(\mu, \Sigma)$	Normal distribution with mean μ and covariance Σ
$q(\cdot)$	Proposal function
$\mathbb{1}$	One-vector
$\mathbf{1}$	Indicator function
$D(\cdot)$	Diagonalization of vector
$D_{\mathrm{KL}}(\cdot\|\|\cdot)$	Kullback-Leibler divergence
ρ_t	Marginal at time t
$\Pi(\rho_0, \rho_1)$	Set of all couplings of marginals ρ_0 and ρ_1
PF(B)	Regular particle filter with bootstrap proposal

Contents

List of Figures

List of Tables

1

Introduction

The ability to track the state of a moving object over time given the data from physical sensors (such as radar, lidar or cameras), is a fundamental challenge across disciplines ranging from signal processing and autonomous navigation to computer vision. The fact that real-world sensors are rarely perfect and often miss data has given rise to methods that estimate an object's true state from a set of noisy and incomplete observations. The goal of such methods is to produce a final estimate that is more accurate than the underlying measurements themselves.

A common way of modeling the system is as a *state space model*. In such a model, the true state $\mathbf{x}_t$ evolves from its previous true state $\mathbf{x}_{t-1}$ without any influence from earlier states. Furthermore, the sensor observation $\mathbf{y}_t$ is assumed to only depend on the true state at that time $\mathbf{x}_t$. Finding the probability distribution of the true state at a given time based on the observations up to that point is called the *filtering problem*, and the sought distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ is analogously named the *filtering distribution*. Formulating the problem in these terms transforms the tracking challenge into a statistical one, allowing us to mathematically infer the object's true trajectory.

Assuming that the problem is linear-Gaussian, the problem can be solved analytically using the *Kalman filter*. However, many real-world applications aren't linear-Gaussian. In these settings, the dynamics or distributions must be approximated in order to apply Kalman filtering methods, leading to biased, non-optimal estimates.

Another approach to overcome the limitations of these linearized Kalman approaches is to make use of *particle filtering*. This method approximates the filtering distribution not by linearizing the setting or approximating the distribution by a Gaussian, but rather by approximating it with a set of weighted particles. These are independently propagated using a simpler *proposal distribution*, and then reweighted to account for the mismatch between the proposal and the true filtering distribution. The advantage is that such methods can be used even when the true filtering distribution cannot be sampled from (by judiciously choosing the proposal distribution), and it can be used in the nonlinear case. Furthermore, particle filters asymptotically converge to the exact filtering distribution as the number of particles goes to infinity. However, the disadvantage is that their weights tend to become concentrated on just a handful of particles after a few iterations, leading to increasingly rougher approximations of the probability distribution. This is called *weight degeneracy*. While the use of enough particles can prevent the approximation from becoming

too coarse, it scales poorly in dimension as progressively more particles are needed. The choice of proposal distribution can affect the severity of the weight degeneracy, however.

In this thesis, we study the *Schrödinger bridge* as a new approach to particle propagation. Originating from statistical mechanics, the Schrödinger bridge finds the mathematically most probable trajectory between two probability distributions, given a reference stochastic process. By framing the particle filter's sequential updates as an optimal transport problem, we develop the Schrödinger bridge into a proposal distribution. The theoretical gain is significant: if the particles accurately represent the filtering distribution at time t, the Schrödinger bridge provides an exact transport path to the filtering distribution at time $t + 1$. If the transport is sufficiently accurate, the propagated particles can potentially be used directly as a representation of the new filtering distribution. The particles would be moved to precisely where they need to be, and the particle filter weight update step could then be bypassed entirely.

Calculating the exact Schrödinger bridge during live tracking requires repeatedly solving a complex optimal transport problem. Since the solution process is too computationally expensive for real-time application, we look at *amortized optimization* methods for a cheaper, approximate solution. These use machine learning to predict the solutions to equations by exploiting the shared structure between similar problem instances. Rather than solving the optimal transport equations from scratch at every time step, we train a neural network offline to map current particle states directly to the target filtering distribution.

However, as the Schrödinger computations depend on multiple approximations, notably from neural networks, the resulting transport may lose its theoretical perfection. If the approximation error is too large for the propagated particles to be used directly as the final distribution, they instead serve as an accurate proposal distribution within a particle filter. Because the approximate Schrödinger bridge prediction is assumed to be close to the true target, the weight corrections should be small, and the weight degeneracy therefore minimal.

While a Schrödinger bridge proposal shows potential, standard particle filters suffer from a second, separate source of weight degeneracy: rather than approximating the filtering distribution, they approximate the full trajectory distribution $p(\mathbf{x}_{1:t}|\mathbf{y}_{1:t})$. Because the trajectory space continuously increases in dimensionality, it causes the particle weights to decay over time. Furthermore, the Schrödinger bridge cannot bridge between full trajectory distributions in a filtering problem, as that would require modifying previous states. Since this is a problem with the structure of the particle filter, no proposal distribution can resolve it. Therefore we consider the *marginal particle filters*, which marginalize out trajectory history, approximating the sought filtering distribution.

Ultimately, the goal of this thesis is to merge particle filters, optimal transport and machine learning to create an accurate and computationally efficient filtering algorithm applicable in nonlinear situations. First, we investigate the integration of the Schrödinger bridge into a tracking framework, both by itself, and as a proposal

for a particle filter. Second, we aim to ensure these theoretical calculations are computationally efficient by applying neural networks to approximate the bridge. Finally, we integrate these machine-learning models within a marginal particle filter approximation, with the aim of yielding a tractable tracking filter that exhibits little to no weight degeneracy.

1.1 Outline

This thesis is structured as follows. Chapter 2 introduces relevant theory about Bayesian filtering, including the problem modeling and particle filters. The Schrödinger bridge problem is defined in Chapter 3, detailing the problem and some ways to solve it, culminating in an integration as a proposal into a particle filter. The practical problems relating to the Schrödinger bridge as a proposal, including the implementation and choice of neural networks used, are described in Chapter 4. The models used to test the Schrödinger proposals are then outlined in Chapter 5. These are then used in Chapter 6 to test the hypotheses that were presented in the thesis. Finally, a discussion of the results is held in Chapter 7. We also discuss possible future research areas.

2

Bayesian filtering

Estimating the true state of an object over time requires dealing with uncertain dynamics and noisy measurements, hence requiring a statistical approach. This chapter provides the mathematical framework to this approach in the context of obtaining data sequentially. Note that the theory holds for any type of state estimation, although we are only interesting in tracking a moving object in this thesis.

We start by formulating the filtering problem in Section 2.1, a mathematical description of the tracking problem. We then present the Kalman filter in Section 2.2, which is an exact analytical solution for idealized linear-Gaussian scenarios. Section 2.3 takes a look at the *unscented Kalman filter* for nonlinear dynamics, which works by linearly approximating parts of the problem to accommodate for the use of the Kalman filter. Section 2.4 then details the particle filtering method and the effect of different proposal functions, which don't make use of any linearizations, but rather approximate the filtering distribution by discretizing it. Lastly, two alternative particle filter methods are covered in Section 2.5.

2.1 Filtering problem

The filtering problem concerns the estimation of a dynamic system's true state based on a sequence of sensor measurements. We formulate the problem as a *state space model*, described by a sequence of latent states $(\mathbf{x}_t)_{t\geq 0}$ and corresponding observations $(\mathbf{y}_t)_{t>0}$.

$$\begin{aligned} &\mathbf{x}_0 \sim \nu, \quad \nu \text{ distribution in } \mathbb{R}^d \\ &\mathbf{y}_t, \mathbf{x}_t \in \mathbb{R}^d \\ &\mathbf{x}_t = f(\mathbf{x}_{t-1}) + q_t \\ &\mathbf{y}_t = h(\mathbf{x}_t) + r_t \end{aligned} \tag{2.1}$$

The state space model used in this thesis is formulated by (2.1). Here, $f : \mathbb{R}^d \to \mathbb{R}^d$ acts as the *prior on the movement* (system dynamics), and $h : \mathbb{R}^d \to \mathbb{R}^d$ is the *measurement function*. We assume that both are known. The random variables q_t and r_t denote i.i.d. process and observation noise respectively.

Figure 2.1 visualizes the model (2.1). It demonstrates how each state relies solely on the previous state, and how each observation depends only on the current state.

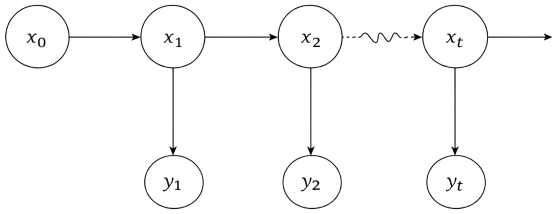


Figure 2.1: Schematic illustration of a state space model chain.

The ultimate goal of filtering is to compute the *filtering distribution* $p(\mathbf{x}_t|\mathbf{y}_{1:t})$. To evaluate this distribution from scratch at time t, one needs to first calculate the full joint posterior distribution over all states and observations up to t. This is given by Bayes' rule as

$$p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t}) = \frac{p(\mathbf{y}_{1:t}|\mathbf{x}_{0:t})\, p(\mathbf{x}_{0:t})}{p(\mathbf{y}_{1:t})}.$$

Estimating the entire historical trajectory simultaneously is a distinct task known as the *smoothing problem*, and the joint probability $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$ is referred to as the *trajectory distribution*.

To isolate the current state and solve the filtering problem, we must marginalize out every previous hidden state $\mathbf{x}_0, ..., \mathbf{x}_{t-1}$, yielding the distribution

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) = \int ... \int p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})\, \mathrm{d}\mathbf{x}_0 ... \mathrm{d}\mathbf{x}_{t-1}.$$

However, evaluating the posterior in this manner requires a full recalculation with every new observation. In a dynamic tracking problem where an estimate is required after each measurement, this is a significant disadvantage. Moreover, the dimensionality of the posterior calculations grows with each measurement, increasing the computational complexity and eventually making it intractable.

To avoid the computational burden of calculating the full posterior at each time step, we instead solve the filtering problem sequentially [14]. By using the Markov properties of the state space model above, the filtering distribution can be computed recursively. This is done in two distinct phases: the *prediction* step, and the *update* step.

Prediction step. Assuming the filtering distribution at the previous time step $p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})$ is known, we calculate the predicted state at time t given our prior knowledge of the dynamics.

$$\begin{aligned} p(\mathbf{x}_t|\mathbf{y}_{1:t-1}) &= \int p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_{1:t-1})\, p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})\, \mathrm{d}\mathbf{x}_{t-1} \\ &= \int p(\mathbf{x}_t|\mathbf{x}_{t-1})\, p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})\, \mathrm{d}\mathbf{x}_{t-1} \end{aligned} \tag{2.2}$$

The last equality holds due to the Markov property, which tells us that the current state depends only on the previous state.

Update step. When a new measurement $\mathbf{y}_t$ becomes available we correct our prediction to obtain the filtering distribution.

$$\begin{aligned} p(\mathbf{x}_t|\mathbf{y}_{1:t}) &= p(\mathbf{x}_t|\mathbf{y}_t, \mathbf{y}_{1:t-1}) = \frac{p(\mathbf{x}_t, \mathbf{y}_t|\mathbf{y}_{1:t-1})}{p(\mathbf{y}_t|\mathbf{y}_{1:t-1})} \\ &= \frac{p(\mathbf{y}_t|\mathbf{x}_t, \mathbf{y}_{1:t-1})\, p(\mathbf{x}_t|\mathbf{y}_{1:t-1})}{p(\mathbf{y}_t|\mathbf{y}_{1:t-1})} \\ &= \frac{p(\mathbf{y}_t|\mathbf{x}_t)\, p(\mathbf{x}_t|\mathbf{y}_{1:t-1})}{\int p(\mathbf{y}_t|\mathbf{x}_t)\, p(\mathbf{x}_t|\mathbf{y}_{1:t-1})\, \mathrm{d}\mathbf{x}_t} \end{aligned} \tag{2.3}$$

Using this recursive method, the calculations of the filtering distribution only depend on the filtering distribution from the previous step $p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})$, the transition model $p(\mathbf{x}_t|\mathbf{x}_{t-1})$, and the measurement model $p(\mathbf{y}_t|\mathbf{x}_t)$, which we all assume to be known. In particular, this formulation ensures that the computational complexity remains constant over time.

2.2 Kalman filter

In Section 2.1, we established that the filtering distribution can be computed sequentially. In practice, the integrals required in the prediction and update steps are often intractable. Because general nonlinear or non-Gaussian models usually lack closed-form analytical solutions, these integrals cannot be evaluated exactly. However, for the special case of linear-Gaussian dynamical systems and measurement functions, the update and prediction steps can be evaluated analytically. The resulting algorithm is known as the *Kalman filter*. It is constructed to minimize the mean squared error to the true state, and it gives us the exact solution to the filtering distribution [9].

Let the general state space model be restricted to the linear-Gaussian case, as follows.

$$\begin{aligned} \mathbf{x}_0 &\sim \mathcal{N}(\mu_0, \Sigma_0) \\ \mathbf{x}_{t+1} &= A\mathbf{x}_t + q_t, \quad & q_t &\sim \mathcal{N}(0, Q) \\ \mathbf{y}_t &= H\mathbf{x}_t + r_t, \quad & r_t &\sim \mathcal{N}(0, R) \end{aligned}$$

Here A and H describe the system dynamics and observation model respectively. Q and R are covariance matrices of the process and observation noise, while μ_0 and Σ_0 define the initial state distribution.

Because Gaussian distributions are closed under linear transformation, the filtering distribution is Gaussian for all t. It is therefore fully described by its mean μ_t and covariance Σ_t. The moments are given by the analytical solutions to (2.2) and (2.3) given below.

Prediction step. Given a posterior estimate at time $t-1$ the predicted state and covariance are

$$\begin{aligned}\mu_{t|t-1} &= A\mu_{t-1} \\ \Sigma_{t|t-1} &= A\Sigma_{t-1}A^\top + Q.\end{aligned}$$

Update step. After observing $\mathbf{y}_t$ the predicted estimate is corrected according to

$$\begin{aligned}v_t &= \mathbf{y}_t - H\mu_{t|t-1} \\ S_t &= H\Sigma_{t|t-1}H^\top + R \\ K_t &= \Sigma_{t|t-1}H^\top S_t^{-1} \\ \mu_t &= \mu_{t|t-1} + K_t v_t \\ \Sigma_t &= \Sigma_{t|t-1} - K_t S_t K_t^\top.\end{aligned}$$

As the Kalman filter is not the central focus of this project, we omit a full derivation and discussion of the algorithm. The interested reader is instead recommended [9] by Masnadi-Shirazi et al. for a full derivation and tutorial on Kalman filters.

While the Kalman filter is theoretically optimal and computationally efficient, it is limited by the assumption of linear-Gaussian models. In real-world scenarios, with highly nonlinear models or skewed noise distributions, the analytical approach fails.

2.3 Unscented Kalman filter

There are several methods that aim to adjust the Kalman filter for nonlinear systems. Consider a system

$$\begin{aligned}\mathbf{x}_{t+1} &= f(\mathbf{x}_t) + q_t, \quad q_t \sim \mathcal{N}(0, Q) \\ \mathbf{y}_t &= h(\mathbf{x}_t) + r_t, \quad r_t \sim \mathcal{N}(0, R)\end{aligned}$$

where f, h are nonlinear functions. A popular variation to the Kalman filter, designed to handle such systems, is the *unscented Kalman filter* (UKF). It is named after the *unscented transform* that it makes use of.

2.3.1 Unscented transform

In order to describe the unscented transform, we first need to define *sigma points* and how to obtain them from a Gaussian distribution.

Definition 2.1 (Sigma points)**.** *A set of n points s_i, which contains the statistics of a given distribution, is called a set of* sigma points *for that distribution. For a Gaussian distribution $\mathcal{N}(\mu, \Sigma)$, sigma points must satisfy*

$$\begin{aligned}\mu &= \frac{1}{n}\sum_i s_i \\ \Sigma &= \frac{1}{n-1}\sum_i (s_i - \mu)(s_i - \mu)^\top.\end{aligned}$$

The minimum amount of sigma points s_i needed to define a d-dimensional distribution $\mathcal{N}(\mu, \Sigma)$ is $d+1$. However, calculating this minimal set often requires computationally costly matrix operations, such as matrix inversion. Instead, it is standard practice to compute a slightly larger symmetric set of $2d+1$ points using the Cholesky decomposition, which is computationally cheaper.

Proposition 2.1 (Computing sigma points)**.** *Let $X \sim \mathcal{N}(\mu, \Sigma)$ be a $d-$dimensional random variable. A symmetric set of $2d+1$ sigma points, defined as*

$$\begin{aligned} s_0 &= \mu \\ s_i &= \mu + \sqrt{d\Sigma_{:,i}}, \quad \forall i \in \{1, ..., d\} \\ s_j &= \mu - \sqrt{d\Sigma_{:,j}}, \quad \forall i \in \{d+1, ..., 2d\}, \end{aligned}$$

exactly captures the true mean μ and true covariance Σ of the distribution, where $\sqrt{\Sigma}$ is the matrix square root, i.e. finding the Cholesky decomposition $LL^\top = \Sigma$.

The unscented transform of a Gaussian distribution $\mathcal{N}(\mu, \Sigma)$ with respect to a nonlinear function g works by finding a set of sigma points for that Gaussian and propagating each point through g. The resulting points uniquely define a new Gaussian distribution $\mathcal{N}(\hat{\mu}, \hat{\Sigma})$ given by

$$\begin{aligned} \hat{\mu} &:= \frac{1}{n} \sum_i g(s_i) \\ \hat{\Sigma} &:= \frac{1}{n-1} \sum_i (g(s_i) - \hat{\mu})(g(s_i) - \hat{\mu})^\top. \end{aligned}$$

$\mathcal{N}(\hat{\mu}, \hat{\Sigma})$ now constitutes a Gaussian approximation of the distribution $g(\mathcal{N}(\mu, \Sigma))$. See Figure 2.2 for an illustration of the method.

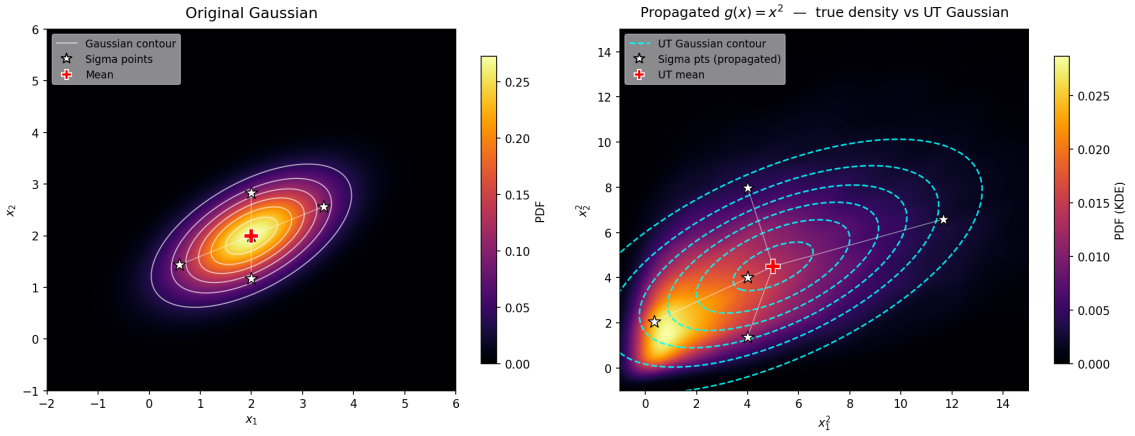


Figure 2.2: Comparison of an exact and unscented transformation by a nonlinear function $g(x, y) = (x^2, y^2)$ (right) of an initial Gaussian distribution (left).

2.3.2 Update and prediction

The prediction and update steps of the UKF are as follows. Note the similarities to their counterparts in the regular Kalman filter.

Prediction step. Given a posterior estimate $\mathcal{N}(\mu, \Sigma)$ at time $t-1$, the predicted state $\mu_{t|t-1}$ and covariance $\Sigma_{t|t-1}$ are obtained by performing an unscented transform with respect to f, that is finding the sigma points s_i of the distribution and calculating

$$\begin{aligned}\mu_{t|t-1} &= \frac{1}{n}\sum_i f(s_i) \\ \Sigma_{t|t-1} &= \frac{1}{n-1}\sum_i (f(s_i) - \mu_{t|t-1})(f(s_i) - \mu_{t|t-1})^\top + Q.\end{aligned}$$

Update step. After observing $\mathbf{y}_t$, we calculate new sigma points z_i using the predicted state and covariance. The updated estimate $\mu_{t|t}, \Sigma_{t|t}$ is calculating

$$\begin{aligned}\hat{z} &= \frac{1}{n}\sum_i h(z_i) \\ S_t &= \frac{1}{n-1}\sum_i (h(z_i) - \hat{z})(h(z_i) - \hat{z})^\top + R \\ C_{sz} &= \frac{1}{n-1}\sum_i (f(s_i) - \mu_{t|t-1})(h(z_i) - \hat{z})^\top \\ K_t &= C_{sz} S_t^{-1} \\ \mu_{t|t} &= \mu_{t|t-1} + K_t(\mathbf{y}_t - \hat{z}) \\ \Sigma_{t|t} &= \Sigma_{t|t-1} - K_t S_t K_t^\top.\end{aligned}$$

Since the UKF, much like the Kalman filter, is not a central part of this project, we omit a full derivation and discussion of the algorithm. For more information on the subject, the interested reader is recommended [7] by Julier and Uhlmann.

The UKF has gained popularity in finding approximate filtering distributions in nonlinear contexts due to its low computational cost and relatively high accuracy. However, approximating the filtering distribution by a Gaussian distribution leads to potential loss of information, such as modes. To account for these problems, one must find other ways to more precisely approximate the integrals in general nonlinear non-Gaussian systems.

2.4 Particle filters

In order to address the limitations of the Kalman filters, we instead turn to *particle filters*. Rather than attempting to compute the exact filtering distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ analytically, particle filters approximate the posterior using a set of weighted samples called *particles*, see Figure 2.3. Unlike the UKF, they can more accurately represent non-Gaussian distributions, since the particle cloud is not constrained to follow any specific parametric shape.

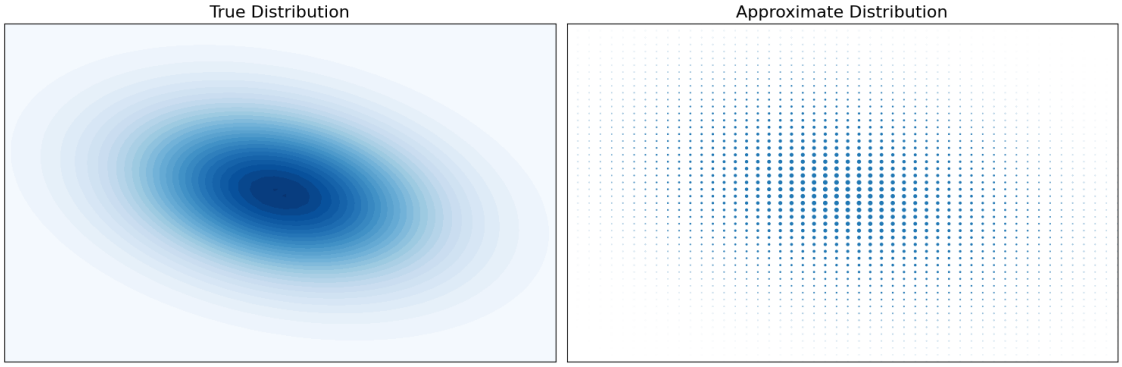


Figure 2.3: The true distribution (left) is approximated by sampling points from a grid and weighting the particles (right). Weights are shown by scaling the particles.

We begin by defining the mathematical background behind the particle filter algorithm in Section 2.4.1. Following this, we discuss weight degeneracy in particle filters in Section 2.4.2. Finally, Section 2.4.3 discusses different proposals used to combat weight degeneracy.

2.4.1 Particle filter algorithm

To represent the posterior as a particle cloud at each time step, the particle filter performs two main operations: it draws sample points (particle states) from the state space, and it assigns each particle an appropriate weights proportional to the true posterior at the sampled points. If we could sample directly from the filtering distribution, the sampled particles would represent the posterior using equal weights.

However, as the true filtering distribution relies on intractable integrals, we cannot sample the exact probability density in the general nonlinear case. To bypass this problem, particle filters employ a Monte Carlo technique known as *importance sampling*.

Theorem 2.1 (Importance sampling, taken from [11])**.** *Let $p(x)$ be a target probability density from which we cannot sample, but which is proportional to a density $\pi(x)$ that we can evaluate. Furthermore, let $q(x)$ be a proposal density from which we sample $\{x^i\}_{i=0}^N$. Let us assume that the support of the proposal covers the support of the target, meaning $q(x) > 0$ wherever $p(x) > 0$.*

We construct an empirical probability measure $\hat{p}_N(x)$ defined as

$$\hat{p}_N(x) = \sum_{i=0}^{N} w^i \delta(x - x^i),$$

where $\delta(\cdot)$ is the Dirac delta function, the unnormalized importance weights are defined as

$$\tilde{w}^i := \frac{\pi(x^i)}{q(x^i)},$$

and $w^i = \frac{\tilde{w}^i}{\sum \tilde{w}^i}$ *are the corresponding normalized weights.*

Then, as the number of samples $N \to \infty$*, the empirical measure* $\hat{p}_N$ *converges weakly to the true target distribution* $p(x)$*.*

Theorem 2.1 allows us to overcome the inability to sample the true posterior by instead sampling from a simpler *proposal distribution* q and weighting the particles.

To apply this principle to our tracking problem, we note that we are ultimately interested in the filtering distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t})$. However, as discussed, the marginalization over the previous states in the prediction step (2.2) is not always tractable. Particle filters work by instead estimating the full joint trajectory distribution $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$, thus avoiding the intractable marginalization integrals.

Note that the conditional trajectory distribution is proportional to the joint distribution

$$p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t}) = \frac{p(\mathbf{x}_{0:t}, \mathbf{y}_{1:t})}{p(\mathbf{y}_{1:t})} \propto p(\mathbf{x}_{0:t}, \mathbf{y}_{1:t}) = p(\mathbf{x}_0) \prod_{k=1}^{t} p(\mathbf{y}_k|\mathbf{x}_k)\, p(\mathbf{x}_k|\mathbf{x}_{k-1})\,,$$

which we can evaluate, but not sample from. By introducing a proposal distribution $q(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$ from which we can sample, we can use importance sampling by introducing non-normalized weights to approximate the distribution $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$ as

$$\tilde{w}(\mathbf{x}_{1:t}) = \frac{p(\mathbf{x}_{0:t}, \mathbf{y}_{1:t})}{q(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})}. \tag{2.4}$$

While (2.4) defines the weights for the full trajectory, recalculating this entire fraction from scratch at every time step is ineffective. As explained in Section 2.1, dynamic tracking benefits from a recursive solution.

To achieve a recursive filter, we must choose a proposal $q(\cdot)$ such that it factorizes sequentially

$$q(\mathbf{x}_{0:t}|\mathbf{y}_{1:t}) = q(\mathbf{x}_{0:t-1}|\mathbf{y}_{1:t-1})\, q(\mathbf{x}_t|\mathbf{y}_t, \mathbf{x}_{t-1})\,. \tag{2.5}$$

This factorization forms the basis of the filter. Instead of generating a new trajectory from scratch at each step, we keep the existing trajectory $\mathbf{x}_{0:t-1}$ and append a new proposed state by sampling from $q(\mathbf{x}_t|\mathbf{y}_t, \mathbf{x}_{t-1})$.

Since the joint distribution in the numerator of (2.4) can be factorized sequentially

$$p(\mathbf{x}_{0:t}, \mathbf{y}_{1:t}) = p(\mathbf{y}_t|\mathbf{x}_t)\, p(\mathbf{x}_t|\mathbf{x}_{t-1})\, p(\mathbf{x}_{0:t-1}, \mathbf{y}_{1:t-1}),$$

the weight update can now be written as

$$\tilde{w}_t^{(i)} = \frac{p\left(\mathbf{y}_t\middle|\mathbf{x}_t^{(i)}\right) p\left(\mathbf{x}_t^{(i)}\middle|\mathbf{x}_{t-1}^{(i)}\right)}{q\left(\mathbf{x}_t^{(i)}\middle|\mathbf{y}_t, \mathbf{x}_{t-1}^{(i)}\right)} \tilde{w}_{t-1}^{(i)}. \tag{2.6}$$

To resolve the missing proportionality constant, the non-normalized weights evaluated in (2.6) are then normalized such that they sum to one.

$$w_t^{(i)} = \frac{\tilde{w}_t^{(i)}}{\sum_{j=1}^{N} \tilde{w}_t^{(j)}} \tag{2.7}$$

The algorithm for this sequential importance sampling is summarized in Algorithm 1 below, using T as the final time step. Note that Algorithm 1 can be used for any proposal $q(\mathbf{x}_t|\mathbf{y}_t, \mathbf{x}_{t-1})$ which satisfies (2.5). For finite N, however, our choice of proposal will affect the algorithm's performance. Note further that particle filters generally contain resampling, which is covered in Section 2.4.2. For the complete particle filter algorithm, see Algorithm 2.

Algorithm 1 Sequential importance sampling

1: $N \leftarrow$#particles
2: $\mathbf{x}_0^{(i)} \sim \nu$ for $i = 1, ..., N$
3: $w_0^{(i)} \leftarrow \frac{1}{N}$ for $i = 1, ..., N$
4: **for** $t = 1, ..., T$ **do**
5: **Sample** $\mathbf{x}_t^{(i)} \sim q\left(\mathbf{x}_t^{(j)} \middle| \mathbf{y}_t, \mathbf{x}_{t-1}^{(j)}\right)$, $i = 1, ..., N$
6: **Update** $\tilde{w}_t^{(i)}$ as (2.6) and **normalize** as (2.7)

In practice, this means that we propose new states for the particles using the proposal distribution q for each new measurement, and then weight these particles to correct for the discrepancy with the true target distribution. An illustration is shown in Figure 2.4. While the theoretical derivation uses the full trajectory $\mathbf{x}_{0:t}$ to avoid difficult integrals, our actual goal is the filtering distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t})$. To obtain this, we discard the historical states $\mathbf{x}_{0:t-1}$ from memory once the new particles and weights have been computed. By keeping only the latest states $\{\mathbf{x}_t^{(i)}\}_{i=1}^N$ and their normalized weights $\{w_t^{(i)}\}_{i=1}^N$, we get the final approximation

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) \approx \sum_{i=1}^{N} w_t^{(i)} \delta(\mathbf{x}_t - \mathbf{x}_t^{(i)}).$$

The particle filter algorithm complexity depends on the proposal q and the amount of particles N. For the proposals defined in Section 2.4.3, the complexity is $\mathcal{O}(N)$.

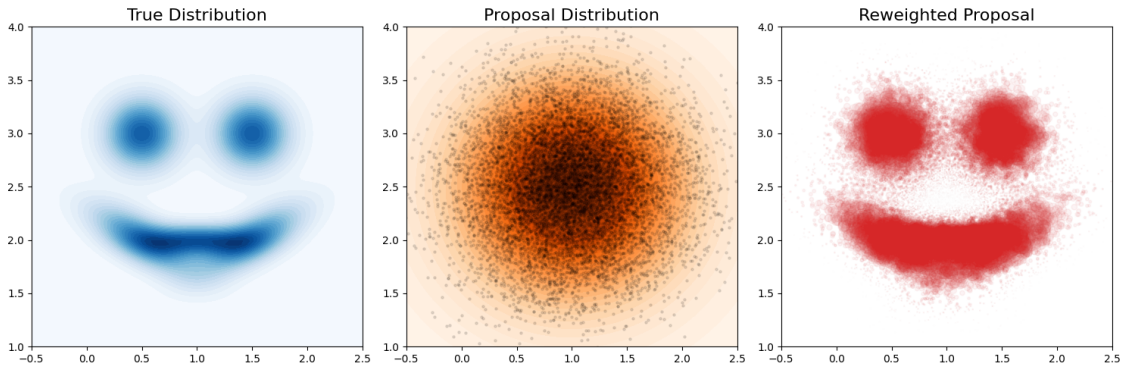


Figure 2.4: The true target distribution (left) is approximated by a simpler normal distribution (middle). The particles are reweighted and the sizes are scaled based on the weights (right).

2.4.2 Weight degeneracy and resampling

A common problem with particle filters is their susceptibility to *weight degeneracy*, that is an increase in weight variance over time. As weights are sequentially updated and normalized, the total weight tends to become concentrated on a small subset of the particles. One cause is the proposal function, which may place particles poorly and cause an immediate weight decay. Another cause is the fact that particle filters calculate the trajectory distribution, rather than the filtering distribution. Since the entire trajectory is used to calculate the weights, the dimension of the state space grows linearly with each step. Consequently, while the number of particles N remains fixed, the size of the trajectory space grows grows exponentially. This makes it increasingly improbable for a finite sample size to accurately cover the space. As a result, particles that are placed poorly at any point in history are penalized with lower weights for the rest of the tracking, independent of their current position. As the weight concentrates unevenly over time, the number of particles effectively contributing to the distribution decreases, thereby reducing the accuracy of the approximation.

An example is illustrated in Figure 2.5 using the prior on the movement as proposal (also called the bootstrap proposal, see Section 2.4.3), where only a single particle approximates the entire distribution after a few iterations. The figure tracks the amount of relevant particles, i.e. those whose weights have never dropped under a certain threshold.

In an effort to maintain a more numerous ecosystem of particles, while avoiding to compute the trajectory of irrelevant particles, we perform *resampling*. Resampling is commonly performed at fixed intervals or when weight variance exceeds some threshold. Particles are then sampled based on their weights, and all weights are subsequently set to uniform. Algorithm 1 is extended with resampling to construct Algorithm 2, the full particle filter algorithm.

Algorithm 2 The Particle Filter

1: $N \leftarrow$#particles
2: $\mathbf{x}_0^{(i)} \sim \nu$ for $i = 1, ..., N$
3: $w_0^{(i)} \leftarrow \frac{1}{N}$ for $i = 1, ..., N$
4: **for** $t = 1, ..., T$ **do**
5: **Sample** $\mathbf{x}_t^{(i)} \sim q\left(\mathbf{x}_t \middle| \mathbf{y}_t, \mathbf{x}_{t-1}^{(j)}\right)$, $i = 1, ..., N$
6: **Update** $w_t^{(i)}$ as (2.6) and **normalize** as (2.7)
7: **Resample** according to some criterion

Figure 2.6 shows the trajectory of the particles in the same scenario as Figure 2.5, given that they are resampled every step. We can see that the filtering distribution is approximated much better, although we note that resampling more often can lead to biased results (such as missing a mode in the distribution).

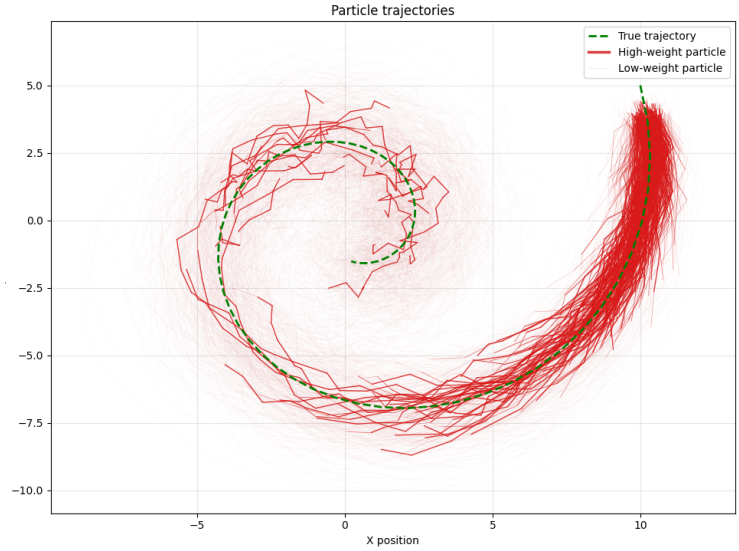


Figure 2.5: Trajectory of 500 particles. Particle weights are shown by varying the opacity and line width of the particle trajectories.

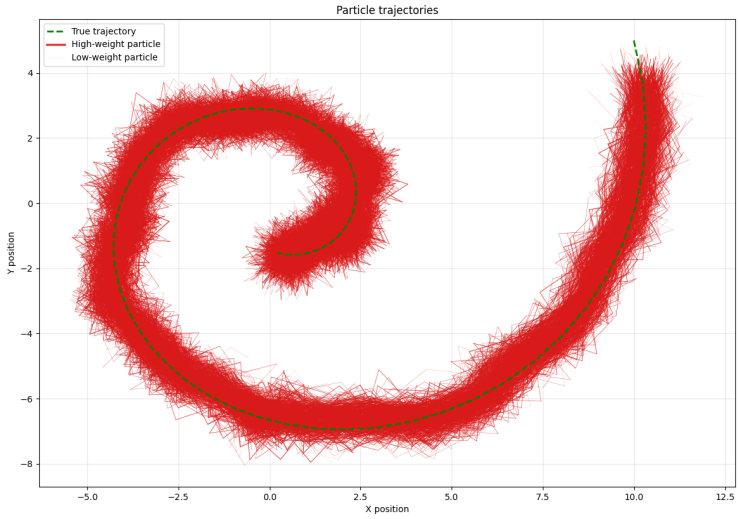


Figure 2.6: Trajectory of 500 particles that are resampled at every step. Particle weights are shown by varying the opacity and line width of the particle trajectories.

An intuitive parallel to biology is that one can see Algorithm 2 as only allowing the fittest (highly weighted) particles to reproduce. The problem then becomes that the more often one resamples, the greater the loss with regards to the particle "gene pool". In biology, this may make a species less resilient to changes to its habitat, and analogously, a loss of particle variety may lead to the loss of accuracy in the distribution approximation (modes, tails etc.).

2.4.3 Common proposal functions

As noted above, the proposal function can affect the weight distribution between the particles, and can in fact greatly impact the weight degeneracy. If a proposal blindly propagates particles, only a few particles land in high-probability regions, causing extreme weight variance. A well-designed proposal, on the other hand, uses more of the available information to steer the particle cloud to favorable locations. This yields evenly distributed weights across the particles, reducing weight degeneracy. Two proposal functions are described in this section: the *bootstrap* and the *locally optimal* (LO) proposals. Particle filters equipped with these proposals are henceforth denoted PF(B) and PF(LO) respectively.

Bootstrap Particle Filter

Using the prior on the movement (2.8) as proposal defines the bootstrap proposal. Because it places virtually no restrictive assumptions on the state space model, PF(B) can handle nonlinear system dynamics and non-Gaussian noise distributions.

$$q\left(\mathbf{x}_t \middle| \mathbf{x}_{t-1}^{(i)}, \mathbf{y}_t\right) = p\left(\mathbf{x}_t \middle| \mathbf{x}_{t-1}^{(i)}\right). \tag{2.8}$$

The weight update (2.6) thus becomes

$$w_t^{(i)} \propto w_{t-1}^{(i)} \frac{p\left(\mathbf{y}_t \middle| \mathbf{x}_t^{(i)}\right) p\left(\mathbf{x}_t^{(i)} \middle| \mathbf{x}_{t-1}^{(i)}\right)}{p\left(\mathbf{x}_t^{(i)} \middle| \mathbf{x}_{t-1}^{(i)}\right)} = w_{t-1}^{(i)} p\left(\mathbf{y}_t \middle| \mathbf{x}_t^{(i)}\right).$$

Because PF(B) proposes particles using only the prior dynamics, it is very susceptible to weight degeneracy when observations are highly informative (such as systems with low measurement noise, or high-dimensional observations). Since the prior proposal does not account for the new observation $\mathbf{y}_t$ before moving the particles, the risk of moving particles to regions where the likelihood is near zero is substantial. The ill-placed particles receive low weights, while a few particles near the true state obtain high weights. One step of PF(B) is shown in Figure 2.7a.

Locally Optimal particle Filter

To account for the lack of observations in the proposal (2.8), we instead condition the proposal on *all* available observations $\mathbf{y}_{1:t}$, including the next one.

$$q\left(\mathbf{x}_t \middle| \mathbf{x}_{t-1}^{(i)}, \mathbf{y}_t\right) = p\left(\mathbf{x}_t \middle| \mathbf{x}_{t-1}^{(i)}, \mathbf{y}_{1:t}\right) \overset{\text{Markov}}{=} p\left(\mathbf{x}_t \middle| \mathbf{x}_{t-1}^{(i)}, \mathbf{y}_t\right). \tag{2.9}$$

We define (2.9) the locally optimal proposal.

Proposition 2.2 (Weight variance minimization proposal)**.** *The proposal distribution which, for any given distribution of particles P_{t-1} at time t, propagates the particles to the distribution P_t with the lowest weight variance is given by* (2.9)*.*

Proof. We wish to solve

$$\min_q \operatorname{Var}\left(\frac{w_t}{w_{t-1}}\right). \tag{2.10}$$

Denote by $\mathbf{x}_{t-1}$ the particles in P_{t-1}, and $\mathbf{x}_t$ the particle in P_t, originating from $\mathbf{x}_{t-1}$, propagated by $q(\mathbf{x}_t|\mathbf{y}_t, \mathbf{x}_{t-1})$. By the law of total variance, we get

$$\mathrm{Var}\left(\frac{w_t}{w_{t-1}}\right) = \underbrace{\mathbb{E}\left[\mathrm{Var}\left(\frac{w_t}{w_{t-1}}\middle|\mathbf{x}_{t-1}, \mathbf{y}_t\right)\right]}_{\text{(I)}:=} + \underbrace{\mathrm{Var}\left(\mathbb{E}\left[\frac{w_t}{w_{t-1}}\middle|\mathbf{x}_{t-1}, \mathbf{y}_t\right]\right)}_{\text{(II)}:=}. \tag{2.11}$$

Note from (2.6) that the weight update between steps $t-1$ and t is

$$\frac{w_t}{w_{t-1}} = C\frac{\tilde{w}_t}{\tilde{w}_{t-1}} = C\frac{p(\mathbf{y}_t|\mathbf{x}_t)\, p(\mathbf{x}_t|\mathbf{x}_{t-1})}{q(\mathbf{x}_t|\mathbf{y}_t, \mathbf{x}_{t-1})}, \tag{2.12}$$

where C is a constant independent of $\mathbf{x}_t$. Setting (2.12) in (2.11), we get

$$\begin{aligned}\mathbb{E}\left[\frac{w_t}{w_{t-1}}\middle|\mathbf{x}_{t-1}, \mathbf{y}_t\right] &= C\int \frac{p(\mathbf{y}_t|\mathbf{x}_t)\, p(\mathbf{x}_t|\mathbf{x}_{t-1})}{q(\mathbf{x}_t|\mathbf{y}_t, \mathbf{x}_{t-1})} q(\mathbf{x}_t|\mathbf{y}_t, \mathbf{x}_{t-1})\, \mathrm{d}\mathbf{x}_t \\ &= Cp(\mathbf{y}_t|\mathbf{x}_{t-1})\,,\end{aligned}$$

which does not depend on q, meaning that (II) does not depend on q either and can be disregarded in the minimization. Furthermore,

$$\mathrm{Var}\left(\frac{w_t}{w_{t-1}}\middle|\mathbf{x}_{t-1}, \mathbf{y}_t\right) = \mathrm{Var}\left(C\frac{p(\mathbf{y}_t|\mathbf{x}_t)\, p(\mathbf{x}_t|\mathbf{x}_{t-1})}{q(\mathbf{x}_t|\mathbf{y}_t, \mathbf{x}_{t-1})}\middle|\mathbf{x}_{t-1}, \mathbf{y}_t\right)$$

is equal to zero if the argument of the variance is constant. Because variances are non-negative, the smallest value (I) can take is zero. Finally, as

$$p(\mathbf{y}_t|\mathbf{x}_t)\, p(\mathbf{x}_t|\mathbf{x}_{t-1}) \propto p(\mathbf{x}_t|\mathbf{y}_t, \mathbf{x}_{t-1})\,,$$

it means that (2.9) minimizes (2.10), given a fixed value of $\mathbf{x}_{t-1}$ and $\mathbf{y}_t$. □

Proposition 2.2 claims that LO is the optimal choice of proposal distribution in terms of minimized weight variance between each successive time steps. Note however that this does not minimize the weight variance over the entire trajectory, but rather greedily chooses each individual step (hence *local*). The weight update in (2.6) can be simplified to

$$\begin{aligned} w_t^{(i)} &\propto w_{t-1}^{(i)} \frac{p\left(\mathbf{y}_t\middle|\mathbf{x}_t^{(i)}\right) p\left(\mathbf{x}_t^{(i)}\middle|\mathbf{x}_{t-1}^{(i)}\right)}{p\left(\mathbf{x}_t^{(i)}\middle|\mathbf{x}_{t-1}^{(i)}, \mathbf{y}_t\right)} = w_{t-1}^{(i)} \frac{p\left(\mathbf{y}_t\middle|\mathbf{x}_t^{(i)}\right) p\left(\mathbf{x}_t^{(i)}\middle|\mathbf{x}_{t-1}^{(i)}\right)}{\frac{p(\mathbf{y}_t|\mathbf{x}_t^{(i)}, \mathbf{x}_{t-1}^{(i)}) p(\mathbf{x}_t^{(i)}|\mathbf{x}_{t-1}^{(i)})}{p(\mathbf{y}_t|\mathbf{x}_{t-1}^{(i)})}} \\ &= w_{t-1}^{(i)} \frac{p\left(\mathbf{y}_t\middle|\mathbf{x}_{t-1}^{(i)}\right) p\left(\mathbf{y}_t\middle|\mathbf{x}_t^{(i)}\right)}{p\left(\mathbf{y}_t\middle|\mathbf{x}_t^{(i)}, \mathbf{x}_{t-1}^{(i)}\right)} = \left[\mathbf{y}_t|\mathbf{x}_{1:t} \sim \mathbf{y}_t|\mathbf{x}_t \text{ by (2.1)}\right] \\ &= w_{t-1}^{(i)} p\left(\mathbf{y}_t\middle|\mathbf{x}_{t-1}^{(i)}\right). \end{aligned} \tag{2.13}$$

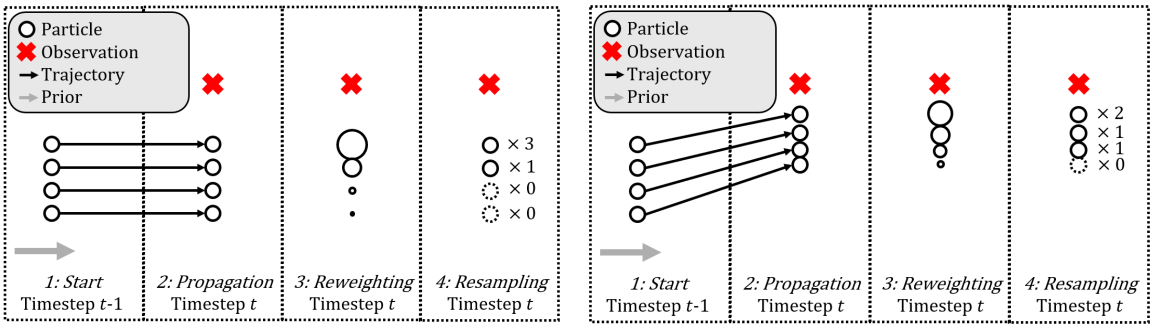
The primary advantage of LO over bootstrap is the ability to reduce weights degeneracy by utilizing the latest measurement to steer particles towards regions of high likelihood. However, it requires the ability to sample from

$$p\left(\mathbf{x}_t\middle|\mathbf{x}_{t-1}^{(i)}, \mathbf{y}_t\right)$$

and evaluate the integral

$$p\left(\mathbf{y}_t \middle| \mathbf{x}_{t-1}^{(i)}\right) = \int p(\mathbf{y}_t|\mathbf{x}_t)\, p\left(\mathbf{x}_t \middle| \mathbf{x}_{t-1}^{(i)}\right) \mathrm{d}\mathbf{x}_t$$

for the weight update. As noted by Arulampalam et al. [1], there are only two cases where these operations have an analytical solution: if the state $\mathbf{x}_t$ belongs to a strictly finite, discrete set, then the intractable integral simplifies into a finite sum, or if the system has linear or nonlinear dynamics but strictly linear measurements and the process and measurement noises are independent Gaussians. One step of PF(LO) is shown in Figure 2.7b.



(a) PF(B). **(b)** PF(LO).

Figure 2.7: Schematic showcase of the difference between PF(B) and PF(LO).

2.5 Variations on particle filters

Despite the reduced weight degeneracy from LO, deterioration over time can still be noticeable. Another way to combat weight degeneracy is by modifying the particle filter algorithm itself. Two variations on particle filters defined by Pitt & Shephard [13] and Klaas et al. [8] are the *auxiliary particle filter* (APF) and *marginal particle filter* (MPF), that each tackle different sources of the problem. Klaas et al. also combine both variations into the *auxiliary marginal particle filter* (AMPF). The notation of a particle filter architecture using a certain proposal is as defined in Section 2.4.3, only replacing PF with APF, MPF or AMPF.

2.5.1 Auxiliary particle filter

Weight degeneracy may develop as a consequence of unlikely observations. The APF aims to diminish this by selecting particles to propagate based on how likely they are to have given rise to the next observation, rather than propagating every particle unconditionally.

As for the regular particle filter, we aim to find a discrete approximation of the filtering distribution. As shown below, this takes the form of a sum over the particles

approximating the filtering distribution in the previous time step.

$$\begin{aligned} p(\mathbf{x}_t|\mathbf{y}_{1:t}) \approx \hat{p}(\mathbf{x}_t|\mathbf{y}_{1:t}) &\propto p(\mathbf{y}_t|\mathbf{x}_t) \sum_{i=1}^{N} w_{t-1}^{(i)} p\Big(\mathbf{x}_t\Big|\mathbf{x}_{t-1}^{(i)}\Big) \\ &= p(\mathbf{y}_t|\mathbf{x}_t) \sum_{i=1}^{N} w_{t-1}^{(i)} \frac{p\Big(\mathbf{y}_t\Big|\mathbf{x}_{t-1}^{(i)}\Big)\, p\Big(\mathbf{x}_t\Big|\mathbf{x}_{t-1}^{(i)}, \mathbf{y}_t\Big)}{p\Big(\mathbf{y}_t\Big|\mathbf{x}_t, \mathbf{x}_{t-1}^{(i)}\Big)} \\ &= \sum_{i=1}^{N} w_{t-1}^{(i)} p\Big(\mathbf{y}_t\Big|\mathbf{x}_{t-1}^{(i)}\Big)\, p\Big(\mathbf{x}_t\Big|\mathbf{x}_{t-1}^{(i)}, \mathbf{y}_t\Big)\,. \end{aligned} \tag{2.14}$$

To isolate the effect of each previous particle on the filtering distribution, we introduce the *auxiliary variable* k to each joint distribution in the mixture model (2.14)

$$\hat{p}(k, \mathbf{x}_t|\mathbf{y}_{1:t}) \propto w_{t-1}^{(k)} p\Big(\mathbf{y}_t\Big|\mathbf{x}_{t-1}^{(k)}\Big)\, p\Big(\mathbf{x}_t\Big|\mathbf{x}_{t-1}^{(k)}, \mathbf{y}_t\Big)\,.$$

Marginalizing out the particles $\mathbf{x}_t$, we calculate the effect that each previous particle index k alone has on the filtering distribution, and lets us quantify how important the corresponding previous particle $\mathbf{x}_{t-1}^{(k)}$ is, given the next observation.

$$\hat{p}(k|\mathbf{y}_{1:t}) \propto w_{t-1}^{(k)} p\Big(\mathbf{y}_t\Big|\mathbf{x}_{t-1}^{(k)}\Big) = w_{t-1}^{(k)} \int p(\mathbf{y}_t|\mathbf{x}_t)\, p\Big(\mathbf{x}_t\Big|\mathbf{x}_{t-1}^{(k)}\Big)\, \mathrm{d}\mathbf{x}_t \tag{2.15}$$

Because the integral in (2.15) is usually intractable, it must be approximated. We do this by replacing the conditioning on $\mathbf{x}_t$ in the likelihood with a deterministic value $\mu_t^{(k)}$ associated with the distribution $p\Big(\mathbf{x}_t\Big|\mathbf{x}_{t-1}^{(k)}\Big)$, such as mean or mode (that represents a likely state). This gives the *simulation weights*

$$\lambda_{t-1}^{(k)} \propto w_{t-1}^{(k)} p\Big(\mathbf{y}_t\Big|\mu_t^{(k)}\Big) \approx \hat{p}(k|\mathbf{y}_{1:t})\,.$$

Using these approximations, we make use of a proposal on the form $q(k, \mathbf{x}_t|\mathbf{y}_{1:t}) = q(k|\mathbf{y}_{1:t})\, q(\mathbf{x}_t|\mathbf{y}_{1:t}, k)$, where

$$\begin{aligned} &q(k|\mathbf{y}_{1:t}) = \lambda_{t-1}^{(k)}, \text{ and} \\ &q(\mathbf{x}_t|\mathbf{y}_{1:t}, k) = q\Big(\mathbf{x}_t\Big|\mathbf{x}_{t-1}^{(k)}, \mathbf{y}_t\Big)\,. \end{aligned}$$

The algorithm works by sampling from the joint density $q(k, \mathbf{x}_t|\mathbf{y}_{1:t})$ N times yielding $\Big(\mathbf{x}_t^{(i)}, k_i\Big)_{i=1}^{N}$, and updating the weights

$$w_t^{(i)} = \frac{\hat{p}\Big(k_i, \mathbf{x}_t^{(i)}\Big|\mathbf{y}_{1:t}\Big)}{q\Big(k_i, \mathbf{x}_t^{(i)}\Big|\mathbf{y}_{1:t}\Big)} \propto \frac{w_{t-1}^{(k_i)} p\Big(\mathbf{y}_t\Big|\mathbf{x}_t^{(i)}\Big)\, p\Big(\mathbf{x}_t^{(i)}\Big|\mathbf{x}_{t-1}^{(k_i)}\Big)}{\lambda_{t-1}^{(k_i)} q\Big(\mathbf{x}_t^{(i)}\Big|\mathbf{x}_{t-1}^{(k_i)}, \mathbf{y}_t\Big)}. \tag{2.16}$$

Note especially that the algorithm allows for duplicates of k, while omitting others entirely. A summary of the algorithm is given in Algorithm 3.

Algorithm 3 Auxiliary Particle Filter

$N \leftarrow$ number of particles
$\mathbf{x}_0^{(i)} \sim \nu$ for $i = 1, ..., N$
$w_0^{(i)} \leftarrow \frac{1}{N}$ for $i = 1, ..., N$
for $t = 1, ..., T$ **do**
  ▷ ***Define*** $\mu_t^{(i)}$ *and* $\lambda_{t-1}^{(i)}$. ◁
  $\mu_t^{(i)} \leftarrow_d p\left(\mathbf{x}_t \middle| \mathbf{x}_{t-1}^{(i)}\right)$, $i = 1, ..., N$
  $\lambda_{t-1}^{(i)} \propto w_{t-1}^{(i)} p\left(\mathbf{y}_t \middle| \mu_t^{(i)}\right)$, $i = 1, ..., N$
  ▷ ***Sample*** k_i, $\mathbf{x}_t^{(i)}$. ◁
  $k_i = j$ with probability $\lambda_{t-1}^{(j)}$, $i = 1, ..., N$
  $\mathbf{x}_t^{(i)} \sim q\left(\mathbf{x}_t \middle| \mathbf{x}_{t-1}^{(k_i)}, \mathbf{y}_t\right)$, $i = 1, ..., N$
  Update $w_t^{(i)}$ as in (2.16) and **norm** as in (2.7)
  Resample according to some criterion

We can think of the auxiliary particle filter as correcting the weights at time $t-1$ for the observation at time t and resampling according to this metric before propagating the particles [16]. This is a heuristic, as we hope to discard particles that we believe won't contribute to the distribution very much in favor of those that we believe will contribute more. Pitt & Shephard [13] and Klaas et al. [8] demonstrate that the added flexibility leads to less weight decay due to unlikely observations compared to regular particle filters. Furthermore, the modifications made to the general particle filter algorithm does not change the complexity, which is still $\mathcal{O}(N)$.

2.5.2 Marginal particle filters

As mentioned in Section 2.4.2, another cause of weight degeneracy is the use of the trajectory distribution $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$ in the standard particle filter, that leads to an ever expanding trajectory space $\mathbf{x}_{0:t}$.

The *marginal particle filter* (MPF) was developed by Klaas et al. [8] to resolve this problem by marginalizing out $\mathbf{x}_0, ..., \mathbf{x}_{t-1}$ and thus approximating $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ directly. They prove that MPFs greatly reduce the variance of the weights compared to standard particle filters and thus also alleviate the weight degeneracy problem.

The filtering distribution at time t is computed as

$$\begin{aligned} p(\mathbf{x}_t|\mathbf{y}_{1:t}) &\propto p(\mathbf{y}_t|\mathbf{x}_t)\, p(\mathbf{x}_t|\mathbf{y}_{1:t-1}) \\ &= p(\mathbf{y}_t|\mathbf{x}_t) \int p(\mathbf{x}_t|\mathbf{x}_{t-1})\, p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})\, \mathrm{d}\mathbf{x}_{t-1}. \end{aligned} \tag{2.17}$$

However, as previously mentioned, the integral in (2.17) is generally intractable. MPFs approximate it by utilizing the fact that we have an approximate representation of the previous filtering distribution $p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})$ in the form of weight/particle

pairs $\left\{\mathbf{x}_{t-1}^{(i)}, w_{t-1}^{(i)}\right\}$. The integral is then approximated as

$$\int p(\mathbf{x}_t|\mathbf{x}_{t-1})\, p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})\, \mathrm{d}\mathbf{x}_{t-1} \approx \sum_{j=1}^{N} w_{t-1}^{(j)} p\left(\mathbf{x}_t \middle| \mathbf{x}_{t-1}^{(j)}\right). \tag{2.18}$$

The proposal to be sampled from in MPFs can be chosen among any distribution with appropriate support, but Klaas et al. note that it is convenient to sample from a distribution which takes a similar form to (2.18), i.e. the mixture of proposals

$$\sum_{j=1}^{N} w_{t-1}^{(j)} q\left(\mathbf{x}_t \middle| \mathbf{y}_t, \mathbf{x}_{t-1}^{(j)}\right).$$

The weight update then becomes

$$w_t^{(i)} \propto p\left(\mathbf{y}_t \middle| \mathbf{x}_t^{(i)}\right) \frac{\sum_{j=1}^{N} w_{t-1}^{(j)} p\left(\mathbf{x}_t^{(i)} \middle| \mathbf{x}_{t-1}^{(j)}\right)}{\sum_{j=1}^{N} w_{t-1}^{(j)} q\left(\mathbf{x}_t^{(i)} \middle| \mathbf{y}_t, \mathbf{x}_{t-1}^{(j)}\right)}, \quad i = 1, ..., N. \tag{2.19}$$

The algorithm is described in Algorithm 4.

Algorithm 4 Marginal Particle Filter

1: $N \leftarrow$ number of particles
2: $\mathbf{x}_0^{(i)} \sim \nu$ for $i = 1, ..., N$
3: $w_0^{(i)} \leftarrow \frac{1}{N}$ for $i = 1, ..., N$
4: **for** $t = 1, ..., T$ **do**
5: **Sample** $\mathbf{x}_t^{(i)} \sim \sum_{j=1}^{N} w_{t-1}^{(j)} q\left(\mathbf{x}_t \middle| \mathbf{y}_t, \mathbf{x}_{t-1}^{(j)}\right)$, $i = 1, ..., N$
6: **Update** $w_t^{(i)}$ as in (2.19) and **norm** as in (2.7)

Intuitively, MPFs on this form work by resampling all particles based on their weights before moving them and updating the steps. The update then marginalizes over every possible previous state that the particles could have had. This also means that the complexity also increases to $\mathcal{O}(N^2)$, compared to $\mathcal{O}(N)$ for a regular or auxiliary particle filter.

2.5.3 Auxiliary marginal particle filter

Klaas et al. show that MPFs are also prone to higher weight variance when unlikely observations appear [8]. They propose the addition of an auxiliary variable to the MPF to define the AMPF.

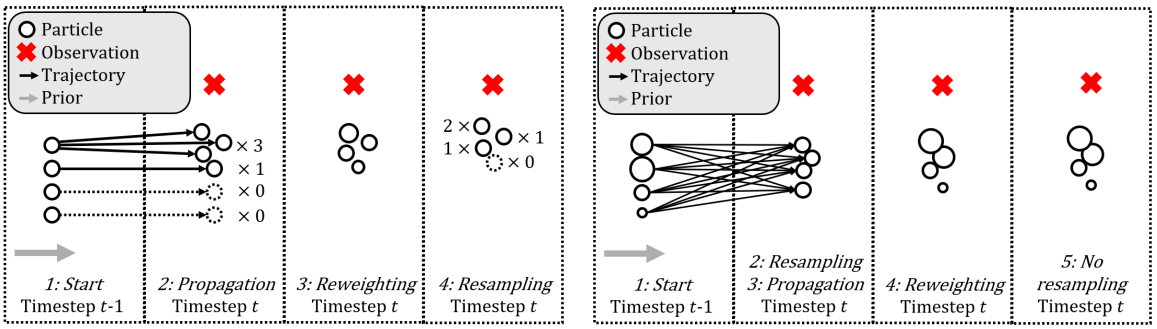
Mathematically, the sampling occurs from the same mixture as MPF, but weighting the mixtures by $\lambda_{t-1}^{(j)}$ as opposed to $w_{t-1}^{(j)}$. The AMPF weight update becomes

$$w_t^{(i)} \propto p\left(\mathbf{y}_t \middle| \mathbf{x}_t^{(i)}\right) \frac{\sum_{j=1}^{N} w_{t-1}^{(j)} p\left(\mathbf{x}_t^{(i)} \middle| \mathbf{x}_{1:t-1}^{(j)}\right)}{\sum_{j=1}^{N} \lambda_{t-1}^{(j)} q\left(\mathbf{x}_t^{(i)} \middle| \mathbf{y}_t, \mathbf{x}_{t-1}^{(j)}\right)}, \quad i = 1, ..., N. \tag{2.20}$$

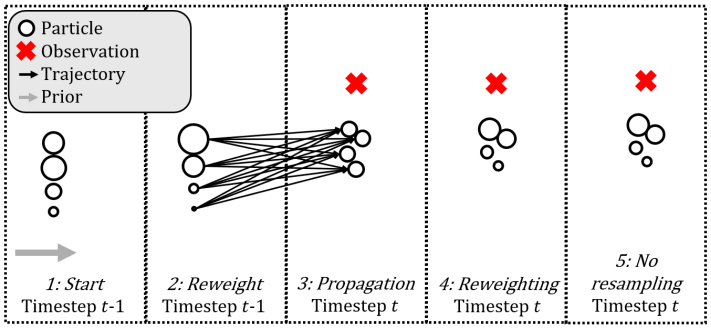
The algorithm is described in Algorithm 5.

Algorithm 5 Auxiliary Marginal Particle Filter

1: $N \leftarrow$ number of particles
2: $\mathbf{x}_0^{(i)} \sim \nu$ for $i = 1, ..., N$
3: $w_0^{(i)} \leftarrow \frac{1}{N}$ for $i = 1, ..., N$
4: **for** $t = 1, ..., T$ **do**
5: ▷ ***Define*** $\mu_t^{(i)}$, $\lambda_{t-1}^{(i)}$. ◁
6: $\mu_t^{(i)} \leftarrow_d p\left(\mathbf{x}_t \middle| \mathbf{x}_{t-1}^{(i)}\right)$, $i = 1, ..., N$
7: $\lambda_{t-1}^{(i)} \propto w_{t-1}^{(i)} p\left(\mathbf{y}_t \middle| \mu_t^{(i)}\right)$, $i = 1, ..., N$
8: ▷ ***Sample*** $\mathbf{x}_t^{(i)}$. ◁
9: $\mathbf{x}_t^{(i)} \sim \sum_{j=1}^{N} \lambda_{k-1}^{(j)} q\left(\mathbf{x}_t \middle| \mathbf{y}_t, \mathbf{x}_{t-1}^{(j)}\right)$, $i = 1, ..., N$
10: ▷ ***Update*** $w_t^{(i)}$. ◁
11: **Update** $w_t^{(i)}$ as in (2.20) and **norm** as in (2.7)



(a) APF(B). **(b)** MPF(B).



(c) AMPF(B).

Figure 2.8: Schematic showcase of the difference between APF, MPF and AMPF, using a bootstrap proposal.

The AMPF can be intuitively explained much like APFs as the particles being resampled according to a combination of their weights and the next observation before being propagated. This is unlike MPFs in general, which resample particles only based on their weights. However, AMPFs differentiate themselves from APFs in that their particles are reweighted not according to their previous states, but

rather according to every possible anterior value. The particles thus "forget" the trajectory, i.e. we are looking at the marginal space rather than the joint one. This last trait is shared with MPFs in general. It also sports the same complexity as the latter, that is $\mathcal{O}(N^2)$.

The conceptual differences between APF, MPF and AMPF are shown in Figure 2.8. This can be compared to Figure 2.7, which illustrates regular particle filters.

2.5.4 Particle filter variations and proposals synergies

Note that the two sums in the MPF weight update (2.19)

$$\sum_{j=1}^{N} w_{t-1}^{(j)} p\left(\mathbf{x}_t^{(i)} \middle| \mathbf{x}_{1:t-1}^{(j)}\right)$$
$$\sum_{j=1}^{N} w_{t-1}^{(j)} q\left(\mathbf{x}_t^{(i)} \middle| \mathbf{y}_t, \mathbf{x}_{t-1}^{(j)}\right)$$

are equal for bootstrap proposal. This means that MPF(B) gives no improvement compared to PF(B). Furthermore, Klaas et al. [8] claim that AMPF(LO) gives no improvement over APF(LO) in the case where we can sample from (2.9) directly and evaluate (2.13) exactly. In fact, the importance weight variance is zero for APF, thus MPF brings no improvements in the theoretical case. In practice, however, approximations in the algorithm may still lead to some weight variance. In all other cases, AMPF has lower weight variance than APF [8, Proposition 1].

3

Schrödinger bridges

While both the LO and bootstrap proposals have advantages and disadvantages, they share the need for an update step which could potentially give rise to weight degeneracy. This raises the question: is it possible to propagate the particles to the filtering distribution directly, avoiding the weight update entirely?

A concept from optimal transport called the *Schrödinger bridge* offers a theoretical framework for this. Given a prior on the physical movement, the Schrödinger bridge finds the most likely law that connects the two known distributions ρ_t and ρ_{t+1}. In a tracking application, these boundary distributions would be the filtering distributions. However, in a live tracking scenario the target distribution ρ_{t+1} is inherently unknown.

It is possible to approximate this target distribution by first running a standard filter and subsequently solving the optimal transport equations numerically. Although this two-step numerical process is computationally heavy and generally too slow for real-time applications, evaluating its performance is a core objective of this thesis. To eventually overcome this computational burden, we will explore learning the bridge dynamics using machine learning. We refer to this integration as the *Schrödinger bridge proposal*. While the full algorithmic implementations of Schrödinger bridge proposals in particle filters are presented in Chapter 4, this chapter introduces the necessary mathematical background.

While particle filters operate on discrete samples, classical Schrödinger bridge theory is rooted in continuous stochastic differential equations. This chapter aims to connect the two fields by looking at the problem through two different perspectives.

- First, we introduce the general continuous SDE formulation of the problem (Sections 3.1 and 3.2) and present its explicit, closed-form Linear-Gaussian solution (Section 3.3). This establishes an intuition for how the bridge operates and provides an exact baseline solution to the problem.
- We then pivot to a factorized formulation of the Schrödinger problem (Section 3.4 and 3.5), a version that lends itself to discrete, approximate solutions well-suited for particle filter usage.

We finish the chapter by discussing the usage of Schrödinger bridges in particle filters, both as a standalone particle transport mechanism, and as a proposal distri-

bution in a particle filter.

3.1 Problem formulation

To understand how the Schrödinger bridge optimally transports probability mass between two filtering distribution, we first define the problem in the continuous-time domain. The Schrödinger bridge aims to find the optimal stochastic process $\mathbb{P}^*$ that connects a known starting distribution ρ_0 at time t_0 to a target distribution ρ_1 at time t_1 with minimal deviation from a prior physical model. Let $\Omega = C([0,1], \mathbb{R}^d)$ be the space of all continuous paths in d-dimensions and let $\mathbb{Q}$ denote the system dynamics governed by the SDE

$$\mathrm{d}\mathbf{x}_t = \mathbf{b}(\mathbf{x}_t, t)\mathrm{d}t + \sigma(t)\mathrm{d}\mathbf{W}_t, \quad \mathbf{x}_0 \sim \rho_0 \tag{3.1}$$

where $\mathbf{W}_t$ is the d-dimensional Brownian motion, $\mathbf{b} : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$ is the drift coefficient representing the physical forces and $\sigma(t)$ is the diffusion coefficient. Then, let $\mathbb{P}$ denote a path measure on Ω, and $\mathbb{P}_t$ the marginal distribution at time t. Furthermore, let $\mathcal{D}(\rho_0, \rho_1) = \{\mathbb{P} : \mathbb{P}_0 = \rho_0, \mathbb{P}_1 = \rho_1\}$ be the set of path measures with marginals starting in ρ_0 and ending in ρ_1. The solution $\mathbb{P}^*$ to the *Schrödinger bridge problem* (SBP) can then be expressed as

$$\mathbb{P}^* = \underset{\mathbb{P} \in \mathcal{D}(\rho_0, \rho_1)}{\operatorname{argmin}} D_{\mathrm{KL}}(\mathbb{P} || \mathbb{Q}), \tag{3.2}$$

where D_{KL} is the Kullback-Leibler divergence.

3.2 General analytical solution

Solving the optimization problem (3.2) directly over the infinite-dimensional path space Ω is typically intractable. However, the problem can be split into two separate stages that offer efficient and tractable solutions: a static boundary problem (Section 3.2.1), and a dynamic trajectory problem (Section 3.2.2) [6].

The static solution determines the optimal coupling between the boundary distributions ρ_0 and ρ_1, completely ignoring the original optimal path problem to focus only on the start and end points. The dynamic solution then constructs the trajectory that connects the paired end distributions from the static solution.

3.2.1 Static problem

Instead of evaluating the entire continuous path measure, the static SBP focuses exclusively on the boundary distributions at t_0 and t_1. We denote a joint probability distribution $\pi(\mathbf{x}_0, \mathbf{x}_1)$ with marginals $\rho_0(\mathbf{x}_0)$ and $\rho_1(\mathbf{x}_1)$ as a coupling. The set of all such valid couplings is denoted as $\Pi(\rho_0, \rho_1)$.

The objective is to find the optimal coupling π^* which minimizes the relative entropy with respect to the joint distribution of the reference process evaluated at the

boundaries $\mathbb{Q}_{01}(\mathbf{x}_0, \mathbf{x}_1)$

$$\pi^* = \underset{\pi(\mathbf{x}_0,\mathbf{x}_1)\in\Pi(\rho_0,\rho_1)}{\operatorname{argmin}} D_{\mathrm{KL}}(\pi||\mathbb{Q}_{01}). \tag{3.3}$$

Here $\mathbb{Q}_{01}(\mathbf{x}_0, \mathbf{x}_1) = \rho_0(\mathbf{x}_0)q(t_0, \mathbf{x}_0, t_1, \mathbf{x}_1)$ describes the likelihood of moving from $\mathbf{x}_0$ to $\mathbf{x}_1$, following the reference dynamics (3.1), where q is the transition density.

3.2.2 Dynamic problem

Once the endpoint pairing π^* is found by the static problem, the optimal time-continuous trajectory can be calculated. The dynamic solution $\mathbb{P}^*$ takes the form of a modified stochastic differential equation. The general solution, provided by Huang [6], is given by the following theorem.

Theorem 3.1. *Assume that $\sigma(t) \in C^1([0,1])$, $\boldsymbol{x} \in \mathrm{R}^d$, and the components of $\boldsymbol{b}(\boldsymbol{x}, t)$ are bounded continuous and satisfy Hölder conditions with respect to $\boldsymbol{x}$. Then the solution probability measure $\mathbb{P}^*$ to the SBP (3.2) is induced by the following SDE with a modified drift:*

$$\mathrm{d}\boldsymbol{x}_t = [\boldsymbol{b}(\boldsymbol{x}_t, t) + \boldsymbol{u}^*(\boldsymbol{x}_t, t)]\mathrm{d}t + \sigma(t)\mathrm{d}\,\boldsymbol{W}_t, \qquad \boldsymbol{x}_0 \sim \rho_0$$

where the drift term

$$\boldsymbol{u}^*(\boldsymbol{x}, t) = \frac{\sigma(t)^2 \int s_{\boldsymbol{x}_1}(\boldsymbol{x}, t) g_t(\boldsymbol{x}, \boldsymbol{x}_0, \boldsymbol{x}_1)\pi^*(\mathrm{d}\boldsymbol{x}_0, \mathrm{d}\boldsymbol{x}_1)}{\int g_t(\boldsymbol{x}, \boldsymbol{x}_0, \boldsymbol{x}_1)\pi^*(\mathrm{d}\boldsymbol{x}_0, \mathrm{d}\boldsymbol{x}_1)}$$

where π^* is the solution to the static problem (3.3) and $s_{x_1}(\boldsymbol{x}, t)$ is the conditional score defined as

$$s_{\boldsymbol{x}_1}(\boldsymbol{x}_t, t) = \nabla_{\boldsymbol{x}_t} \log q(t, \boldsymbol{x}_t, 1, \boldsymbol{x}_1)$$

and

$$g_t(\boldsymbol{x}, \boldsymbol{x}_0, \boldsymbol{x}_1) = \frac{q(0, \boldsymbol{x}_0, t, \boldsymbol{x})q(t, \boldsymbol{x}, 1, \boldsymbol{x}_1)}{q(0, \boldsymbol{x}_0, 1, \boldsymbol{x}_1)}.$$

The modified drift term $\mathbf{u}^*$ is a correction to the reference drift $\mathbf{b}$ that enforces the marginal constraints of the SBP. The correction biases the dynamics toward paths consistent with the optimal coupling π^*. The function g_t corresponds to the likelihood of the reference process passing through the state $\mathbf{x}$ given the system dynamics and the endpoint states, while the score term $s_{\mathbf{x}_1}$ describes how variation in the current state changes the likelihood of reaching the final state $\mathbf{x}_1$.

3.3 Explicit linear-Gaussian solution

The general solution to the Schrödinger bridge in Theorem 3.1 relies on integrals that are usually intractable. However, the SBP admits a closed-form analytical solution in the case where the reference dynamics $\mathbb{Q}$ is linear and when the marginal

constrains ρ_0, ρ_1 are both Gaussian. We present the explicit solution below suggested by Zhang and Stumpf [17], as a mathematically exact baseline to the Schrödinger problem. We will refer to this solution as the Gaussian Schrödinger bridge.

Let the state be $\mathbf{X}_t \in \mathbb{R}^d$. We then limit the general drift $\mathbf{b}(x,t)$ to an affine form, modeling the reference dynamics as a multivariate Ornstein-Uhlenbeck process

$$\mathrm{d}\mathbf{X}_t = \mathbf{A}(\mathbf{X}_t - \mathbf{m})\mathrm{d}t + \boldsymbol{\sigma}\mathrm{d}\mathbf{W}_t$$

where $\mathbf{A} \in \mathbb{R}^{d\times d}$ is the drift matrix, $\mathbf{m} \in \mathbb{R}^d$ is the mean and $\boldsymbol{\sigma} \in \mathbb{R}^{d\times d}$ is the diffusion matrix. The marginal constraints are given by $\rho_0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$, $\rho_1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$.

Theorem 3.2 (Gaussian Schrödinger bridge)**.** *Given the linear reference dynamics and Gaussian boundary marginals defined above, the optimal joint distribution connecting the initial and end state (the static problem solution) is given by the multivariate Gaussian*

$$\boldsymbol{\pi}^* = \mathcal{N}\left(\begin{bmatrix} \bar{\boldsymbol{\mu}}_0 \\ \bar{\boldsymbol{\mu}}_1 \end{bmatrix}, \begin{bmatrix} \bar{\boldsymbol{\Sigma}}_0 & \bar{\boldsymbol{\Sigma}}_{01} \\ \bar{\boldsymbol{\Sigma}}_{01}^\top & \bar{\boldsymbol{\Sigma}}_1 \end{bmatrix} \right),$$

where $\bar{\boldsymbol{\Sigma}}_{01}$ minimizes the transport cost relative to the reference dynamics, given by

$$\bar{\boldsymbol{\Sigma}}_{01} = \bar{\boldsymbol{\Sigma}}_0^{1/2}\left(\bar{\boldsymbol{\Sigma}}_0^{1/2}\bar{\boldsymbol{\Sigma}}_1\bar{\boldsymbol{\Sigma}}_0^{1/2} + \frac{1}{4}\boldsymbol{I}\right)^{1/2}\bar{\boldsymbol{\Sigma}}_0^{-1/2} - \frac{1}{2}\boldsymbol{I}.$$

The trajectory connecting the initial and end state (the dynamic problem solution) is a Gaussian process characterized by a time varying mean $\boldsymbol{\mu}_t$ and covariance $\boldsymbol{\Sigma}_t$

$$\mathrm{d}\boldsymbol{X}_t = \Big(\dot{\boldsymbol{\mu}}_t + \mathbf{S}_t^\top \boldsymbol{\Sigma}_t^{-1}(\boldsymbol{X}_t - \boldsymbol{\mu}_t)\Big)\mathrm{d}t + \boldsymbol{\sigma}\,\mathrm{d}\,\boldsymbol{W}_t, \qquad \boldsymbol{X}_0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$$

where the mean and covariance interpolate between the boundaries according to

$$\begin{aligned} \boldsymbol{\mu}_t &= \boldsymbol{W}_t^{(0)}\bar{\boldsymbol{\mu}}_0 + \boldsymbol{W}_t^{(1)}\bar{\boldsymbol{\mu}}_1 + \boldsymbol{d}_t \\ \boldsymbol{\Sigma}_t &= \boldsymbol{\Omega}_t + \boldsymbol{W}_t^{(0)}\bar{\boldsymbol{\Sigma}}_0\,\boldsymbol{W}_t^{(0)\top} + \boldsymbol{W}_t^{(1)}\bar{\boldsymbol{\Sigma}}_1\,\boldsymbol{W}_t^{(1)\top} + \boldsymbol{W}_t^{(0)}\bar{\boldsymbol{\Sigma}}_{01}\,\boldsymbol{W}_t^{(1)\top} + \boldsymbol{W}_t^{(1)}\bar{\boldsymbol{\Sigma}}_{01}^\top\,\boldsymbol{W}_t^{(0)\top} \end{aligned}$$

The moments above are constructed using transformed parameters that align the boundary constraints with the trajectory of the prior, accounting for any potential rotations in the system.

- $\bar{\boldsymbol{\mu}}_0 = \boldsymbol{\Phi}_1^{-1/2}(e^{t_1\boldsymbol{A}}(\boldsymbol{\mu}_0 - \boldsymbol{m}) + \boldsymbol{m})$
- $\bar{\boldsymbol{\Sigma}}_0 = \boldsymbol{\Phi}_1^{-1/2}e^{t_1\boldsymbol{A}}\boldsymbol{\Sigma}_0 e^{t_1\boldsymbol{A}^\top}\boldsymbol{\Phi}_1^{-1/2}$
- $\bar{\boldsymbol{\mu}}_1 = \boldsymbol{\Phi}_1^{-1/2}\boldsymbol{\mu}_1$
- $\bar{\boldsymbol{\Sigma}}_1 = \boldsymbol{\Phi}_1^{-1/2}\boldsymbol{\Sigma}_1\boldsymbol{\Phi}_1^{-1/2}$

where $\boldsymbol{\Phi}_t = \int_0^t e^{(t-s)\boldsymbol{A}}\boldsymbol{\sigma}\boldsymbol{\sigma}^\top e^{(t-s)\boldsymbol{A}^\top}\mathrm{d}s$.

The dynamic solution is also governed by mixing weights matrices, where $\boldsymbol{W}_t^{(0)}$ fades the influence of the start over time, $\boldsymbol{W}_t^{(1)}$ increases the influence of the target and $\boldsymbol{d}_t$ pulls the path toward the equilibrium $\boldsymbol{m}$.

- $\boldsymbol{\Gamma}_t = \boldsymbol{\Phi}_t e^{(t_1-t)\boldsymbol{A}^\top} \boldsymbol{\Phi}_1^{-1}$
- $\boldsymbol{W}_t^{(0)} = \left(e^{-(t_1-t)\boldsymbol{A}} - \boldsymbol{\Gamma}_t\right)\boldsymbol{\Phi}_1^{1/2}$
- $\boldsymbol{W}_t^{(1)} = \boldsymbol{\Gamma}_t \boldsymbol{\Phi}_1^{1/2}$
- $\boldsymbol{d}_t = \left(\boldsymbol{I} - e^{-(t_1-t)\boldsymbol{A}}\right)\boldsymbol{m}$
- $\boldsymbol{\Omega}_t = \boldsymbol{\Phi}_t - \boldsymbol{\Gamma}_t e^{(1-t)\boldsymbol{A}} \boldsymbol{\Phi}_t$

Finally, $\dot{\boldsymbol{\mu}}$ is the time derivative of the mean and with

- $\boldsymbol{S}_t = (\partial_{t'}\boldsymbol{\Omega}_{t,t'})(t) + \boldsymbol{W}_t^{(0)}\bar{\boldsymbol{\Sigma}}_0\,\dot{\boldsymbol{W}}_t^{(0)\top} + \boldsymbol{W}_t^{(0)}\bar{\boldsymbol{\Sigma}}_{01}\,\dot{\boldsymbol{W}}_t^{(1)\top} + \boldsymbol{W}_t^{(1)}\bar{\boldsymbol{\Sigma}}_{01}^\top\,\dot{\boldsymbol{W}}_t^{(0)\top} + \boldsymbol{W}_t^{(1)}\bar{\boldsymbol{\Sigma}}_1\,\dot{\boldsymbol{W}}_t^{(1)\top}$
- $(\partial t'\boldsymbol{\Omega}_{t,t'})(t) = \boldsymbol{\Phi}_t\boldsymbol{A}^\top - \boldsymbol{\Gamma}_t e^{(t_1-t)\boldsymbol{A}}(-\boldsymbol{A}\boldsymbol{\Phi}_t + e^{t\boldsymbol{A}}\boldsymbol{\sigma}\boldsymbol{\sigma}^\top e^{t\boldsymbol{A}^\top})$

Due to the mathematical complexity and limited relevance of the analytical solution to the data-driven focus of this thesis, a full derivation falls outside of the scope of this thesis. A full discussion of the Gaussian solution with proof, can be found in [17].

3.4 Factorization of the Schrödinger bridge

The explicit SDE solution derived in the previous section is exact but limited in the same way as the Kalman filter, as it is only applicable to linear-Gaussian problems. We seek a solution to the Schrödinger bridge that can be applied to arbitrary, non-linear distributions. We thus step away from the continuous drift correction $\mathbf{u}^*(x,t)$ defined in Section 3.2, and return to the static boundary problem. By reformulating the static coupling, we arrive at a factorized expression for the static Schrödinger bridge π^* that naturally lends itself to data-driven approximations [12].

We remind ourselves of the objective: we seek an optimal coupling π^* that minimizes the relative entropy $D_{\mathrm{KL}}(\pi||\mathbb{Q}_{01})$ with respect to the prior joint distribution $\mathbb{Q}_{01}(\mathbf{x}_0,\mathbf{x}_1) = \rho_0(\mathbf{x}_0)q(t_0,\mathbf{x}_0,t_1,\mathbf{x}_1)$, subject to the marginal constraints ρ_0 and ρ_1. This optimization problem can be formulated using the method of Lagrange multipliers. We construct a Lagrangian functional $\mathcal{L}(\pi;\lambda,\mu)$ with multipliers $\lambda(\mathbf{x}_0)$ and $\mu(\mathbf{x}_1)$ for the initial and final marginal constraints, respectively.

$$\begin{aligned}\mathcal{L}(\pi;\lambda,\mu) = &\iint \left[\log\frac{\pi(\mathbf{x}_0,\mathbf{x}_1)}{\mathbb{Q}_{01}(\mathbf{x}_0,\mathbf{x}_1)}\right]\pi(\mathbf{x}_0,\mathbf{x}_1)\mathrm{d}\mathbf{x}_0\mathrm{d}\mathbf{x}_1\\ &+\int\lambda(\mathbf{x}_0)\left[\int\pi(\mathbf{x}_0,\mathbf{x}_1)\mathrm{d}\mathbf{x}_1 - \rho_0(\mathbf{x}_0)\right]\mathrm{d}\mathbf{x}_0\\ &+\int\mu(\mathbf{x}_1)\left[\int\pi(\mathbf{x}_0,\mathbf{x}_1)\mathrm{d}\mathbf{x}_0 - \rho_1(\mathbf{x}_1)\right]\mathrm{d}\mathbf{x}_1\end{aligned}$$

To find the optimal coupling, we set the first variation of the Lagrangian with respect to π equal to zero.

$$1 + \log\pi^*(\mathbf{x}_0,\mathbf{x}_1) - \log q(t_0,\mathbf{x}_0,t_1,\mathbf{x}_1) - \log\rho_0(\mathbf{x}_0) + \lambda(\mathbf{x}_0) + \mu(\mathbf{x}_1) = 0$$

Rearranging this expression, we get

$$\frac{\pi^*(\mathbf{x}_0, \mathbf{x}_1)}{q(t_0, \mathbf{x}_0, t_1, \mathbf{x}_1)} = \exp[\log \rho_0(\mathbf{x}_0) - 1 - \lambda(\mathbf{x}_0)] \exp[-\mu(\mathbf{x}_1)].$$

Notice that the ratio of the optimal coupling to the prior transition consists of two independent factors: the first depending only on $\mathbf{x}_0$ and the second only on $\mathbf{x}_1$. By defining two continuous potential functions $\hat{\varphi}_0(\mathbf{x}_0) := \exp[\log \rho_0(\mathbf{x}_0) - 1 - \lambda(\mathbf{x}_0)]$ and $\varphi_1(\mathbf{x}_1) := \exp[-\mu(\mathbf{x}_1)]$, the optimal coupling can be written in the factorized form

$$\pi^*(\mathbf{x}_0, \mathbf{x}_1) = \hat{\varphi}_0(\mathbf{x}_0) q(t_0, \mathbf{x}_0, t_1, \mathbf{x}_1) \varphi_1(\mathbf{x}_1). \tag{3.4}$$

For the factorization to hold, the potentials must satisfy

$$\begin{aligned} \rho_0(\mathbf{x}_0) &= \hat{\varphi}_0(\mathbf{x}_0) \int q(t_0, \mathbf{x}_0, t_1, \mathbf{x}_1) \varphi_1(\mathbf{x}_1) \mathrm{d}\mathbf{x}_1 \\ \rho_1(\mathbf{x}_1) &= \varphi_1(\mathbf{x}_1) \int q(t_0, \mathbf{x}_0, t_1, \mathbf{x}_1) \hat{\varphi}_0(\mathbf{x}_0) \mathrm{d}\mathbf{x}_0. \end{aligned}$$

Once the potentials are found, any intermediate state is given by the product $\rho_t(\mathbf{z}) = \varphi_t(\mathbf{z})\hat{\varphi}_t(\mathbf{z})$, solving the dynamic problem. Here, $\hat{\varphi}_t(\mathbf{z})$ is the forward evolution of the factor $\hat{\varphi}_0$ under the prior, and $\varphi_t(\mathbf{z})$ is the backward evolution of the factor φ_1, given by

$$\begin{aligned} \hat{\varphi}_t(t, \mathbf{x}_t) &= \int q(t_0, \mathbf{x}_0, t, \mathbf{x}_t) \hat{\varphi}_0(\mathbf{x}_0) \mathrm{d}\mathbf{x}_0 \\ \varphi_t(t, \mathbf{x}_t) &= \int q(t, \mathbf{x}_t, t_1, \mathbf{x}_1) \varphi_1(\mathbf{x}_1) \mathrm{d}\mathbf{x}_1. \end{aligned} \tag{3.6}$$

The two potentials $\hat{\varphi}_0$ and φ_1 adjust the prior dynamics to fit the marginal distributions, serving a similar role to the correcting drift $\mathbf{u}^*$ above.

3.5 Data driven approximations

In practical applications, such as particle filtering, the marginal distributions ρ_0 and ρ_1 are rarely available as continuous analytical functions. Instead, they are represented by finite sets of samples $\{\mathbf{x}_0^{(i)}\}_{i=1}^m$ and $\{\mathbf{x}_1^{(j)}\}_{j=1}^n$. Consequently, the continuous boundary integrals (3.6) governing the potentials $\hat{\varphi}_0$ and φ_1 cannot be solved directly. We look at two methods to approximate these potentials both based on the Sinkhorn algorithm: one discrete matrix solution and one based on maximum likelihood estimation.

3.5.1 Discrete Sinkhorn

The discrete Sinkhorn algorithm solves the static SBP by evaluating the prior transition dynamics strictly through the pairwise connections between the specific data points. The dynamics is captured by constructing a matrix $K \in \mathbb{R}^{m \times n}$, with entries $K_{ij} = q(t_0, \mathbf{x}_0^{(i)}, t_1, \mathbf{x}_1^{(j)})$ representing the likelihood of moving from sample i to target j under the prior dynamics. Moreover, the functions $\hat{\varphi}_0$ and φ_1 are replaced by

strictly positive finite-dimensional vectors $u \in \mathbb{R}^m$ and $v \in \mathbb{R}^n$. If we define $D(w)$ to be the diagonal matrix with elements given by a generic vector w, the optimal discrete coupling matrix π^* is then given by

$$\pi^* = D(u)KD(v)^{-1},$$

where u and v are chosen such that the rows of π^* sum to ρ_0 and the columns of π^* sum to ρ_1. This is the discrete analogue to the factorized expression (3.4). To find these vectors we use Algorithm 6.

Algorithm 6 Sinkhorn

Input: ρ_0, ρ_1 and K
$u_0, v_0 \leftarrow \mathbb{1}$
$\pi_0 \leftarrow D(u_0)KD(v_0)^{-1}$
$k \leftarrow 0$
while $\pi^\top \mathbb{1} \not\approx \rho_0$ or $\pi \mathbb{1} \not\approx \rho_1$ **do**
    $u_{k+1} \leftarrow D(\pi_{2k}\mathbb{1})^{-1}\rho_1$
    $\pi_{2k+1} \leftarrow D(u_{k+1})\pi_{2k}$
    $v_{k+1} \leftarrow D(\rho_0)^{-1}\pi_{2k+1}^\top \mathbb{1}$
    $\pi_{2k+2} \leftarrow \pi_{2k+1}D(v_{k+1})^{-1}$
    $k \leftarrow k + 1$

Algorithm 6 operates by alternating between two steps. It first scales the variables to ensure that the rows of the coupling matrix sum to the initial marginal ρ_0. It then scales the variables to ensure the columns sum to the target marginal ρ_1. Because enforcing the column constraint inherently breaks the previously established row constraint, the algorithm must alternate between these two operations iteratively until convergence is achieved and both marginal constraints are simultaneously satisfied.

Intuitively, the algorithm could be seen as trying to fit under a blanket which is exactly as long as you. At each iteration, you first cover your head with the blanket, leaving your feet cold. You then draw the blanket over your feet, leaving your head cold. At each iteration, the blanket rotates slightly such that it eventually covers both your feet and head perfectly.

While the Sinkhorn algorithm is well documented, relatively simple to implement and proven to converge, it scales poorly in the particle filter context. Because it requires constructing and repeatedly updating the $N \times N$ K matrix, its computational complexity scales quadratically $\mathcal{O}(N^2)$. This makes the calculations slow for larger data sets.

3.5.2 Continuous Sinkhorn

A method to solve the SBP based on maximum likelihood estimation was suggested by Pavon et. al. [12]. Unlike the discrete Sinkhorn algorithm, which restricts the solution to a finite transition matrix, this method parameterizes the potentials as continuous functions. This continuous approximation allows us to evaluate the

Schrödinger potentials across the entire state space, rather than being restricted to the empirical data points.

To build this continuous solution, the algorithm is conceptually divided into two parts. First, we examine the *half-bridge* problem, which only imposes one of the boundary constraints. Second, we expand the same logic into the *full-bridge* problem, which alternates between enforcing the initial and final constraints until both are satisfied simultaneously, a continuous analogue to the discrete Sinkhorn algorithm.

Half-bridge problem We consider the static SBP, but instead of solving the fully constrained smoothing problem, we define the half-bridge problem where only the final target constraint is imposed.

$$\pi^* = \arg\min_{\pi} \iint \pi(\mathbf{x}_0, \mathbf{x}_1) \log\left(\frac{\pi(\mathbf{x}_0, \mathbf{x}_1)}{\rho_0(\mathbf{x}_0) q(t_0, \mathbf{x}_0, t_1, \mathbf{x}_1)}\right) d\mathbf{x}_0 d\mathbf{x}_1$$

$$\text{subject to:} \quad \int \pi(\mathbf{x}_0, \mathbf{x}_1) d\mathbf{x}_0 = \rho_1(\mathbf{x}_1)$$

Because we are operating under the constraint that we only have finite sets of samples from the start and end distributions, we must formulate a data-driven approach to this problem, namely: given a known potential $\hat{\varphi}_1(\mathbf{x}_1) \geq 0$ and a set of n samples $\{\mathbf{x}_1^{(j)}\}$ from the target distribution $\rho_1(\mathbf{x}_1)$, our goal is to find the potential $\varphi_1(\mathbf{x}_1)$ such that the factorization $\hat{\varphi}_1(\mathbf{x}_1)\varphi_1(\mathbf{x}_1) = \rho_1(\mathbf{x}_1)$ holds.

We solve this problem as a maximum likelihood estimation.

$$\varphi_1 = \arg\max_{\varphi_1(\mathbf{x}_1)\geq 0} \sum_j \log(\hat{\varphi}_1(\mathbf{x}_1^{(j)})\varphi_1(\mathbf{x}_1^{(j)})), \qquad \text{subject to} \int \hat{\varphi}_1(\mathbf{x}_1)\varphi_1(\mathbf{x}_1) d\mathbf{x}_1 = 1$$

Note that because we are optimizing with respect to φ_1, any terms independent of φ_1 act as constants and can be omitted. Now if we assume infinitely many samples (equivalently, assuming $\rho_1(\mathbf{x}_1)$ is known), the discrete empirical mean converges to the expected value over the target distribution

$$\varphi_1 = \arg\max_{\varphi_1 \geq 0} \int \log(\varphi_1(\mathbf{x}_1))\rho_1(\mathbf{x}_1) d\mathbf{x}_1, \quad \text{subject to} \int \hat{\varphi}_1(\mathbf{x}_1)\varphi_1(\mathbf{x}_1) d\mathbf{x}_1 = 1.$$

To enforce the positivity constraint of φ_1 we propose an exponential parameterization

$$\varphi_1(\mathbf{x}_1) = e^{g(\mathbf{x}_1)}.$$

Substituting this transformation, the optimization problem becomes

$$\max_g \int g(\mathbf{x}_1)\rho_1(\mathbf{x}_1) d\mathbf{x}_1 \quad \text{s.t.} \int \hat{\varphi}_1(\mathbf{x}_1) e^{g(\mathbf{x}_1)} d\mathbf{x}_1 = 1.$$

We then relax the constraint by introducing a Lagrange multiplier λ

$$\max_g \min_\lambda \mathcal{L}(g, \lambda) = \int g(\mathbf{x}_1)\rho_1(\mathbf{x}_1) d\mathbf{x}_1 - \lambda\Big(\int \hat{\varphi}_1(\mathbf{x}_1) e^{g(\mathbf{x}_1)} d\mathbf{x}_1 - 1\Big).$$

First we maximize the Lagrangian over g. Setting the functional derivative to zero yields

$$\frac{\delta \mathcal{L}}{\delta g} = \rho_1(\mathbf{x}_1) - \lambda \hat{\varphi}_1(\mathbf{x}_1) e^{g(\mathbf{x}_1)} = 0 \implies g(\mathbf{x}_1) = \log\left(\frac{\rho_1(\mathbf{x}_1)}{\lambda \hat{\varphi}_1(\mathbf{x}_1)}\right).$$

Substituting the optimal $g(\mathbf{x}_1)$ into the Lagrangian and minimizing over λ gives

$$\min_{\lambda}[-\log(\lambda) + \lambda] \Rightarrow \lambda = 1.$$

Since λ is known we get an unconstrained estimation problem

$$\max_{g} \mathcal{L}(g) = \int g(\mathbf{x}_1)\rho_1(\mathbf{x}_1)d\mathbf{x}_1 - \int \hat{\varphi}_1(\mathbf{x}_1)e^{g(\mathbf{x}_1)}d\mathbf{x}_1. \tag{3.7}$$

The exact analytical solution is $g(\mathbf{x}_1) = \log\left(\frac{\rho_1(\mathbf{x}_1)}{\hat{\varphi}_1(\mathbf{x}_1)}\right)$, or $\varphi_1(\mathbf{x}_1) = \frac{\rho_1(\mathbf{x}_1)}{\hat{\varphi}_1(\mathbf{x}_1)}$. However, because we only have a finite number of samples in practice, the continuous integrals must be replaced by their empirical counterparts. The first term simply becomes the empirical mean. For a set of n unweighted samples $\{\mathbf{x}_1^{(j)}\}_{j=1}^n \sim \rho_1$, this is

$$\int g(\mathbf{x}_1)\rho_1(\mathbf{x}_1)\mathrm{d}\mathbf{x}_1 \approx \frac{1}{n}\sum_{j=1}^{n} g(\mathbf{x}_1^{(j)}).$$

For the second term, we use importance sampling by introducing a sampleable proposal distribution $\tilde{\rho}$ and drawing $\tilde{n}$ samples $\{\tilde{\mathbf{x}}_1^{(k)}\}_{k=1}^{\tilde{n}}$. The integral can then be approximated by

$$\int \hat{\varphi}_1(\mathbf{x}_1)e^{g(\mathbf{x}_1)}\mathrm{d}\mathbf{x}_1 = \int \frac{\hat{\varphi}_1(\mathbf{x}_1)e^{g(\mathbf{x}_1)}}{\tilde{\rho}(\mathbf{x}_1)}\tilde{\rho}(\mathbf{x}_1)\mathrm{d}\mathbf{x}_1 \approx \frac{1}{\tilde{n}}\sum_{k=1}^{\tilde{n}} \frac{\hat{\varphi}_1(\tilde{\mathbf{x}}_1^{(k)})e^{g(\tilde{\mathbf{x}}_1^{(k)})}}{\tilde{\rho}(\tilde{\mathbf{x}}_1^{(k)})}.$$

Maximizing 3.7 using these discrete approximations of the integrals gives us an estimate of φ_1.

Full-bridge problem While the half-bridge above guarantees convergence to the target distribution ρ_1, it leaves the initial distribution unconstrained. To fulfill both boundary conditions simultaneously, we must find both the potentials $\hat{\varphi}_0$ and φ_1. Finding them requires an alternating optimization algorithm, corresponding to the continuous Sinkhorn algorithm.

We begin by parameterizing both potentials to ensure they remain strictly positive, $\hat{\varphi}_0(x_0) = \exp(f(x_0))$ and $\varphi_1(x_1) = \exp(g(x_1))$. By extending the maximum likelihood framework of the half-bridge (Pavon et al. [12]), the problem is solved by alternating between two optimization objectives.

Suppose we begin by fixing $\hat{\varphi}_0$ and optimizing for φ_1. The objective function is analogous to the half-bridge formulation (3.7). The first term simply becomes the empirical mean over the target samples $\{\mathbf{x}_1^{(j)}\} \sim \rho_1$. For the second term, however, we do not have direct access to the propagated potential $\hat{\varphi}_1$. Because only the potential at time step 0 ($\hat{\varphi}_0 = \exp(f(\mathbf{x}_0))$) is accessible, we must obtain $\hat{\varphi}_1(\mathbf{x}_1)$ by propagating it through the transition dynamics

$$\hat{\varphi}_1(\mathbf{x}_1) = \int p(\mathbf{x}_1 \mid \mathbf{x}_0)\hat{\varphi}_0(\mathbf{x}_0)\mathrm{d}\mathbf{x}_0.$$

Substituting this propagation into the second term of the loss yields a double integral

$$\begin{aligned}\int \hat{\varphi}_1(\mathbf{x}_1)e^{g(\mathbf{x}_1)}\mathrm{d}\mathbf{x}_1 &= \int \left[\int p(\mathbf{x}_1 \mid \mathbf{x}_0)e^{f(\mathbf{x}_0)}\mathrm{d}\mathbf{x}_0\right] e^{g(\mathbf{x}_1)}\mathrm{d}\mathbf{x}_1 \\ &= \iint p(\mathbf{x}_1 \mid \mathbf{x}_0)e^{f(\mathbf{x}_0)+g(\mathbf{x}_1)}\mathrm{d}\mathbf{x}_0\mathrm{d}\mathbf{x}_1.\end{aligned}$$

To approximate the double integral, we again apply importance sampling. By introducing a sampleable proposal distribution $\tilde{\rho}_0$ for the initial state, we can approximate the outer integral using $\tilde{m}$ samples $\{\tilde{\mathbf{x}}_0^{(i)}\}_{i=1}^{\tilde{m}} \sim \tilde{\rho}_0$.

$$\iint \frac{p(\mathbf{x}_1 \mid \mathbf{x}_0)e^{f(\mathbf{x}_0)+g(\mathbf{x}_1)}}{\tilde{\rho}_0(\mathbf{x}_0)}\tilde{\rho}_0(\mathbf{x}_0)\mathrm{d}\mathbf{x}_0\mathrm{d}\mathbf{x}_1 \approx \frac{1}{\tilde{m}}\sum_{i=1}^{\tilde{m}}\left[\int p(\mathbf{x}_1 \mid \tilde{\mathbf{x}}_0^{(i)})e^{g(\mathbf{x}_1)}\mathrm{d}\mathbf{x}_1\right]\frac{e^{f(\tilde{\mathbf{x}}_0^{(i)})}}{\tilde{\rho}_0(\tilde{\mathbf{x}}_0^{(i)})}.$$

Second, we can approximate the remaining integral by sampling $\hat{n}$ candidates $\{\hat{\mathbf{x}}_1^{(i,k)}\}_{k=1}^{\hat{n}}$ from the prior $p(\mathbf{x}_1|\tilde{\mathbf{x}}_0^{(i)})$. This yields the final estimator:

$$\int \hat{\varphi}_1(\mathbf{x}_1)e^{g(\mathbf{x}_1)}\mathrm{d}\mathbf{x}_1 \approx \frac{1}{\tilde{m}\hat{n}}\sum_{i=1}^{\tilde{m}}\sum_{k=1}^{\hat{n}}\frac{e^{f(\tilde{\mathbf{x}}_0^{(i)})+g(\hat{\mathbf{x}}_1^{(i,k)})}}{\tilde{\rho}_0(\tilde{\mathbf{x}}_0^{(i)})}.$$

Combining these approximations gives us the total cost function

$$g = \arg\max_g \sum_j w_{\text{target}}^{(j)} g(\mathbf{x}_1^{(j)}) - \frac{1}{\tilde{m}\hat{n}}\sum_{i=1}^{\tilde{m}}\sum_{k=1}^{\hat{n}}\frac{e^{f(\tilde{\mathbf{x}}_0^{(i)})+g(\hat{\mathbf{x}}_1^{(i,k)})}}{\tilde{\rho}_0(\tilde{\mathbf{x}}_0^{(i)})}.$$

Conversely, when updating $\hat{\varphi}_0$ (holding φ_1 fixed), we maximize the symmetrical objective

$$\max_f \mathcal{L}_0(f) = \int f(\mathbf{x}_0)\rho_0(\mathbf{x}_0)\mathrm{d}\mathbf{x}_0 - \iint e^{f(\mathbf{x}_0)+g(\mathbf{x}_1)}p(\mathbf{x}_1|\mathbf{x}_0)\mathrm{d}\mathbf{x}_0\mathrm{d}\mathbf{x}_1,$$

obtained in the same fashion as above.

3.6 Schrödinger bridge proposal

Returning to the question posed in the beginning of this chapter: Is it possible to propagate the particles to the filtering distribution directly without needing an update step? In theory, the Schrödinger bridge does exactly this.

If the initial distribution ρ_0 and the target distribution ρ_1 correspond exactly to the true filtering distribution at successive time steps, the solution to the Schrödinger problem represents the optimal transport between them. By sampling from the optimal coupling $\pi^*(\mathbf{x}_{t+1}|\mathbf{x}_t)$, particles at time t are transported directly to the exact filtering distribution at time $t+1$.

In this ideal scenario where ρ_0 and ρ_1 are known, the particles land exactly where they belong according to the true posterior. If used in an MPF, this causes zero weight degeneracy. The prediction and update step of the traditional particle filter reduce into a single, optimal propagation step, meaning we don't need to perform any updates. A schematic illustration is shown in Figure 3.1.

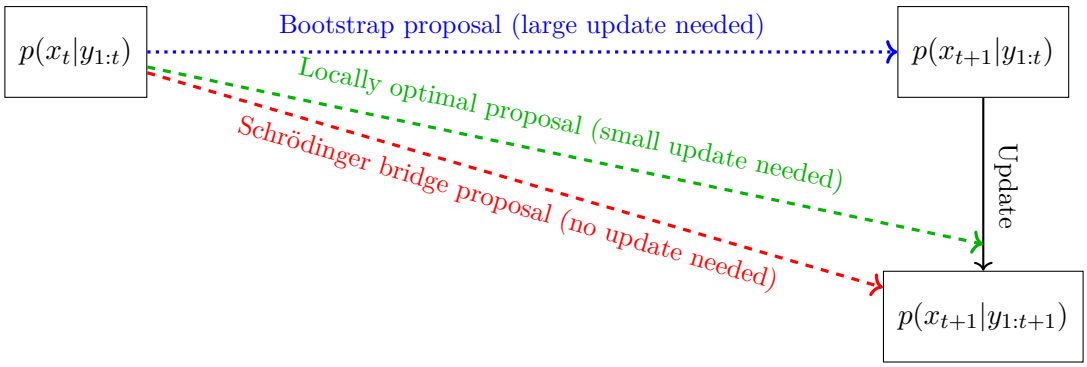
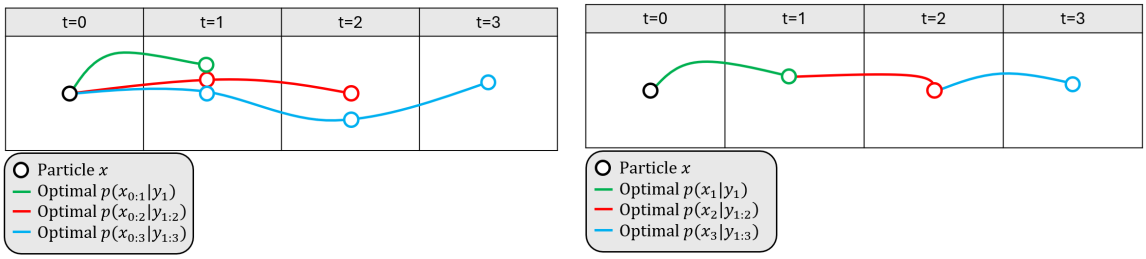


Figure 3.1: Comparing algorithmical steps for the three particle filters.

Note that the use of a Schrödinger bridge as a proposal to a standard particle filter does not necessarily achieve zero weight degeneracy as the Schrödinger bridge only guarantees bridging between filtering distributions. Standard particle filters approximate the full trajectory distributions $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$. As new observations arrive, the true optimal past trajectory may change in light of the new evidence. Because particle filters only allow sampling of the latest state without modifying the past states, bridging the final marginals cannot resolve misalignment in the historical trajectory. Consequently, an update of the weights is still required in this case. See Figure 3.2 for an illustration.



(a) Optimal trajectory distributions. **(b)** Optimal filtering distributions.

Figure 3.2: Optimal past states change for the optimal trajectory distribution, but stay the same for the filtering distribution.

However, using the Schrödinger bridge solution for any type of particle filter might seem paradoxical. In fact, it requires the filtering distribution ρ_1, but finding this distribution is the entire objective of the filtering problem. In order to use the Schrödinger bridge proposal in a particle filter, we discuss several ways of estimating ρ_1 in Chapter 4.

4

Implementing the Schrödinger proposal

Chapter 3 establishes that if the initial distribution ρ_0 and the target distribution ρ_1 are fully known, the Schrödinger bridge provides an optimal transport of particles. However, finding this target distribution is the very objective of the filter itself.

To utilize the optimal transport of the Schrödinger framework without prior knowledge of the target, we can solve the problem using a two-step process. First, we approximate the target distribution using a standard proposal distribution, generating a finite set of candidate samples representing the target boundary (introduced in Section 3.5). Once this discrete target is computed, we can explicitly solve the Schrödinger bridge to find the exact optimal transport path between the initial and target states. While this two-step process yields an exact solution, it is computationally heavy. Therefore, we aim to speed up this process through amortized learning. By using the exact filter to generate training data offline, we can train a neural network to learn the optimal transport. During live tracking, this amortized model bypasses the two step process, yielding a fast approximation of the Schrödinger bridge.

The chapter is structured as follows. We begin by validating our machine-learning approach in a linear-Gaussian setting, where the target boundary can be acquired analytically via a Kalman filter and the Schrödinger bridge possesses a closed-form solution (Section 4.1). To address nonlinear, non Gaussian tracking scenarios, we develop the *discrete state Schrödinger* (DSS) proposal in Section 4.2.1. This establishes our exact but computationally expensive two-step discrete method used to find the target distribution and solve the transport without closed-form assumptions. Using data generated by the DSS, we then introduce a machine-learned approximation for the Schrödinger half-bridge in Section 4.2.2. Finally, we address the implementation and approximation of the marginal particle filter, used to achieve the ideal of a particle filter without any weight degeneracy.

4.1 Linear-Gaussian baseline

To verify that neural networks can indeed learn the optimal transport paths using the SBP, we implement an initial supervised method based on the Gaussian SBP solution from Section 3.3.

4.1.1 Data generation

Training data is first generated by simulating a series of independent tracking trajectories. For each trajectory, the data generation process is designed to pair a particle's current state with its optimal target state at the next time step, as dictated by the SBP.

We begin by fixing the model parameters for the system dynamics, the time step length, and the noise covariances, Q and R. Because the system is linear and Gaussian, the true posterior filtering distribution at any given time step $t+1$ can be calculated exactly. We pass the entire sequence of simulated observations, along with the system dynamics and specific noise parameters, through a standard Kalman filter. This yields the true filtering distribution as a sequence of target means μ_{t+1} and target covariances Σ_{t+1}. These statistical moments serve as the terminal boundary conditions, ρ_1, for the SBP at each respective time step t.

With the target boundaries established, we sequentially propagate a cloud of particles forward in time. For a given time step t, the initial boundary condition ρ_0 is defined by the empirical mean and covariance of the current particle cloud. To formulate the Gaussian SBP, we combine the empirical initial distribution with the Kalman filter's target distribution and the continuous system dynamics to formulate the Gaussian SBP. Using the closed-form solution (Theorem 3.1), we extract the optimal drift parameters. Once the drift parameters are extracted, the resulting optimally controlled SDE is simulated forward over the continuous time interval $[t, t+1]$ using finely discretized time steps. The terminal positions of the particles at the end of the simulated step serve as our optimal target states $\mathbf{x}_{t+1}$.

For every individual particle in the cloud at time t, a single training sample is constructed containing: the particle's initial position $\mathbf{x}_t$, all sensor observations $\{\mathbf{y}_{1:t+1}\}$, and the final position $\mathbf{x}_{t+1}$ generated by the SBP simulation.

4.1.2 Network parameterization

Our objective is to train a neural network to output a Gaussian distribution for each particle's forward step

$$p_\theta(\mathbf{x}_{t+1}|\mathcal{I}_t) = \mathcal{N}(\mathbf{x}_{t+1}; \mu_\theta(\mathcal{I}_t), \Sigma_\theta(\mathcal{I}_t))$$

where $\mathcal{I}_t$ is the input context at time t, and θ represents the network weights. The network is designed to directly output the mean vector $\mu_\theta \in \mathbb{R}^d$ and the covariance matrix $\Sigma_\theta \in \mathbb{R}^{d\times d}$. To ensure mathematical validity and computational stability, we constrain Σ_θ to be a diagonal matrix, which assumes conditional independence between the state dimensions given the input context. To guarantee that the variances along the diagonal are strictly positive, the network's final variance layer applies a softplus activation function.

4.1.3 Architectural variants

Standard proposal distributions in particle filtering typically rely on a strict Markovian assumption, conditioning the next state solely on the immediate previous state

and the most recent observation: $q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t)$. However, relying only on $\mathbf{y}_t$ may leave out relevant information needed by the network to accurately capture the target distribution. We therefore relax the Markovian assumption to explore the impact of including more historical measurements.

To achieve this, we explore two neural network architectures: a multilayer perceptron (MLP) and a random reservoir. Both architectures share the same output structure and are trained using the Adam optimizer, utilizing an 80/20 training-validation. However, they differ in how they process past observations. Specific hyperparameters, including layer counts, hidden dimensions, learning rates, and epochs, are detailed in Appendix B.3.

Multilayer perceptron with observation windows

The MLP architecture processes the current state alongside a fixed history of observations. For a given particle, the input vector consists of the particle's current state $\mathbf{x}_t$, the incoming sensor measurement $\mathbf{y}_{t+1}$, and a window of L previous measurements.

$$\mathcal{I}_t = [\mathbf{x}_t, \mathbf{y}_{t+1}, \mathbf{y}_t, \mathbf{y}_{t-1}, \ldots, \mathbf{y}_{t-L+1}]$$

We evaluate this model across varying window sizes (e.g., $L \in \{0, 1, 2, 3\}$) to assess the impact of historical context on the network's predictive accuracy. Structurally, the MLP is constructed as a deep feedforward network. The final output layer is a linear transformation projecting the final hidden representation into a $2 \cdot d$-dimensional vector, representing the mean and the diagonal elements of the covariance matrix. The networks are optimized in two stages, initially using MSE as loss, followed by a NLL loss phase.

Random reservoir

To incorporate the full history of prior observations, we must utilize an architecture capable of handling variable-length sequential data, overcoming the fixed-input limitation of the static MLP. To achieve this, the random reservoir architecture is split into two distinct components: a fixed untrained RNN encoder – implemented as a random reservoir (see Appendix B.2) – and a trainable feedforward policy network.

The reservoir processes the sequence of observations one step at a time, storing the accumulating history in a hidden state. However, while the random reservoir effectively captures deep historical context, it can struggle to correctly prioritize the most recent observations. Because we want newer information to have a larger impact on the outcome, we add the same fixed window of L recent observations used in the MLP to the network's input context. The network thus evaluates the current local state, the recent observation window, and the recurrent hidden state h_t.

$$\mathcal{I}_t = \left([\mathbf{x}_t, \mathbf{y}_{t+1}, \mathbf{y}_t, \ldots, \mathbf{y}_{t-L+1}], h_t\right)$$

This input vector is passed through the feedforward network, which uses the same structure as the MLP.

4.1.4 Usage and initialization

Once trained, the network can be used to propagate individual particles. Passing the current particle states and the other relevant inputs (depending on the architecture) through the network yields the optimal mean and covariance for each particle. The particle cloud is then propagated forward by drawing samples from this parametrized Gaussian distribution $\mathbf{x}_{t+1} \sim \mathcal{N}(\mu_\theta, \Sigma_\theta)$.

Because both the MLP and RNN architectures rely on an explicit window of L historical measurements to function properly, a practical constraint arises: the neural networks cannot be evaluated during the very first steps of a tracking scenario. To initialize the system, the Kalman filter has therefore been used to propagate the particles until a sufficient number of previous observations have been gathered to fully populate the network's required input structure.

4.2 General nonlinear implementation

Having implemented a Schrödinger bridge solution in a linear-Gaussian setting, we now seek a solution applicable to nonlinear, non-Gaussian tracking scenarios where the target distribution cannot be determined analytically. Because we can no longer rely on a closed-form Kalman filter to provide our target boundaries, we must find another way to compute them. Furthermore, because the resulting distributions may be highly complex and non-Gaussian, we will not rely on a network parametrization restricted to a mean and variance.

We split the method into three distinct phases: first, formulating the exact DSS proposal to establish the theoretical optimal propagation method, second, training an evaluator network to learn the potential function φ, and third, training a normalizing flow (NF) to sample from that learned potential.

4.2.1 Discrete state Schrödinger bridge

We begin by formulating and implementing the DSS proposal as a theoretically optimal tracking method. The DSS is based on an evaluation of the transition probabilities between a set of N current particles at time t and a set of candidate particles at time $t+1$, utilizing the Sinkhorn algorithm introduced in Section 3.5.

By solving the discrete optimal transport, the DSS method yields a propagation step that theoretically eliminates any weight degeneration when integrated with a marginal particle filter. Because of this, it becomes possible to remove the weight update step entirely. The DSS proposal is therefore used not only as a benchmark in our results, but also serves as an exact data generator for our subsequent neural networks.

To construct a target distribution, we sample $\alpha > 1$ candidate particles from some proposal distribution $\hat{q}\left(\cdot \middle| \mathbf{x}_t^{(j)}, \mathbf{y}_t\right)$ for all $j = 1, ..., N$, giving a total of $L = \alpha N$ candidate particles $\hat{\mathbf{x}}_{t+1}$. We consider different approaches as to the choice of $\hat{q}$. One approach to sample these L points is by doing so uniformly from the same support

as the filtering distribution (which in a practical sense is generally finite). As a uniform distribution evenly distributes points across space, it wastes computational resources on evaluating regions with near-zero probability mass. As a result, we need a large α to get a sufficient resolution of the target space. Furthermore, the uniform approach scales poorly as the state dimensionality increases. Another approach is to generate the candidate points by sampling from a standard proposal distribution such as bootstrap or LO. This strategy directs the candidate points into high-probability areas, requiring fewer particles to effectively capture the relevant regions of the state space. Note that the two proposals also have the same support as the filtering distribution, but that LO, while giving generally better sample points than bootstrap, can only be used in the linear case.

In order to solve for the optimal transport between these sets, we must define for given observations $\mathbf{y}_{1:t}$ the boundary marginal densities $p_0(\mathbf{x}_t) := p(\mathbf{x}_t|\mathbf{y}_{1:t})$ and $p_1(\mathbf{x}_t) := p(\mathbf{x}_{t+1}|\mathbf{y}_{1:t+1})$. The initial marginal p_0 is already represented by the weights and locations of the N current particles comprising the filter's state at time t. To obtain the target boundary p_1, note the following.

$$\begin{aligned} p(\mathbf{x}_{t:t+1}|\mathbf{y}_{1:t+1}) &\propto p(\mathbf{y}_{t+1}|\mathbf{x}_{t:t+1}, \mathbf{y}_{1:t})\, p(\mathbf{x}_{t:t+1}|\mathbf{y}_{1:t}) \\ &\propto p(\mathbf{y}_{t+1}|\mathbf{x}_{t+1})\, p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{y}_{1:t})\, p(\mathbf{x}_t|\mathbf{y}_{1:t}) \\ &= p(\mathbf{y}_{t+1}|\mathbf{x}_{t+1})\, p(\mathbf{x}_{t+1}|\mathbf{x}_t)\, p(\mathbf{x}_t|\mathbf{y}_{1:t})\,. \end{aligned} \tag{4.1}$$

Using the joint distribution from (4.1), we can marginalize over the previous particles.

$$\begin{aligned} p_1(\mathbf{x}_{t+1}) = p(\mathbf{x}_{t+1}|\mathbf{y}_{1:t+1}) &= \int p(\mathbf{y}_{t+1}|\mathbf{x}_{t+1})\, p(\mathbf{x}_{t+1}|\mathbf{x}_t)\, p(\mathbf{x}_t|\mathbf{y}_{1:t})\, \mathrm{d}\mathbf{x}_t \\ &= \int p(\mathbf{y}_{t+1}|\mathbf{x}_{t+1})\, \frac{p(\mathbf{x}_{t+1}|\mathbf{x}_t)}{\hat{q}(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{y}_{t+1})} p(\mathbf{x}_t|\mathbf{y}_{1:t})\, \hat{q}(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{y}_{t+1})\, \mathrm{d}\mathbf{x}_t \end{aligned}$$

Therefore, on a small region A, we approximately have

$$\int_A p_1(\mathbf{x}_{t+1}) \mathrm{d}\mathbf{x}_{t+1} \approx \sum_{i=1}^{L} \mathbf{1}_A(\hat{\mathbf{x}}_{t+1}^{(i)}) p\Big(\mathbf{y}_{t+1}\Big|\hat{\mathbf{x}}_{t+1}^{(i)}\Big) \sum_{j=1}^{N} \frac{p\Big(\hat{\mathbf{x}}_{t+1}^{(i)}\Big|\mathbf{x}_t^{(j)}\Big)}{\hat{q}\Big(\hat{\mathbf{x}}_{t+1}^{(i)}\Big|\mathbf{x}_t^{(j)}, \mathbf{y}_{t+1}\Big)} w_t^{(j)},$$

where $\hat{\mathbf{x}}_{t+1}^{(i)} \sim \hat{q}\Big(\cdot\Big|\mathbf{x}_t^{(\lceil i/\alpha \rceil)}, \mathbf{y}_{t+1}\Big)$ as noted above. Therefore, we set the weights $\hat{w}_{t+1}$ corresponding to the candidate points as

$$\hat{w}_{t+1}^{(i)} = p\Big(\mathbf{y}_{t+1}\Big|\hat{\mathbf{x}}_{t+1}^{(i)}\Big) \sum_{j=1}^{N} \frac{p\Big(\hat{\mathbf{x}}_{t+1}^{(i)}\Big|\mathbf{x}_t^{(j)}\Big)}{\hat{q}\Big(\hat{\mathbf{x}}_{t+1}^{(i)}\Big|\mathbf{x}_t^{(j)}, \mathbf{y}_{t+1}\Big)} w_t^{(j)}$$

Next, we need to define the respective prior corresponding to the unconditional dynamics matrix K. Again, correcting for the distribution of $\hat{\mathbf{x}}_{t+1}$, the matrix

is

$$K_{ij} = \frac{p\left(\hat{\mathbf{x}}_{t+1}^{(i)} \middle| \mathbf{x}_t^{(j)}\right)}{\hat{q}\left(\hat{\mathbf{x}}_{t+1}^{(i)} \middle| \mathbf{x}_t^{(j)}, \mathbf{y}_{t+1}\right)} w_t^{(j)}, \tag{4.2}$$

where we used in similar fashion that

$$\int_A p(\mathbf{x}_{t+1}|\mathbf{x}_t)\, p(\mathbf{x}_t|\mathbf{y}_{1:t})\, \mathrm{d}(\mathbf{x}_t, \mathbf{x}_{t+1}) = \sum_{i=1}^{L}\sum_{j=1}^{N} \mathbf{1}_A(\mathbf{x}_t^{(j)}, \hat{\mathbf{x}}_{t+1}^{(i)}) \frac{p\left(\hat{\mathbf{x}}_{t+1}^{(i)} \middle| \mathbf{x}_t^{(j)}\right)}{\hat{q}\left(\hat{\mathbf{x}}_{t+1}^{(i)} \middle| \mathbf{x}_t^{(j)}, \mathbf{y}_{t+1}\right)} w_t^{(j)}.$$

With the marginal constraints defined by $p_0 = w_t$ and $p_1 = \hat{w}_{t+1}$, and the reference process by (4.2), we can use the Sinkhorn algorithm to find the optimal coupling matrix $\pi^* \in [0,1]^{N \times L}$.

The entries of π^* represent the transition probabilities between every pair of N previous particle and L candidate particles. We can then sample N proposed particles at time $t+1$ by sampling one index i based on the normalized probability vector given by the j:th column of π^*, for all $j = 1, ..., N$. We then set the new particles as $\mathbf{x}_{t+1}^{(j)} = \hat{\mathbf{x}}_{t+1}^{(i)}$, and the DSS proposal evaluation $q\left(\mathbf{x}_{t+1}^{(j)} \middle| \mathbf{x}_t^{(j)}, \mathbf{y}_{t+1}\right) := \pi_{ij}^*$. Finally, the weights at time $t+1$ are given by

$$w_{t+1}^{(j)} = \frac{p\left(\mathbf{y}_{t+1} \middle| \mathbf{x}_{t+1}^{(j)}\right) p\left(\mathbf{x}_{t+1}^{(j)} \middle| \mathbf{x}_t^{(j)}\right)}{q\left(\mathbf{x}_{t+1}^{(j)} \middle| \mathbf{x}_t^{(j)}, \mathbf{y}_{t+1}\right)}.$$

See Figure 4.1 for an illustration.

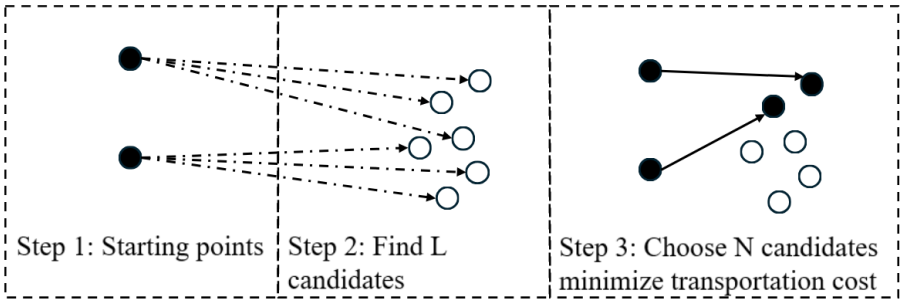


Figure 4.1: Illustration of PF(DSS). The algorithm generates $L = N\alpha = 2 \cdot 3$ candidate particles using another proposal distribution $\hat{q}$. It then selects the next N particles from the optimal transport defined by π^*.

While the DSS provides a rigorous discrete solution to the missing target boundary, its reliance on generating L candidate particles and executing the Sinkhorn algorithm yields a complexity of at least $O(N^2)$. This makes the DSS impractical for live tracking. We can still use it offline to construct a comprehensive training dataset. For every individual time step, a single training sample captures the initial particle cloud x_{start}, the predicted cloud propagated strictly through the prior dynamics x_{pred}, the full set of L candidate target points x_{target} alongside their evaluated DSS weights w_{target}, and the corresponding sensor measurement y.

4.2.2 Network parametrization

Next, we train neural networks to learn the Schrödinger transportation. Unlike the Gaussian baseline, we parameterize this solution using three interacting neural networks: a set of potential networks $\hat{\varphi}_0$-net φ_1-net to approximate the optimal transport, and a normalizing flow to draw samples.

Evaluating the potentials: the $\hat{\varphi}$-net and φ-net

To satisfy both boundary conditions of the full bridge, we use two networks to approximate the log-potentials defined in Section 3.5. Recall that the continuous potential functions are parameterized as $\hat{\varphi}_0(\mathbf{x}_0) = e^{f(\mathbf{x}_0)}$ and $\varphi_1(\mathbf{x}_1) = e^{g(\mathbf{x}_1)}$. We design the neural networks to approximate the exponents of these functions. For a given input context $\mathcal{I}$ and a candidate state $\mathbf{x}$, denoted together as x, the networks' outputs are defined as $\hat{\varphi}_{\text{net}}(x) = \log(\hat{\varphi}_0(x))$ and $\varphi_{\text{net}}(x) = \log(\varphi_1(x))$.

The networks are trained via an alternating optimization procedure, mimicking the full-bridge solution. The objective is to train the networks to minimize the negative of their respective objectives. To optimize this via our generated dataset, the continuous integrals are replaced with discrete empirical approximations.

When solving for $\hat{\varphi}_{\text{net}}$ while φ_{net} is frozen, we minimize

$$\mathcal{L}(\hat{\varphi}_{\text{net}}) = -\sum_i w^{(i)}_{\text{start}} \hat{\varphi}_{\text{net}}(\mathbf{x}^{(i)}_{\text{start}}) + \frac{1}{\tilde{m}\hat{n}} \sum_{i=1}^{m} \sum_{k=1}^{\hat{n}} \frac{\exp\left(\hat{\varphi}_{\text{net}}(\mathbf{x}^{(i)}_{\text{start}}) + \varphi_{\text{net}}(\hat{\mathbf{x}}^{(i,k)}_{\text{pred}})\right)}{\tilde{\rho}_0(\mathbf{x}^{(i)}_{\text{start}})}.$$

Similarly, when updating φ_{net} while $\hat{\varphi}_{\text{net}}$ is frozen, we minimize:

$$\mathcal{L}(\varphi_{\text{net}}) = -\sum_j w^{(j)}_{\text{target}} \varphi_{\text{net}}(\mathbf{x}^{(j)}_{\text{target}}) + \frac{1}{\tilde{m}\hat{n}} \sum_{i=1}^{m} \sum_{k=1}^{\hat{n}} \frac{\exp\left(\hat{\varphi}_{\text{net}}(\mathbf{x}^{(i)}_{\text{start}}) + \varphi_{\text{net}}(\hat{\mathbf{x}}^{(i,k)}_{\text{pred}})\right)}{\tilde{\rho}_0(\mathbf{x}^{(i)}_{\text{start}})}.$$

The first term in both loss functions represents the mean over the respective boundary distributions, this is approximated using the weighted particles.

To compute the second (importance sampling) term, we take $\tilde{m} = N$ samples $\mathbf{x}^{(i)}_{\text{start}}$ from the dataset, which acts as our proposal distribution $\tilde{\rho}_0$. We then construct $\hat{\mathbf{x}}^{(i,k)}_{\text{pred}}$ by propagating the particles through the prior with added stochastic noise.

Evaluating this term requires computing the explicit probability density $\tilde{\rho}_0(\mathbf{x}^{(i)}_{\text{start}})$ in the denominator. Because our empirical dataset consists of discrete samples, we construct a continuous probability landscape by implementing a Gaussian Kernel Density Estimator. By placing an Gaussian kernel G over each of the offline-generated starting samples, we produce a smooth, sampleable density estimate for the initial distribution

$$\tilde{\rho}_0(\mathbf{x}) = \frac{1}{M} \sum_{l=1}^{M} G(\mathbf{x} - \mathbf{x}^{(l)}_{\text{start}}),$$

By evaluating this distribution for each starting particle, we obtain the necessary continuous probability measure for the denominator.

Sampling via normalizing flows

While the potential networks are designed such that they can evaluate points points in the state space, they are incapable of generating discrete samples. To propagate particles, we construct a secondary generative model capable of sampling from the conditional Schrödinger bridge distribution wich is expressed as

$$\begin{aligned}\pi^*(\mathbf{x}_1|\mathbf{x}_0) &= \frac{\pi^*(\mathbf{x}_0,\mathbf{x}_1)}{\int \pi^*(\mathbf{x}_0,\mathbf{x}_1)\mathrm{d}\mathbf{x}_1} = \frac{\hat{\varphi}_0(\mathbf{x}_0)q(t_0,\mathbf{x}_0,t_1,\mathbf{x}_1)\varphi_1(\mathbf{x}_1)}{\int \hat{\varphi}_0(\mathbf{x}_0)q(t_0,\mathbf{x}_0,t_1,\mathbf{x}_1)\varphi_1(\mathbf{x}_1)\mathrm{d}\mathbf{x}_1} \\ &= \frac{q(t_0,\mathbf{x}_0,t_1,\mathbf{x}_1)\varphi_1(\mathbf{x}_1)}{\int q(t_0,\mathbf{x}_0,t_1,\mathbf{x}_1)\varphi_1(\mathbf{x}_1)d\mathbf{x}_1} \propto q(t_0,\mathbf{x}_0,t_1,\mathbf{x}_1)\varphi_1(\mathbf{x}_1).\end{aligned}$$

or, using our potential network $\pi^*(\mathbf{x}_{t+1}|\mathbf{x}_t) \propto q(\mathbf{x}_{t+1}|\mathbf{x}_t)e^{\varphi_{net}(\mathbf{x}_{t+1})}$.

To achieve this, we implement a conditional RealNVP Normalizing Flow (NF). Normalizing flows possess the distinct advantage of being able to both explicitly evaluate a probability density and draw samples from it. Structurally, an NF constructs a complex probability distribution by applying a sequence of invertible transformations to samples drawn from a simple, tractable base distribution, we have used a standard multivariate Gaussian $z \sim \mathcal{N}(0, \mathbf{I})$.

The NF is trained to match the unnormalized target density, defined as $q_{\mathrm{NF}} = q(x_{t+1}|\mathbf{x}_t)\exp(\varphi_{\mathrm{net}}(x_{t+1}))$. Optimization is achieved by minimizing the Kullback-Leibler divergence between the flow's own generated distribution and the target density. To capture the tracking history, the flow is conditioned on the same input context $\mathcal{I}_t$ as its corresponding φ-net. Evaluated empirically over batches of samples drawn directly from the flow itself ($\mathbf{x} \sim q_{\mathrm{NF}}$), the loss function simplifies to

$$\begin{aligned}D_{\mathrm{KL}}(q_{\mathrm{NF}}(\cdot|\mathbf{x}_t)||q(\cdot|\mathbf{x}_t)\exp(\varphi_{\mathrm{net}}(\cdot))) &= \int \log\left(\frac{q_{\mathrm{NF}}(x|\mathbf{x}_t)}{q(x|\mathbf{x}_t)\exp(\varphi_{\mathrm{net}}(x)))}\right) q_{\mathrm{NF}}(x|\mathbf{x}_t)\mathrm{d}x \\ &\approx \sum_{x\sim q_{\mathrm{NF}}(\cdot|\mathbf{x}_t)} \log\left(\frac{q_{\mathrm{NF}}(x|\mathbf{x}_t)}{q(x|\mathbf{x}_t)\exp(\varphi_{\mathrm{net}}(x))}\right) \\ &= \sum_{x\sim q_{\mathrm{NF}}(\cdot|\mathbf{x}_t)} \log(q_{\mathrm{NF}}(x|\mathbf{x}_t)) - (\log(q(x|\mathbf{x}_t)) + \varphi_{\mathrm{net}}(x))\end{aligned}$$

By minimizing this objective across the training dataset, the sequence of invertible transformations learns to warp the simple base Gaussian directly into the complex, nonlinear optimal transport geometry dictated by the φ-network.

Once the NF is trained, we can use it to sample and evaluate points from $\pi^*(\mathbf{x}_1|\mathbf{x}_0)$. During live tracking, we can draw samples from the NF for the next time step, based on the current particle cloud and the measurement. The propagated particles can be used by themselves as an estimation of the filtering distribution, but the distribution can also be used as a proposal in a particle filter.

4.2.3 Architectural variants

To build the φ-network and its corresponding NF, we explore the exact same architectural variants utilized in our Gaussian baseline: a static multilayer perceptron utilizing a fixed window of L observations, and a random reservoir combined with the recent observation window. The NF follows the same architecture choice as the φ-network it has been trained on.

The structure of these networks is similar to the once defined in Section 4.1.3. With the difference of the φ-net and the NF being an unsupervised task. Because the optimization objectives evaluate distributional properties rather than calculating direct errors against known ground-truth data, the training process does not utilize a training-validation split (as for the supervised Gaussian baseline). Instead, the networks are optimized directly across the entirety of the offline-generated DSS datasets. Specific hyperparameter configurations for the nonlinear networks are detailed in Appendix B.3.

4.2.4 Usage and initialization

Once the NF is fully trained against the learned potential network, the model acts as a proposal distribution for the specific problem it has been trained on. During live inference, the DSS and the evaluator network φ_{net} are entirely discarded. At each time step, the current particle cloud and the required historical sensor measurements are fed into the NF. The latter draws samples for the next time step, propagating the particle cloud directly according to the learned optimal transport $\pi^*(x_1|x_0)$. As for the Gaussian baseline solution, a standard filtering initialization is required until the necessary number of measurements are acquired.

4.3 Approximating the marginal particle filter

If using the Schrödinger solution as a proposal distribution, the theoretical ideal of zero weight degeneracy requires the use of an MPF. However, implementing the MPF in a real-world setting requires us to overcome its computational complexity of $\mathcal{O}(N^2)$ (compared to $\mathcal{O}(N)$ for regular particle filters), due to the approximation of the integral (2.18) shown again below.

$$\int w_{t-1} \mathrm{d} f_i(\mathbf{x}_{t-1}|\mathbf{y}_{t-1}) = \int w_{t-1} f_i(\mathbf{x}_{t-1}|\mathbf{y}_{t-1}) \mathrm{d}\mathbf{x}_{t-1} \tag{4.3}$$

The integral becomes a sum on the form

$$\sum_{j=1}^{N} f_i(\mathbf{x}_{t-1}^{(j)}) w(\mathbf{x}_{t-1}^{(j)}),$$

for all $i = 1, ..., N$, and $f_i(\cdot)$ designating either $p\left(\mathbf{x}_t^{(i)} \middle| \cdot\right)$ or $q\left(\mathbf{x}_t^{(i)} \middle| \mathbf{y}_t, \cdot\right)$. The advantage of the marginal approach is that the particles have a lower weight variance, which does not build up over time.

Klaas et al. [8] propose more efficient computations of the update using k-d trees with an average complexity of $\mathcal{O}(N \log(N))$ or even $\mathcal{O}(N)$ in the linear-Gaussian case. However, this approximation method only works for proposals where the transition probability is proportional to some distance metric, which is not necessarily the case for the Schrödinger bridge solutions. See Appendix A for more information on this approach.

Instead, we approximate the integral (4.3) by performing importance sampling using a base distribution $r(\mathbf{x}_{t-1})$. We can then rewrite the integral as

$$\int w_{t-1} \frac{f_i(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})}{r(\mathbf{x}_{t-1})} \mathrm{d}r(\mathbf{x}_{t-1}) \approx \sum_{k=1}^{n} w(\hat{\mathbf{x}}_{t-1}^{(j)}) \frac{f_i(\hat{\mathbf{x}}_{t-1}^{(k)})}{r(\hat{\mathbf{x}}_{t-1}^{(j)})}, \quad \hat{\mathbf{x}}_{t-1}^{(k)} \sim r$$

for $k = 1, ..., n$, which is also an unbiased estimation of (4.3). This approach reduces the complexity to $\mathcal{O}(nN)$, where n can be a constant or a function of N (such as the logarithm) depending on the desired approximation.

The base r can be any distribution with the same support as f_i. The variance of this estimator decreases as $fw - \mu r$, achieving zero-variance if $r(\mathbf{x}) \propto |f(\mathbf{x})|w(\mathbf{x})$ for all relevant $\mathbf{x}$ [11]. While the exact expression is costly to evaluate, a reasonable and readily available choice for r are the weights w themselves, as they provide information on the magnitude of the expression $|f(\mathbf{x})|w(\mathbf{x})$. In this thesis, when approximating the MPF we use importance sampling, evaluating both $r \sim \mathcal{U}$ and $r \sim w$.

5

Experimental setup

To evaluate the filtering methods detailed in this thesis, we establish a benchmarking framework consisting of two tracking scenarios: a linear-Gaussian baseline and a more complex nonlinear environment. Because the analytic solution of the Schrödinger bridge relies on continuous-time dynamics, the linear-Gaussian target is initially formulated as a continuous *stochastic differential equation* (SDE) before being discretized for use in the particle filter. As the nonlinear scenario relies solely on discrete approximations rather than closed-form analytic solutions, its target dynamics are formulated directly as a discrete-time state-space model. Finally, these dynamic models are paired with a standard linear Cartesian measurement model and a nonlinear polar measurement model.

5.1 Motion models

We begin by defining the motion models which are responsible for generating the true target trajectories, which the filters attempt to estimate.

5.1.1 Linear motion model: the moth

To establish a baseline, a simple two-dimensional linear scenario was devised. The model represents a clockwise rotational motion that converges toward the origin. We first define the system as a continuous SDE governed. Following the Ornstein-Uhlenbeck from Section 3.3, and setting the attractor $m = 0$, the continuous dynamics are modeled as

$$\mathrm{d}X_t = AX_t\mathrm{d}t + \sigma\mathrm{d}W_t$$

where $X_t \in \mathbb{R}^2$ denotes the position of the target, W_t is a standard Brownian motion, σ is the continuous diffusion matrix, and $A \in \mathbb{R}^{2\times2}$ is the continuous drift matrix governing the rotation and decay of the trajectory.

To control the shape of the spiral, the drift matrix is parameterized as

$$A = -\epsilon I + CJ$$

where I is the 2×2 identity matrix and $J = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ is a 90-degree rotation matrix. The parameter $\epsilon > 0$ controls decay rate of the spiral toward its center, while $\omega > 0$ dictates the clockwise angular velocity.

Because the particle filters evaluated in our experiments operate on discrete time steps, we must compute the discrete-time equivalent of this model over a time step Δt. The continuous system transforms into the discrete-time state-space model

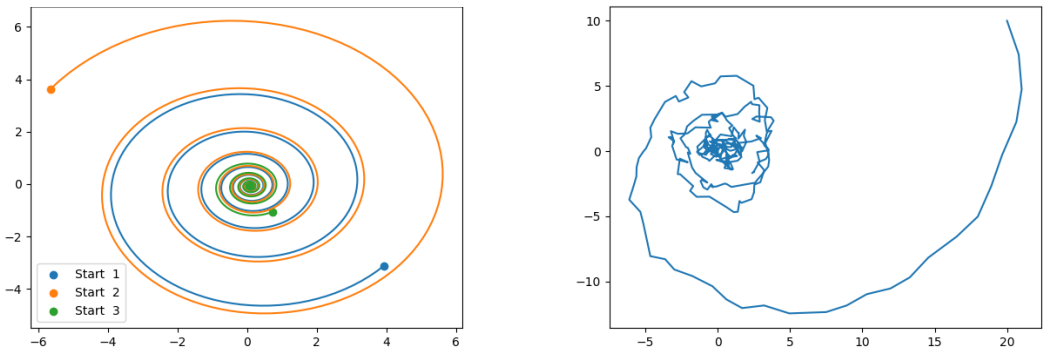
$$x_{t+1} = \Phi x_t + w_t$$

where the discrete process noise is distributed as $w_t \sim \mathcal{N}(0, Q)$.

The discrete transition matrix Φ is derived by taking the exponential of the continuous drift

$$\Phi = e^{(-\epsilon I + CJ)\Delta t} = e^{-\epsilon \Delta t}\Big(\cos(C\Delta t)I + \sin(C\Delta t)J\Big)$$

For our simulations we used a time step of $\Delta t = 0.05$, a decay factor $\epsilon = 0.3$, and a rotation factor $C = 1.2$. Note that regardless of the starting position, the trajectory eventually converges to the origin after circling it several times, as illustrated in Figure 5.1a. Because this spiraling behavior resembles a moth drawn to a lamp, we refer to this as the moth model. Figure 5.1b displays the resulting trajectory with the covariance matrix set to $Q = \text{diag}(0.12, 0.12)$.



(a) Movement independent of start point. **(b)** Trajectory with process noise.

Figure 5.1: Dynamics of the linear moth model.

5.1.2 Nonlinear motion model: The plane

To evaluate the filtering methods under more complex conditions, we introduce a nonlinear scenario representing an aircraft navigating toward the nearest available safe base in unstable weather. Let there be N possible landing bases, denoted as $\mathcal{B} := (B_i)_{i=1}^N$ where $B_i \in \mathbb{R}^2$, $i = 1, ..., N$. The aircraft's movement is dictated by two forces: a desire to steer towards the closest base, and an environmental drift caused by wind currents. The wind is modeled as the gradient of a known atmospheric pressure, defined by the function $D : \mathbb{R}^2 \to \mathbb{R}$. The dynamics of the aircraft are

governed by the following system of equations

$$B^*(x_t) := \arg\min_{B \in \mathcal{B}} \|x_t - B\|^2$$
$$f(x_t) := \frac{B^*(x_t) - x_t}{\|B^*(x_t) - x_t\|^2}$$
$$x_{t+1} = x_t + \Big(f(x_t) - \eta \nabla D(x_t)\Big) \cdot \Delta t + w_t$$

Here, $B^*(\mathbf{x}_t)$ identifies the nearest base from the current position, $f(\mathbf{x}_t)$ is the steering force towards $B^*(\mathbf{x}_t)$, and w_t represents the state-transition noise. Because the target base may change depending on the spatial location, the model can cause a multimodal filtering distribution. We refer to this scenario as the *nonlinear plane model*. Figure 5.2 illustrates the dynamics of model.

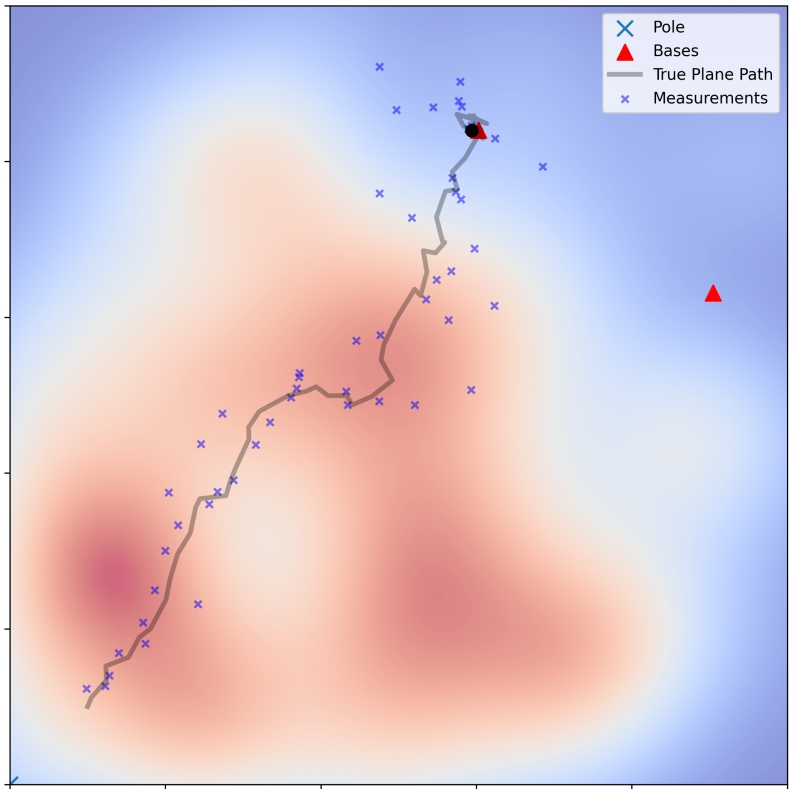


Figure 5.2: Nonlinear plane model.

The air pressure is kept the same for all runs of the nonlinear plane scenario (in order to ease the neural network learning), whereas the bases are randomly generated for each run. The air pressure D was originally calculated as

$$D(x) := \sum_{i=1}^{100} \mathcal{N}(x; \mu_i, 0.1),$$

where $\mu_i \sim \mathcal{U}([0,1]^2)$. The heatmap of D can be seen in the background of Figure 5.2. Bases are sampled from $\sqrt{\mathcal{U}([0,1]^2)}$, i.e. a uniform distribution from which the

square root has been applied. The last operation was applied in order to move the bases away from the starting point $x_0 = (0.1, 0.1)^\top$, in order to make the run more interesting.

5.2 Measurement models

With the target trajectories established, we implement two measurement models to generate observation data: one linear Cartesian model, and one nonlinear polar model. The difference between the two measurement models is showcased in Figure 5.4.

5.2.1 Linear Cartesian Measurement

The Cartesian measurement model assumes that the sensors directly observe the target's coordinates (x, y), corrupted by additive Gaussian noise v_t. This measurement model is strictly linear, defined as

$$y_t = h_{\text{cart}}(x_t) + v_t = x_t + v_t$$

The Linear measurement model has been paired with the moth model for out evaluations, where the noise v_t is drawn from a zero-mean Gaussian with covariance $R_{\text{cart}} = \text{diag}(0.45,\, 0.45)$.

5.2.2 Nonlinear Polar Measurement

For the nonlinear measurement model we let a sensor be located at a fixed pole $s = (s_1, s_2)^\top$ (in our experiments, $s = (0, 0)^\top$). To construct an observation, the Cartesian coordinates of the current state $\mathbf{x}_t$ is first transformed into polar coordinates (ρ, θ) representing the range and the bearing relative to the sensor location

$$\begin{aligned}\rho_t &= \|x_t - s\|_2 \\ \theta_t &= \text{arctan2}(x_{t,2} - s_2, x_{t,1} - s_1)\end{aligned}$$

Independent Gaussian noise is then added to the polar coordinates. Let the polar noise be distributed as $v \sim \mathcal{N}(0, R)$, where the covariance matrix is defined as $R = \text{diag}(\sigma_\rho^2, \sigma_\theta^2)$. In our simulations tracking the plane, we have used the noise variance $\sigma_\rho^2 = 0.001$ and $\sigma_\theta^2 = 0.5$. The resulting polar coordinates $\tilde{\rho}_t = \rho_t + v_\rho$ and $\tilde{\theta}_t = \theta_t + v_t$, are then converted back to Cartesian form. This results in a final observation of

$$y_t = h_{\text{polar}}(x_t, v) = s + \begin{pmatrix} \tilde{\rho}_t \cos\!\left(\tilde{\theta}_t\right) \\ \tilde{\rho}_t \sin\!\left(\tilde{\theta}_t\right) \end{pmatrix}$$

We paired this measurement model with the nonlinear plane dynamics to simulate an observation mechanism akin to the type of noise present in real-world radar systems. Because the noise is added in the polar domain prior to the conversion, the

resulting Cartesian uncertainty distribution becomes a “banana-shaped” probability mass around the true state. See Figure 5.3 for an illustration.

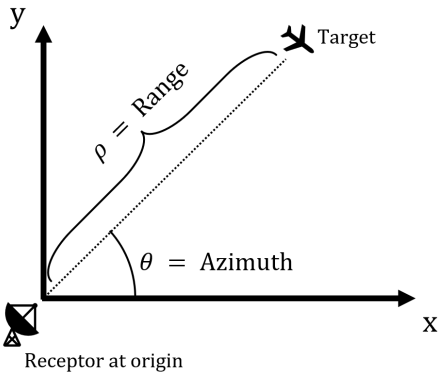


Figure 5.3: Polar coordinates w.r.t. pole at origin.

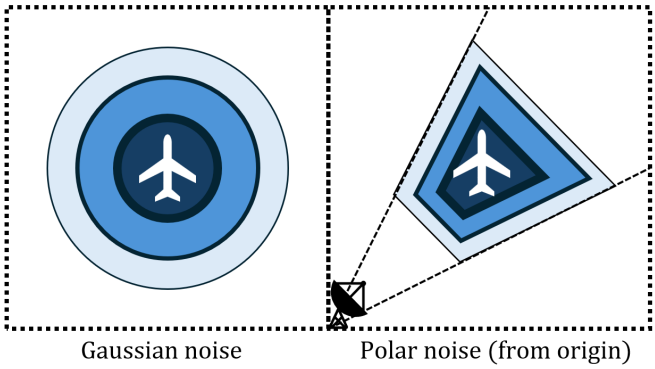


Figure 5.4: Gaussian vs polar noise.

6

Results and discussion

In this chapter, we evaluate and analyze the performance of various combinations of proposal distributions and filter architectures. These combinations are tested on the models described in Chapter 5, specifically utilizing the linear moth motion model paired with the Cartesian measurement model, and the nonlinear plane model paired with the polar measurement model.

The chapter is structured as follows: Section 6.1 introduces the metrics used to evaluate the tracking performance. Section 6.2 details the experimental results for the linear scenario by first establishing the performance of the exact DSS proposal, comparing it as a standalone propagation mechanism against its use within a particle filter. Following this exact solution, we evaluate the amortized solutions discussed in Sections 4.1 and 4.2.1. Section 6.3 then extends this same analysis to the nonlinear tracking scenario. Finally, Section 6.4 analyzes the computational and accuracy trade-offs of the various approximation strategies applied to the MPF.

6.1 Evaluation metrics

In order to effectively quantify weight degeneracy, we use the concept of *effective sample size* (ESS) in this thesis.

Definition 6.1 (Effective sample size)**.** *ESS at time t is defined as*

$$\frac{\left(\sum_{i=1}^{N} \tilde{w}_t^{(i)}\right)^2}{\sum_{i=1}^{N} \left(\tilde{w}_t^{(i)}\right)^2},$$

or for normalized weights

$$\frac{1}{\sum_{i=1}^{N} \left(w_t^{(i)}\right)^2}. \tag{6.1}$$

The attainable values for ESS as defined in (6.1) are $[1, N]$, only achieving its upper limit when all weights are constant (minimum degeneracy), and its lower limit when all weight is concentrated on a single particle (maximum degeneracy).

As described in Section 2.4.2, resampling can improve the quality of the filtering distribution approximation, but also decrease it should the resampling occur too

often. A rule of thumb described by [14] and used in this thesis is resampling only when ESS attains a value under γN for some $\gamma \in (0, 1)$. In the results below, we use $\gamma = 0.1$ if not said otherwise.

Note that while ESS quantifies weight degeneracy, it cannot be the sole measure to evaluate the tracking performance. For example, a constant proposal function $q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t) := C$ will usually give very poor tracking while always boasting perfect ESS. Furthermore, resampling at every step will give a higher ESS than only resampling when ESS drops below a certain threshold, even though the latter may give better tracking results. We will thus present ESS in conjunction with the *mean squared error* (MSE), as well as the average amount of resamples as a measure of the ESS stability.

Note that empirical computation times are excluded from our evaluation metrics. While the theoretical complexities have been discussed, the code implementations developed for this thesis have not prioritized software optimization. Because the implementations vary significantly in their degree of code efficiency, we deem any direct runtime comparisons to be misleading regarding the true performance of the algorithms.

6.2 Linear model: The moth

The goal of this section is to demonstrate how our Schrödinger bridge implementations compares to established filtering methods in the linear moth model, where the analytical Kalman filter is applicable.

Specifically, we evaluate the exact DSS proposal along with our neural network approximations, both tested as a standalone propagation mechanism and as a weighted proposal. We compare these Schrödinger approaches to the optimal Kalman filter, as well as the bootstrap and LO proposals implemented within various particle filter architectures.

6.2.1 Parameter consideration

We begin by acknowledging that the performance of any particle filter is heavily dependent on the specific scenario parameters. Notably, the number of particles affects tracking accuracy, while the relationship between the system noise Q and the measurement noise R dictates the rate of weight decay.

In Appendix C.1, baseline tests are presented to confirm the expected theoretical behavior of the bootstrap and LO proposals. When using a massive number of particles (N = 100,000), both PF(B) and PF(LO) follow the Kalman filter estimation with high accuracy. However, when the particle count is smaller ($N = 150$), the deviation from the Kalman filter becomes noticeable. The bootstrap proposal performs worse in this setting than the LO proposal. This is attributed to the combination of fewer particles and bootstrap's need for frequent resampling, which results in poor particle diversity and a poor approximation of the true filtering distribution.

Furthermore, under extreme noise conditions, where the dynamics are very uncertain and the observations are highly informative the ESS of PF(B) remains below 10% during the entire run while PF(LO) is much more stable. This results in an inaccurate estimation of the filtering distribution and causes a noticeable difference in tracking ability to PF(LO). The cause of PF(B)'s poor preference is the exclusion of the measurement in the propagation step.

Building on the insights from above, the following evaluations for the linear moth model uses noise parameters chosen (see Section 5) to simulate a realistic tracking environment. Furthermore, because using the DSS proposal and running the MPF is highly computationally intensive and time consuming, the particle count was restricted to N = 100 for all simulations. The DSS performance is also affected by the number of target particles proposed to it. For all experiments we have used an extra sampling factor of 100.

6.2.2 Discrete state Schrödinger proposal

Figure 6.1 demonstrates the tracking performance of the DSS algorithm compared to the LO proposal. In the figure, a distinction is made between two versions of the DSS proposal. "WlessDSS" refers to the use of the DSS proposal to propagate the particles, but the particles are not given any weights. While Reg(DSS) represents PF(DSS), where particles receive weight updates.

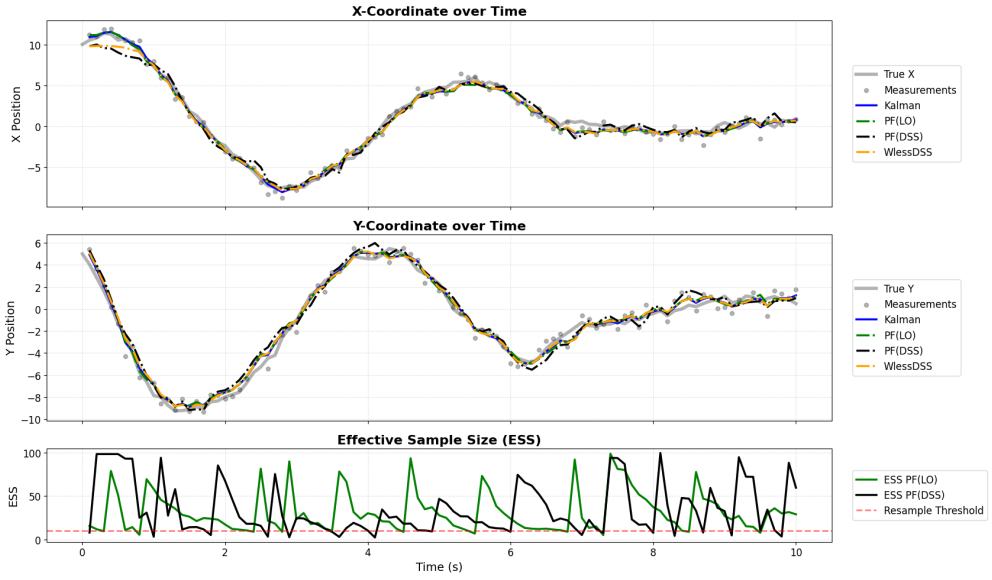


Figure 6.1: Tracking performance comparison between the unweighted DSS propagation mechanism, the weighted DSS, and PF(LO).

As the figure is only one example of the outcome, and might be hard to read, we have summarized the performance of the different proposals in Table 6.1. The table presents the average values calculated over 50 runs, using 100 particles, with a resampling whenever the ESS $< 10\%$. It categorizes the performance across four metrics: mean ESS for one run (with a max value of 100), the average number of

resampling triggered per run, MSE compared to the true path and MSE compared to the Kalman filter. The final row of the table shows the performance of the unweighted DSS proposal.

Table 6.1: Performance comparison of filter architectures and proposal distributions on the linear moth model. Values represent the mean over 50 independent runs using 100 particles. Resample threshold 0.1.

Filter	Proposal	ESS	# Resamples	MSE (vs True)	MSE (vs Kalman)
Kalman	-	-	-	0.343 ± 0.041	-
Regular PF	Bootstap	27.0 ± 1.6	21.0 ± 1.4	0.380 ± 0.070	0.033 ± 0.008
	LO	32.6 ± 2.5	11.6 ± 1.4	0.362 ± 0.048	0.023 ± 0.005
	DSS	29.5 ± 2.3	19.5 ± 2.7	0.417 ± 0.055	0.075 ± 0.025
Unweighted	DSS	-	-	0.382 ± 0.061	0.043 ± 0.027

Looking at Figure 6.1, the most immediate observation is that the estimated trajectories from all proposals are very close, with the plotted lines lying almost on top of one another. Because the distances in this tracking scenario are quite small, all tested proposals successfully provide a good approximation of the true target state. It is therefore difficult to compare the differences in MSE.

When examining the numerical metrics in the "Regular" filter section of Table 6.1, some differences emerge. We see that the DSS proposal's performance falls somewhere between the bootstrap and LO proposals in terms of filter stability. It yields a mean ESS and a resampling frequency between the values of the other two methods. Despite this moderate stability, the MSE for the weighted DSS is the highest in the group, both compared to the true path and the Kalman filter. Thus, the use of the DSS proposal in a standard particle filter does not appear to be strictly favorable in this specific case.

Furthermore, we note that the ESS for PF(DSS) is not close to being constant. This is expected, since we cannot bridge between optimal trajectory distributions without changing the previous positions of the particles (as was argued in Section 3.6). Consequently, because the resulting particle weights are non-uniform, the weighted state estimate diverges from the standalone propagation. This is visible in the figure, where the weighted DSS trajectory deviates from the unweighted DSS path. We also see that the unweighted DSS has a mean MSE in the same range as PF(LO) filter, they are particularly similar when compared to the true path.

6.2.3 Effect of filter architectures

While the DSS proposal in a regular particle filter did not demonstrated great potential in the previous section, changing the filter architecture significantly impacts its performance. Table 6.2 presents the results of utilizing the DSS as a proposal within the APF, MPF, and AMPF architectures (visualizations of these results are provided in Appendix C.2).

Table 6.2: Performance comparison of filter architectures and proposal distributions. Values represent the mean and standard deviation over 50 independent runs.

Filter	Proposal	ESS	MSE (vs True)	MSE (vs Kalman)
Kalman	-	-	0.343 ± 0.041 -	
Regular	Bootstap	54.8 ± 1.7	0.366 ± 0.060	0.015 ± 0.004
	LO	69.6 ± 1.9	0.367 ± 0.053	0.009 ± 0.002
	DSS	54.8 ± 2.5	0.394 ± 0.067	0.057 ± 0.038
APF	Bootstap	64.8 ± 1.8	0.401 ± 0.054	0.044 ± 0.014
	LO	82.7 ± 1.1	0.390 ± 0.057	0.032 ± 0.011
	DSS	72.6 ± 1.8	0.417 ± 0.046	0.060 ± 0.023
MPF	Bootstap	24.1 ± 1.6	0.391 ± 0.067	0.048 ± 0.011
	LO	31.3 ± 1.6	0.392 ± 0.064	0.030 ± 0.006
	DSS	100.0 ± 0.0	0.383 ± 0.054	0.050 ± 0.025
AMPF	Bootstap	65.5 ± 1.9	0.378 ± 0.047	0.031 ± 0.006
	LO	83.1 ± 1.3	0.374 ± 0.048	0.025 ± 0.004
	DSS	71.3 ± 1.7	0.410 ± 0.054	0.062 ± 0.023
Unweighted	DSS	-	0.382 ± 0.061	0.043 ± 0.027

One interesting finding from Table 6.2 is the performance of MPF(DSS). This combination yields a constant mean ESS of 100. This empirically validates the theoretical ideal discussed in Section 3.6. Furthermore, a constant ESS implies that the state estimation should be nearly identical to that of the unweighted DSS baseline. This is confirmed by both the data in the table and Figure 6.2, which showcase the unweighted DSS and the MPF(DSS) paths following each other closely.

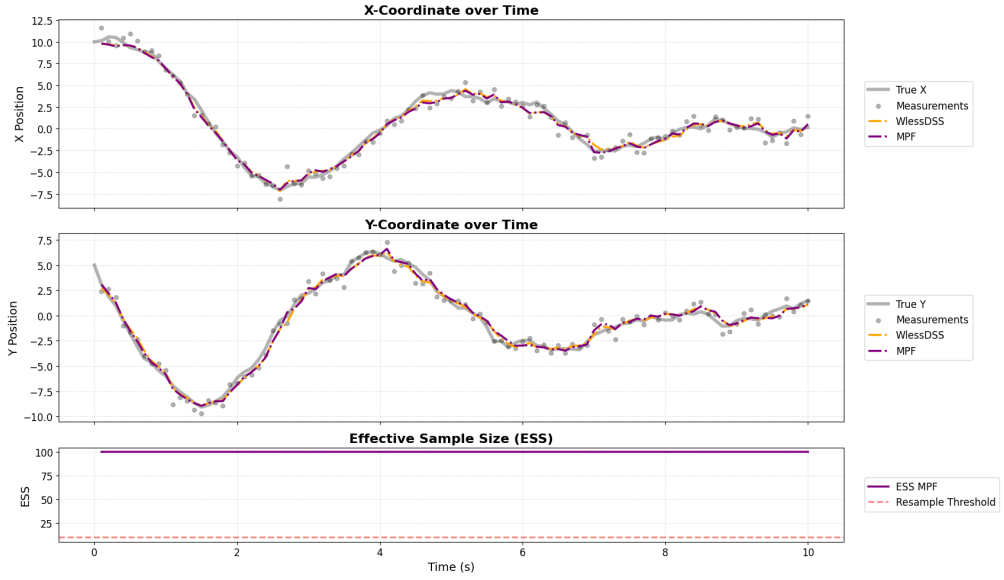


Figure 6.2: Tracking performance comparison between the unweighted DSS and the MPF(DSS).

Interestingly, the AMPF does not maintain the same level of ESS stability when

using the DSS proposal. The ESS is no longer constant, which we believed to be caused by the heuristic nature of the auxiliary filter step, namely that the choice of the deterministic value $\mu_t^{(k)}$ may inaccurately place more weights on some particles. Introducing this heuristic approximation ruins the exact calculations of the Sinkhorn algorithm and marginalization.

Again we note that the all filter and proposal combinations give a good MSE, with no noticeable best performances.

6.2.4 Gaussian Schrödinger bridge

In this section, we discuss the results of the machine-learned approximation of the Gaussian Schrödinger bridge. The primary goal is to demonstrate that amortized learning can be used to execute the repeated calculations of the Schrödinger bridge. First, we consider the performance of the exact analytical Gaussian bridge, which uses the Kalman filter to define the target distribution ρ_1. In Figure 6.3, we the exact Gaussian Schrödinger bridge transports the particles such that they perfectly align with the Kalman estimation.

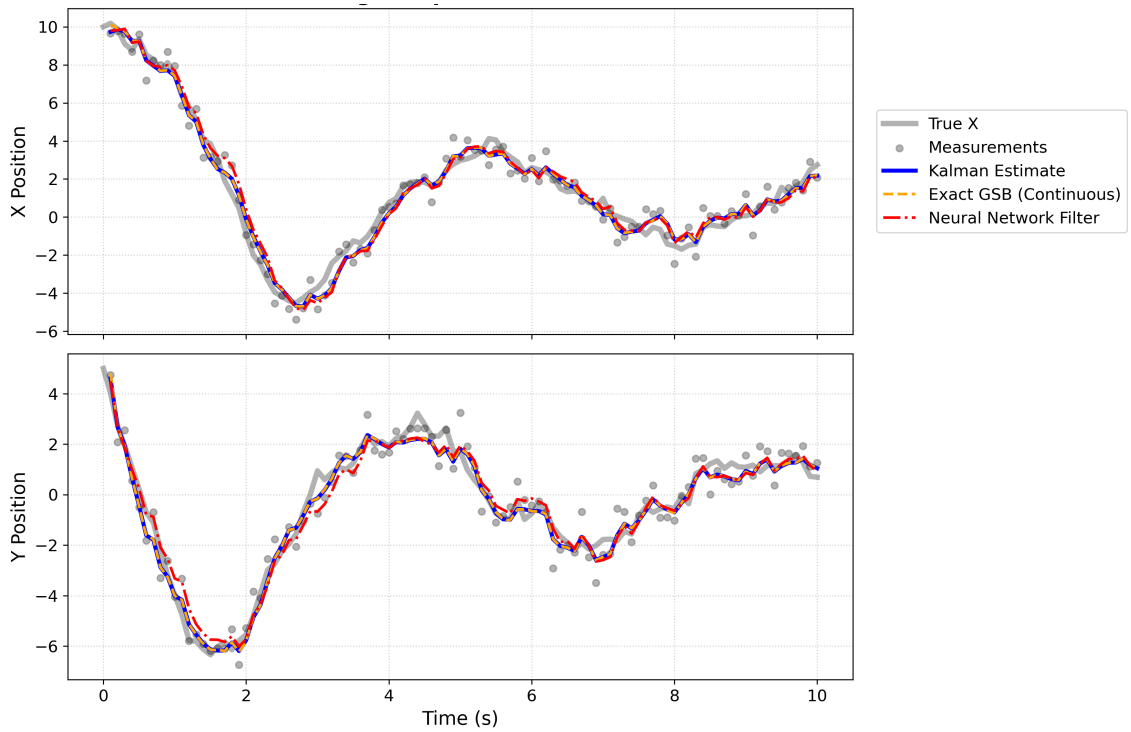


Figure 6.3: Tracking ability of the analytical Schrödinger solution and the trained network.

Figure 6.3 also illustrates the tracking performance of the learned Schrödinger bridge. This approximation is done by the MLP network with 3 previous measurements, proposed in Section 4.1. The networks showed similar results when previous measurements were included. No obvious difference was seen when including all observations in the random reservoir.

In the networks we observe a slightly larger deviation from the Kalman filter compared to the filters from the previous section, resulting in an MSE to the true path of 0.503 ± 0.074 and an MSE to the Kalman filter of 0.160 ± 0.037. Nonetheless, we still conclude from the figure that the network is capable of following the general path of the moth. This network only operates without particle weights. It serves strictly as a proof-of-concept, to show that it is possible to train a neural network to solve a Schördinger bridge problem. However, this unweighted state estimation already closely aligns with the true posterior distribution, an example can be seen of this in Figure 6.4.It is therefore probable that using this network as a proposal within a particle filter would yield low weight variance.

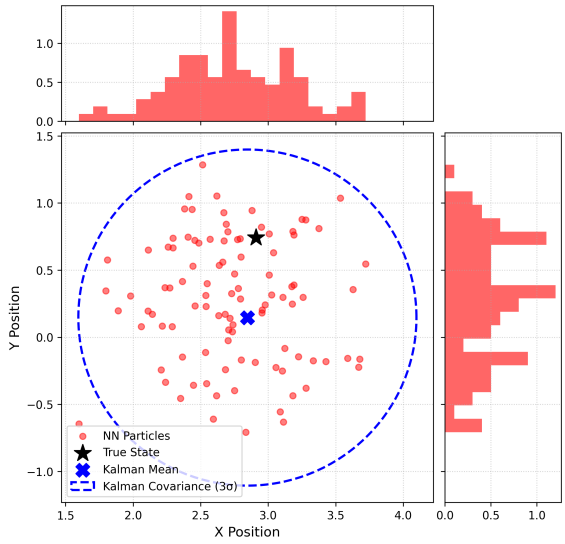


Figure 6.4: Spread of generated particles from a random step during tracking.

6.2.5 Performance of networks

In this section, we evaluate the implementation of the neural networks approximating the half-bridge, consisting of the φ-network and the NF. Unlike the Gaussian Schrödinger bridge, these can be applied in the nonlinear case.

The potential networks

Since we are primarily interested in the accuracy of the target distribution, and the starting distribution is already fixed by our initial particle cloud, our evaluation focuses exclusively on the forward φ-network. As described in previously this network suffices to calculate the conditional transition $\pi^*(\mathbf{x}_1 \mid \mathbf{x}_0)$ used to propagate the particles.

Because the φ-network outputs a continuous function that can only be evaluated (and not directly sampled), we assess its performance by computing its values over a spatial grid of potential states $\mathbf{x}_1$. Specifically, we evaluate the unnormalized target density $\pi(\mathbf{x}_1) \propto \exp(\varphi_{\text{net}}(\mathbf{x}_1)) \sum q(\mathbf{x}_1|\mathbf{x}_0)$, which maps the probability landscape given the entire initial particle distribution at $\mathbf{x}_0$.

Overall, training the φ-network yielded a spatial distribution that successfully covers the target points, exhibiting an approximately Gaussian shape with an accurate mean but an excessively large variance. We observed similar results independent of the number of historical observations provided to the network, and independent of whether a random reservoir was used or not. Illustrated in Figure 6.5 are the results from an MLP conditioned on three previous observations ($L = 3$). We see that the high-probability regions correctly cluster around the true target location. However, while the resulting spatial distribution is roughly spherical, it does not perfectly capture the desired Gaussian as it in general produces a distribution with a larger variance than the target distribution.

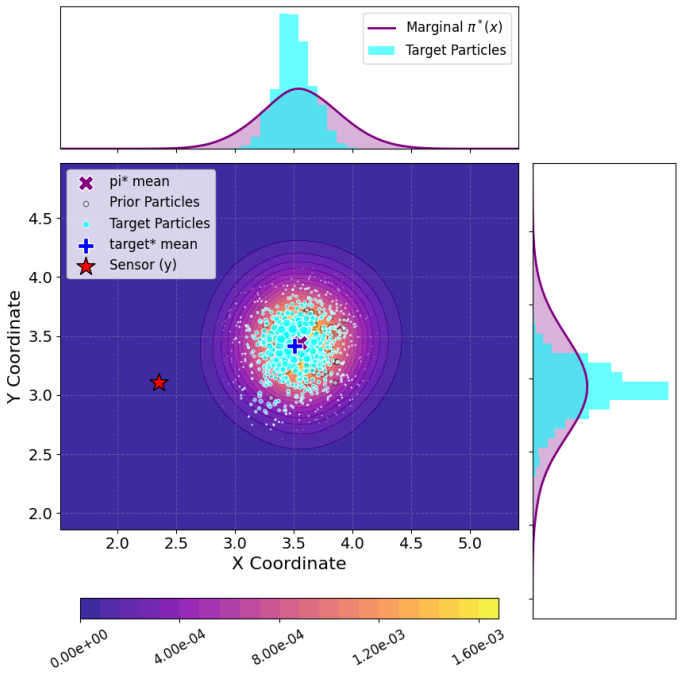


Figure 6.5: Comparing the trained $\pi^*(\mathbf{x}_1) \propto \exp(\varphi_{\text{net}}(\mathbf{x}_1)) \sum q(\mathbf{x}_1|\mathbf{x}_0)$ distribution to the training data (predicted and target particles).

6.2.5.1 Normalizing flow

Implementing and training the NF proved more challenging than the potential networks, with several persistent issues during the training phase.

Initially, we hypothesized that using a random reservoir would yield superior results by providing the model with the particle's full historical trajectory. While, the φ-network produced comparable results regardless of the number of historical observations provided, the NF models reacted poorly to the recurrent structure. As shown in Figure 6.6, the RNN struggled to locate regions of high probability, often generating entirely inaccurate distributions. Due to the severe training issues, the attempt to use the full trajectory as input was abandoned in favor of the MLP.

To retain some historical context without the RNN, the MLP was instead conditioned on three previous observations, as for the results of the φ network in

Figure 6.5. This adjustment gave more accurate results. Figure 6.7 illustrates a favorable training outcome using this approach, demonstrating that the network is capable of generate point with a similar distribution as the target particles it is trying to imitate.

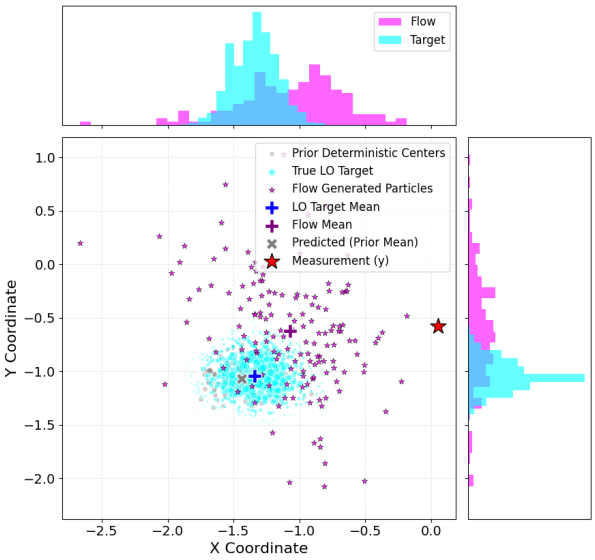


Figure 6.6: Generated $\mathbf{x}_{t+1}$ points by the NF with the full trajectory as input.

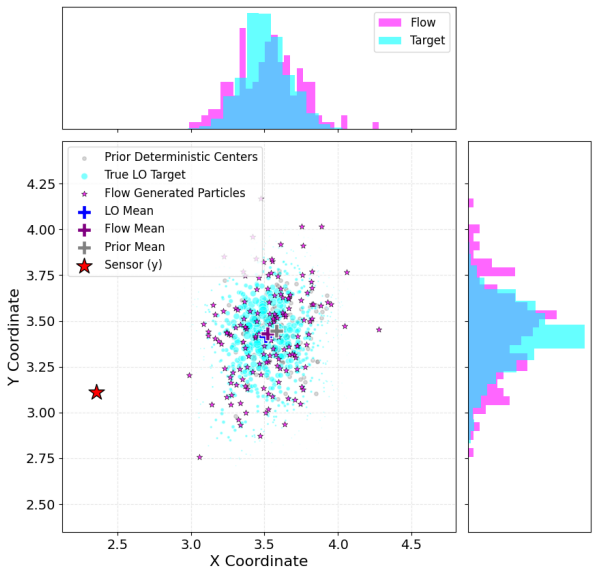


Figure 6.7: Generated $\mathbf{x}_{t+1}$ points by the NF with the latest ant three previous observations as input.

6.2.5.2 Tracking performance

Following the single-step distribution evaluations of the previous section, we deployed the trained NF model within the linear moth scenario (we specifically evaluate

Table 6.3: Performance comparison of filter architectures and with the NF as proposal. Values represent the mean over 50 independent runs using 100 particles.

Filter	Proposal	ESS	MSE (vs True)	MSE (vs Kalman)
Regular	NF	57.4 ± 2.0	0.349 ± 0.042	0.020 ± 0.006
APF	NF	65.4 ± 2.0	0.387 ± 0.058	0.031 ± 0.006
MPF	NF	57.4 ± 2.5	0.379 ± 0.060	0.039 ± 0.024
AMPF	NF	66.1 ± 1.7	0.366 ± 0.052	0.032 ± 0.005
Unweighted	NF	-	1.100 ± 0.307	0.799 ± 0.213

the MLP with input: current state, latest observation, three previous observation, using the DSS proposal to initiate the filter). The network was evaluated both as an unweighted transport option and as a proposal in the filter architectures.

Figure 6.8 illustrates the tracking performance of the unweighted NF network. Across all test runs, the generated trajectories are excessively smooth. Compared to the initial proof-of-concept results from Section 6.2.4, the NF estimated trajectory is further from the truth. It seems like the network relies too heavily on the prior dynamics, and fails to include the measurements properly. Consequently, the standalone network provides a poor state estimate.

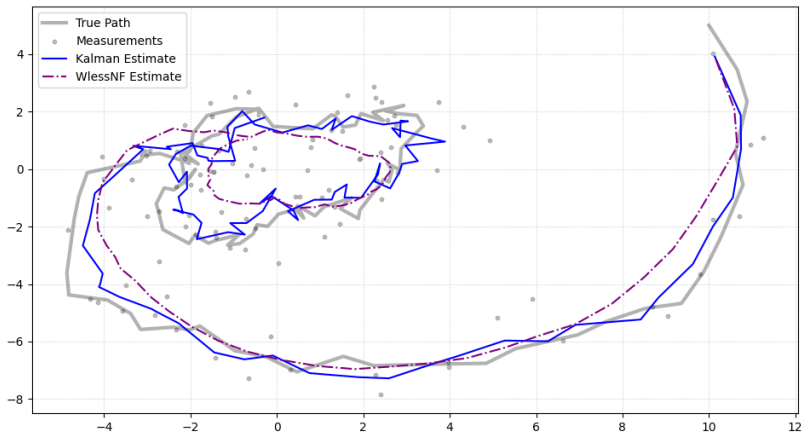


Figure 6.8: Tracking performance of the unweighted NF.

While integrating the NF model as a proposal distribution within a particle filter significantly improves its estimation accuracy compared to the standalone configuration, its performance remains lower than PF(DSS). Comparing Table 6.3 to our previous results, the NF performs comparable to the bootstap proposal. This is consistent with the observation that the NF mostly relies on the prior dynamics.

When the NF proposal is integrated into the MPF, the weight degeneracy is still noticeable (see Figure 6.9). We have thus not managed to approximate the DSS proposal well enough to achieve a constant ESS. We note that in this context, where the MPF with a Schrödinger proposal does not give constant ESS, the AMPF(NF) yields a superior solution to the higher mean ESS and lower MSE.

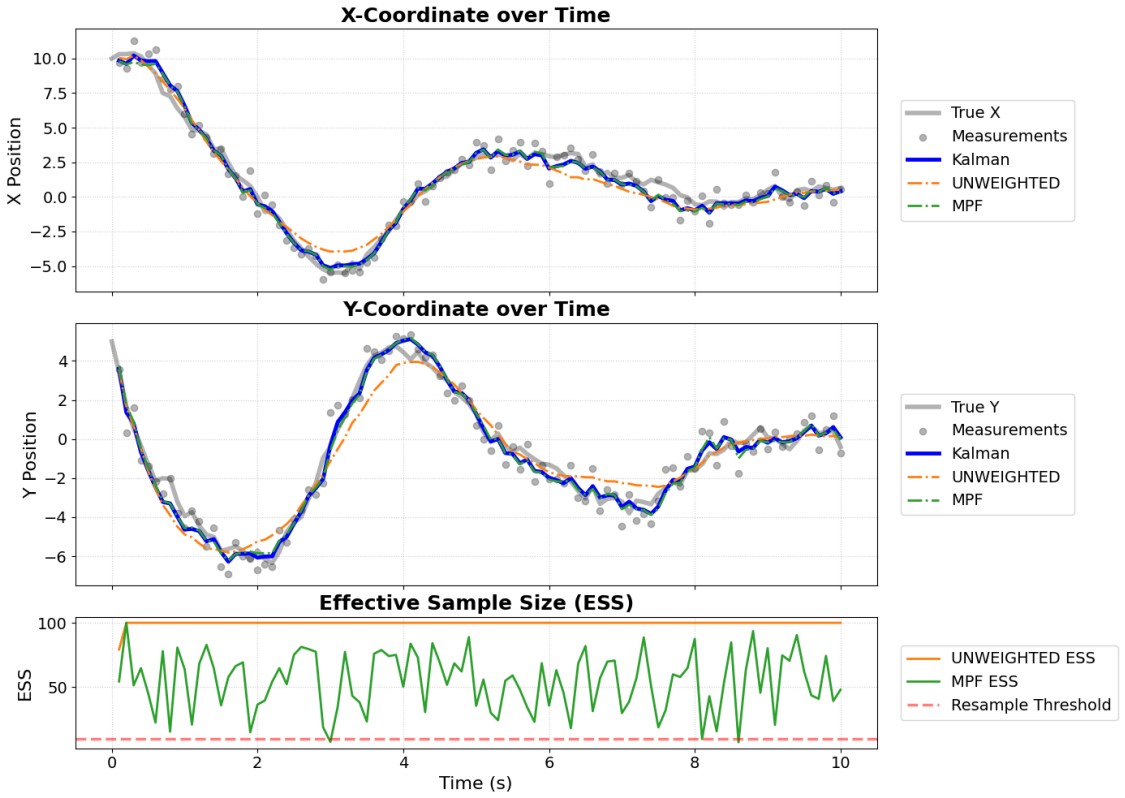


Figure 6.9: Tracking performance of the generative NF network, without weights and as a proposal in the MPF.

6.3 Nonlinear model: The plane

In order to showcase the methods presented in this thesis in a nonlinear scenario, we turn to the polar noise plane model. Note that, much like for the linear moth model, the values of the motion and measurement noises as well as the particle amount greatly affect the results. Since varying the noises has already been studied in Section 6.2.1 above, we let these values be constants which were judged reasonable for the nonlinear plane model. See Figure 6.11 for an example path. The amount of particles is also set to a constant $N = 100$ for the same reason as for the linear moth above.

6.3.1 Discrete state Schrödinger bridge

Unlike for the linear moth scenario, no exact benchmarking method exists in the nonlinear case. The LO proposal cannot be used as described in this thesis either since the scenario is nonlinear, and we are left with two methods: the bootstrap proposal, and the UKF. The UKF has the advantage of being very fast and versatile, but its accuracy cannot be improved by increasing amount of particles N, whereas the bootstrap proposal is somewhat slower but converges to the filtering distribution as $N \to \infty$. In order to get an idea of their performance, we turn to Table 6.4. We see that the UKF and BPF track the object decently, although with substantially higher error compared to the other filters. We also see from the table that the BPF

on average resamples the most of any proposal tested. A visualization of the average MSE and ESS per trajectory step is given in Figure 6.10.

Table 6.4: Average metrics over 50 runs, 100 particles, $\alpha = 100$ for DSS, trajectory length 50, and $\gamma = 0.1$ resampling threshold.

Filter	Proposal	Mean ESS	MSE (to true state) $\times 10^{-4}$	# Resamples
UKF		-	32.6 ± 124.6	-
Regular	Bootstrap	13.1 ± 10.2	8.9 ± 8.9	27.0
	DSS	35.5 ± 23.9	8.2 ± 6.8	5.4
Weightless	DSS	-	7.7 ± 6.4	-

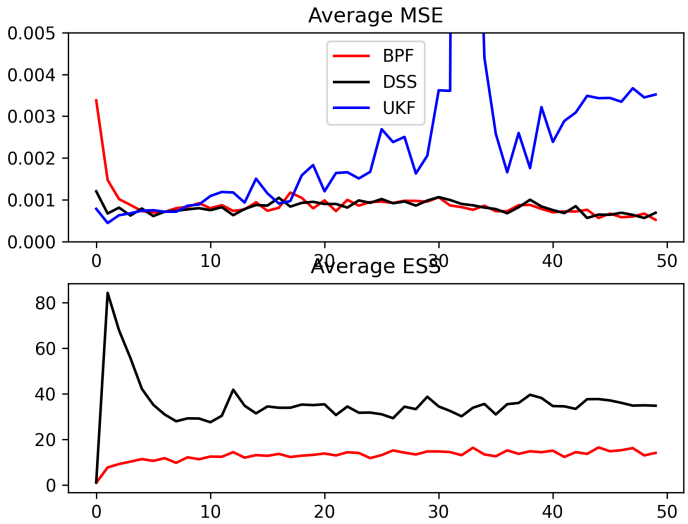


Figure 6.10: Average metrics per trajectory step over 50 runs. Computed with resampling threshold $\gamma = 0.1$. Comparing PF(DSS), UKF and PF(B).

The PF(DSS) on the other hand has both a substantially lower MSE compared to the UKF, but also resamples rather seldom compared to PF(B). Looking at Figure 6.11, we see that both PF(DSS) and PF(B) diverge very little from one another in the beginning of the trajectory, whereas the UKF at times give sharp outliers due to its approximations.

Furthermore, we see from Table 6.4 that the MSE for the weightless DSS is slightly lower than its particle filter counterparts, which was argued for in Section 3.6. We additionally see from Figure 6.12 that it doesn't get worse over time, although we only have 50 steps.

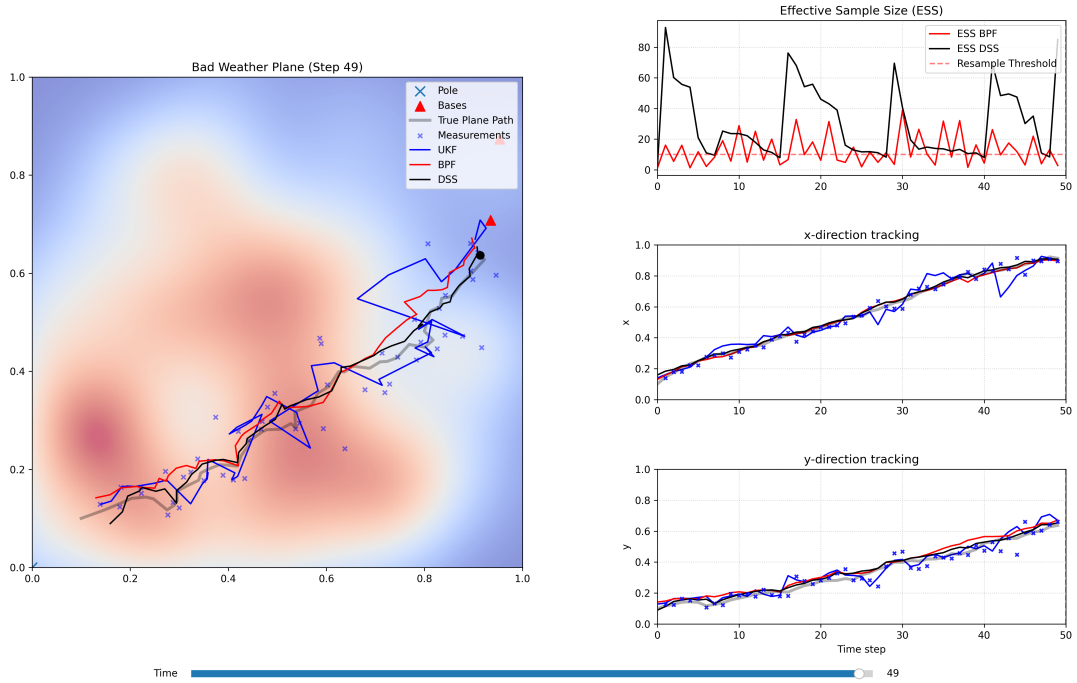


Figure 6.11: Tracking comparison for PF(B), PF(DSS) and UKF (left). ESS and one-dimensional views of tracking (right).

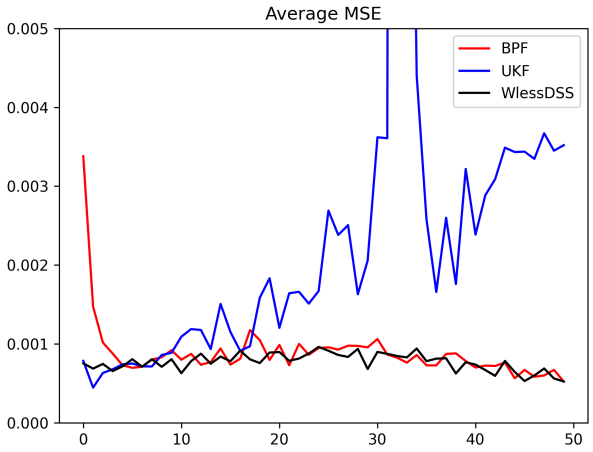


Figure 6.12: Average MSE for weightless DSS compared to UKF and PF(B) over 50 runs.

6.3.2 Marginal particle filters with Schrödinger proposals

We see from Table 6.5 that the MSE of MPF(DSS) stays more or less the same. Much like for the linear moth scenario, the ESS is almost maximal.

Table 6.5: Average metrics for the different filter architectures. Average taken over 50 runs, 100 particles, $\alpha = 100$ for DSS, trajectory length 50.

Filter	Proposal	Mean ESS	MSE (to true) $\times 10^{-4}$
MPF	DSS	99.9 ± 0.3	7.8 ± 6.4
AMPF	DSS	69.9 ± 22.2	7.9 ± 6.4

Looking back at the table, we see that using AMPF(DSS) gives similar values to the MSE, but substantially lower ESS compared to MPF(DSS). This is motivated in the same way as for the linear moth model. A tracking comparison between MPF(DSS) and AMPF(DSS), as well as PF(B) with many particles acting as a reference process is given in Figure 6.13. We see visually that MPF(DSS) seems to follow the reference process almost perfectly, while AMPF(DSS) diverges from it at times.

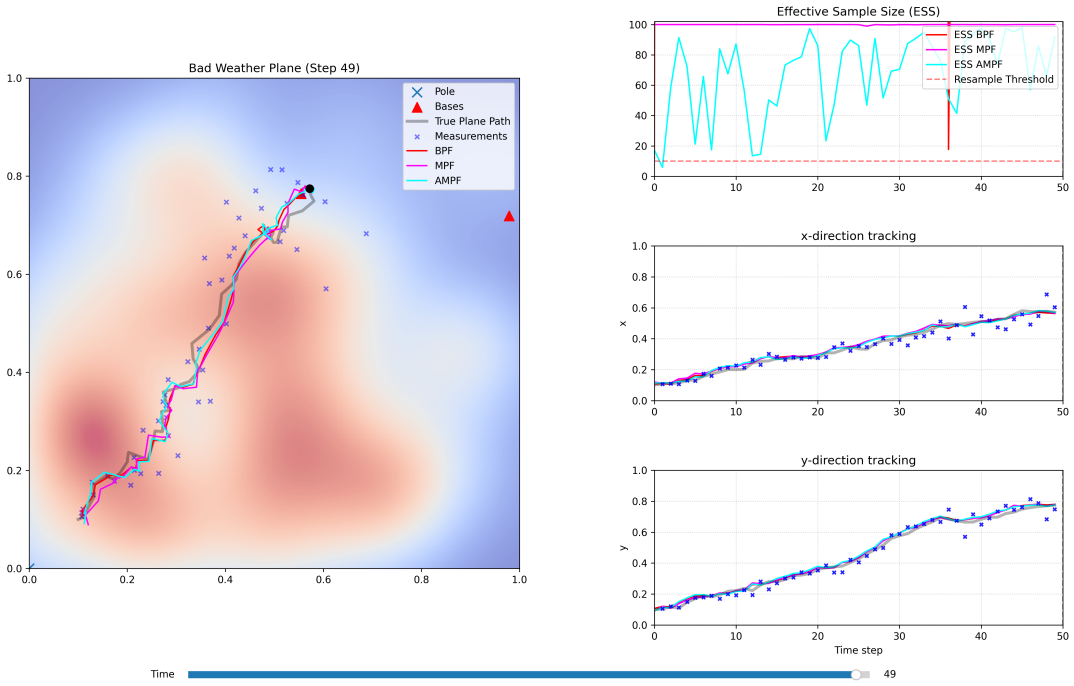
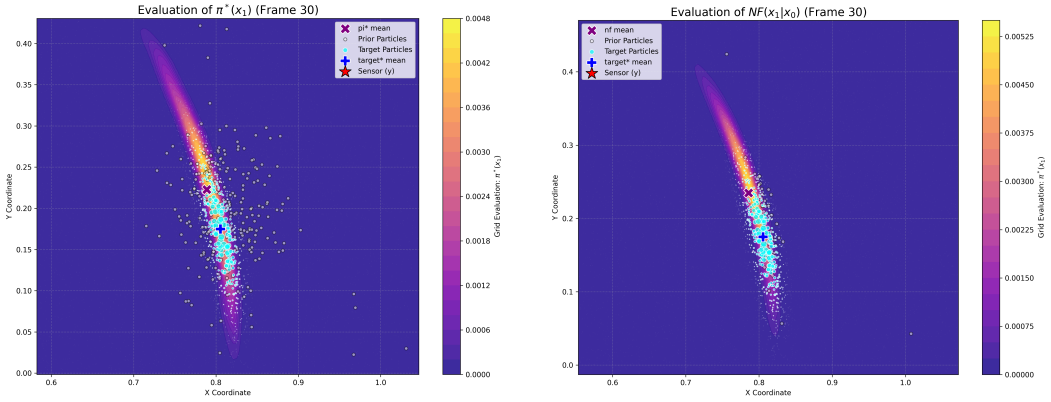


Figure 6.13: Tracking comparison for PF(B) with 15,000 particles, MPF(DSS) and AMPF(DSS) (left). ESS and one-dimensional views of tracking (right).

6.3.3 Performance of networks

Since PF(DSS) could be successfully implemented in a nonlinear scenario while giving good results, we now turn to the faster but more approximation-heavy neural networks. Much like for the moth, inputting a random reservoir to the networks did not yield very good results. We thus opted to instead only input three previous observations to the networks ($L = 3$). We evaluate the networks in the same manner as for the moth, plotting the training data (prior and target particles) on top of a heat-map of the networks' evaluations, see Figure 6.14. In this way, we can look for

potential discrepancies between the distributions. We see that while the networks seem to have found decent approximations of the true distribution, they seem to create a slightly biased and more spread-out distribution than the true one. While both distributions coincide rather well, the NF gives an even more biased distribution than the φ-network, since it was trained on an approximate distribution.



(a) φ network. **(b)** Normalizing flow.

Figure 6.14: Comparing the trained distribution with the training data (prior and target particles).

We further see from Table 6.6 and Figure 6.16 that the amount of resamples of PF(NF) is higher than that of PF(DSS), with lower average ESS. The tracking is still better than that of the UKF, further motivated by Figure 6.16, and we do see that the trajectory found in Figure 6.15 is not as stable as that of PF(DSS), although it does seem to present fewer outliers than the UKF. Note that the ESS plateau in Figure 6.16 stems from the fact that we use PF(DSS) for the first three trajectory steps, since the NF inputs the previous three observations.

Table 6.6: Average metrics for the NF proposal. Average taken over 50 runs, 100 particles, $\alpha = 100$ for DSSPF, trajectory length 50, and $\gamma = 0.1$ resampling threshold for the regular particle filter.

Filter	Proposal	Mean ESS	MSE (to true) $\times 10^{-4}$	Resamples
Regular	NF	20.6 ± 9.9	9.1 ± 9.2	21.1
Weightless	NF	-	15.0 ± 14.9	-
MPF	NF	41.6 ± 9.4	7.6 ± 8.6	-
AMPF	NF	18.9 ± 10.3	8.4 ± 8.5	-

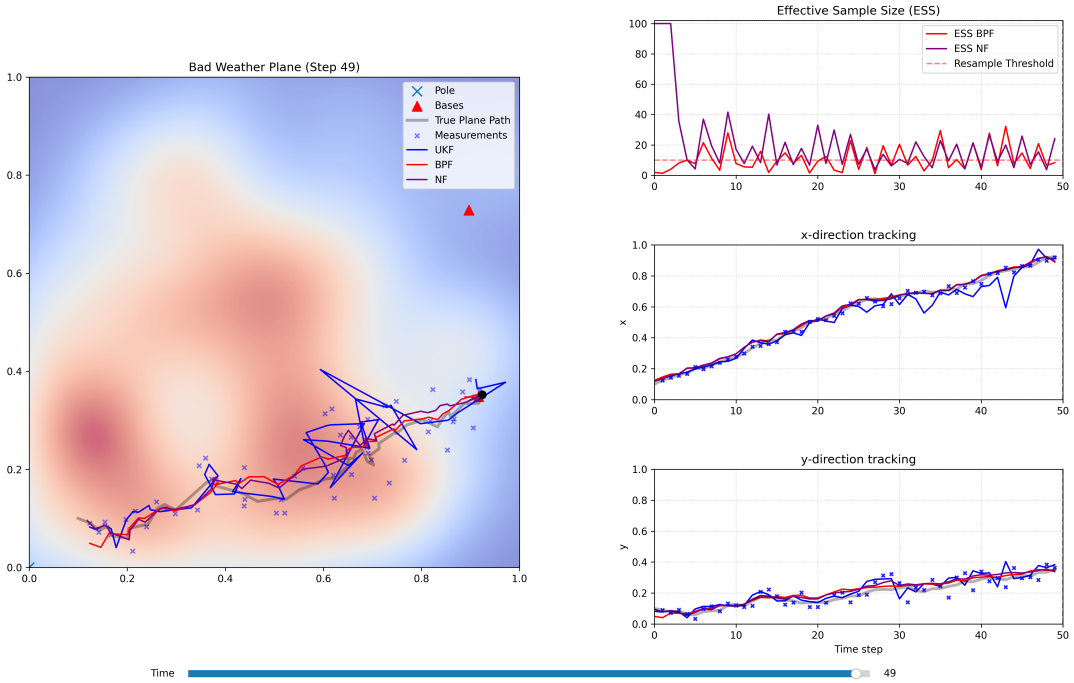


Figure 6.15: Tracking comparison for PF(B), PF(NF) and UKF (left). ESS and one-dimensional views of tracking (right).

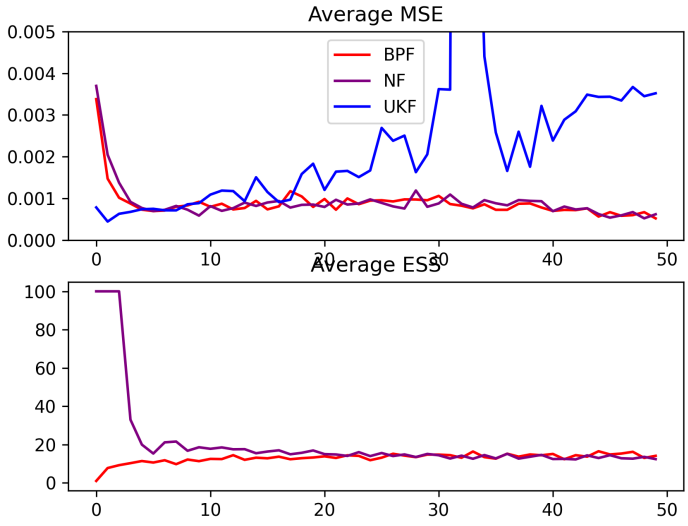


Figure 6.16: Average metrics per trajectory step over 50 runs, resampling threshold $\gamma = 0.1$. Comparing PF(NF), UKF and PF(B).

We further see from Table 6.6 that the MSE for the weightless NF, unlike DSS, performs worse than its particle filter counterpart. While a weightless approach is computationally cheaper (especially compared to the MPF), it neither allows for

keeping track of the ESS, nor adjusting the distribution for errors in the approximation. This is especially relevant for the NF proposal, which contains even more approximations than the DSS proposal and is thus likely to lose accuracy over time. In fact, we see from Figure 6.17 that the MSE gets worse over time for the weightless NF.

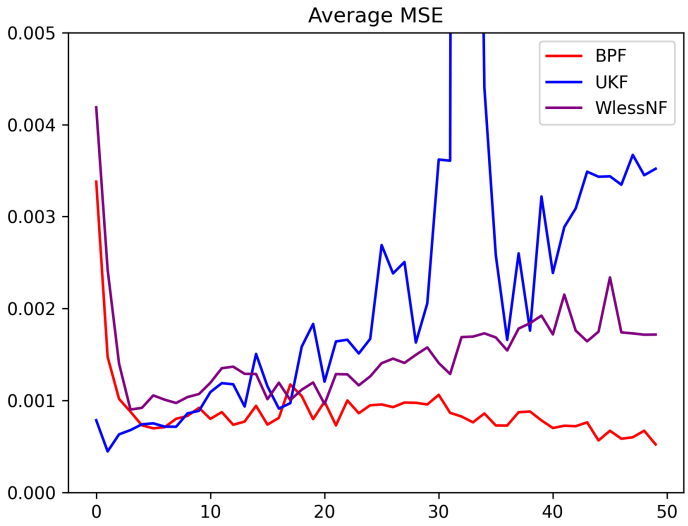


Figure 6.17: Average MSE for the weightless NF compared to PF(B) and UKF over 50 runs.

Finally, the table indicates a substantial improvement to the MSE when using MPF(NF). The ESS is not as good as that of MPF(DSS), however, although substantially higher than PF(NF). This further motivates the fact that the NF may not be as fit to be used in a weightless scenario compared to the DSS proposal. AMPF(NF) behaves similarly to AMPF(DSS), in the sense of lost ESS, for the same reasons. The MSE for AMPF(NF) is also notably higher than MPF(NF).

6.4 Approximating the marginal particle filter

As we have seen, two alternatives to using Schrödinger bridge proposals in filtering distributions is by either removing the update step, or using the MPF. The former alternative works well for very accurate proposals, but fail as the approximation quality worsens, while the latter is computationally very heavy. A compromise would be to approximate the update step of the MPF to keep oversight over the weights and ESS, while evolving in the filtering distribution (as opposed to the trajectory distribution). Two methods to do so have been presented in Section 4.3: using n random indices in the update step, or sampling them based on their corresponding weight. In Table 6.7 are given average metrics for both methods, using $n = 5$ and $n = 30$, using MPF(NF) and AMPF(NF). Only the NF proposal has been studied due to time limitation and since this is the most likely proposal to be approximated

in a real-world application.

Table 6.7: Average metrics for different MPF(NF) approximation strategies. Average over 50 runs, 100 particles, trajectory length 50, and $\gamma = 0.1$ resampling threshold.

Particle filter	Approximation type	n	ESS	MSE (to true) $\times 10^{-4}$
Regular PF	No approximation	-	20.6 ± 9.9	9.1 ± 9.2
MPF	Weight	30	40.0 ± 9.4	7.7 ± 6.4
		5	38.2 ± 8.0	8.2 ± 8.2
	Random	30	40.1 ± 8.6	7.8 ± 7.0
		5	38.7 ± 9.4	7.7 ± 8.1
	No approximation	-	41.6 ± 9.4	7.6 ± 8.6
AMPF	Weight	30	33.1 ± 10.7	8.2 ± 7.0
		5	31.9 ± 9.9	8.3 ± 7.3
	Random	30	34.1 ± 10.1	7.8 ± 6.7
		5	31.5 ± 10.0	8.1 ± 6.9
	No approximation	-	18.9 ± 10.3	8.4 ± 8.5

Note that the approximate values for the ESS and MSE seem to lie somewhere between the unapproximated marginal filters and the regular particle filter. We further note that the MPF again surpasses the AMPF in terms of ESS, and seem to give slightly better MSE. Due to the similarity of all of the values, it is difficult to conclude whether the weight or random approximation is best. It is clear however, that a higher n gives better ESS and MSE, however.

7

Conclusion

The goal of this thesis was to study the extent to which Schrödinger bridges can be used for Bayesian filtering. We implemented three proposals solving the Schrödinger bridge problem in different ways:

- the discrete state Schrödinger bridge, providing an upper bound for how well a Schrödinger bridge-based proposal can perform, but too computationally demanding to be used in real-world situations,
- the Gaussian Schrödinger bridge (GSB), demonstrating that it is possible to learn the goal distribution using neural networks, but only applicable in the linear-Gaussian case, and
- the learned half-bridge, which can also be applied in nonlinear scenarios.

We demonstrated empirically that the DSS can rival or surpass the most widely used solutions to the filtering problem in both the linear and nonlinear case. This was especially evident when used in an MPF, or in the weightless setting, where the proposal is used without importance weight updates. These results suggest that Schrödinger bridge-based proposals can theoretically provide very good solutions to the filtering problem.

We further demonstrated that the goal distribution can be approximated using a neural network: in the linear case via the GSB, and in the nonlinear case via the φ-network and NF. The NF struggled to accurately track the in both the linear and nonlinear case. IN the nonlinear case, when used within an MPF, it did not achieve constant weights but manage to track well, outperforming both the Bootstrap proposal and the UKF.

Since the MPF is computationally more demanding than the regular particle filter, approximations to the algorithm were evaluated. We studied several such approximation strategies together with the NF proposal and found that, while they lead to worse tracking performance than the exact MPF, they still outperformed both the weightless NF and the NF used in a regular particle filter.

While this thesis concludes that Schrödinger bridges can indeed be effectively utilized for Bayesian filtering, several aspects with potential for improvement and directions for future research were identified. The design choices relating to the neural network architecture are discussed in Section 7.1, and ideas for future research on the topic are presented in Section 7.2.

7.1 Neural network architecture

The neural networks used for the half-bridge may still be substantially improved upon. As noted from Figure 6.14, the learned distributions do not perfectly overlap with the target points: they are both more spread out and skewed relative to the true distribution. Below, some possible causes and solutions to this problem are discussed.

In order to guarantee exact evaluation of the learned distribution, and by extension a mathematically correct particle filter update step, a normalizing flow was used. However, normalizing flows can only represent distributions expressible through an invertible function, which limits their expressive power and may help explain the network's difficulty in learning the true distribution. Since a network approximating the evaluation of the distribution already exists in the form of φ, an alternative approach would be to use a more expressive generative architecture – such as a generative adversarial network or a variational autoencoder – for sampling, and rely on φ for evaluation. This would sidestep the invertibility constraint while preserving the ability to evaluate the learned distribution, although it cannot be used in a particle filter.

Furthermore, as noted in Section 4.1.4, incorporating previous measurements into the input via a random reservoir may lead to better results. However, we were unable to successfully train this architecture, as the network struggled to simultaneously learn to process the reservoir state and learn the filtering distribution. A possible remedy would be to first train the network without the reservoir component, and only introduce it later.

7.2 Future work

There exist multiple ways to build upon this thesis. Firstly, testing the proposals on real-world scenarios may challenge them more than our experiments did, and reveal flaws that our models were too simple to expose. For example, the synthetic nature of the data used in this thesis may be much more regular than real-world data, potentially giving a skewed picture of the proposals' actual performance.

Furthermore, it may be relevant to consider additional metrics beyond those used in this thesis. For instance, calculation time was not measured, as doing so would require optimizing the code, which falls outside the scope of this thesis. However, as argued above, calculation time is important for real-world use of filtering methods. Large computational overheads may negate the gain from using fewer particles, in which case a simpler proposal may be preferable. Additionally, metrics such as the Wasserstein distance may be better suited to comparing the learned distribution with the true one, rather than taking the MSE between the weighted mean of the particle distribution and the true path. This is especially relevant in highly nonlinear scenarios, where the weighted mean and the mode do not coincide, or where several modes are present.

Finally, we have shown that combining the Schrödinger bridge proposals with MPFs greatly improves their performance, but that the MPF needs to be approximated to be computationally viable. We studied several approximation strategies based on approximating the update sums via importance sampling, but another approach proposed by Klaas et al. [8] is to make use of k-d trees. We did not implement this, as k-d trees rely on the assumption that the probability of a previous particle giving rise to a later one is determined by some distance metric – which is not necessarily the case for a learned evaluation function. Nevertheless, this approach may provide a useful heuristic for selecting which points to include in the update sum, and has the additional advantage of allowing one to bound the maximum tolerated error of the sum. Whether this assumption holds sufficiently well in the context of a learned evaluation function remains an open question, and we believe it would be worth exploring empirically.

Bibliography

[1] M. Arulampalam et al. “A tutorial on particle filtersfor online nonlinear/non-Gaussian Bayesian tracking. In: Connection BetweenForest Resources and Wood Quality: Modelling Approaches and Simulation Soft-ware”. In: *IEEE Trans.* 50 (Jan. 2002).

[2] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. *Density estimation using Real NVP*. 2017. arXiv: 1605.08803 [cs.LG]. URL: https://arxiv.org/abs/1605.08803.

[3] Alexander Gray and Andrew Moore. “`N-Body’Problems in Statistical Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by T. Leen, T. Dietterich, and V. Tresp. Vol. 13. MIT Press, 2000. URL: https://proceedings.neurips.cc/paper_files/paper/2000/file/7385db9a3f11415bc0e9e2625fae3734-Paper.pdf.

[4] Alexander Gray and Andrew Moore. “Nonparametric Density Estimation: Toward Computational Tractability”. In: May 2003. DOI: 10.1137/1.9781611972733.19.

[5] Hristov, Hristo. *Introduction to K-D Trees*. https://www.baeldung.com/cs/k-d-trees. Accessed: 2026-02-06.

[6] Hanwen Huang. *A Closed-Form Diffusion Model for Learnring Dynamics from Marginal Observations*. Nov. 2025. DOI: 10.48550/arXiv.2511.07786.

[7] Simon J. Julier and Jeffrey K. Uhlmann. “New extension of the Kalman filter to nonlinear systems”. In: *Signal Processing, Sensor Fusion, and Target Recognition VI*. Ed. by Ivan Kadar. Vol. 3068. International Society for Optics and Photonics. SPIE, 1997, pp. 182–193. DOI: 10.1117/12.280797. URL: https://doi.org/10.1117/12.280797.

[8] Mike Klaas, Nando de Freitas, and Arnaud Doucet. *Toward Practical N2 Monte Carlo: the Marginal Particle Filter*. 2012. arXiv: 1207.1396 [stat.CO]. URL: https://arxiv.org/abs/1207.1396.

[9] Hamed Masnadi-Shirazi, Alireza Masnadi-Shirazi, and Mohammad Amir Dastgheib. *A Step by Step Mathematical Derivation and Tutorial on Kalman Filters*. Oct. 2019. DOI: 10.48550/arXiv.1910.03558.

[10] Bernhard Mehlig. “Supervised Recurrent Networks”. In: *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers*. Cambridge University Press, 2021, pp. 157–176.

[11] Art B. Owen. “Importance sampling”. In: *Monte Carlo theory, methods and examples*. https://artowen.su.domains/mc/, 2013. Chap. 9.

[12] Michele Pavon, Esteban G Tabak, and Giulio Trigila. *The data-driven Schroedinger bridge*. 2018. arXiv: 1806.01364 [math.OC]. URL: https://arxiv.org/abs/1806.01364.

[13] Michael Pitt and Neil Shephard. "Filtering via Simulation: Auxiliary Particle Filters". In: *Journal of the American Statistical Association* 94 (Nov. 1997). DOI: 10.2307/2670179.

[14] Simo Särkää and Lennart Svensson. *Bayesian Filtering and Smoothing. Second Edition*. Cambridge University Press, 2023, p. 240.

[15] Michael te Vrugt. "An Introduction to Reservoir Computing". In: *Artificial Intelligence and Intelligent Matter*. Springer Nature Switzerland, 2026, pp. 53–66. ISBN: 9783032041296. DOI: 10.1007/978-3-032-04129-6_4. URL: http://dx.doi.org/10.1007/978-3-032-04129-6_4.

[16] Nick Whiteley and Adam M. Johansen. "Auxiliary particle filtering : recent developments". In: *Bayesian time series models*. Ed. by David Barber, A. Taylan Cemgil, and Silvia Chiappa. Cambridge: Cambridge University Press, Sept. 2011, pp. 52–81. ISBN: 9780521196765. DOI: 10.1017/CBO9780511984679.004. URL: https://doi.org/10.1017/CBO9780511984679.004.

[17] Stephen Y. Zhang and Michael P H Stumpf. *Learning non-equilibrium diffusions with Schrödinger bridges: from exactly solvable to simulation-free*. 2026. arXiv: 2505.16644 [stat.ML]. URL: https://arxiv.org/abs/2505.16644.

A

Kernel density estimation

In the marginal particle filter, the added complexity comes from identifying the probability of every particle $\mathbf{x}_t^{(i)}$ originating from each particle $\mathbf{x}_{t-1}^{(j)}$. Such a system, in which $j = 1, ..., N$ particles affect each of $i = 1, ..., M$, particles is called an *N-body problem*. One concept from N-body learning is called *kernel density estimation* (KDE). While KDEs are mostly used in the context of probability density estimation, it was noted by [8] that it can also be used to approximate the *influence* q_i of a set of *sources* $x_j \in \mathcal{X}$ with associated weights ω_j on a set of *targets* $y_i \in \mathcal{Y}$ with an *influence function* (or *kernel*) $K(x_j, y_j)$.

$$q_i = \sum_{j=1}^{N} \omega_j K(x_j, y_i),\ i = 1, .., M \tag{A.1}$$

The exact approach to calculating q would be to perform all point pair calculations, with a complexity of $\mathcal{O}(d)$, where $d = |\{x \in \mathcal{X},\ y \in \mathcal{Y} | K(x, y) \neq 0\}|$. For a strictly non-zero kernel, such as an infinite-tailed Gaussian, this means $\mathcal{O}(MN)$ complexity, even though some points only influence each other infinitesimally. Below, we detail how to approximately find the influence (A.1) with an average of $\mathcal{O}(N \log^2(M))$ time (an even more approximate algorithm with $\mathcal{O}(N \log(M))$ time complexity exists but is not explained in this thesis) by combining sources and/or targets according to some specified error ε using *single/dual k-d search*, utilizing a data structure called *k-d tree* (see Sections A.2, A.3 and A.1 respectively).

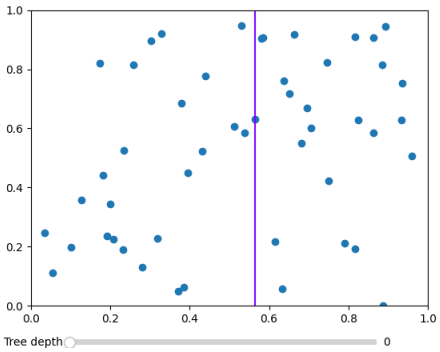
It is further explained by [8] that the MPF update step can be mapped to this problem directly:

$$\left\{x_j\right\} := \left\{\mathbf{x}_{t-1}^{(j)}\right\},\quad \left\{y_i\right\} := \left\{\mathbf{x}_t^{(i)}\right\},\quad \left\{w_j\right\} := \left\{w_{t-1}^{(j)}\right\},\quad \text{and}$$
$$K(x_j, y_i) := q\Big(\mathbf{x}_t^{(i)} = y_i \Big| \mathbf{x}_{t-1}^{(j)} = x_j, \mathbf{y}_t\Big) \text{ or } p\Big(\mathbf{x}_t^{(i)} = y_i \Big| \mathbf{x}_{t-1}^{(j)} = x_j\Big).$$

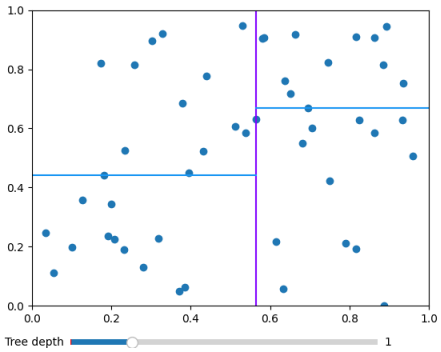
A.1 k-d tree

Following the explanation of [5], *k-dimensional trees* (k-d trees) are binary trees that partition a k-dimensional space. Every non-leaf node corresponds to a k-dimensional hyperplane perpendicular to one of the dimensional axes that partitions the space in two, such that the left subtree corresponds to the hyperspace to the left of the

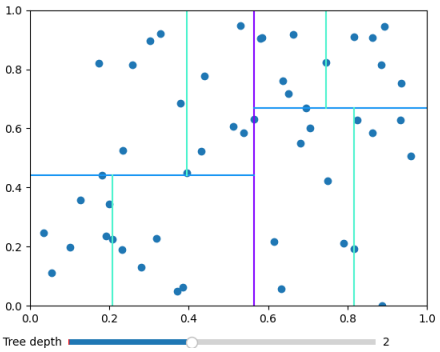
hyperplane and correspondingly for the right subtree. In the context of this thesis, the axes are chosen by circling through the k dimensions and the position of the hyperplane such that it intersects the median point of that axis.



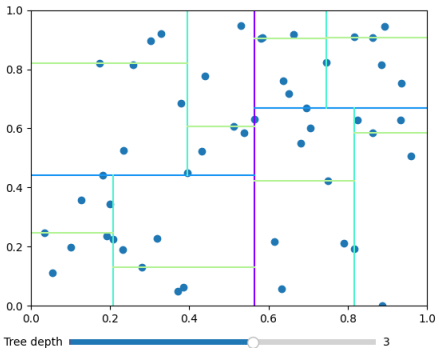
(a) Plane partition at root.



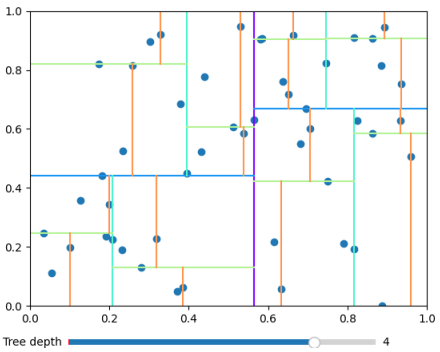
(b) Plane partition at tree depth 1.



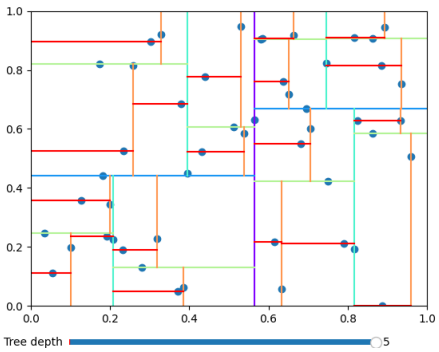
(c) Plane partition at tree depth 2.



(d) Plane partition at tree depth 3.



(e) Plane partition at tree depth 4.



(f) Plane partition at tree depth 5.

Figure A.1: Plane partition at different k-d tree node depths.

Figure A.1 showcases how k-d trees partition the 2-dimensional space when diving deeper down its branches. Note that every node is traversed by a line in the leaves, meaning that every space partition is empty. This motivates why we claimed that

leaf nodes do not partition the hyperspace. Inner nodes however contain points, which are additionally assigned weights. The complexity of building a k-d tree on N points with the above specifications is either $\mathcal{O}(N \log(N))$ or $\mathcal{O}(N \log^2(N))$ depending on whether the median is approximated or found.

A.2 Single-tree search KDE

We construct one k-d tree of the sources $X \subset \mathcal{X}$ with weights $\{w_j\}$. For every point $y_i \in Y \subset \mathcal{Y}$, we recursively calculate the approximate influence q_i by descending the tree. See Algorithm 7.

Algorithm 7 Single-tree search KDE

Using X source points, Y target points
Build k-d tree from X setting weights w_j and getting root x_{root}
for $i = 1, ..., M$ **do**
    Set $q_i = 0$
    SingleTreeSearch(x_{root}, y_i, q_i)

procedure SingleTreeSearch(x, y, q)
    if IsLeaf(x) **then**
        $q_{y_{\text{index}}} \leftarrow q_{y_{\text{index}}} + x_{\text{weight}} K(x, y)$
        return
    Calculate $d_{\min}, d_{\max}, K_{\min}, K_{\max}$
    if $x_{\text{total\_weight}}(K_{\max} - K_{\min}) < \varepsilon$ **then**
        $q \leftarrow q + x_{\text{total\_weight}} K(x_{\text{centroid}}, y)$
        return
    SingleTreeSearch(x_{left}, y, q)
    SingleTreeSearch(x_{right}, y, q)

A.3 Dual-tree search

While single-tree search only approximates the source points by grouping close-by or small points together, *dual-tree search* also approximates the target points with a k-d tree. It is explained by [3] that the search can be thought of as a simultaneous traversal of two trees, meaning we also group similar or far-away target points together. The search is thus based on node-node comparisons, rather than point-node comparisons as for single-tree search. They further show that dual-tree search can speed up many N-body problems compared to single-tree search, although no claim is made in the specific KDE case.

The algorithm is as follows, and greatly resembles Algorithm 7. Note however that weights have no meaningful interpretation for the target tree, and are all equally set to $\frac{1}{M}$. Also note that the distances now correspond to the lower and upper bounds for any points in both chunks, see Figure A.2.

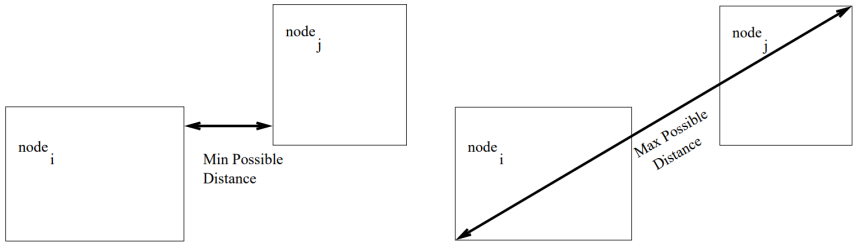


Figure A.2: The lower and upper bound on pairwise distances between the points contained in each of two k-d tree nodes, taken from [4].

A.4 Application to marginal particle filters

In the MPF weight update step, we wish to approximate

1. $\sum_j w_j p(y_i|x_j)$, and
2. $\sum_j w_j q(y_i|x_j, \mathbf{y}_t)$

for $i = 1, ..., M$. Let us therefore define $X_p := \left\{\mathbb{E}\left[p(\mathbf{x}_t|\mathbf{x}_{t-1}^{(j)})\right] \middle| j = 1, ..., N\right\}$ and $X_q := \left\{\mathbb{E}\left[q(\mathbf{x}_t|\mathbf{x}_{t-1}^{(j)}, \mathbf{y}_t)\right] \middle| j = 1, ..., N\right\}$. In linear-Gaussian models, these sets can be calculated explicitly. Note that for both of these proposals, the kernels to evaluate the influence of X_p and X_q on Y are Gaussian with known covariance matrices. Since the Gaussian kernel between two points depends on the distance between them, it makes sense to make use of dual-tree search to partition the points into spatial chunks to approximate the influence. We thus approximate both 1. and 2. by performing a dual-tree search with target Y, the only difference being the sources X_p and X_q respectively.

We believe that this method can be used to speed up one type of nonlinear model, where points are propagated as

$$y = f(x) + \nu,$$

where $p(y|x) = p(y - x)$. That, is, models with some nonlinear known motion model $f(\cdot)$ onto which some noise depending on some distance metric has been added (e.g. Gaussian noise). There exist, however, proposals and models for which we believe to be incompatible with KDE using dual-tree search.

KDE using dual-tree search requires the evaluation of the minimum and maximum influence bound between nodes. This is not always possible. For example, for proposals which do not make use of a distance metric, points on the edge of the chunks may not give the highest/lowest influence. Partitioning the state-space into chunks thus has no meaningful interpretation. This problem also persists in the general nonlinear case.

Note that [4] who devised the dual-tree search only looked at kernels depending on a distance $K(x, y) = K(y - x)$.

Algorithm 8 Dual-tree search KDE

Using X source points, Y target points
Build k-d tree from X with weights w_j and Y with weights $\{\frac{1}{M}\}$, and getting roots x_{root} and y_{root} respectively
Set $q_i = 0$ for $i = 1, ..., M$
DualTreeSearch(x_{root}, y_i, q_i)

procedure DualTreeSearch(x, y, q)
    if IsLeaf(x) and IsLeaf(y) **then**
        $q_{y_{\text{index}}} \leftarrow q_{y_{\text{index}}} + x_{\text{weight}} K(x, y)$
        return
    Calculate $d_{\min}, d_{\max}, K_{\min}, K_{\max}$
    if $x_{\text{total\_weight}} \cdot y_{\text{size}} \cdot (K_{\max} - K_{\min}) < \varepsilon$ **then**
        $q_{y_{\text{indeces}}} \leftarrow q_{y_{\text{indeces}}} + x_{\text{total\_weight}} K(x_{\text{centroid}}, y_{\text{centroid}})$
        return
    if $x_{\text{size}} > y_{\text{size}}$ **then**
        DualTreeSearch(x_{left}, y, q)
        DualTreeSearch(x_{right}, y, q)
    else
        DualTreeSearch(x, y_{left}, q)
        DualTreeSearch(x, y_{right}, q)

B

Neural networks

In this appendix, we give an introduction to two neural network architectures used in this thesis.

B.1 Normalizing flows

A *normalizing flow* (NF) is a neural network architecture that estimates a probability distribution π on $\mathbb{R}^D$ given samples from it. It creates an approximate distribution $\tilde{\pi}$ by modifying another, simpler distribution d by means of some invertible *coupling function* f. Letting $x \sim \pi$ be an observed variable and $z = f(x) \sim d$ be its corresponding latent variable from this simpler distribution, we can now compute

$$p(x) = p(f(x)) \left|\det\left(J(x)\right)\right| = p(z) \left|\det\left(J(x)\right)\right|, \quad J(x) = \frac{\partial f(x)}{\partial x^T} \text{ Jacobian},$$

meaning it is possible to evaluate the approximate distribution $\tilde{\pi}$ exactly. This stands in contrast to other models such as variational autoencoders and generative adversarial networks, which can similarly be used to sample a distribution, but cannot evaluate this distribution exactly. For that reason, they may not be adequate for a use inside of a particle filter, since they cannot generally compensate for the mismatch between the sampled distribution and the true filtering distribution.

In order to allow for efficient calculations, $f(z)$ needs to be chosen such that (i) its inverse and (ii) the determinant of its Jacobian are easy to compute. One such model was developed by Dinh et al. [2] and called *RealNVP*. It makes use of *affine coupling functions* $f(x, d; s, t) = sx_{1:d}+t$, which operate on a subset of the dimensions $1, ..., d < D$. Values in the dimensions $d + 1 : D$ are kept constant. It is trivial to invert, and its Jacobian is

$$J(x) = \begin{pmatrix} I_d & 0 \\ \frac{\mathrm{d}f(x_{d+1:D})}{\mathrm{d}x_{1:d}^T} & D(\exp(s)). \end{pmatrix}$$

This is a triangular matrix, meaning its determinant can be efficiently computed as $\exp(\mathbb{1}s)$. The network is thus trained to output s and t for each of its layers. Note that since not all dimensions are modified by any single coupling function, they are made to alternate between affecting dimensions $1 : d_1$ and $d_2 + 1 : D$.

One could imagine the latent distribution d to be an elastic balloon with some kind of pattern on it wrapped around a ball. In this analogy, the NF creates a more intricate

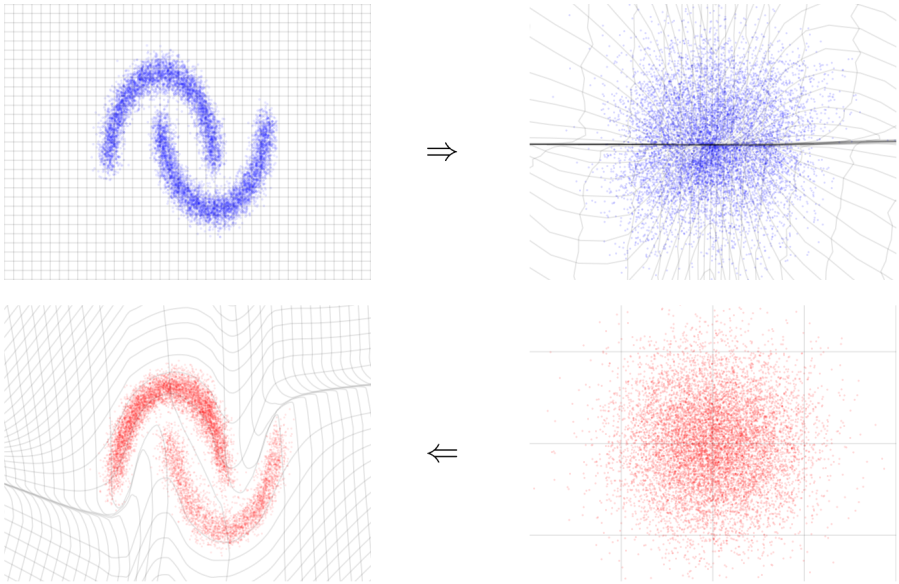


Figure B.1: Learned invertible mapping between π and d. Image from Dinh et al. [2].

geometric shape that the balloon is wrapped over. The pattern gets stretched and displaced on this new geometric shape, but a point on the ball still corresponds to exactly one point on the geometric shape, and vice versa. A 2D example of this is given in Figure B.1.

B.2 Recurrent Neural Network

MLPs are unidirectional in forward propagation, where each neuron feeds into the next neuron layer from left to right, and have a fixed input size. This is called a *feed-forward network*. A generalization of feed-forward networks allows for any node to feed into any other node. These are called *recurrent neural networks* (RNN). A feature of RNNs is that they allow processing of information several times using the same weights by creating a loop containing the node in the neuron architecture. An schematic illustration is given in Figure B.2.

RNNs are especially useful in our setting with a start state $\mathbf{x}_t$ and observations $\mathbf{y}_{1:t+1}$ [15]. In this way, we are flexible with respect to the amount of observations which may vary in real-world applications. Furthermore, using the same network to propagate the hidden states makes sense for physical systems and allows for better generalizations, as opposed to treating every timestep t differently.

A common problem with RNNs is their training difficulty [15]. In fact, updating RNN weights require more calculations than their feed-forward counterparts. Furthermore, RNNs are often forced to perform the same calculations many times within the same training epoch, for example recurring over different overlapping subsets of the same time series.

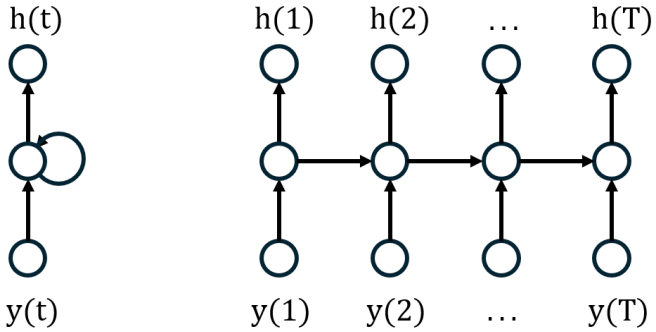
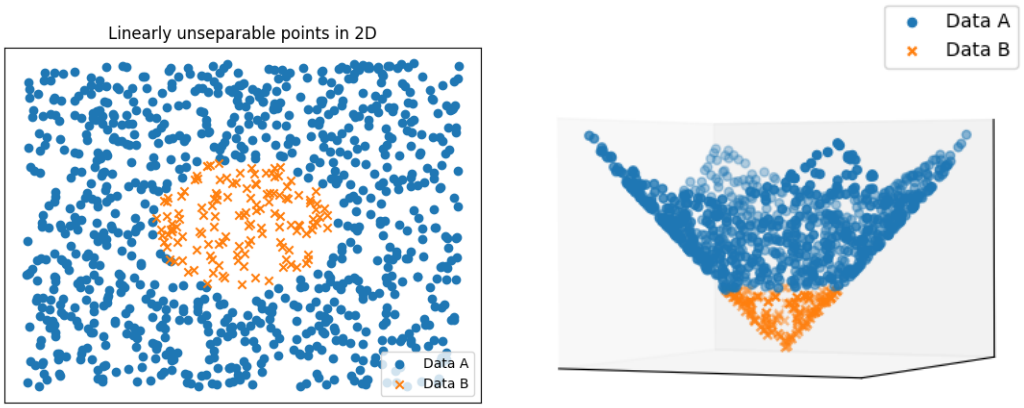


Figure B.2: Left: recurrent network with one input terminal, one hidden neuron, and one output neuron. Right: same network but unfolded in time. Figure adapted from [10].

B.2.1 Random reservoirs

A solution to this problem is found in random reservoirs, which have yielded good results in predicting chaotic dynamics [10]. They encode the input to a high dimension, but only the output layer is ever trained [15]. The idea behind random reservoirs is to randomly encode the input and training the output layer to decode it. By making the hidden state have a high dimension, we hope to be able to linearly separate a greater amount of important information from the input, which has been nonlinearly mixed up with irrelevant information [15]. For an example, see Figure B.3.



(a) Data in 2D. **(b)** Data in 3D.

Figure B.3: Data nonlinearly mixed up with irrelevant information. The data can only be linearly separated if projected to a higher dimension.

As the weights of the reservoirs are completely random, reservoirs are at risk of becoming unstable. Necessary criteria to ensure that the output of the network does not blow up are given by [10], but we do not cover them in this thesis.

B.3 Network architectures and training hyperparameters

The following tables detail the specific hyperparameter configurations utilized for both the evaluator networks (φ-networks) and the conditional NF. The MLP architectures was trained and evaluated across both the linear moth and nonlinear plane tracking scenarios, whereas the reservoirs were not.

B.3.1 potential network configurations

To solve the full-bridge problem, dual Multi-Layer Perceptrons were trained via an alternating optimization procedure to approximate the potentials. Both networks share the same structural configuration. The training scheme was configured to alternate the frozen network every 5 epochs. Between the evaluation scenarios, the only architectural variation is the inclusion of the base coordinates when tracking the nonlinear plane model.

Table B.1: Hyperparameter configurations for the φ-network.

Hyperparameter	**φ-network**
State Dimension	2
Observation Dimension	2
Hidden Dimension	64
Previous Observations (L)	{0,1,2,3}
Base Coordinates Included	Yes (Plane) / No (Moth)
Batch Size	32
Optimizer	Adam
Initial Learning Rate	1×10^{-3}
Gradient Clipping (Max Norm)	1.0
LR Scheduler	ReduceLROnPlateau
Scheduler Factor / Patience	0.5 / 15 epochs
Training Epochs	400

B.3.2 Normalizing flow configurations

The NFs were constructed using conditional RealNVP layers. Similar to the evaluator networks, the NFs include the base coordinates as an additional structural input when evaluating the nonlinear plane model. Furthermore, the base conditioning dimension natively increases for the random reservoir variant, as it concatenates the standard condition variables (size 5) directly with the pre-computed hidden state vector (size 64) extracted from the random reservoir.

Table B.2: Hyperparameter configurations for the conditional Normalizing Flows.

Hyperparameter	**NF**
State Dimension	2
Conditioning Dimension	5 or 69 (5 + 64 hidden)
Base Coordinates Included	Yes (Plane) / No (Moth)
Number of Flow Layers	12
Batch Size	32
Optimizer	Adam
Initial Learning Rate	1×10^{-3}
LR Scheduler	ReduceLROnPlateau
Scheduler Factor / Patience	0.5 / 15 epochs
Training Epochs	200
Input State Jitter	Normal Noise ($\sigma = 0.2$)

B.3.3 Specific Training Mechanics

Beyond the standard hyperparameters, a few specific data-handling and optimization techniques were applied during the Normalizing Flow training to stabilize convergence:

- **Dynamic KL loss weighting:** The reverse KL divergence loss was scaled using a cosine decay schedule during the first 50 epochs. The weighting factor started at 50 and smoothly decayed to 1, where it remained for the final 50 epochs.
- **Input jitter:** During the training of the NF, structural noise (jitter) was added to the prior state input points. This was sampled from a standard Gaussian and scaled by a factor of 0.2 to prevent the flow from overfitting to exact discrete particle coordinates.

C

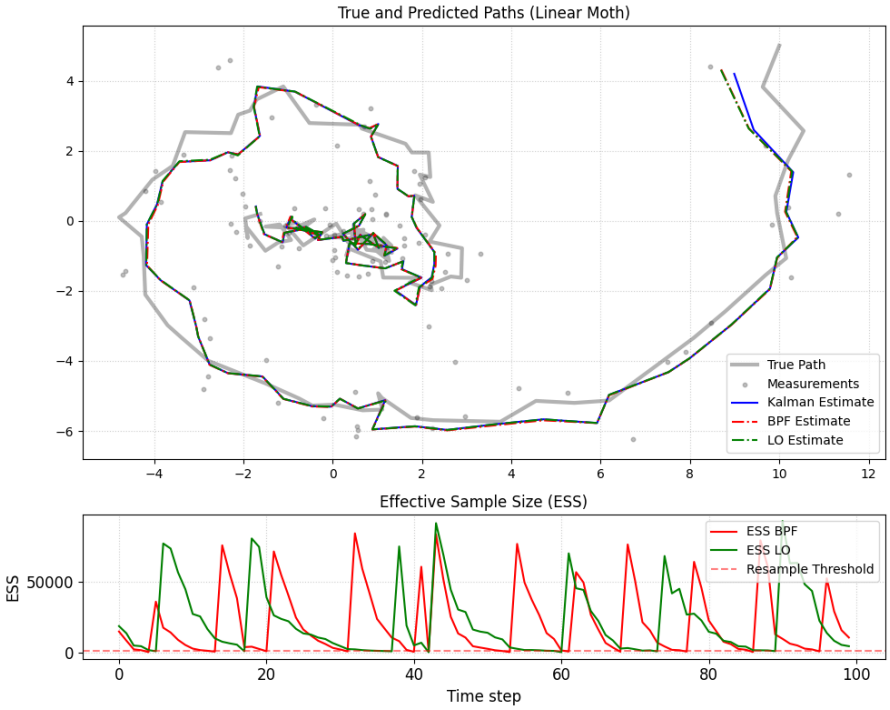
Showcase of proposal and particle filter performances in the linear scenario

In this appendix are presented examples highlighting statements made in the results about the linear moth scenario. In Section C.1 is presented the effect of varying the movement and observation noises R and Q, as well as the particle amount N, using the LO and bootstrap proposals. In Section C.2, we present the difference of using the LO, bootstrap and DSS proposals for differnt particle filter architectures: APF, MPF and AMPF.

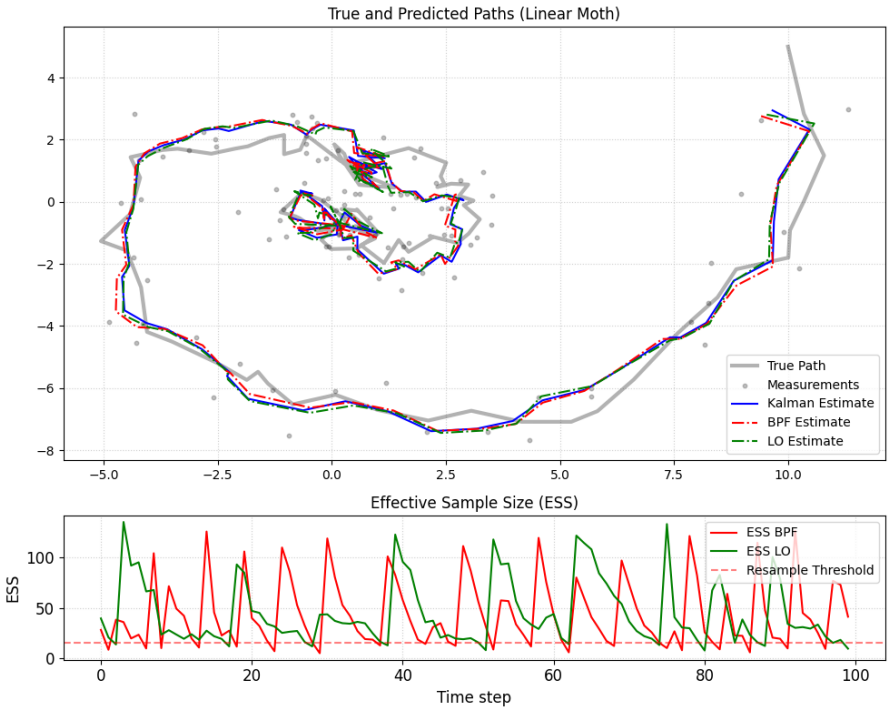
C.1 Varying scenario settings on LO and bootstrap proposals

The performance of the particle filters is affected by the exact scenario settings. Two parameters having a great affect on the results are the number of particles and the relationship between the noise parameters. Below we show the effect of changing these two when using PF(B) and PF(LO). We acknowledge that other parameters also affect the results, such as the length of the observation time steps.

In Figure C.1 is presented an example of how the number of particles affects the results. We see that using a large number of particles makes the filter align with the Kalman filter, that is they converge to the true filtering distribution. When the amount of particles is lower, however, we see that the proposals deviate from the Kalman filter.



(a) 100,000 particles.



(b) 150 particles.

Figure C.1: Comparison of PF(B) and PF(LO) performances when varying the particle amount.

When increasing the size of the movement noise and lowering that of the measurement, we see a larger difference between the performance of PF(B) and PF(LO).

In fact, the ESS of the former is very low due to the very information -rich measurements, whereas the latter is more resilient to the change. This is exemplified in Figure C.2.

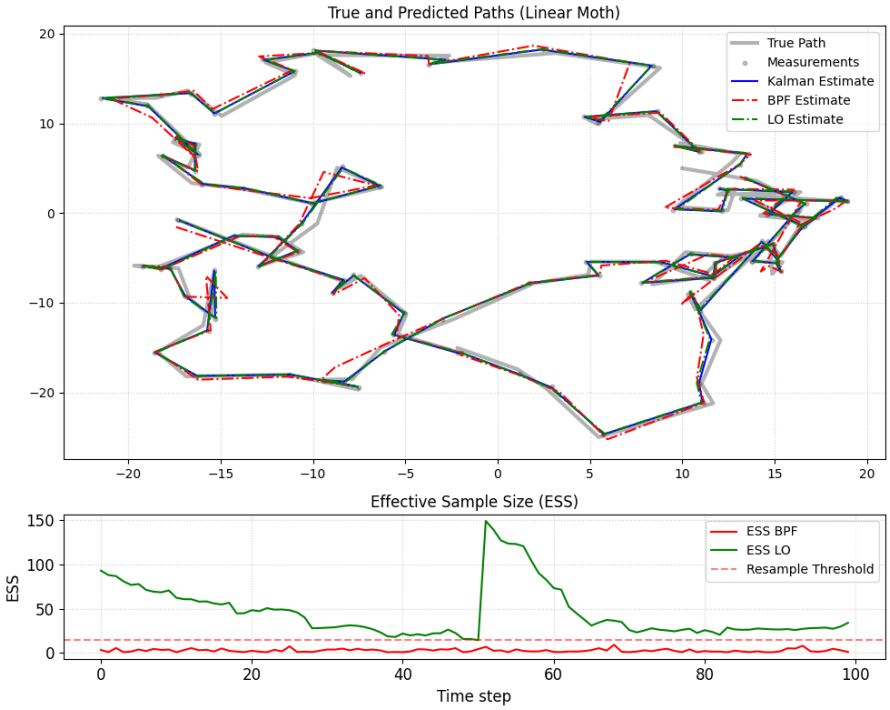


Figure C.2: Performance of the LO and bootstrap proposals for high movement noise but low measurement noise, 150 particles.

C.2 Effect of using different particle filter architectures

Below are presented the performance of the bootstrap, LO and DSS proposals in a regular particle filter (Figure C.3), an APF (Figure C.4), MPF (Figure C.5), and AMPF (Figure C.6). For simplicity, $N = 150$ and the same values for R and Q ahve been chosen as in Figure C.1. While the performances of the two auxiliary architectures seem to be similar, we see that that the MPF(DSS) has near-constant maximal ESS, whereas the MPF(B) and MPF(LO) vary much more and are significantly lower.

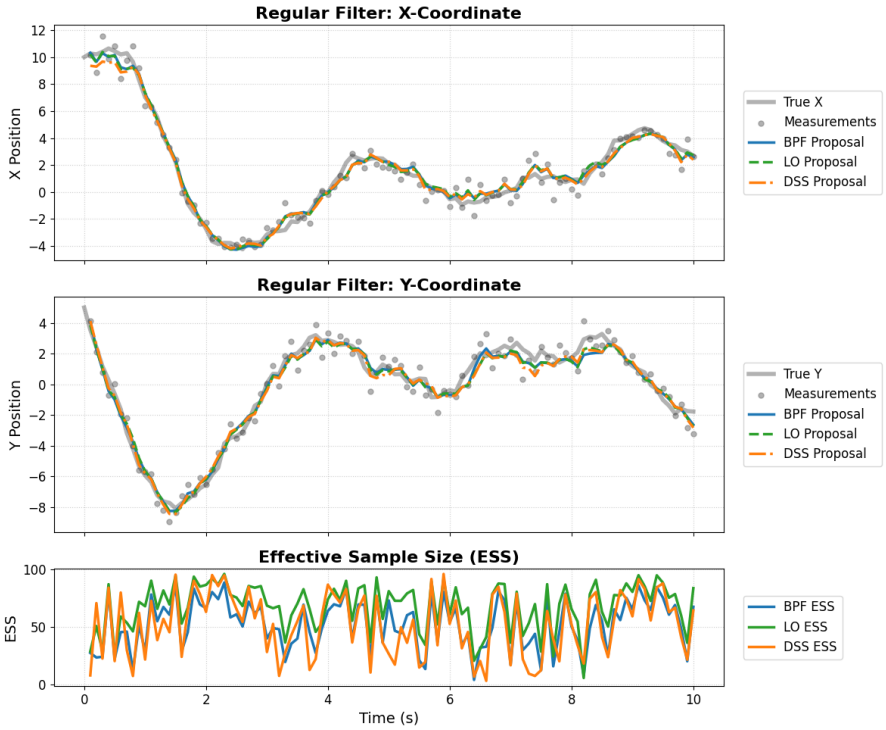


Figure C.3: PF(B), PF(LO) and PF(DSS) for 100 particles.

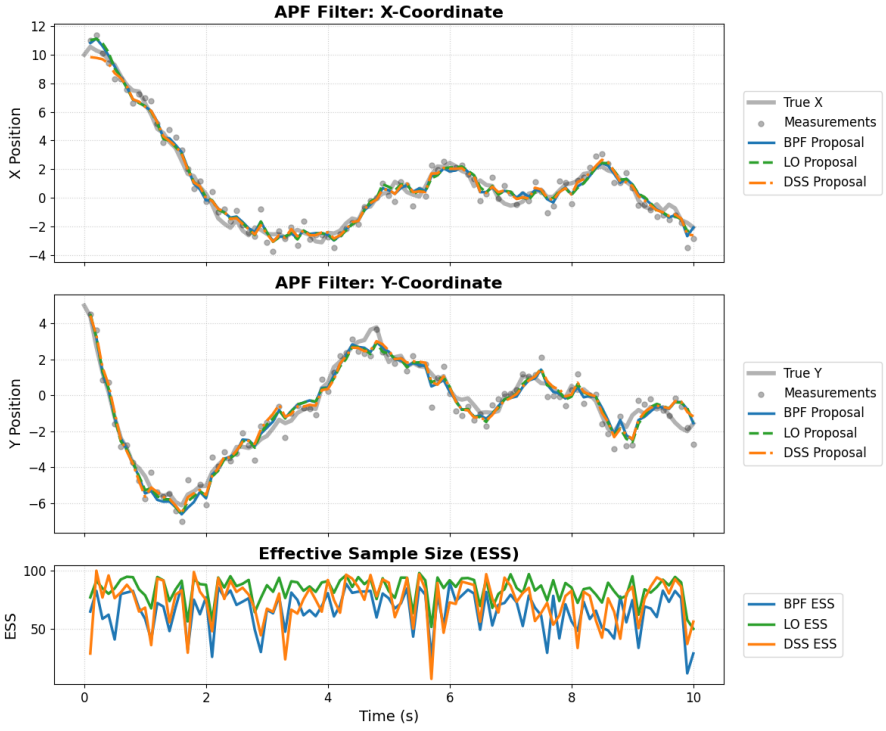


Figure C.4: APF(B), APF(LO) and APF(DSS) for 100 particles.

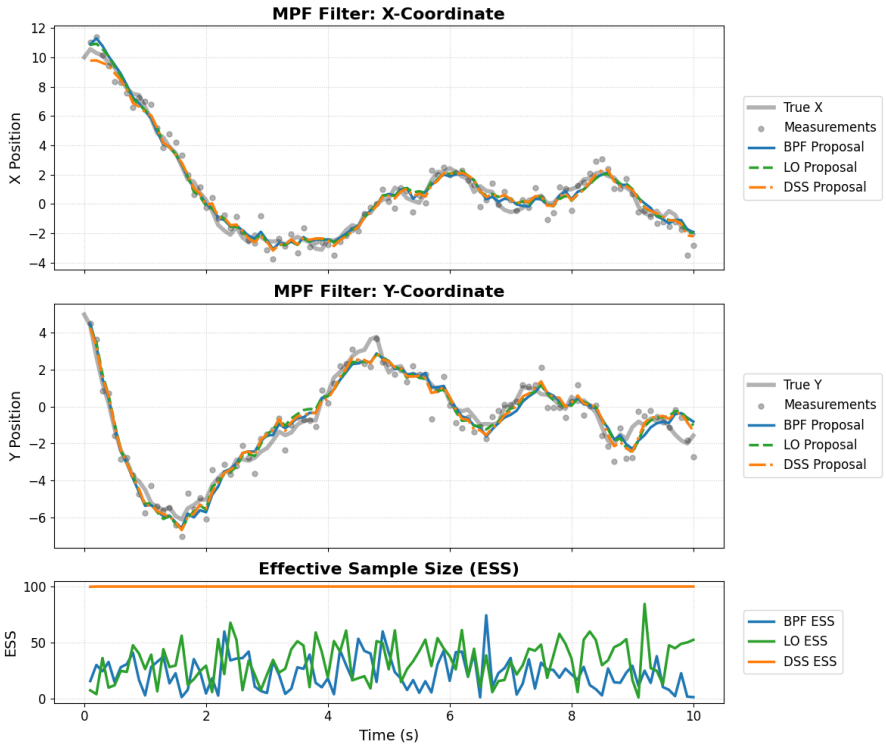


Figure C.5: MPF(B), MPF(LO) and MPF(DSS) for 100 particles.

Figure C.6: AMPF(B), AMPF(LO) and AMPF(DSS) for 100 particles.

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY