UNIVERSITY OF CALIFORNIA,
IRVINE

End-to-end Differentiable Learning of Particle Belief Propagation Algorithms

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Ali Younis

Dissertation Committee:
Professor Erik B. Sudderth, Chair
Professor Alexander Ihler
Professor Padhraic Smyth

2025

TABLE OF CONTENTS

Page

LIST OF FIGURES **v**

LIST OF TABLES **vii**

ACKNOWLEDGMENTS **viii**

VITA **ix**

ABSTRACT OF THE DISSERTATION **xi**

1 Introduction **1**
1.1 Overview of Contributions 2

2 End-to-end Learnable Particle Filters **5**
2.1 Introduction 6
2.2 Sequential State Estimation via Generative Models 7
2.2.1 Particle Filters 7
2.2.2 Regularized Particle Filters 9
2.3 Conditional State Estimation via Discriminative Particle Filters 10
2.3.1 Prior Work on Discriminative Particle Filters 12
2.4 Stable Gradient Estimation for Mixture Model Resampling 16
2.4.1 Implicit Reparameterization Gradients 16
2.4.2 Importance Weighted Samples Gradient Estimator 18
2.4.3 Instability of Implicit Reparameterization Gradients 19
2.5 Mixture Density Particle Filters 22
2.6 Experiments 26
2.6.1 Bearings Only Tracking Task 27
2.6.2 Deepmind Maze Tracking Task 29
2.6.3 House3D Tracking Task 31
2.6.4 Computation Requirements 33
2.7 Limitations 34
2.8 Discussion 34

Appendices **36**
Appendix 2.A Parameterization of the Neural Dynamics and Measurement Models 36
Appendix 2.B Additional Experiment Results 37
Appendix 2.C Experiment Details 49

3 End-to-end Learnable Particle Smoothers **60**
3.1 Introduction 61
3.2 Reduced Variance Particle Resampling 62
3.2.1 Multinomial Resampling 63
3.2.2 Stratified Resampling 63
3.2.3 Residual Resampling 64
3.2.4 Integration into Resampling from Mixture Distributions 64
3.3 From Filtering to Smoothing 65
3.3.1 Forward-Filtering, Backward Smoothing 66
3.3.2 Two Filter Smoothing 66
3.4 Mixture Density Particle Smoothers 68
3.4.1 Training Loss and Gradient Computation 71
3.4.2 72
3.4.3 Computational Requirements 72
3.5 Experiments 73
3.5.1 Bearings Only Tracking Task 73
3.5.2 City Scale Global Localization Task 75
3.5.3 Mapillary Geo-Localization (MGL) Dataset 78
3.5.4 KITTI Dataset 83
3.6 Limitations 85
3.7 Discussion 85

Appendices **86**
Appendix 3.A Additional Experiment Results 86
3.A.1 MGL Dataset Additional Results 86
3.A.2 Kitti Dataset Additional Results 87
Appendix 3.B Additional Experiment Details 89
3.B.1 Bearings Only Tracking Task 89
3.B.2 Global Localization Task with Mapillary Geo-Location Dataset and KITTI Datasets 92

4 End-to-end Learnable Particle Belief Propagation **99**
4.1 Introduction 100
4.2 Belief Propagation 102
4.2.1 Pairwise Markov Random Fields 102
4.2.2 Belief Propagation 103
4.2.3 Discrete Belief Propagation 105
4.3 Belief Propagation in Continuous Spaces 106
4.3.1 Gaussian Belief Propagation 107
4.3.2 Nonparametric Belief Propagation 107

4.3.3 Pull Message Passing Nonparametric Belief Propagation 109
4.3.4 Particle Belief Propagation 110
4.4 Towards End-to-End Learnable Belief Propagation 112
4.4.1 Differentiable Nonparametric Belief Propagation 113
4.5 Mixture Density Belief Propagation 113
4.5.1 Differentiable Resampling from the Node Marginals 114
4.5.2 Application to Dynamic Systems 116
4.5.3 Decoupling Resampling and Marginals Bandwidths 118
4.5.4 Drawing Full Graph Samples 120
4.5.5 Training Procedure 121
4.6 Experiments 121
4.6.1 Bearings Only Tracking Task 122
4.6.2 Articulated Spider Pose Estimation Task 124
4.6.3 Human Mesh Recovery from Video 130
4.6.4 Results 133
4.7 Discussion 137

5 Conclusion and Future Directions **138**
5.1 Summary of Methods and Contributions 138
5.2 Suggestions for Future Work 140
5.2.1 Mixture Density Belief Propagation for Loopy Graphs 140
5.2.2 Mixture Density Belief Propagation Multi-Person Tracking 141
5.2.3 Mixture Density Particle Filter Simultaneous Localization and Mapping using 3D Gaussian Splatting 141

Bibliography **143**

LIST OF FIGURES

Page

2.1 Generative and discriminative sequential state estimation graphical models . 7
2.2 Illustration of particle and particle weight perturbations under Implicit Reparameterization Gradients and Importance Weighted Sample Gradients regimes. 17
2.3 A simple temporal prediction problem illustrating instability of Implicit Reparameterization Gradients and benefits of Importance Weighted Sample Gradients 21
2.4 Mixture Density Particle Filter Flow Diagram 23
2.5 The Mixture Density Particle Filter algorithm 25
2.6 Bearings-Only tracking task box plot results for particle filter methods . . . 28
2.7 Bearings-Only tracking task example trajectories for particle filter methods . 29
2.8 Deepmind Maze tracking task box plot results for particle filter methods . . 30
2.9 House3D tracking task example trajectories with various (multiple mode) initialization strategies for mixture density particle filters 32
2.10 House3D tracking task example trajectories for various particle filter methods 33
2.11 Computation time plots for various particle filters methods on Deepmind Maze tracking task 33

3.1 Mixture Density Particle Smoother flow diagram 69
3.2 The Mixture Density Particle Smoother algorithm 71
3.3 Bearings-Only tracking Task box plot results for particle filter and smoother methods 74
3.4 Global localization task measurement model 79
3.5 Position and angle error recall curves on the Mapillary Geo-Localization dataset 81
3.6 Example trajectories from the Mapillary Geo-Localization dataset for mixture density particle filters and smoothers 82
3.7 Example learned dynamics from the forward filter of the mixture density particle smoother trained on the Mapillary Geo-Localization dataset 82
3.8 Recall of position and angle using the Mapillary Geo-Localization (MGL) dataset 83
3.9 Lateral and longitudinal errors are in the vehicle frame of reference. 83
3.10 Position recall using the KITTI dataset 84

4.1 Examples of Probabilistic Graphical Models 101
4.2 Message Passing in Belief Propagation 104
4.3 Example Message Passing Schedule for Belief Propagation 105

4.4 Mixture Density Belief Propagation Flow Diagram 114
4.5 The Mixture Density Belief Propagation Message Passing Algorithm 117
4.6 Probabilistic Graphical Models with System Dynamics 118
4.7 The Mixture Density Belief Propagation Algorithm 119
4.8 Probabilistic Graphical Models For Bearings-Only Tracking Task 123
4.9 Bearings-Only tracking Task box plot results for Mixture Density Belief Propagation 123
4.10 Probabilistic Graphical Model for the Articulated Spider Pose Estimation Task 125
4.11 Negative Log-Likelihood Curve for the Articulated Spider Pose Estimation Task 127
4.12 Example sequence for Mixture Density Belief Propagation on the Articulated Spider Pose Estimation Task 128
4.13 Example sequence for Mixture Density Belief Propagation on the Articulated Spider Pose Estimation Task 129
4.14 Graphical model structure for the Human Mesh Reconstruction task 132
4.15 Example sequence for the Human Mesh Recovery Task 134
4.16 Example sequence for the Human Mesh Recovery Task 135
4.17 Example sequence for the Human Mesh Recovery Task 136

5.1 Proposed framework for 3D Gaussian Splatting based Simultaneousness Localization and Mapping using Mixture Density Particle Filters. 142

LIST OF TABLES

Page

2.1 Results of various particle filter methods on the House3D tracking task . . . 31

ACKNOWLEDGMENTS

Firstly I would like to thank Erik Sudderth for being a supportive advisor over the course of my PhD journey. I would also like to thank the other members of the Learning, Inference and Vision Lab who have been wonderful to work alongside: Gabe Hope, Harry Bendekgey, Debora Sujono, Sakshi Agarwal, Federica Zoe Ricci, Mehrnaz Motamed, Daniele Micheletti, Aaron Mui and Jimin Heo.

I would also like to thank my committee members Padhraic Smyth and Alex Ihler. Padhraic was the instructor of one of my early machine learning courses during my M.S. studies. Padhraics lectures sparked my curiosity of machine learning, leading me to eventually return to UCI to pursue this PhD. Prior research by Alex was the inspiration for this PhD. I have deeply enjoyed reading all his past work and building on the solid foundation Alex has laid in the area of belief propagation.

I would like to thank my good friends Will, Kaeleigh, Nam, Courtney, Andrea and Milan for being with me every step of this PhD and making sure that I step away from my computer every once in a while.

I would like to express my deepest gratitude to my parents for their unconditional support and guidance as I complete graduate school. You have always been my greatest supporters and I would not be where I am without you.

Finally I would like to thank my dearest wife Saadia Nur. Thank you for being the calm in the chaos before conference paper submissions. Thank you for being the spark of joy during late nights debugging. Thank you for being the listening ear when I was frustrated. Thank you for being by my side throughout this PhD.

This research supported in part by NSF Robust Intelligence Award No. IIS-1816365 and ONR Award No. N00014-23-1-2712.

VITA

Ali Younis

EDUCATION

Doctor of Philosophy in Computer Science **2025**
University of California, Irvine *Irvine, California*

Masters in Computer Science **2018**
University of California, Irvine *Irvine, California*

Bachelors of Science in Computer Science and Engineering **2017**
University of California, Irvine *Irvine, California*

RESEARCH EXPERIENCE

Graduate Student Researcher **2020-2025**
University of California, Irvine *Irvine, California*

Undergraduate Research Assistant **2013-2017**
University of California, Irvine *Irvine, California*

INDUSTRY EXPERIENCE

Software Engineering Intern **Summer 2024**
Qualcomm Inc. *San Diego, California*

Software Engineering Intern **Summer 2023**
Qualcomm Inc. *San Diego, California*

Software Engineer **06/2020–10/2020**
Modal AI *San Diego, California*

Software Engineer **08/2018–06/2020**
Tyvak Nano-Satellite Systems, Inc. *Irvine, California*

Software Engineering Intern **4 Summers from 2014-2017**
Qualcomm Inc. Corporate R&D *San Diego, California*

PUBLICATIONS

A. Younis, E. Sudderth. "Learning to be Smooth: An End-to-End Differentiable Particle Smoother". *Neural Information Processing Systems (Neurips) 2024.*

A. Younis, E. Sudderth. "Differentiable and Stable Long-Range Tracking of Multiple Posterior Modes". *Neural Information Processing Systems (Neurips) 2023.*

S. Agarwal, G. Hope, **A. Younis**, E. Sudderth. "A Decoder Suffices for Query-Adaptive Variational Inference". *Uncertainty in Artificial Intelligence (UAI) 2023.*

R. Trimananda, **A. Younis**, T. KWA, B. Demsky, G. Xu. "Securing Smart Home Devices against Compromised Cloud Servers". *Poster at HotEdge, June 2020*

R. Trimananda, **A. Younis**, B. Wang, B. Xu, B. Demsky, G. Xu. "Vigilia: Securing smart home edge computing". *Symposium of Edge Computing (SEC) October 2018.*

ABSTRACT OF THE DISSERTATION

End-to-end Differentiable Learning of Particle Belief Propagation Algorithms

By

Ali Younis

Doctor of Philosophy in Computer Science

University of California, Irvine, 2025

Professor Erik B. Sudderth, Chair

Estimating the temporal state of a system from image sequences is an important task for many vision and robotics applications. A number of classical frameworks for state estimation have been proposed, but often these methods require human experts to specify the system dynamics and measurement model, requiring simplifying assumptions that hurt performance. With the increasing abundance of real-world training data, there is enormous potential to boost accuracy by using deep learning to learn state estimation algorithms, but there are also substantial technical challenges in properly accounting for uncertainty. We propose solutions to these challenges by developing end-to-end differentiable particle based solutions which can accurately model uncertainties and allow for embedding of neural networks in our algorithms.

Concretely, we first create an end-to-end learnable particle filter that uses flexible neural networks to propagate multimodal, particle-based representations of state uncertainty. Our gradient estimators are unbiased and have substantially lower variance than existing, differentiable (but biased) particle filters. We apply our end-to-end learnable particle filter to the difficult task of visual localization in unknown environments, and show large improvements over prior work. We then expand on our particle filtering method to create the first end-to-end learnable particle smoother, which incorporates information from fu-

ture as well as past observations, and apply this particle smoother to the real-world task of city-scale geo-localization using camera and planimetric map data. We compare to state-of-the-art baselines for visual geo-localization, and again show superior performance. Finally, we develop an end-to-end learnable particle belief propagation algorithm for inference on tree-structured graphical models, where the state of each node evolves over time based on unknown system dynamics. A key application of differentiable particle belief propagation is learning to estimate the articulated pose of dynamic human bodies.

Chapter 1

Introduction

Many tasks in computer vision and robotics require estimating the state of a dynamic system as it evolves through time. Examples of such tasks include object tracking, localization, and articulated pose estimation. Often these systems are highly non-linear with complex observation spaces that are high-dimensional and difficult to use in their raw forms (such as image data). In the past, solving these difficult tasks required a human expert to handcraft models which capture the complex dynamics of the system and model the observation generation process well. Unfortunately, many tasks are too complex to write down concrete useful equations for and thus simplified versions of the true dynamics and measurement models are used, leading to worse performing models.

In the era of big data we have access to large swaths of labeled data, observations with ground truth labels, with more and more datasets being released everyday. Further with the advent of deep learning, where flexible neural networks are used to model complex functions, and extremely fast parallel computing on graphics processing units (GPU), solving these complex robotics and computer vision tasks without a human expert becomes possible. Models can instead be learnable neural networks, trained on large datasets able to produce accurate

predictions for even the hardest of tasks. By feeding the neural networks the observation and the ground truth labels, these models can learn complex relationships between what they see and what they should predict. Indeed recent trends in deep learning have been to design bigger and bigger models which ingest ever growing amounts of data to produce models with amazing accuracy.

Though solving state estimation tasks for dynamic systems is possible using big neural network models alone, blending classical state estimation techniques with deep learning can offer additional benefits due to the implicit knowledge encoded in the classical methods. More specifically if we can represent the task as a graphical model, encoding conditional independence and local state interactions, then we can apply a wide range of classic filtering, smoothing and belief propagation methods to solve these tasks. Further if we are able to make these method differentiable, we can use deep learning within these classic frameworks, creating inference methods that blend the best of well studied graphical model inference algorithms with the latest advancements in neural design, achieving great accuracy improvements.

1.1 Overview of Contributions

In this dissertation we explore the blending of classical particle filtering, particle smoothing and belief propagation techniques with neural networks to produce learnable inference methods on a variety of graphical models. Applying neural networks to these classic inference methods brings them into age of deep learning and showcases how classical methods are still relevant and useful. Further, making these inference methods end-to-end learnable from data enables their application to a broad range of difficult to model problems which we explore in more detail in subsequent chapters.

We summarize the contributions of each chapter below.

Chapter 2: End-to-End Learnable Particle Filters

In this chapter, we explore end-to-end learnable particle filtering (PF) methods. We begin with an overview of discriminative particle filtering, which underpins many recent advances in differentiable particle filtering. We then examine existing differentiable PFs, identifying several key limitations. Next, we analyze *Implicit Reparameterization Gradients*:w as a technique for differentiable resampling from mixture distributions, revealing its inherent gradient instability when the mixture being sampled has multiple distinct modes. To overcome this gradient instability we introduce our novel *Importance Weighted Sample Gradients* method for computing unbiased and stable gradients for samples from a mixture distribution with respect to the mixture parameters. Using importance weighted sample gradients we construct a novel unbiased end-to-end differentiable particle filter method which we name *Mixture Density Particle Filter* and showcase its superior performance on a variety of tracking tasks when compared to other differentiable particle filter methods.

Chapter 3: End-to-End Learnable Particle Smoothers

We begin this chapter by exploring several reduced-variance particle resampling techniques and how they can be integrated into the resampling step of our mixture density particle filter from chapter 2. Next we explore moving from particle filters to particle smoothers and introduce two variants of classical particle smoothing. Building on the classical Two-Filter smoother, we develop the first, to the best of our knowledge, end-to-end differentiable particle smoothing technique, which we call *Mixture Density Particle Smoothing*. Our mixture density particle smoother borrows several components from our mixture density particle filter as well as our importance weighted sample gradients method from chapter 2 to pro-

duce a smoother with orders of magnitude performance improvements over classical particle smoother methods. Finally we showcase the superior performance of our mixture density particle smoothers on the difficult task of city-scale global localization when compared to a variety of methods specifically designed for this task.

Chapter 4: End-to-End Learnable Particle Belief Propagation

In this chapter we tackle end-to-end learnable belief propagation (BP) for tree-structured graphical models and their use in complex articulated pose of dynamic bodies problems. Starting from the classical sum-product algorithm for inference in discrete state spaces, we quickly move onto particle-based belief propagation inference methods for high dimensional continuous spaces. Borrowing ideas from the classical nonparametric BP and particle BP methods, we propose a novel end-to-end differentiable belief propagation method which we call *Mixture Density Belief Propagation*. We showcase our methods' superior performance when compared to earlier works in differentiable nonparametric BP, as well as the application of our method to the complex task of articulated pose estimation for dynamic human bodies from video sequences.

Chapter 2

End-to-end Learnable Particle Filters

In this chapter we explore particle filters [1, 2, 3, 4] and various techniques to make them end-to-end differentiable, i.e. able to end-to-end train their internal dynamics and measurement models from data. Looking at several existing works in end-to-end differentiable particle filters [5, 6, 7, 8], we demonstrate various deficiencies in existing methods from bias in the produced gradients to poor allocation of particles. Later we look at methods for differentiable resampling from mixture distributions, namely Implicit Reparameterization Gradients (IRG) [9, 10], and showcase its dramatic failures when resampling from mixtures with multiple disconnected modes. We also develop a novel unbiased differentiable resampling technique which we call Importance Weighted Sample Gradients (IWSG) which produces stable gradients. Finally we integrate IWSG into particle filtering to produce our novel Mixture Density Particle Filter (MDPF) and showcase superior performance when compared to existing works.

2.1 Introduction

A *particle filter* (PF) [1, 2, 3, 4] uses weighted samples to provide a flexible, nonparametric representation of uncertainty in continuous latent states. Classical PFs are generative models that typically require human experts to specify the latent state dynamics and measurement, often making simplifying assumptions about the true dynamics and measurement models in order to create tractable models. Such generative models may be inaccurate or even unavailable for high-dimensional observations like images.

With the growing popularity of deep learning and abundance of time-series data, recent work has instead sought to discriminatively learn particle filter variants that (unlike conventional recurrent neural networks) capture uncertainty in latent states [5, 6, 7, 8]. These approaches are especially promising for high-dimensional observations like images, where learning accurate generative models is extremely challenging.

The key challenge in learning discriminative PFs is the non-differentiable particle resampling step, which is key to robustly maintaining diverse particle representations, but inhibits end-to-end learning by preventing gradient propagation. Prior discriminative PFs have typically used heuristic relaxations of discrete particle resampling, or biased learning by truncating gradients at resampling steps. These methods severely compromise the effectiveness of PF training; prior work has often assumed exact dynamical models are known, or used very large numbers of particles for test data, due to these limitations.

In this chapter we discuss existing differentiable particle filtering methods and highlight there drawbacks and deficiencies. We also demonstrate dramatic failures of popular reparameterization-based gradient estimators for mixture models. This motivates our *importance weighted samples gradient* estimator, which provides the foundation for the *mixture density particle filter*, our unbiased end-to-end differentiable particle filter method.

2.2 Sequential State Estimation via Generative Models

Algorithms for sequential state estimation compute or approximate the posterior distribution of the system state x_t, given a sequence of observations y_t and (optionally) input actions a_t, at discrete times $t = 1, \ldots, T$. Sequential state estimation is classically formulated as inference in a generative model like the *hidden Markov model* (HMM) or state space model of Fig. 2.1, with dynamics defined by state transition probabilities $p(x_t \mid x_{t-1}, a_t)$, and observations generated via likelihoods $p(y_t \mid x_t)$.

Given a known Markov prior on latent state sequences, and a corresponding observation sequence, the *filtered* state posterior $p(x_t \mid y_t, y_{t-1}, \ldots, y_1)$ can in principle be computed via Bayesian inference. When the means of state dynamics and observations likelihoods are linear functions, and noise is Gaussian, exact filtered state posteriors are Gaussian and may be efficiently computed via the *Kalman filter* (KF) [11, 12, 13]. The non-Gaussian state posteriors of general state space models may be approximated via Gaussians [14], but estimating the posterior mean and covariance is in general challenging, and cannot faithfully capture the multimodal posteriors produced by missing or ambiguous data.

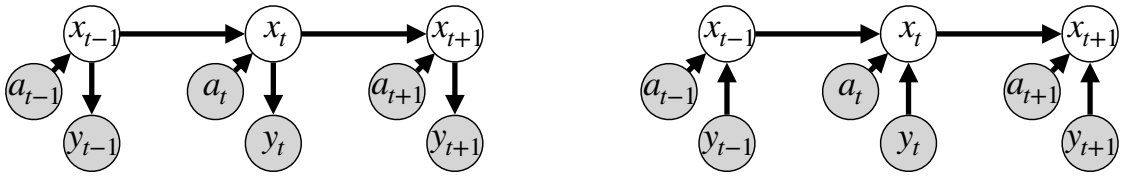


Figure 2.1: Sequential state estimation may be formulated via either generative (*left*) or discriminative (*right*) graphical models. In either case, dynamics of states x_t are influenced by actions a_t, and estimated via data y_t.

2.2.1 Particle Filters

To flexibly approximate general state posteriors, particle filters (PF) [1, 2, 3, 4] represent possible states nonparametrically via a collection of weighted samples or *particles*. The classical

PF parameterizes the state posterior at time t by a set of N particles $x_t^{(:)} = \{x_t^{(1)}, \ldots, x_t^{(N)}\}$ with associated weights $w_t^{(:)} = \{w_t^{(1)}, \ldots, w_t^{(N)}\}$. PFs recursively update the state posterior given the latest observation y_t and action a_t, and may be flexibly applied to a broad range of models; they only assume that the state dynamics may be simulated, and the observation likelihood may be evaluated.

Particle Proposal. To produce particle locations $x_t^{(:)}$ at time t, a model of the state transition dynamics is applied individually to each particle $x_{t-1}^{(i)}$, conditioned on external actions a_t if available:

$$x_t^{(i)} \sim p(x_t \mid x_{t-1} = x_{t-1}^{(i)}, a_t). \tag{2.1}$$

Measurement Update. After a new particle set $x_t^{(:)}$ has been proposed, particle weights must be updated to account for the latest observation y_t via the known likelihood function:

$$w_t^{(i)} \propto p(y_t \mid x_t^{(i)}) \cdot w_{t-1}^{(i)}. \tag{2.2}$$

The updated weights are then normalized so that $\sum_{i=1}^{N} w_t^{(i)} = 1$. This weight update is motivated by importance sampling principles, because (asymptotically, as $N \to \infty$) it provides an unbiased approximation of the true state posterior $p(x_t \mid y_t, \ldots, y_1) \propto p(y_t \mid x_t) \cdot p(x_t \mid y_{t-1}, \ldots, y_1)$

Particle Resampling. Due to the stochastic nature of PFs, over time particles will slowly diverge to regions with low posterior probability, and will thus be given little weight in the measurement step. If all but a few particles have negligible weight, the diversity and effective representational power of the particle set is diminished, resulting in poor estimates of the state posterior.

To address this, particle resampling is used to maintain diversity of the particle set over time,

and may be performed at every iteration or more selectively when the "effective" sample size becomes too small [15]. During resampling, a new uniformly-weighted particle set $\hat{x}_t^{(:)}$ is constructed by discrete resampling with replacement. Each resampled particle is a copy of some existing particle, where copies are sampled with probability proportional to the particle weights:

$$\hat{x}_t^{(i)} = x_t^{(j)}, \qquad j \sim \text{Cat}(w_t^{(1)}, \ldots, w_t^{(N)}). \tag{2.3}$$

After resampling, assigning uniform weights $\hat{w}_t^{(i)} = \frac{1}{N}$ maintains an unbiased approximate state posterior. This discrete resampling is non-differentiable, preventing gradient estimation via standard reparameterization [16, 17, 18], and inhibiting end-to-end learning of dynamics and measurement models.

2.2.2 Regularized Particle Filters

The regularized PF [19, 20] acknowledges that a small set of discrete samples will never *exactly* align with the true continuous state, and thus estimates a continuous state density $m(x_t \mid x_t^{(:)}, w_t^{(:)}, \beta)$ by convolving particles with a continuous kernel function K with bandwidth β:

$$m(x_t \mid x_t^{(:)}, w_t^{(:)}, \beta) = \sum_{i=1}^{N} w_t^{(i)} \cdot K(x_t - x_t^{(i)}; \beta). \tag{2.4}$$

The kernel function could be a Gaussian, in which case β is a (dimension-specific) standard deviation. The extensive literature on *kernel density estimation* (KDE) [21] provides

theoretical guidance on the choice of kernel. Quadratic Epanechnikov kernels take

$$K(u;\beta) = \begin{cases} \frac{3}{4}\left(1-(u/\beta)^2\right), & \text{if} |u/\beta| \leq 1 \\ 0, & \text{otherwise} \end{cases} \tag{2.5}$$

otherwise, and asymptotically minimize the mean-squared-error in the estimation of the underlying continuous density. A number of methods have been proposed for selecting the smoothing bandwidth β [21, 22, 23], but they often have asymptotic justifications, and can be unreliable for small N.

Attractively, regularized particle filters resample particles $\hat{x}_t^{(i)} \sim m(x_t \mid x_t^{(:)}, w_t^{(:)}, \beta)$ from the continuous mixture density rather than the discrete particle set. By resampling from a continuous distribution, regularized particle filters ensure that no duplicates exist in the resampled particle set, increasing diversity while still concentrating particles in regions of the state space with high posterior probability. Our proposed *mixture density particle filter* generalizes regularized particle filters, using bandwidths β that are tuned jointly with discriminative models of the state dynamics and observation likelihoods.

2.3 Conditional State Estimation via Discriminative Particle Filters

While classical inference algorithms like kalman filters and particle filters are effective in many domains, they require faithful generative models. A human expert must typically design most aspects of the state dynamics and observation likelihoods, and algorithms that use PFs to learn generative models often struggle to scale beyond low-dimensional, parametric models [24]. For complex systems with high-dimensional observations like images, learning generative models of the observation likelihoods is often impractical, and arguably more

challenging than estimating latent states given an observed data sequence. We instead learn *discriminative* models of the distribution of the state *conditioned* on observations (see Fig. 2.1), and replace manually engineered generative models with deep neural networks learned from training sequences with (potentially sparse) latent state observations.

Discriminative Kalman Filters. Discriminative modeling has been used to learn conditional variants of the KF [25, 26] where the posterior is parameterized by a Gaussian state space model, and state transition and observation emission models are produced from data by trained neural networks. Like *conditional random field* (CRF) [27] models for discrete data, discriminative KFs do not learn likelihoods of observations, but instead condition on them. Discriminative KFs desirably learn a posterior covariance, and thus do not simply output a state prediction like conventional recurrent neural networks. But, they are limited to unimodal, Gaussian approximations of state uncertainty.

Discriminative Particle Filters. It is attractive to integrate similar discriminative learning principles with PFs, but the technical challenges are substantially greater due to the nonparametric particle representation of posterior uncertainty, and the need for particle resampling to maintain diversity.

Like in generative PFs, discriminative PFs update the particle locations using a *dynamics model* as in Eq. (2.1). Unlike classic PFs, the particle weights are not updated using a generative likelihood function. Instead, discriminative PFs compute new weights $w_t^{(\cdot)}$ using a *measurement model* function $\ell(x_t; y_t)$ trained to properly account for uncertainty in the discriminative particle posterior:

$$w_t^{(i)} \propto \ell(x_t^{(i)}; y_t) \cdot w_{t-1}^{(i)}. \tag{2.6}$$

Here $\ell(x_t; y_t)$ is a (differentiable) function optimized to improve the accuracy of the discriminative particle posterior, rather than a generative likelihood.

Creating a learnable discriminative PF requires parameterizing the dynamics and measurement models as differentiable functions, such as deep neural networks. This is straightforward for the measurement model $\ell(x_t^{(i)}; y_t)$, which may be defined via any feed-forward neural network architecture like those typically used for classification (see Fig. 2.4). For the dynamics model, the neural network does not simply need to score particles; it must be used for stochastic simulation. Using reparameterization [16, 17, 18], dynamics simulation is decomposed as sampling from a standard Gaussian distribution, and then transforming that sample using a learned neural network, possibly conditioned on action a_t:

$$x_t^{(i)} = f(\eta_t^{(i)}; x_{t-1}^{(i)}, a_t), \qquad \eta_t^{(i)} \sim N(0, I). \tag{2.7}$$

The dynamics model $f(\eta; x_{t-1}, a_t)$ is a feed-forward neural network that deterministically processes noise η, conditioned on x_{t-1} and a_t, to implement the dynamical sampling of Eq. (2.1). The neural network f may be flexibly parameterized because discriminative particle filters only require simulation of the dynamical model, not explicit evaluation of the implied conditional state density.

2.3.1 Prior Work on Discriminative Particle Filters

Specialized, heuristic variants of discriminative PFs have been previously used for tasks like robot localization [28] and multi-object tracking [29]. Principled end-to-end learning of a discriminative PF requires propagating gradients through the PF algorithm, including the discrete resampling step. Zhu et al. [30] explore replacing the PF resampling step with a learned (deterministic) particle transform, but find that exploding gradient magnitudes prohibit end-to-end training.

Truncated-Gradient Particle Filter (TG-PF). The first so-called "differentiable" particle filter [5] actually treated the particle resampling step as a non-differentiable discrete

resampling operation, and simply truncated all gradients to zero when backpropagating through this resampling. Though simple, this approach leads to biased gradients, and often produces ineffective models because *back-propagation through time* (BPTT [31]) is not possible. Perhaps due to these limitations, experiments in Jonschkowski et al. [5] assumed a simplified training scenario where the ground-truth dynamics are known, and only the measurement model must be learned.

Discrete Importance Sampling Particle Filter (DIS-PF). Ścibior et al. [6] propose a gradient estimator for generative PFs that may also be adapted to discriminative PFs. They develop their estimator by invoking prior work on score-function gradient estimators [32], but we provide a simpler derivation via importance sampling principles. Instead of discretely resampling particles as in Eq. (2.3), a separate set of weights $v_t^{(:)}$ is defined and used to generate samples:

$$\hat{x}_t^{(i)} = x_t^{(j)}, \qquad j \sim \mathrm{Cat}(v_t^{(1)}, \ldots, v_t^{(N)}). \tag{2.8}$$

To account for discrepancies between these resampling weights $v^{(:)}$ and true weights $w^{(:)}$, the resampled particle weights $\hat{w}_t^{(i)}$ are defined via importance sampling. The choice of $v_t^{(:)}$ critically impacts the effectiveness of the resampling step. To maintain the standard PF update of Eq. (2.3), we set $v_t^{(i)} = w_t^{(i)}$, yielding the following resampled particle weights and associated gradients:

$$\hat{w}_t^{(i)} = \frac{w_t^{(i)}}{v_t^{(i)}\big|_{v_t^{(i)}=w_t^{(i)}}} = 1, \qquad \nabla_\phi \hat{w}_t^{(i)} = \frac{\nabla_\phi w_t^{(i)}}{v_t^{(i)}\big|_{v_t^{(i)}=w_t^{(i)}}}. \tag{2.9}$$

Intuitively, particle locations are resampled according to the current weights in the forward pass of DIS-PF. Then when computing gradients, perturbations of the observation and dynamics models are accounted for by changes in the associated particle weights.

A drawback of this discrete importance sampling is known as the *ancestor problem*: since

the resampled particle set is constructed from a subset of the pre-resampled particles, no direct gradients exist for particles that were not chosen during resampling; they are only indirectly influenced by the weight normalization step. This increases the DIS gradient estimator variance, slowing training.

Soft Resampling Particle Filter (SR-PF). Karkus et al. [7] propose a *soft-resampling* (SR) mixing of the particle weights with a discrete uniform distribution before resampling: $v_t^{(i)} = (1-\lambda)w_t^{(i)} + \lambda/N$. Viewing particle locations as fixed, SR-PF evaluates the resampled particle weights and gradients as

$$\hat{w}_t^{(i)} = \frac{w_t^{(i)}}{(1-\lambda)w_t^{(i)} + \lambda/N}, \qquad \nabla_\phi \hat{w}_t^{(i)} = \nabla_\phi \left(\frac{w_t^{(i)}}{(1-\lambda)w_t^{(i)} + \lambda/N} \right). \tag{2.10}$$

While the SR-PF weight update is differentiable, it does not avoid the ancestor problem. By resampling low-weight particles more frequently, SR-PF will degrade overall PF performance when N is not large. More subtly, the gradient of Eq. (2.10) has (potentially substantial) bias: it assumes that perturbations of model parameters influence particle weights, but *not* the outcome of discrete resampling (2.8). Indeed in some experiments in [7], smoothing is actually disabled by setting $\lambda = 0$.

Concrete Particle Filter (C-PF). The Gumbel-softmax or Concrete distribution [33, 34] approximates discrete sampling by interpolation with a continuous distribution, enabling reparameterized gradient estimation. Each resampled particle is a convex combination of the weighted particle set:

$$\hat{x}_t^{(i)} = \sum_{j=1}^{N} \alpha_{ij} x_t^{(j)}, \qquad \alpha_{ij} = \frac{\exp((\log(w_t^{(j)}) + G_{ij})/\lambda)}{\sum_{k=1}^{N} \exp((\log(w_t^{(k)}) + G_{ik})/\lambda)}, \qquad G_{ij} \sim \text{Gumbel}. \tag{2.11}$$

Gradients of (2.11) are biased with respect to discrete resampling due to the non-learned "temperature" hyperparameter $\lambda > 0$. When λ is large, the highly-biased relaxation will interpolate between modes, and produce many low-probability particles. Bias decreases as

$\lambda \to 0$, but in this limit the variance is huge. Even with careful tuning of λ, Concrete relaxations are most effective for small N. Maddison et al. [34] focus on models with binary latent variables, and find that performance degrades even for $N = 8$, far smaller than the N needed for practical PFs. While we provide C-PF as a baseline, we are unaware of prior work successfully incorporating Concrete relaxations in PFs.

Optimal Transport Particle Filter (OT-PF). OT-PF [8] modifies PFs by replacing stochastic resampling with the optimization-based solution of an entropy-regularized *optimal transport* (OT) problem. OT seeks a probabilistic correspondence between particles $x_t^{(:)}$ with weights $w_t^{(:)}$ as in (2.6), and a corresponding set of uniformly weighted particles, by minimizing a Wasserstein metric $\mathcal{W}_{2,\lambda}^2$:

$$\min_{\tilde{\alpha}\in[0,1]^{N\times N}} \sum_{i,j=1}^{N} \tilde{\alpha}_{ij}\left(||x_t^{(i)} - x_t^{(j)}||^2 + \lambda \log \frac{\tilde{\alpha}_{ij}}{N^{-1} w_t^{(j)}}\right), \text{ s.t. } \sum_{j=1}^{N} \tilde{\alpha}_{ij} = \frac{1}{N}, \sum_{i=1}^{N} \tilde{\alpha}_{ij} = w_t^{(j)}. \quad (2.12)$$

Entropy regularization is required for differentiability OT-PF uses this entropy-regularized mapping to interpolate between particles: $\hat{x}_t^{(i)} = \sum_{j=1}^{N} N\tilde{\alpha}_{ij} x_t^{(j)}$. This assignment approximates the results of true stochastic resampling in the limit as $N \to \infty$, but lacks accuracy guarantees for moderate N.

OT-PF solves a *regularized* OT problem which relaxes discrete resampling, producing biased gradients that are reminiscent (but distinct) from C-PF. Minimization of Eq. (2.12) via the Sinkhorn algorithm [35] requires $\mathcal{O}(N^2)$ operations. This OT problem must be solved at each step of both training and test, and in practice, OT-PF is substantially slower than all competing discriminative PFs (see chapter 2.6.4). Further OT-PF accuracy is highly sensitive to the entropy regularization parameter $\lambda > 0$, which cannot be optimized; it must be set via an expensive hyperparameter search.

2.4 Stable Gradient Estimation for Mixture Model Resampling

Training our discriminative *mixture density particle filter* (MDPF) requires unbiased and low-variance estimates of gradients of samples from continuous mixtures. Mixture sampling is classically decomposed into discrete (selecting a mixture component) and continuous (sampling from that component) steps. The Gumbel-softmax or Concrete distribution [33, 34] provides a reparameterizable relaxation of discrete resampling, but may have substantial bias. Some work has also applied OT methods for gradient estimation in restricted families of Gaussian mixtures [36, 37]. We instead develop importance-sampling estimators that are unbiased, computationally efficient, and low variance.

2.4.1 Implicit Reparameterization Gradients

Direct reparameterization of samples, as used for the latent Gaussian distributions in variational autoencoders [16, 17, 18], cannot be easily applied to mixture models. Reparameterized sampling requires an invertible standardization function $S_\phi(z) = \epsilon$ that transforms a sample z, drawn from a distribution parameterized by ϕ, to an auxiliary variable ϵ independent of ϕ. For mixture distributions, the inverse of $S_\phi(z)$ cannot be easily computed.

The *implicit reparameterization gradients* (IRG) estimator [9, 10] avoids explicit inversion of the standardization function $S_\phi(z)$. Using implicit differentiation, reparameterization can be achieved so long as a computable (analytically or numerically) and invertible $S_\phi(z)$ exists, without the need to explicitly compute its inverse. IRG gradients are expressed via the Jacobian of $S_\phi(z)$:

$$\nabla_\phi z = -(\nabla_z S_\phi(z))^{-1} \nabla_\phi S_\phi(z). \tag{2.13}$$

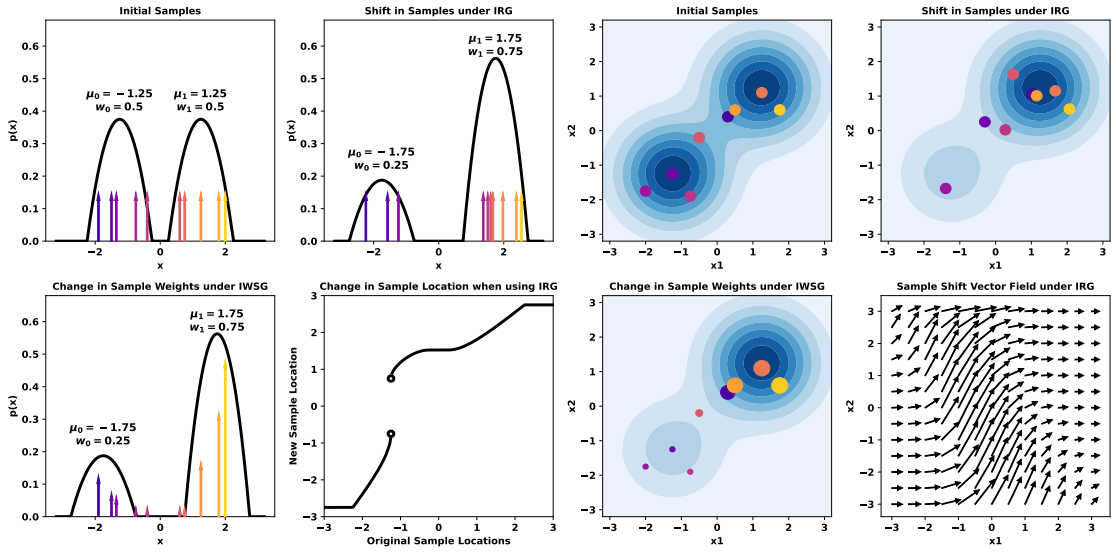


Figure 2.2: *Left:* Example changes in samples from a 1D mixture model with two Epanechnikov [38] components. Under IRG, changes in the mixture are shown by dramatic sample shifting, where some samples must change modes to account for the changes in the relative weights of each mixture mode. Explicitly plotting the particle transformation induced by IRG reveals a discontinuity (bottom). In contrast, IWSG smoothly reweights samples. *Right:* Example changes in samples from a 2D mixture of two Gaussians. IRG again induces large shifts as particles change modes, as demonstrated by the vector field (bottom).

For univariate distributions, the cumulative distribution function (CDF) $M_\phi(z)$ is a valid standardization function. For higher dimensions, the multivariate distributional transform [39] is used:

$$S_\phi(z) = \Big(M_\phi(z_1), M_\phi(z_2 \mid z_1), \ldots, M_\phi(z_D \mid z_1, \ldots, z_{D-1})\Big). \tag{2.14}$$

While IRG may be applied to any distribution with continuous CDF, and has been explicitly suggested (without experimental validation) for mixtures [9], we show that it may have enormous variance.

2.4.2 Importance Weighted Samples Gradient Estimator

We propose a novel alternative method for computing gradients of samples from a continuous mixture distribution. Our *importance weighted sample gradient* (IWSG) estimator employs importance sampling for unbiased gradient estimation. IWSG is related to the DIS-PF gradient estimator for discrete resampling [6], but we instead consider continuous mixture distributions with arbitrary component distributions. This resolves the ancestry problem present in Ścibior et al. [6], since particle locations could be sampled from multiple overlapping mixture components, whose parameters will all have non-zero gradients.

Formally, we wish to draw samples $z^{(i)} \sim m(z \mid \phi)$ from a mixture with parameters ϕ. Instead of sampling from $m(z \mid \phi)$ directly, IWSG samples from some proposal distribution $z^{(i)} \sim q(z)$. Importance weights $w^{(i)}$, and associated gradients, for these samples then equal

$$w^{(i)} = \frac{m(z^{(i)} \mid \phi)}{q(z^{(i)})}, \qquad \nabla_\phi w^{(i)} = \frac{\nabla_\phi m(z^{(i)} \mid \phi)}{q(z^{(i)})}. \tag{2.15}$$

Setting the proposal distribution $q(z) = m(z \mid \phi_0)$ to be a mixture model with the "current" parameters ϕ_0 at which gradients are being evaluated, the IWSG gradient estimator (2.15) becomes

$$w^{(i)} = \frac{m(z^{(i)} \mid \phi)\big|_{\phi=\phi_0}}{m(z^{(i)} \mid \phi_0)} = 1, \qquad \nabla_\phi w^{(i)} = \frac{\nabla_\phi m(z^{(i)} \mid \phi)\big|_{\phi=\phi_0}}{m(z^{(i)} \mid \phi_0)}. \tag{2.16}$$

While current samples are given weight one because they are exactly sampled from a mixture with the current parameters ϕ_0, gradients account for how importance weights will change as mixture parameters ϕ deviate from ϕ_0. Note that gradients are *not* taken with respect to the proposal $q(z) = m(z \mid \phi_0)$ in the denominator of $w^{(i)}$, since sample locations $z^{(i)}$ are not altered by gradient updates; changes in the associated importance weights are sufficient for unbiased gradient estimation.

2.4.3 Instability of Implicit Reparameterization Gradients

Reparameterization methods capture changes to a given distribution $p(z|\phi)$, and thus to its parameters ϕ, by shifting the samples drawn from that distribution. When $p(z|\phi)$ is unimodal, this induces a smooth shift in the samples. When $p(z|\phi)$ contains multiple non-overlapping modes, these shifts are no longer smooth as samples jump from one mode to another; see Fig. 2.2. For mixtures with finitely supported kernels (like the Epanechnikov) and non-overlapping modes, discontinuities exist in $S_\phi^{-1}(\epsilon)$, and IRG estimates may have infinite variance. These discontinuities correspond to samples shifting from one mode to another, and the resulting non-smooth sample perturbations will destabilize backpropagation algorithms for gradient estimation. While the inverse CDF is always continuous for Gaussian mixtures, it may still have near-infinite slope when modes are widely separated, and induce similar IRG instabilities.

In contrast, our IWSG estimator reweights particles instead of shifting them. This reweighting is smooth and does not suffer from any discontinuities.

To illustrate this, we create a simple discriminative PF with linear-Gaussian dynamics (2.17) and a bimodal observation likelihood (2.18):

$$p(x_t \mid x_{t-1}, a_t) = \text{Norm}(x_t \mid Ax_{t-1} + Ba_t, \sigma^2), \tag{2.17}$$

$$p(y_t \mid x_t) = w_1\text{Norm}(y_t \mid C_1x_t + c_1, \gamma^2) + w_2\text{Norm}(y_t \mid C_2x_t + c_2, \gamma^2). \tag{2.18}$$

Using a fixed dataset and gradient descent, we train discriminative PFs using the IRG estimator, our IWSG estimator, as well as with biased gradients that are truncated at each resampling step [5]. We initialize with perturbed parameter values and aim to learn the values of all dynamics and likelihood parameters, except for σ and γ which are fixed to their true values. We compare the stability and convergence of these three PF methods when

learning the parameter values. For this problem, we decouple the resampling and posterior bandwidth parameters as in our A-MDPF model. For resampling we set $\tilde{\beta} = 0.05$, and for the posterior we set $\beta = 0.5$.

Exact posterior inference for this model is possible via a mixture KF [13, 40] which represents the state posterior as a Gaussian mixture. Dynamics are applied to each Gaussian component of the mixture using the KF update equations [13], keeping the number of components the same. When applying the measurement model, we simply multiply the KF mixture with the observation likelihood mixture, increasing the number of mixture components in the KF posterior. The number of components grows exponentially with time, motivating PFs for long-term tracking, thus the mixture KF is not a practical general inference algorithm for long (and possibly very long) sequences, but exact inference with a mixture KF remains remains tractable over a few time steps.

To make the learning problem identifiable and to enforce the constraints $w_1 + w_2 = 1$, $w_1 \geq 0$, $w_2 \geq 0$, we parameterize w_1 and w_2 within the PF and KF models via the logistic of a single learnable parameter v:

$$w_1 = \frac{1}{1 + \exp(v)}, \qquad w_2 = \frac{1}{1 + \exp(-v)}. \tag{2.19}$$

To learn the dynamics and measurement model parameters, we setup a synthetic training problem with a dataset of 1000 sequences, each of length 5 time-steps. Using stochastic gradient descent with batch size 64, we train to minimize the negative log-likelihood loss of the true state given the posterior distribution at only the final time-step. For PF models, we use kernel density estimation [21] to estimate a mixture distribution using the final weighted particle set. For the mixture KF, the posterior distribution is simply a mixture model and no further processing is needed to compute the loss.

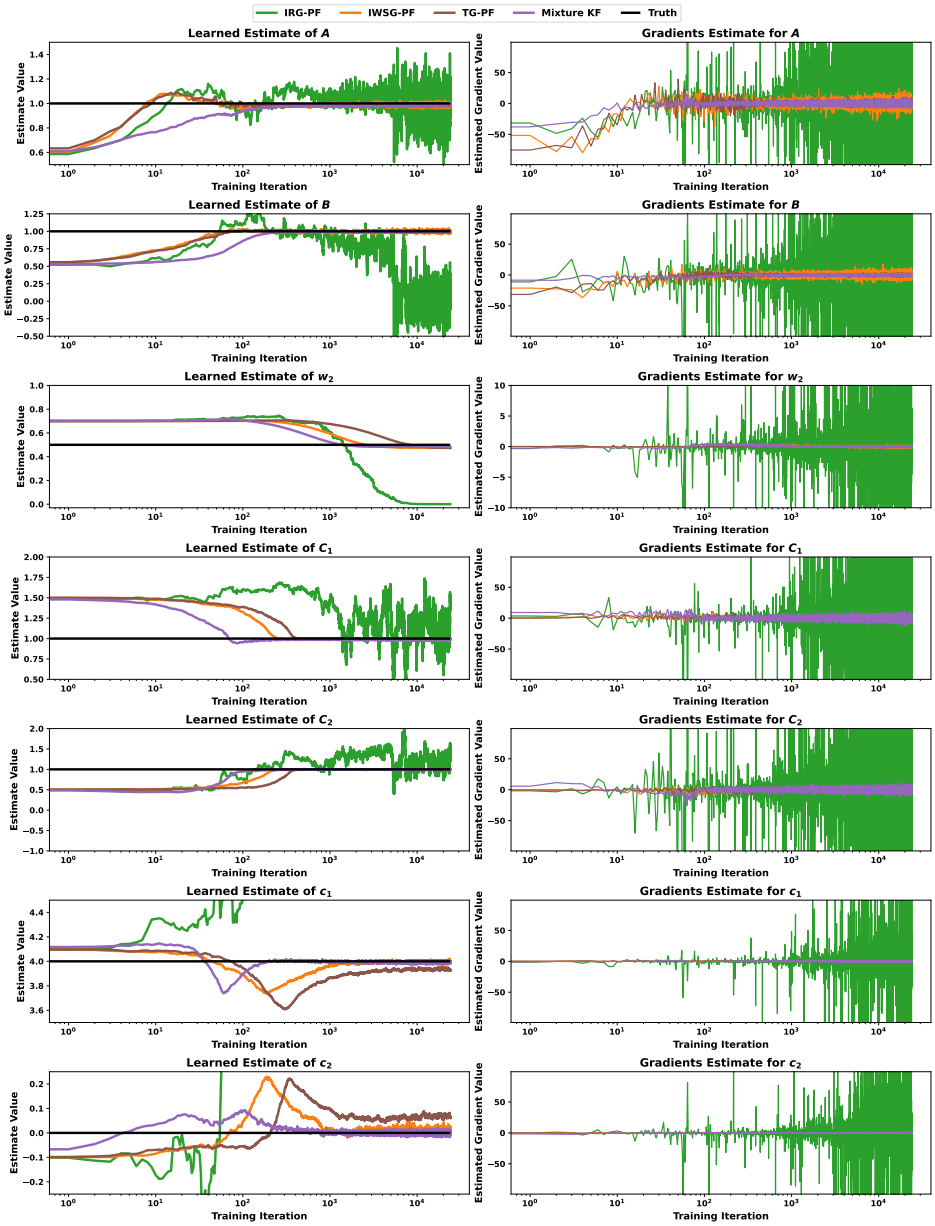


Figure 2.3: A simple temporal prediction problem where IRG has highly unstable gradient estimates. We use $N = 25$ particles when training IRG-PF, IWSG-PF, and truncated gradients (TG-PF). We show the learned values for parameters of the linear system and their gradients during training. IRG produces unstable gradients which prevent parameters from converging at all, but IWSG allows for smooth convergence of all parameters. IWSG is faster than biased TG, and nearly as effective as (expensive, and for general models intractable) mixture KF. TG-PF fails to learn correct values for c_1 and c_2 due to biases from truncating gradients at resampling.

Fig. 2.3 shows the estimates and gradients for a subset of the learned parameters, and several gradient estimators. IRG's unstable gradients prevent convergence of all parameters. For

training (generative) marginal PFs [41] which approximate marginals with mixtures, Lai et al. [42] found that IRG gradient variance was so large that biased estimators converged faster, but provided no detailed analysis. In contrast, IWSG converges smoothly for all parameters, and more rapidly than truncated gradients. Further we see that for parameters c_1 and c_2, the TG-PF method does not converge to the true value. TG-PF truncates gradients at the resampling, resulting in biased gradients. When the loss is computed at the final time, only information gained from that final time-step is present when updating the parameters, leading to biased parameter updates.

2.5 Mixture Density Particle Filters

Our *mixture density particle filter* (MDPF) (Fig. 2.5) is a discriminative model inspired by regularized PFs, where resampling uses KDEs centered on the current particles. We apply IWSG to this mixture resampling, achieving unbiased and low-variance gradient estimates for our fully-differentiable PF.

MDPF does not incorporate human-specified dynamics or measurement models; some prior work assumed known dynamics to simplify learning [5, 7]. These models are parameterized as deep neural networks (NNs), and trained via stochastic gradient descent to minimize the negative-log-likelihood of (potentially sparsely labeled) states x_t, given observations y_t and (optionally) actions a_t. As shown in Fig. 2.4, dynamics and measurement models are flexibly composed from smaller NNs. Our MDPF places no constraints on the functional form of NNs, allowing for modern architectures such as CNNs [43] and Spatial Transformers [44] to be used. See Appendix for implementation details.

The MDPF uses KDE mixture distributions as in Eq. (2.4) to define state posteriors used to evaluate the training loss, as well as resampling. A bandwidth parameter β is thus required

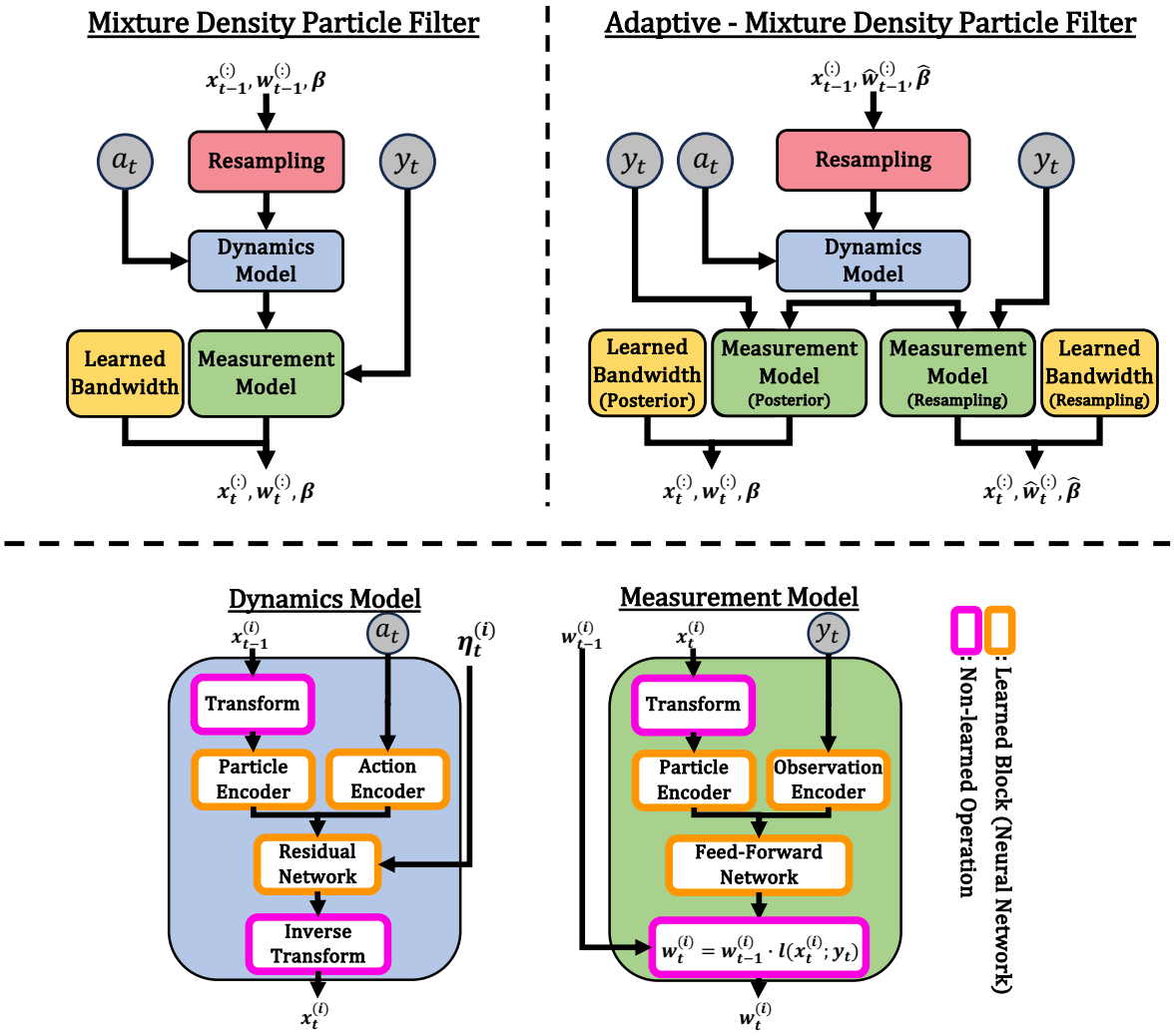


Figure 2.4: *Top Left:* Our MDPF method showing the various sub-components. *Top Right:* Our A-MDPF method with decoupled measurement models and bandwidths. *Bottom:* Dynamics and measurement model structures used in MDPF and A-MDPF. The dynamics and measurement models are composed of several neural networks as well as some fixed transforms, which convert the angular state dimensions of particles into a vector representation.

for each state dimension. Rather than setting β via the classic heuristics discussed in chapter 2.2.2, we make β a learnable parameter, optimizing it using end-to-end learning along with the dynamics and measurement models. This contrasts with prior work including SR-PF [7] and C-PF and OT-PF [8], which all include relaxation hyperparameters that must be tuned via multiple (potentially expensive) training trials.

We also extend the MDPF by decoupling the mixtures used for particle resampling and posterior state estimation. Using two separate measurement models, we compute two sets of particle weights $\tilde{w}_t^{(:)}$ and $w_t^{(:)}$, one for resampling and the other for posterior state estimation; see Fig. 2.4. Each distribution is also given a separate bandwidth $\tilde{\beta}$, β. This *adaptive* mixture density PF (A-MDPF) allows the uncertainty in the estimation of the current latent state, and the degree of exploration in the particle resampling step, to be decoupled and separately optimized during training. Note that training of separate posterior and resampling distributions is *impossible* for PFs that truncate temporal gradients; it is only feasible due to our unbiased and low-variance IWSG estimator of resampling gradients.

Mixture Density Particle Filtering:

Given observations $y_{1:T}$ and actions $a_{1:T}$ with N being the number of particles

1. Initialize particle set $\{x_1^{(:)}, w_1^{(:)}, \beta\}$ using initial known state or via some other method. Normalize weights such that $\sum_{i=1}^{N} w_1^{(i)} = 1$
2. For $t = 2, ..., T$ **and** $i = 1, ..., N$
   (a) Resample particle from mixture distribution

$$\widetilde{x}_t^{(i)} \sim m(x_{t-1}^{(:)}, w_{t-1}^{(:)}, \beta), \qquad \widetilde{w}_t^{(i)} = \frac{1}{N}$$

   (b) Apply noisy system dynamics to particle:

$$x_t^{(i)} = f(\widetilde{x}_t^{(i)}, a_t, \eta), \qquad \eta \sim N(0, 1)$$

   (c) Compute particle weight (normalized such that $\sum_{i=1}^{N} w_t^{(i)} = 1$)

$$w_t^{(i)} = \widetilde{w}_t^{(i)} \cdot l(x_t^{(i)}; y_t)$$

3. **Output:** $\{x_{1:T}^{(:)}, \widetilde{w}_{1:T}^{(:)}, w_{1:T}^{(:)}, \beta\}$

Figure 2.5: The Mixture Density Particle Filter

2.6 Experiments

We evaluate our MDPF and A-MDPF on a variety state estimation tasks in complex, but simulated, environments. We compare to TG-PF [5], SR-PF [7], OT-PF [8], DIS-PF [6], as well as C-PF which uses the Concrete distribution [34, 33] to relax discrete resampling. We also compare to variants of MDPF: Truncated-Gradient-MDPF (TG-MDPF) is similar but stops gradients at each particle resampling, and IRG-MDPF replaces our IWSG estimator with the unstable IRG estimator. Finally, we compare to a LSTM [45] which produces point predictions via uninterpretable latent states.

For all tasks, we estimate posterior distributions of a 3D (translation and angle) state of a robot, $x_t = (x, y, \theta)$. We use Gaussian kernels for position components, and von Mises kernels for angular components, of all KDEs. As is common in *recurrent neural network* (RNN) training, we employ conservative gradient clipping [46], as well as truncated-BPTT [47] where gradients are propagated for 4 time-steps. All methods are initialized as the true state with added noise.

We train and evaluate MDPFs to minimize the following *negative-log-likelihood* (NLL) loss:

$$\mathcal{L}_{\text{NLL}} = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} -\log(m(x_t | x_t^{(:)}, w_t^{(:)}, \beta)). \tag{2.20}$$

We use sparsely labeled training data to highlight the need for effective multi-time-step gradients during training. Dense labeling of data is often expensive, so many real-world datasets have similarly sparse labels. During training, we label true states every 4th time-step ($\mathcal{T} = \{4, 8, 12, \ldots\}$), and use densely labeled datasets ($\mathcal{T} = \{1, 2, 3, \ldots\}$) for evaluation. See the Appendix for details.

After training to minimize NLL, we optionally refine our MDPFs to minimize the *mean-squared-error* (MSE) of the weighted particle mean, and report the induced Root-MSE

(RMSE). Baseline methods do not estimate continuous posteriors, and thus Eq. (2.20) cannot be used for training. Instead we train and evaluate these methods with MSE and RMSE, before freezing the dynamics and measurement models. Then we construct KDEs, and optimize a bandwidth to report NLL values for comparison.

2.6.1 Bearings Only Tracking Task

In the bearings only tracking task, we track the state of a variable-velocity car using noisy bearing observations from a radar station to the car generated as

$$y_t \sim \alpha \cdot \text{Uniform}(-\pi, \pi) + (1 - \alpha) \cdot \text{VonMises}(\psi(x_t), \kappa),$$

where $\psi(x_t)$ is the true bearing and $\alpha = 0.15$. Actions are not available. We use 5000 training and 1000 validation trajectories of length $T = 17$, and 5000 evaluation trajectories of length $T = 150$.

Each method is evaluated and trained 11 times using $N = 25$ particles. To further highlight the stability of our IWSG estimator, we also train each method (except for IRG-MDPF) with the Epanechnikov kernel replacing the Gaussian in the KDE. Fig. 2.6 shows statistics of performance across 11 training runs. The very poor performance of IRG-MDPF is due to unstable gradients; without aggressive gradient clipping, IRG-MDPF fails to optimize at all. The benefits of IWSG are highlighted by the inferior performance of TG-MDPF. Stable multi-time-step gradients clearly benefit learning and are enabled by IWSG. Qualitative results are shown in Fig. 2.7.

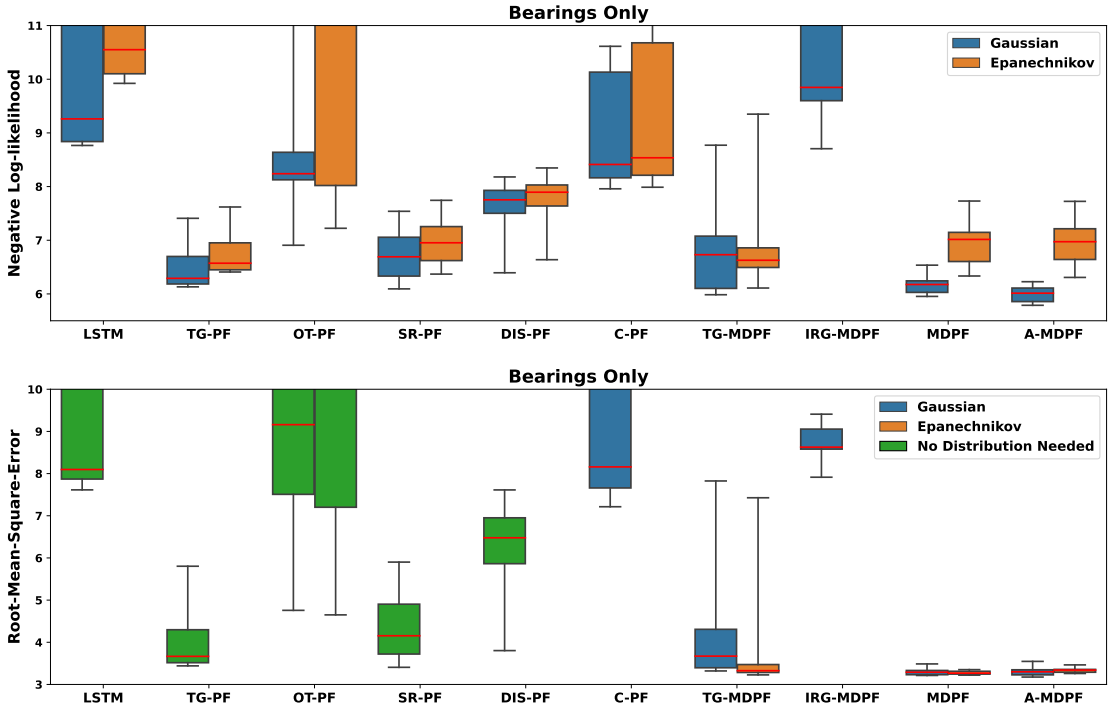


Figure 2.6: Box plots showing median (red line), inter-quartile (colored box) and range (whiskers) over several training runs on the Bearings-Only tracking task (11 runs). MDPF and A-MDPF consistently perform well on both the NLL and RMSE metrics. Our IWSG estimator is critical: IRG-MDPF performs very poorly due to unstable gradients, TG-MDPF is inconsistent and sometimes becomes trapped in local optima, and baselines with biased gradient estimators typically have inferior performance. Note that IRG-MDPF does not support Epanechnikov kernels, which induce discontinuous CDFs. The importance of our IWSG in MDPF is highlighted by the inconsistent performance of TG-MDPF where it is clear that multi-time-step gradients are beneficial.

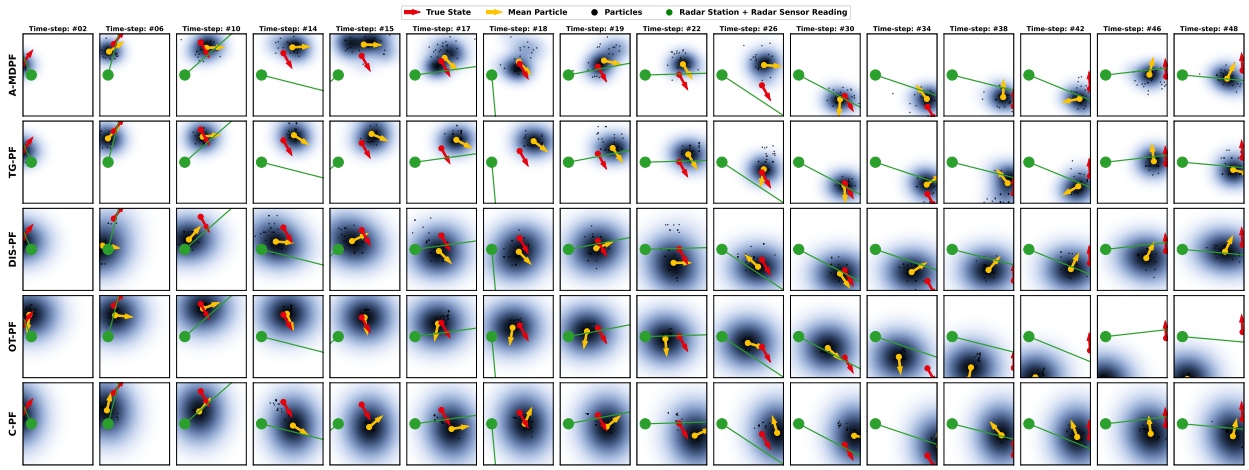


Figure 2.7: Example trajectories from the Bearings-Only tracking task. The radar station (green circle) produces bearing observations (green line) from the radar station to the car (red arrow). Note that observations are noisy and occasionally totally incorrect (see $t = 18$). We plot the posterior distribution of the current state (blue cloud), the mean particle (yellow arrow), and the $N = 25$ particles (black dots). A-MDPF is able to capture multiple modes in the state posterior (see $t = 15, 17, 18, 19$) while successfully tracking the true state of the car. Other models (DIS-PF, OT-PF, C-PF) spread their posterior distribution due to poor tracking of the true state.

2.6.2 Deepmind Maze Tracking Task

In the Deepmind-Maze tracking task, adapted from [5], we wish to track the 3D state of a robot as it moves through modified versions of the maze environments from Deepmind Lab [48] using noisy odometry as actions, and images (from the robot's perspective) as observations. We alter the experimental setup of [5] by increasing the noise of the actions by five times, and using $N = 25$ particles for training and evaluation. We train and evaluate on trajectories from the 3 unique mazes separately, using 5000 trajectories of length $T = 17$ from each maze for training, and 1000 trajectories of length $T = 99$ for evaluation. We show results over multiple training and evaluation runs in Fig. 2.8, and include qualitative results in the Appendix.

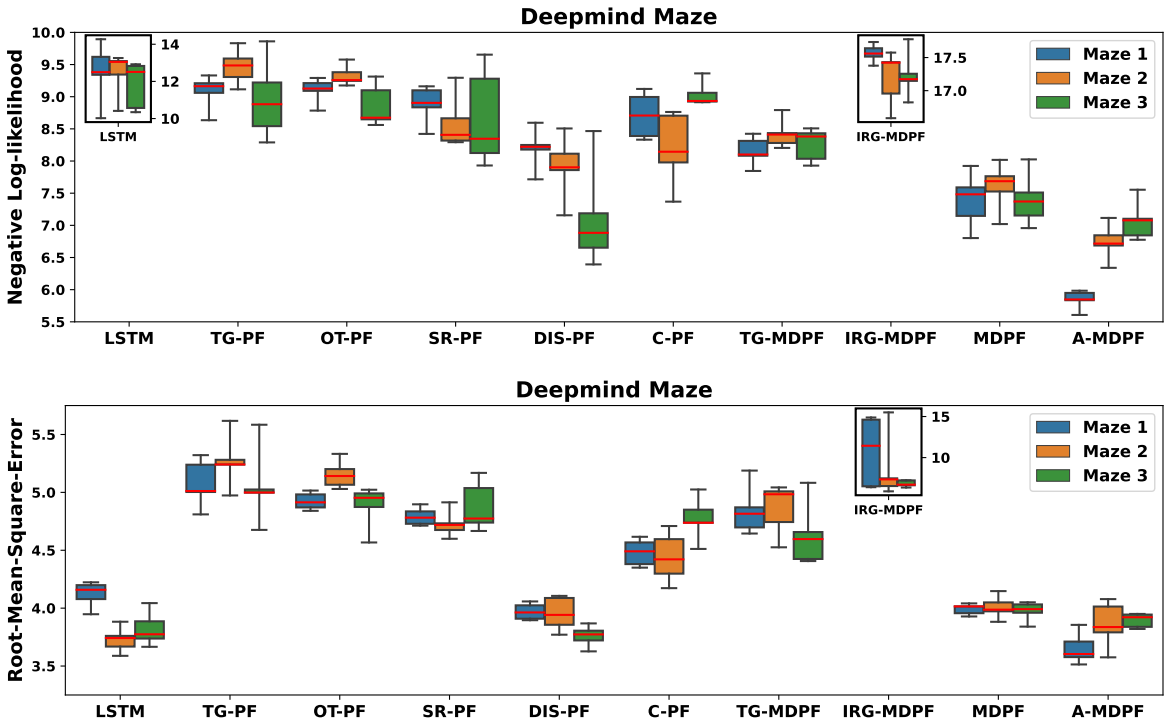


Figure 2.8: Box plots showing median (red line), inter-quartile (colored box) and range (whiskers) over several training runs on the Deepmind-Maze tracking task (5 runs). MDPF and A-MDPF consistently perform well on both the NLL and RMSE metrics. IRG-MDPF performs very poorly due to unstable gradients. In contrast our IWSG estimator performs very well and is stable. The importance of our IWSG in MDPF is highlighted by the inconsistent performance of TG-MDPF where it is clear that multi-time-step gradients are beneficial. LSTM achieves low RMSE in the Deepmind-Maze task, but we find it simply propagates the noisy actions blindly. DIS-PF performs well for Maze 3, but has larger variability and sometimes performs worse than our more stable A-MDPF.

2.6.3 House3D Tracking Task

This 3D state tracking task is adapted from [7], where a robot navigates single-level apartment environments from the SUNCG dataset [49] using the House3D simulator [50], with noisy odometry as action and RGB images as observations (see Fig. 2.9). Floor plans are input into measurement model via a modern Spatial Transformation Network architecture [44]. Training data consists of 74800 trajectories (length $T = 24$) from 199 unique environments, with evaluation data being 820 trajectories (length $T = 100$) from 47 previously unseen floor plans. We train and evaluate with $N = 50$ particles.

Method	NLL↓	RMSE↓
LSTM	12.54 ± 3.43	48.51 ± 27.16
TG-PF	15.50 ± 2.21	206.32 ± 109.44
OT-PF	14.87 ± 2.31	159.57 ± 109.65
SR-PF	14.92 ± 2.57	191.32 ± 129.28
DIS-PF	15.06 ± 2.02	212.58 ± 133.69
TG-MDPF	8.38 ± 2.53	35.77 ± 16.67
MDPF	8.18 ± 2.72	30.59 ± 12.98
A-MDPF	**8.09 ± 3.49**	**30.51 ± 12.32**

Table 2.1: Mean ± interquartile range of evaluation metrics for all House3D test sequences.

Due to the larger computational demands of House3D, we train each method once, reporting results in Table 2.1. Interestingly we find that TG-PF, OT-PF, SR-PF, and DIS-PF are unable to learn useful dynamics or measurement models. The LSTM model achieves better performance, but looking deeper we discover that it ignores observations completely, and simply blindly propagates the estimated state using actions with good dynamics (i.e., dead-reckoning). This strategy is effective for noise-free actions, but in reality the actions are noisy and thus the LSTMs estimated state diverges from the true state after a moderate number of time-steps. In contrast, our MDPF methods are able to learn good dynamics and measurement models that generalizes well to the unseen environments.

We also investigate MDPFs capacity for multimodal tracking in Fig. 2.9, by initializing A-

MDPF with particles spread throughout the state space. When initialized with random particles or with particles clustered around regions with similar observations, A-MDPF does not have sufficient information to collapse the posterior and thus maintains multiple posterior modes. As the robot moves, more evidence is collected, allowing A-MDPF to collapse the state posterior to fewer distinct modes, before eventually collapsing its estimate into one mode containing the true state. Note that such estimation of multiple posterior modes is impossible for standard RNNs like the LSTM.

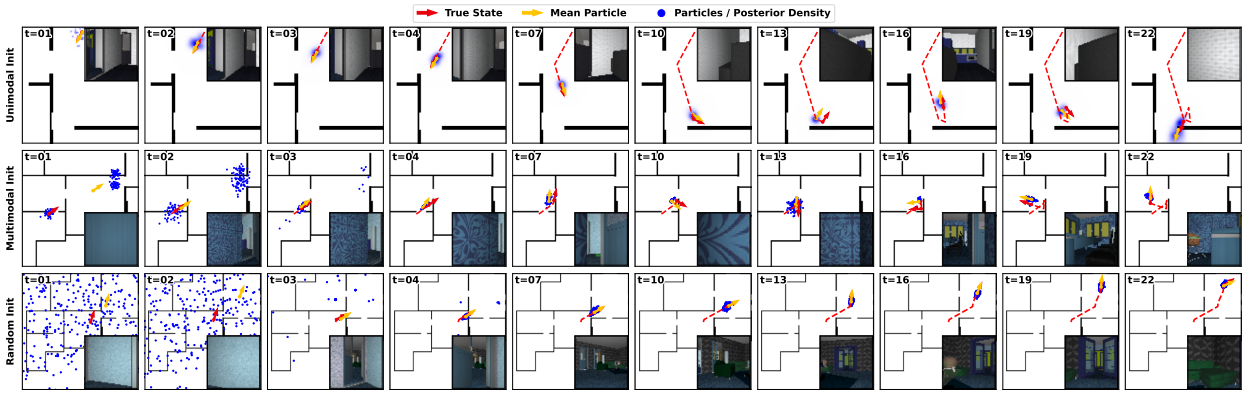


Figure 2.9: Example trajectories (rows), with observations in corner, from the House3D task using A-MDPF. *Top:* Initialized with $N = 50$ particles set to the true state with moderate noise, the posterior (blue cloud) closely tracks the robot. *Middle:* Multi-modal initialization with $N = 150$ particles (blue dots). *Bottom:* Naive initialization with $N = 1000$ particles drawn uniformly. A-MDPF maintains multiple modes until enough observations have been seen to disambiguate the true state.

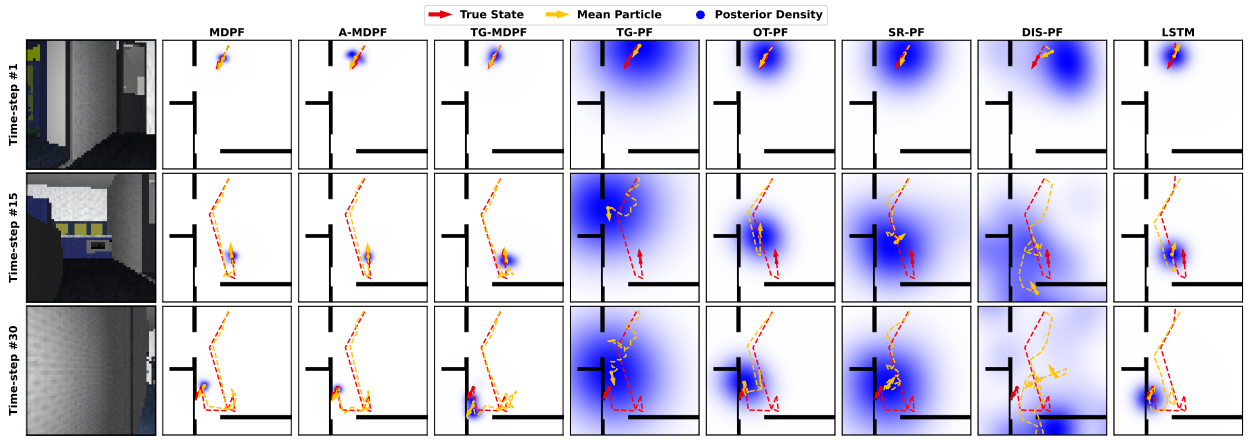


Figure 2.10: Example posteriors for three time steps (rows) of House3D tracking, for several discriminative PFs given the same low-noise initialization. We show the current true state and state history (red arrow and dashes), the posterior distribution of the current state (blue cloud, with darker being higher probability), and the estimated mean state and history (yellow arrow and dashes). Several baselines completely fail to track the robot.

2.6.4 Computation Requirements

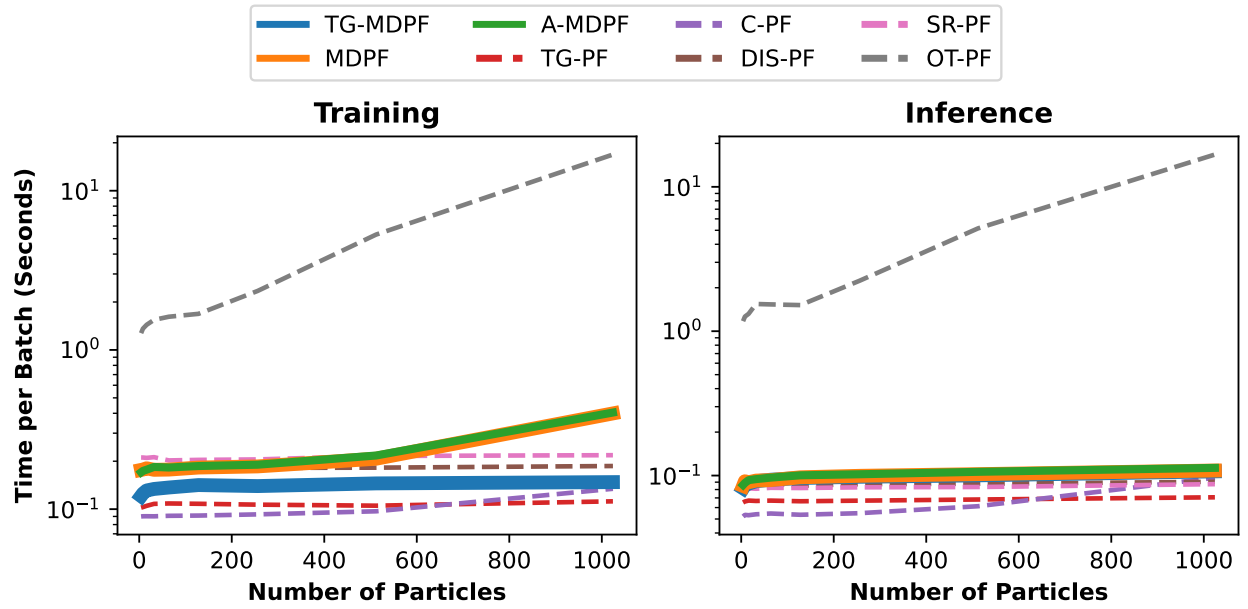


Figure 2.11: Computational time for various particle filters versus the number of particles on the Deepmind Maze task. A batch size of 24 is used. During training, the quadratic run-time of MDPF and A-MDPF is due to gradient computation. During inference MDPF and A-MDPF have linear run-time requirements. In contrast OT-PF must solve a complex optimal transport problem at both training and inference resulting in overall slow run-times.

The computational requirements for our MDPF and A-MDPF methods are quadratic with the number of particle during training, $O(N^2)$, due to gradient computation. During training, computing gradients using IWSG requires evaluation a mixture distribution with N components for each of the N particles. During inference, the computation requirement is linear in the number of particles, $O(N)$, since we mearly have to sample from the resampling mixture distribution rather than computing gradients for the samples. This is highlighted in fig. 2.11. Other methods except OT-PF have computational complexity $O(N)$ during both inference and training. OT-PF requires solving an optimal transport problem with complexity $O(N^2)$ during both training and inference [8]. In practice we find that OT-PF has significant computational requirements resulting in slow run-times both at inference and testing.

2.7 Limitations

A known limitation of all particle filters is there inability to scale to very high-dimensional states. Sparsity increases as the dimension grows, and thus more particles (and consequently computation) are required to maintain the expressiveness and accuracy of the state posterior. Our MDPF is not immune to this issue, but we conjecture that end-to-end training of discriminative PFs will allow particles to be more effectively allocated within large state spaces, scaling better than classic particle filters.

2.8 Discussion

We have developed a novel importance-sampling estimator for gradients of samples drawn from a continuous mixture model. Our results highlight fundamental flaws in the application of (implicit) reparameterization gradients to any mixture model with multiple modes. Ap-

plying our gradient estimator to the resampling step of a regularized discriminative PF, we obtain a fully end-to-end differentiable MDPF that robustly learns accurate models across multiple training runs, and has much greater accuracy than the biased particle filters proposed in prior work. While our experiments have focused on synthetic tracking problems in complex virtual environments, future applications of our MDPF to real-world vision and robotics data are very promising.

Appendix

2.A Parameterization of the Neural Dynamics and Measurement Models

In our implementation of MDPF and A-MDPF, we define the dynamics and measurement models as a set of small learnable neural networks, Fig. 2.4. The dynamics model uses two encoder networks which encode the particles and the actions into latent dimensions. The particle encoder is applied individually to each particle, allowing for parallelization. The encoded actions and particles are used as input, along with zero-mean unit variance Gaussian noise, into a residual network to produce new particles.

The measurement model is also comprised of several small neural networks. Particles and observations are encoded into latent representations before being used as input into a simple feed-forward network. Our method makes no restrictions on the network architectures allowing modern architectures to be used. For high dimensional observations, such as images, or for more complex problems a sophisticated architecture such as Convolutional Neural Networks [43] or Spatial Transformer Networks [44] can be implemented.

Since particle states are interpretable, they are often in a representation not suitable for neural networks. For example angles contain a discontinuities (at 0 and 2π) which hinders

their direct use with neural networks. To address this, non-learned transforms are applied to the particles, converting them into representations that are better suited for neural networks. In the case of angles transforms convert angles to and from unit vectors:

$$T(\theta) = (\cos(\theta), \sin(\theta)) \qquad T^{-1}(u, v) = \text{atan2}(u, v) \tag{2.21}$$

2.B Additional Experiment Results

In this section we give additional qualitative experiment results for the various tracking tasks.

Bearings Only Tracking Task

In fig. 2.12 we show additional qualitative results for various PF methods where it is apparent that A-MDPF and MDPF both track the true state well whereas other methods have wide posteriors due to their poor tracking ability. Figs. 2.13 2.14 2.15, 2.16, 2.17, 2.18 and 2.19 show additional trajectories for the various PF methods.

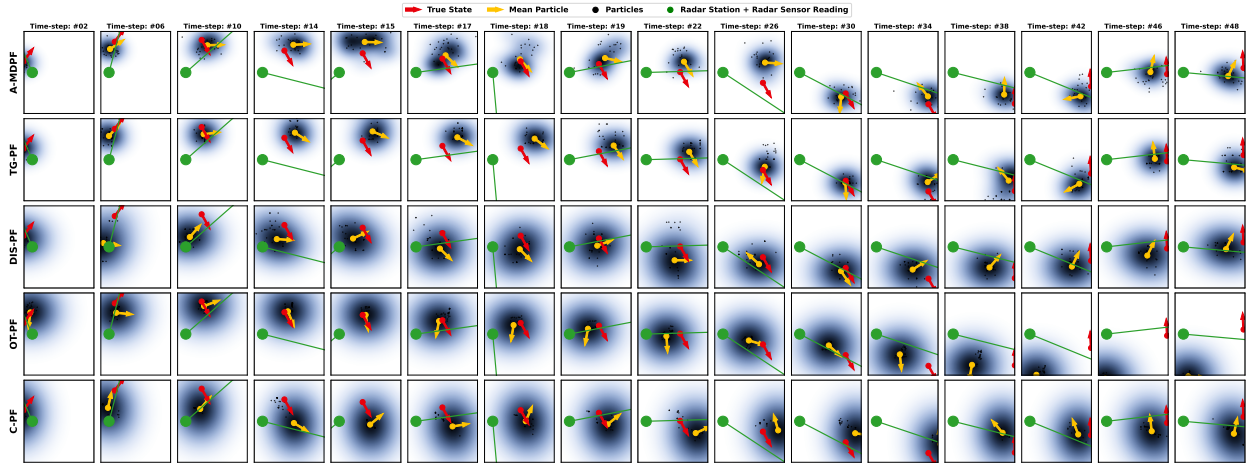


Figure 2.12: Example trajectory with various PF methods shown for the bearings only tracking task. The radar station (green circle) produces bearings observations (green line) from the radar station to the car (red arrow). Note that observations are noisy and occasionally totally incorrect. Shown is the posterior distribution of the current state (blue cloud, with darker being higher probability), the mean particle (yellow arrow) and the N = 25 particle set (black dots). A-MDPF and MDPF both have tight posterior distributions over the true state while maintaining tracking. Other methods such as DIS-PF, OT-PF and C-PF have wide posteriors showing their uncertainty due to their poor ability to track the true state.

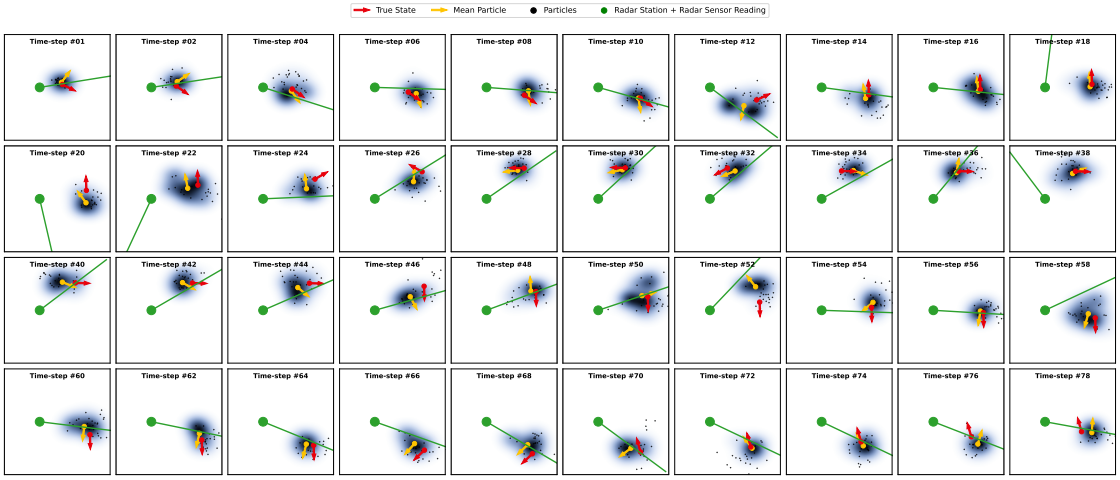


Figure 2.13: Example trajectory using A-MDPF shown for the bearings only tracking task. The radar station (green circle) produces noisy (and occasionally incorrect) bearings observations (green line) from the radar station to the car (red arrow). A-MDPF maintains a tight posterior distribution (blue cloud, with darker being higher probability) over the true state (red arrow) over time showing the effectiveness of our method. The mean particle (yellow arrow) and the N = 25 particle set (black dots) are also shown.

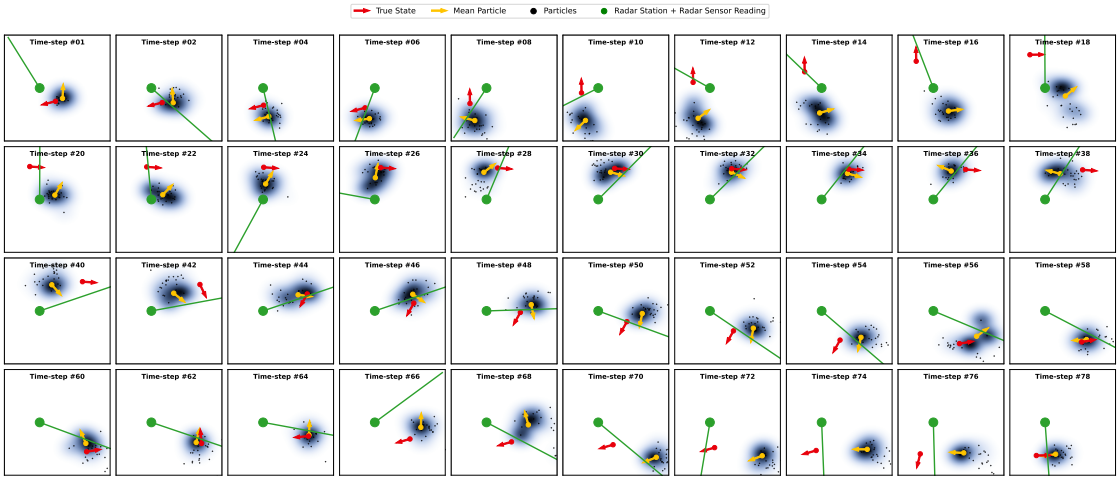


Figure 2.14: Another example trajectory using A-MDPF shown for the bearings only tracking task.

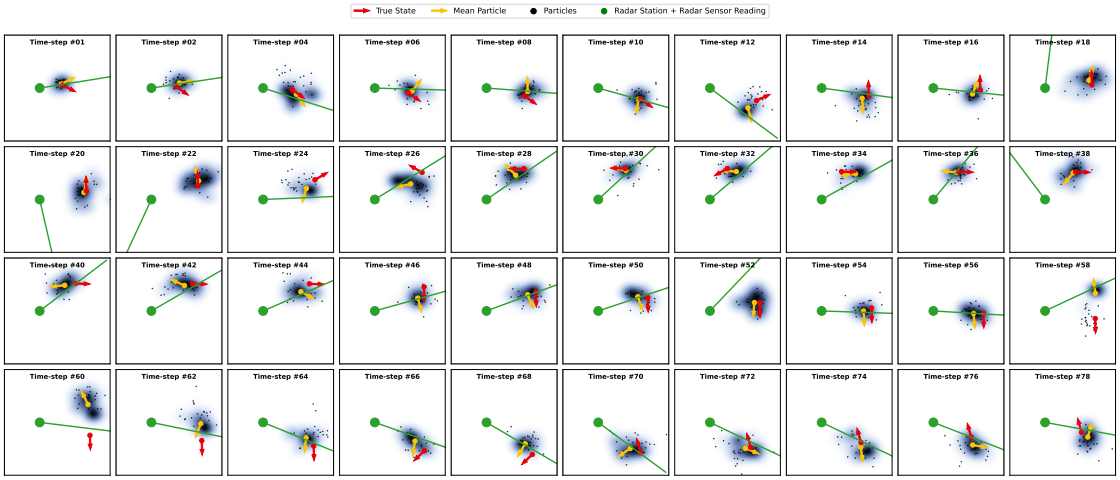


Figure 2.15: Example trajectory using MDPF shown for the bearings only tracking task. Similar to A-MDPF in Figs. 2.13 and 2.14, MDPF accurately tracks the true state (red arrow) with a tight posterior density (blue cloud).

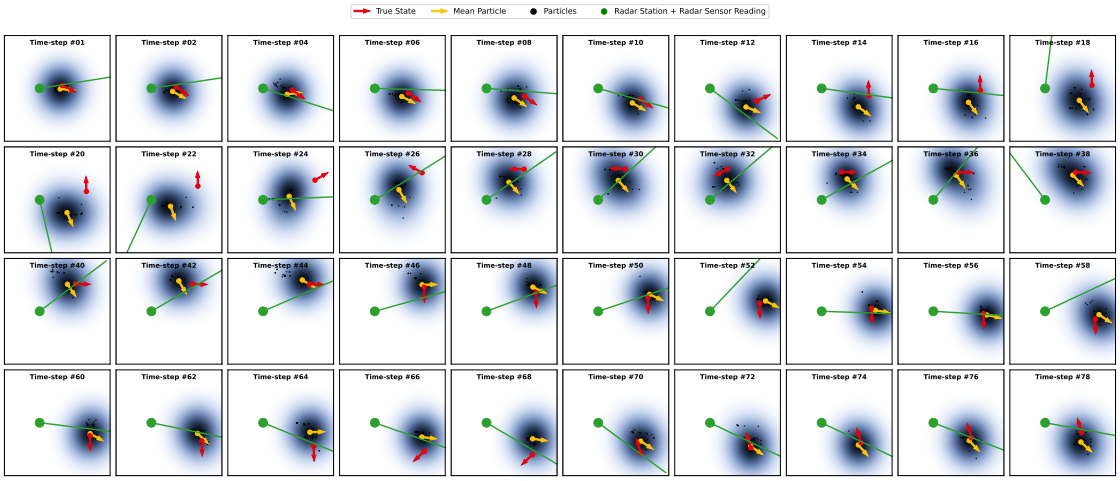


Figure 2.16: Example trajectory using DIS-PF shown for the bearings only tracking task. Due to the poorly learned dynamics and measurement models, DIS-PF often loses track of the true state (red arrow) as seen in time-steps $18 - 28$. This results in a large estimated posterior density (blue cloud) to account for the incorrectly placed particles (black dots) caused by poor dynamics and particle weighting.

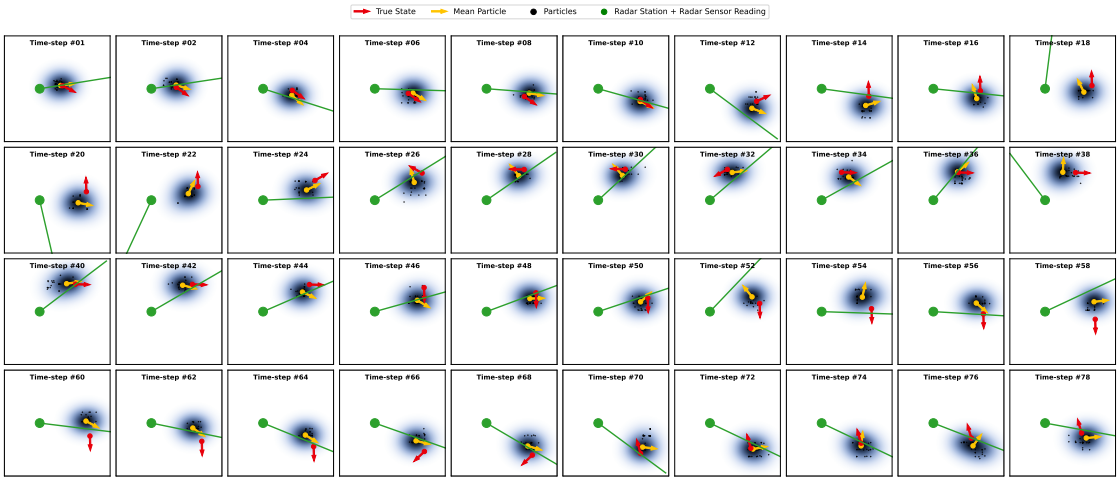


Figure 2.17: Example trajectory using SR-PF shown for the bearings only tracking task.

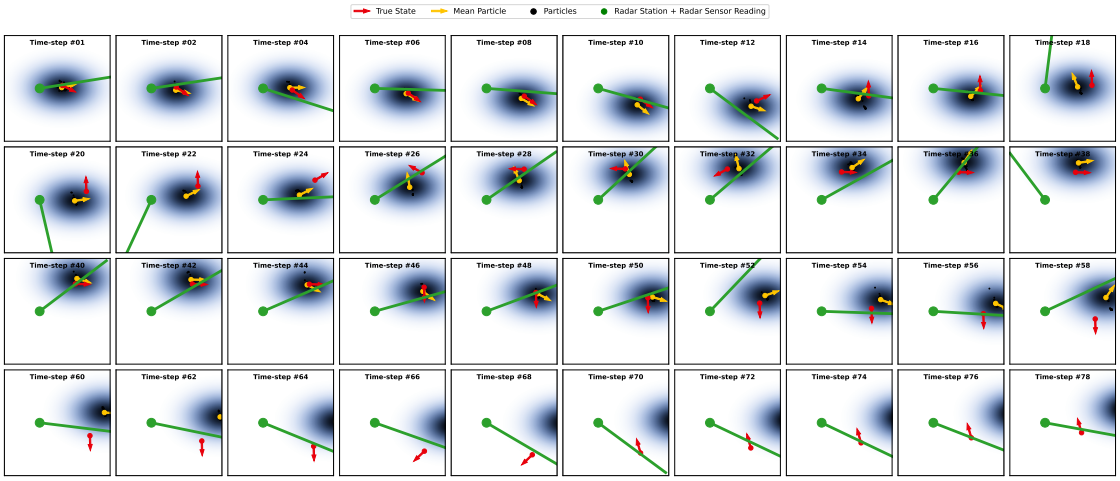


Figure 2.18: Example trajectory using OT-PF shown for the bearings only tracking task. Like DIS-PF in fig. 2.16, OT-PF has trouble tracking the true state (red) due to poorly learned dynamics and measurement models.

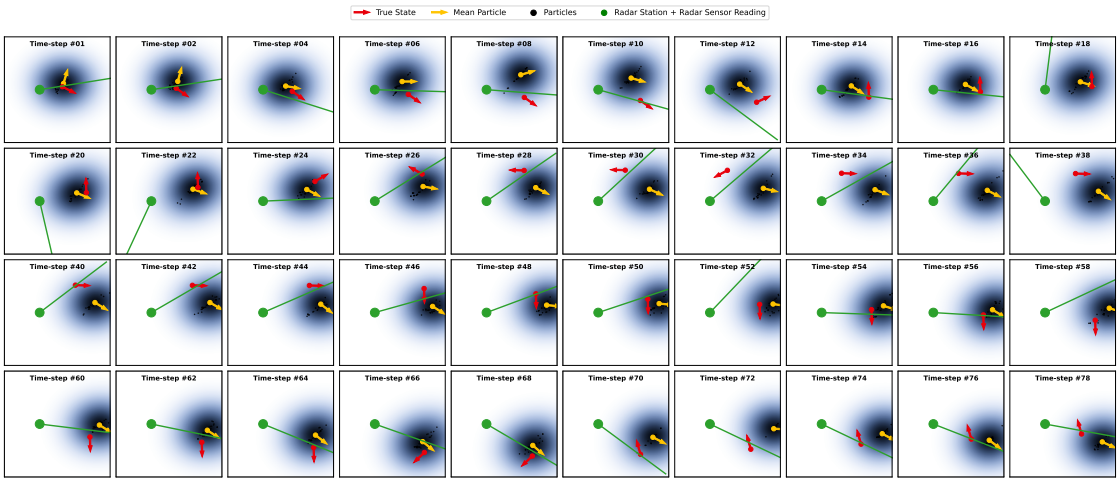


Figure 2.19: Example trajectory using C-PF shown for the bearings only tracking task. C-PF is unable to accurately track the true state (red) as shown by the relatively non-moving mean particle (yellow arrow).

Deepmind Maze Tracking Task

In Figs. 2.20, 2.21 and 2.22 we show additional qualitative results for various PF methods for mazes 1, 2 and 3, respectively. Again it is apparent that A-MDPF and MDPF both track the true state well whereas other methods have wide posteriors due to their poor tracking ability.

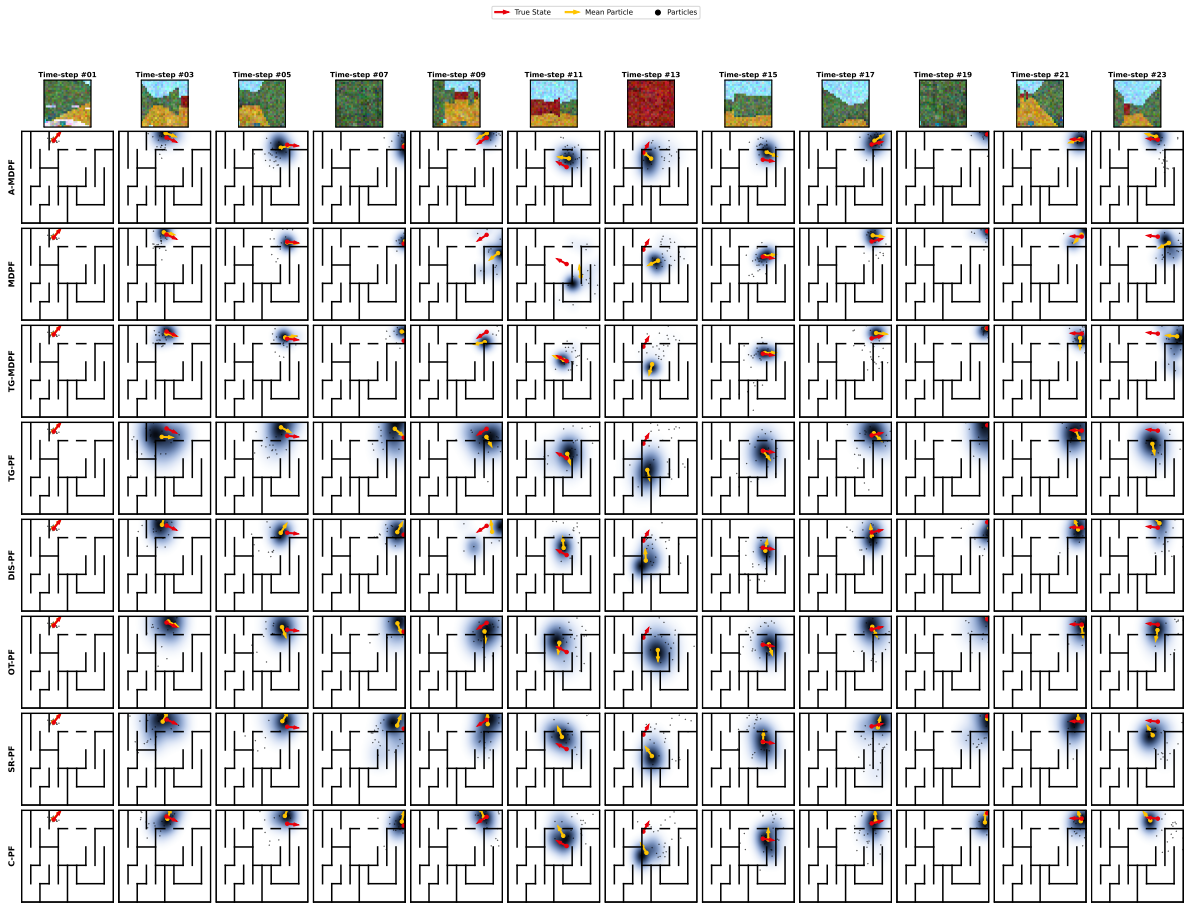


Figure 2.20: Example trajectory with various PF methods shown for the Deepmind maze tracking task for maze 1. Shown is the current true state (red arrow), the posterior estimate of the current state (blue cloud, with darker being higher probability), the mean particle (yellow arrow) and the N = 25 particle set (black dots).

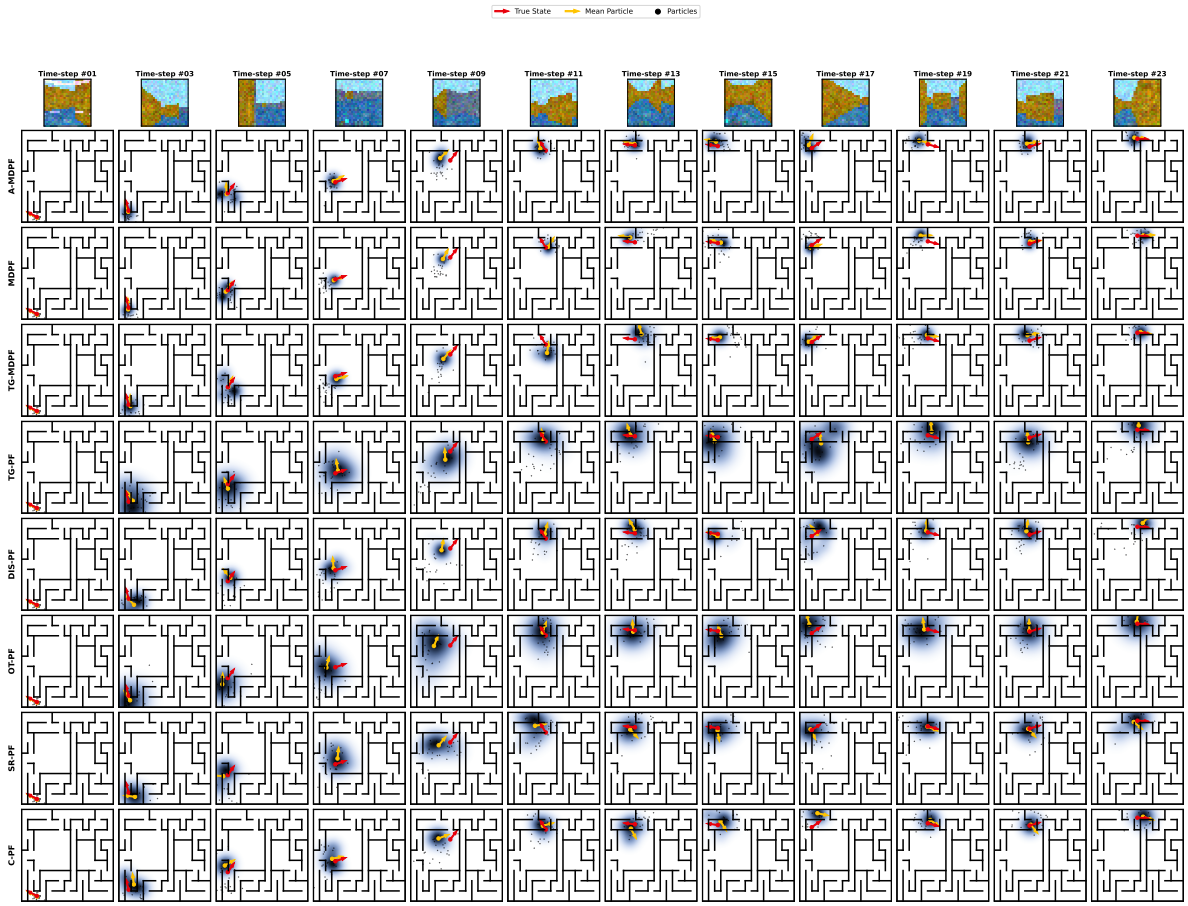


Figure 2.21: Example trajectory with various PF methods shown for the Deepmind maze tracking task for maze 2. Shown is the current true state (red arrow), the posterior estimate of the current state (blue cloud, with darker being higher probability), the mean particle (yellow arrow) and the N = 25 particle set (black dots).

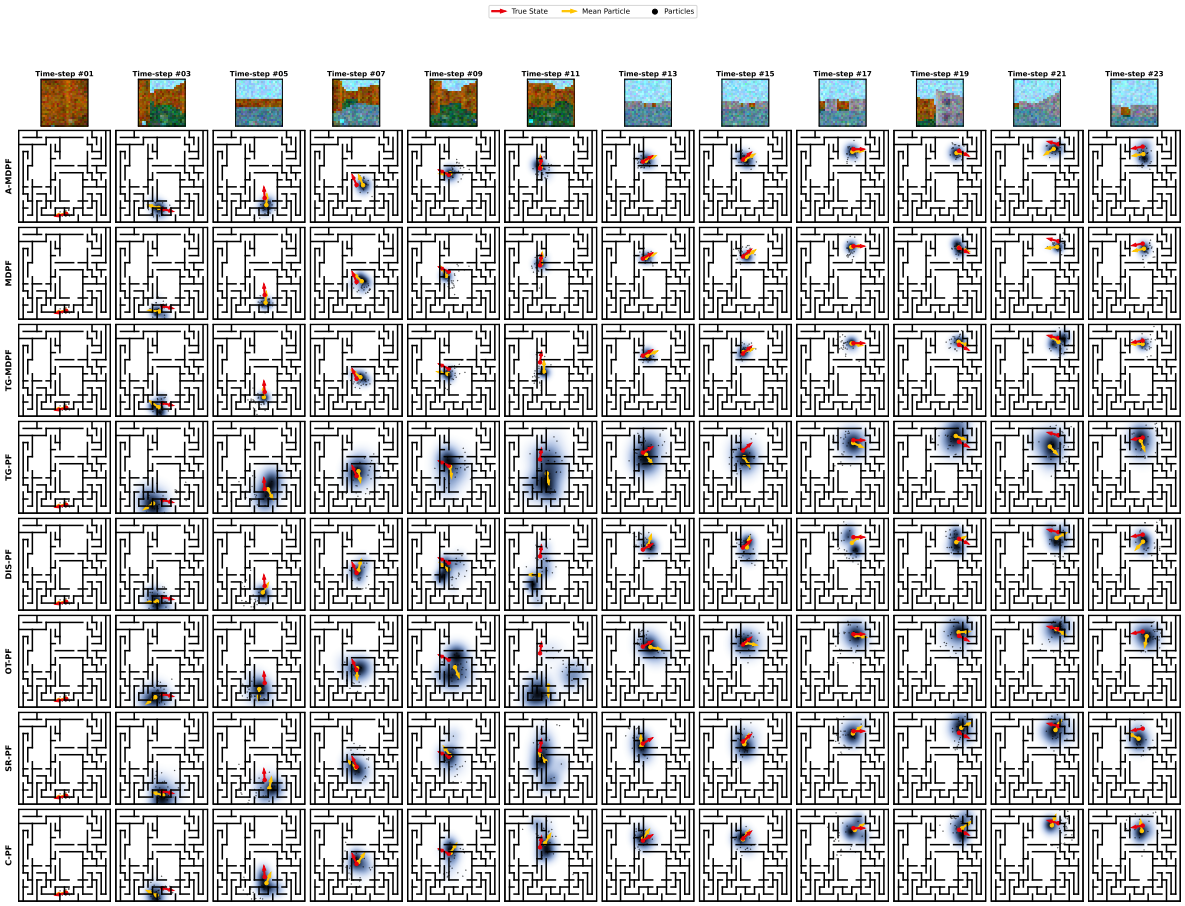


Figure 2.22: Example trajectory with various PF methods shown for the Deepmind maze tracking task for maze 3. Shown is the current true state (red arrow), the posterior estimate of the current state (blue cloud, with darker being higher probability), the mean particle (yellow arrow) and the N = 25 particle set (black dots).

House3D Tracking Task

Figs. 2.23 and 2.24 show example trajectories for the House3D tracking task using A-MDPF and MDPF respectively. We see that our A-MDPF and MDPF methods both successfully track the true state over time as evident by the tight posterior over the true state as well as the closeness of the mean state to the true state. Fig. 2.25 shows an example trajectory using the LSTM model. It is clear that the LSTM model is ignoring observations and instead is blind propagating its state estimate using good dynamics and the actions (dead-reckoning). In later time-steps, the LSTM estimate drifts away from the true state however the predicted state trajectory looks similar to that of the true state, but with a growing offset, indicating blind propagation.

Fig. 2.26 shows qualitative results when using OT-PF. OT-PF is unable to learn good dynamics and measurement models and quickly diverges away from true state. Figs. 2.27, 2.28 and 2.29 give example trajectories for TG-PF, DIS-PF and SR-PF respectively. These methods fail to learn usable dynamics or measurement models with their trajectories essentially being random.

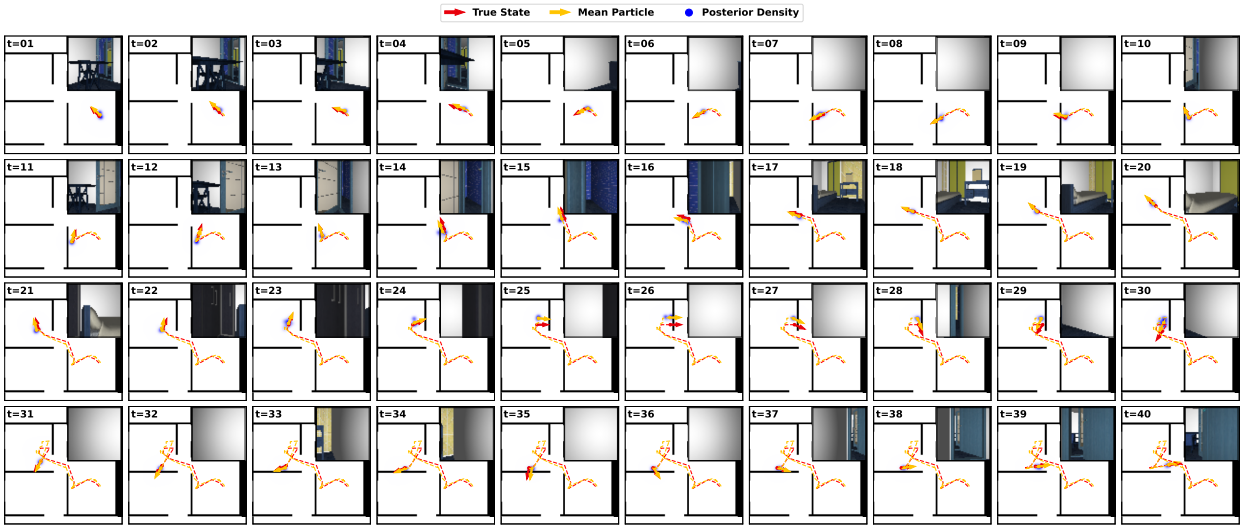


Figure 2.23: Example trajectory for the House3D tracking task using A-MDPF. The posterior density of the state is tight showing the confidence A-MDPF has in its estimate. The estimated posterior density (blue cloud, with darker being higher probability) also converges on the true state (red arrow) showing its accuracy. The mean particle (yellow arrow) closely tracks the true state. The observations are shown as inset images in the plots.

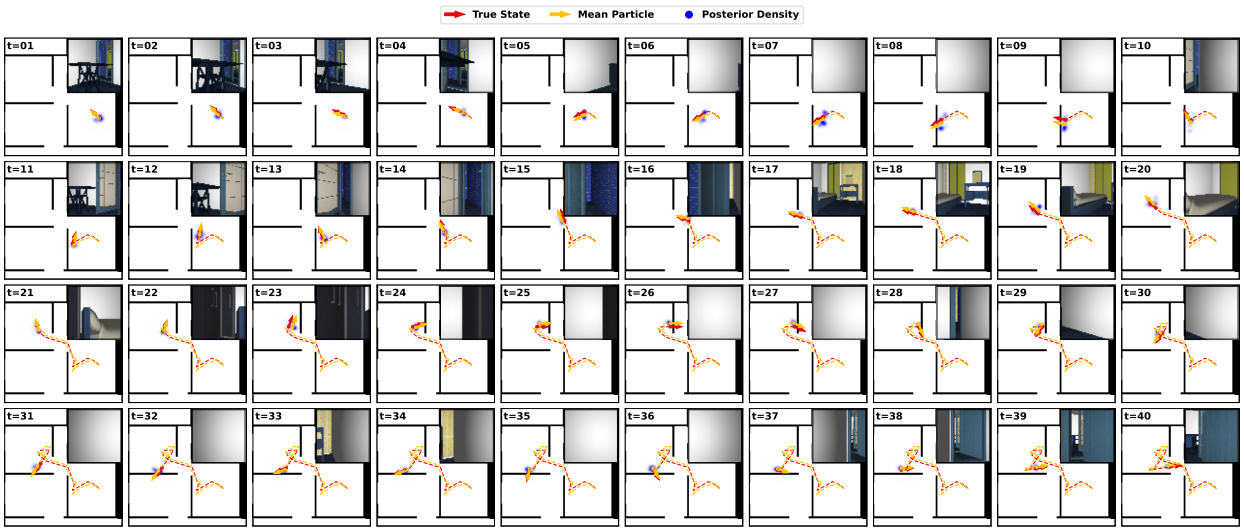


Figure 2.24: Example trajectory for the House3D tracking task using MDPF. Similar to Fig. 2.23, MDPF accurately tracks the true state.

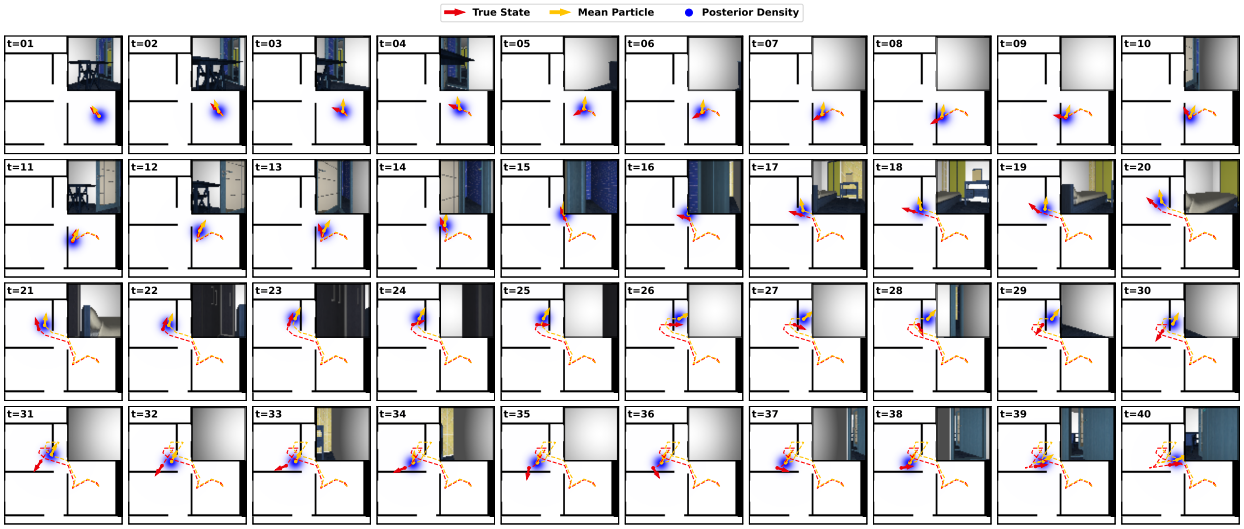


Figure 2.25: Example trajectory for the House3D tracking task using the LSTM model. The LSTM model does not us observations but rather blind-propagates the estimated state using the available noisy actions. This is seen by the true state (shown in red) and the estimated state (shown in yellow) having similar shapes but with increasing error.

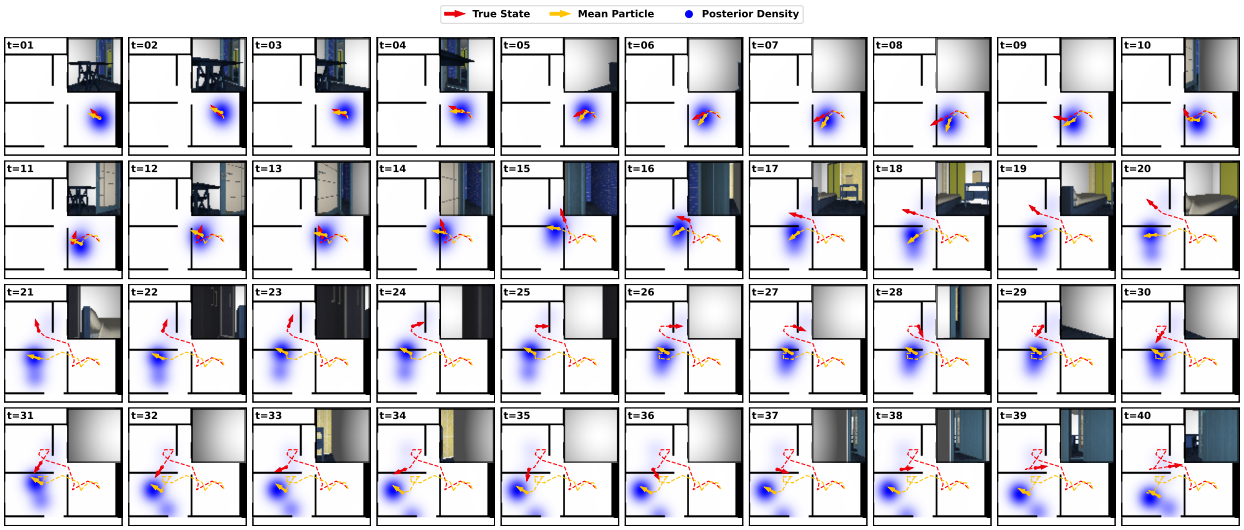


Figure 2.26: Example trajectory for the House3D tracking task using the OT-PF. OT-PF is unable to learn good dynamics or measurement models and thus the estimated posterior (blue cloud) diverges away from the true state (shown in red) after a couple time-steps.

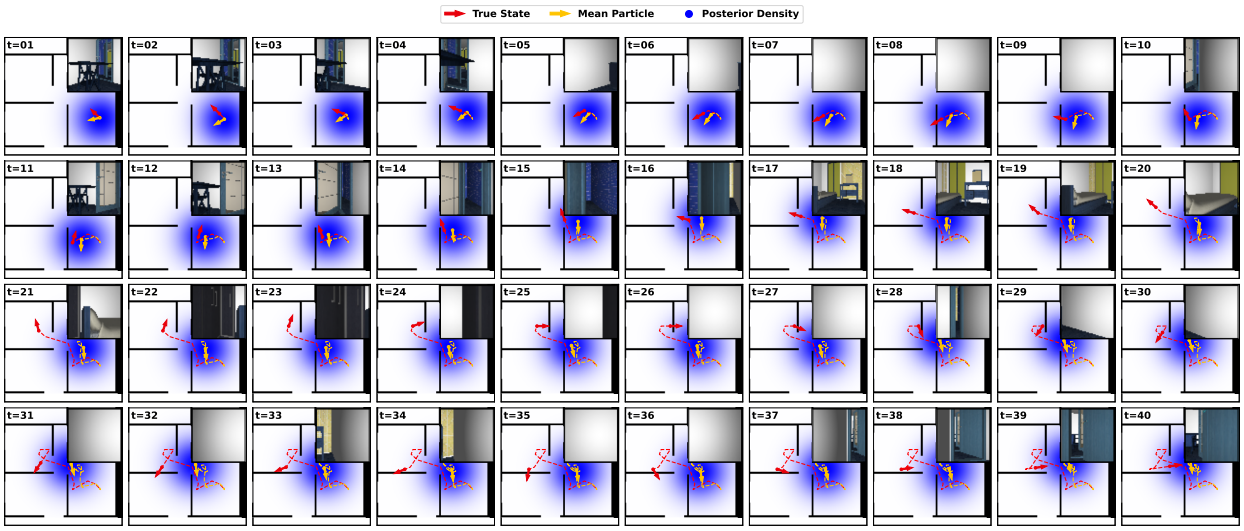


Figure 2.27: Example trajectory for the House3D tracking task using the TG-PF. TG-PF is unable to learn a usable dynamic or measurement model resulting in fairly random output. This is reflected in the large uncertainty of the estimated posterior (blue cloud) as well as the random nature of the mean state (shown in yellow).

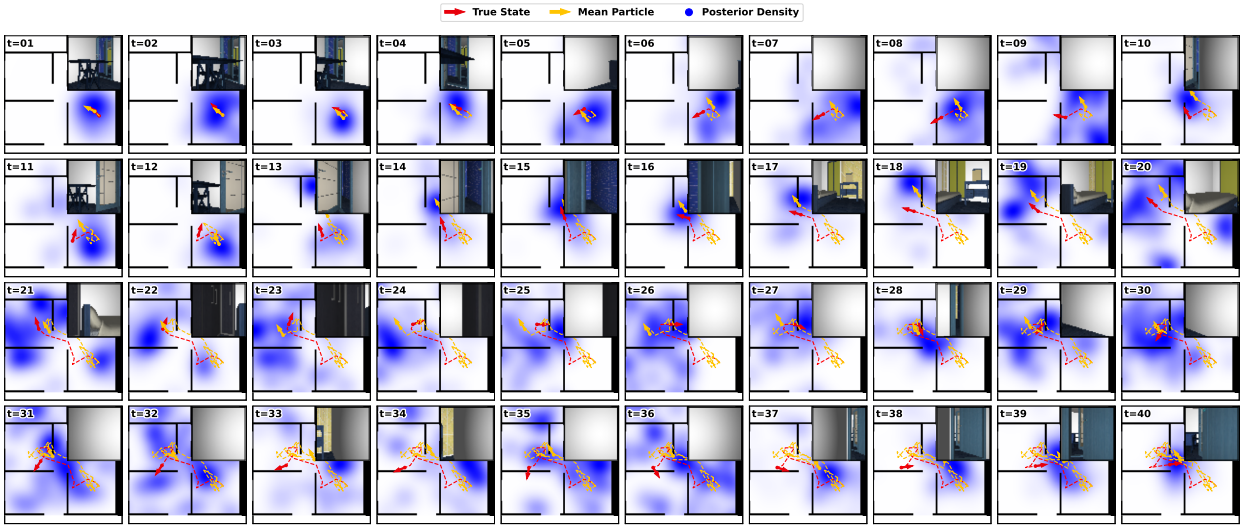


Figure 2.28: Example trajectory for the House3D tracking task using the DIS-PF. Similar to TG-PF in Fig. 2.27, DIS-PF is unable to learn a usable dynamic or measurement model and therefore is unable to collapse the estimated posterior density (blue cloud) onto the true state (shown in red).

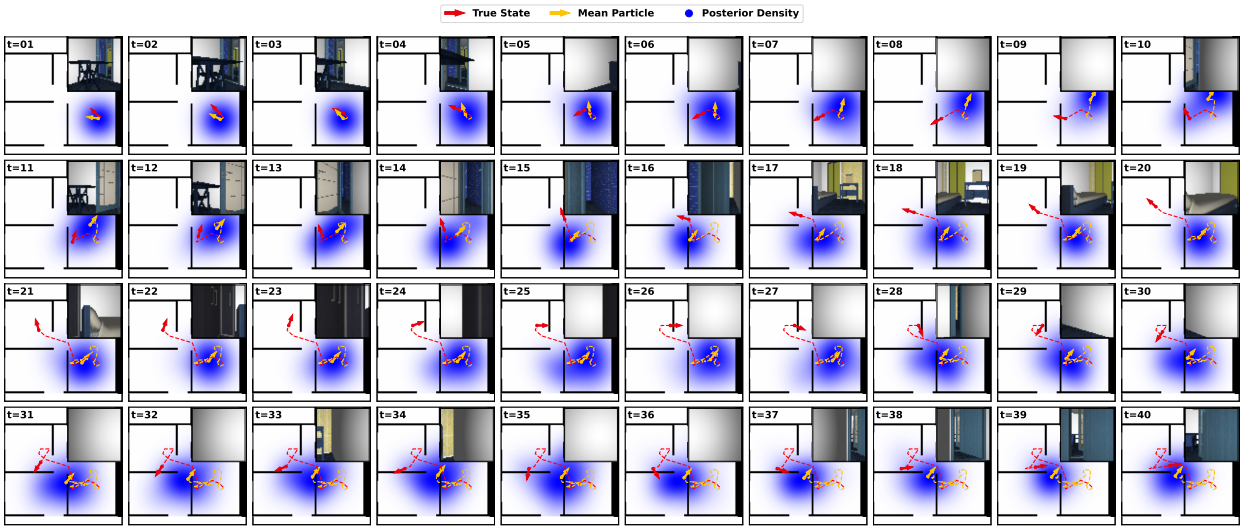


Figure 2.29: Example trajectory for the House3D tracking task using the SR-PF. Similar to TG-PF in Fig. 2.27 and DIS-PF in Fig. 2.28, SR-PF is unable to learn a usable dynamic or measurement model.

2.C Experiment Details

In this section give additional details for the various experiments (tasks). We also provide specific neural architectures used in each task.

Mean Squared Error Loss

When training the comparison PF methods, we use the Mean-Squared Error (MSE) loss. This is computed as the mean squared error between the sparse true states s_t and the mean particle at each time t:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} (x_t - \sum_{i=1}^{N} w_t^{(i)} x_t^{(i)})^2 \tag{2.22}$$

Here $\mathcal{T}$ are the indices of the labeled true states. During training, sparsely labeled true states are used and thus $\mathcal{T} = \{4, 8, 12, \ldots\}$. During evaluation we use dense labels, $\mathcal{T} = \{1, 2, 3, 4, \ldots\}$

For angular dimensions of $x_t^{(i)}$, we compute the mean particle by converting the angle into a unit vector, averaging the unit vectors (with normalization) and converting the mean unit vector back into an angle. Further we take special care when computing square differences for angles due to the discontinuity present at 0 and 2π.

Pre-Training Measurement Model

We pre-train the measurement model for each task using the sparse true states. To pre-train, a random true state from the dataset is selected. A particle set is constructed by drawing samples from a distribution centered around the select true state and weights for this particle set are computed using the measurement model and the observation associated with the selected true state. A mixture distribution is then fit using KDE with a manually specified bandwidth. Negative Log-Likelihood loss of the true state is used as the training loss. We find pre-training not to be sensitive to choice of bandwidth so long as it is not too small.

The dynamics model cannot be pre-trained because of the lack of true state pairs due to the sparse true state training labels and thus must be trained for the first time within the full PF algorithm.

Additional General Task and Training Details

For all tasks we wish to track the 3D (translation and angle) state of a robot, $s = (x, y, \theta)$, over time. This state space contains an angle dimension which is not suitable for neural networks. To address this issue, we convert particles into 4D using a transform $T(s) = (x, y, \sin(\theta), \cos(\theta))$ before using them as input into a neural network. Further we convert back into the original 3D representation to retrieve the a 3D particle after applying neural

Table 2.2: Hyper-parameters use for various PF methods.

Parameter	Value
SR-PF [7] λ	0.1
OT-PF [8] λ (Bearings Only)	0.5
OT-PF [8] λ (Deepmind)	0.01
OT-PF [8] λ (House3D)	0.01
OT-PF [8] scaling	0.9
OT-PF [8] threshold	1e-3
OT-PF [8] max iterations	500
C-PF [33, 34] λ	0.5

networks to the particles.

We use the Adam [51] optimizer with default parameters, tuning the learning rate for each task. We use a batch size of 64 during training and decrease the learning rate by a factor of 10 when the validation loss reaches a plateau. We use Truncated-Back-Propagation-Through-Time (T-BPTT) [47], truncating the gradients every 4 time-steps.

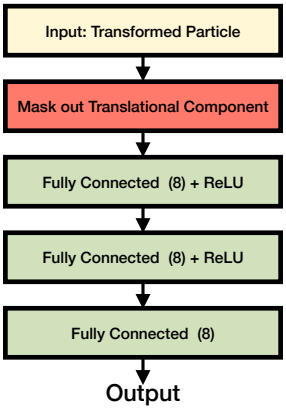
For all tasks we use the Gaussian distribution for the positional components of the state and the Von Mises distribution for the angular components when applying KDE. Here β is a dimension-specific bandwidth corresponding to the two standard deviations for the Gaussians and a concentration for the Von Mises.

Some methods have non-learnable hyper-parameters that must be set. The values of the hyper-parameters are chosen via brief hyper-parameter search with the final chosen values shown in table 2.2. The hyper-parameter values are the same for all tasks unless otherwise specified. We find the regularization parameter λ for OT-PF to be sensitive to the specific datasets.

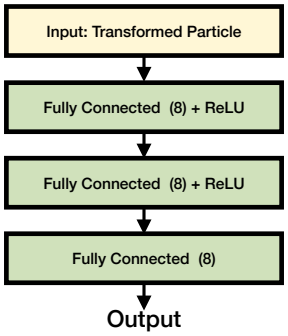
We choose to make the dynamics position invariant. This prevents the learned PF models from memorizing the state space based on position but rather forces the learned dynamics to the true dynamics. Position invariance can be achieved by masking out the translational

dimensions of the particle before input into the dynamics model.

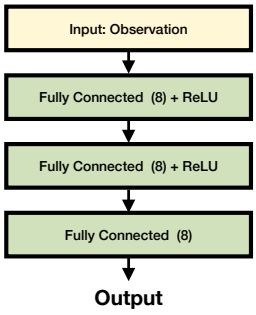
Bearings Only Tracking Task



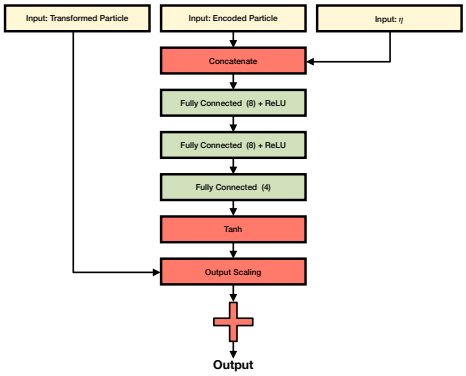
(a) Particle encoder for dynamics Model.



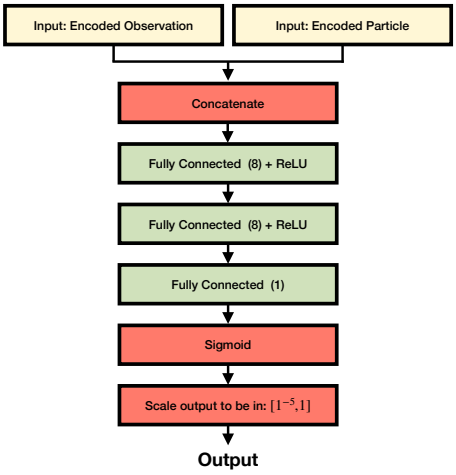
(b) Particle encoder for measurement model.



(c) Observation encoder.



(d) Dynamics model residual network, with actions omitted. Output scaling all dimensions to be within [-0.99, 0.99].



(e) Measurement model weights network.

Figure 2.30: Neural architecture for the networks within the dynamics and measurement models for the Bearings-Only tracking task.

In this section we give additional task specific information and network architecture details for the bearings only tracking task.

Additional Experiment Details

When training MDPF using the Negative Log-Likelihood (NLL) loss, we set the learning rate (LR) of the neural networks to be 0.0005 and the bandwidth learning rate to 0.00005. When training using MSE loss, we use 0.0001 and 0.00001 for the neural network and bandwidth learning rates respectively. For the discrete PF methods, we use LR = 0.0005 for training the neural network using the MSE loss. We use the same bandwidth for optimizing the bandwidth using the NLL loss when the neural networks are held fixed. For the LSTM [45], we use 0.001 as the learning rate for training the networks and LR = 0.0005 when optimizing the bandwidth parameter using NLL.

We apply gradient norm clipping, clipping the norm of the gradients to be ≤ 100.0.

In addition to using the Gaussian distribution for KDE, we also train with the Epanechnikov distribution applied to the translational dimensions of the state. To do this we first train using Gaussian kernels before re-training with the Epanechnikov, using the already trained dynamics and measurement models as a starting point.

The observations o_t are generated as noisy bearings from the radar station to the car:

$$o_t \sim \alpha \cdot \text{Uniform}(-\pi, \pi) + (1 - \alpha) \cdot \text{VonMises}(\psi_t, \kappa)$$

where ψ_t is the true bearing, $\alpha = 0.15$ and $\kappa = 50$. Observations o_t are angles and not suitable as input into a neural network. As such we convert the angular representation into a unit vector as is done with the angle dimensions of the particles.

Network Architectures

Fig. 2.30 gives details on the neural network architectures used for the dynamics and measurement models within the PF for the bearings only tracking task. Since this task does not provide actions, the action encoder has been omitted.

Deepmind Maze Tracking Task

In this section we give additional task specific information and network architecture details for the Deepmind maze tracking task.

Additional Experiment Details

The Deepmind maze tracking task was first proposed in [5]. In this task, a robot moves through modified versions of the maze environments from Deepmind Lab [48]. Three distinct mazes are provided and we train and test on each maze individually. Example robot trajectories as well as example observations for each maze are shown in Fig. 2.31. We modify this task by increasing the noise applied to the actions by 5 fold as well as using sparsely labeled true states during training.

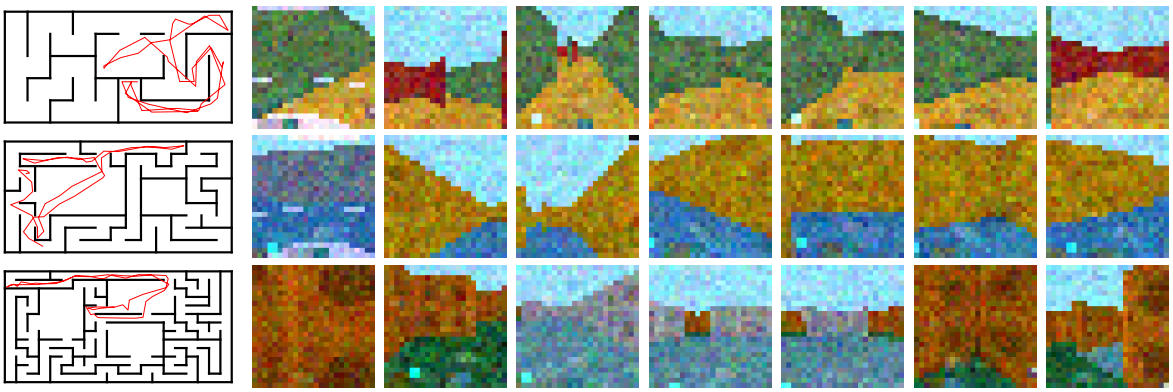


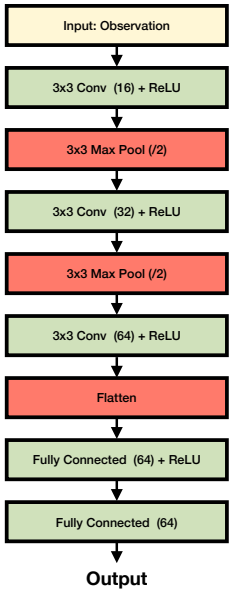
Figure 2.31: Example trajectories and observations from each of the three mazes from the Deepmind maze tracking task.

When training MDPF using the Negative Log-Likelihood (NLL) loss, we set the learning rate (LR) of the neural networks to be 0.0005 and the bandwidth learning rate to 0.00005. When training using MSE loss, we use 0.0005 and 0.00005 for the neural network and bandwidth learning rates respectively. For the discrete PF methods, we use LR = 0.0005 for training the neural network using the MSE loss. We use the same bandwidth for optimizing the bandwidth using the NLL loss when the neural networks are held fixed. For the LSTM, we use 0.001 as the learning rate for training the networks and LR = 0.0005 when optimizing the bandwidth parameter using NLL.

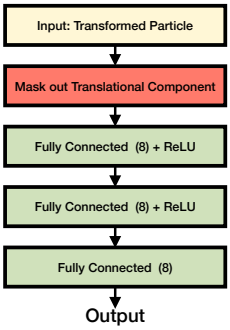
We apply gradient norm clipping, clipping the norm of the gradients to be ≤ 500.0. Further we limit the bandwidth for the angle dimension such that $\beta \leq 4.0$. We do this only for this task as non-MDPF methods were prone to learning an unreasonably large bandwidth resulting in a near uniform distribution for the angular part of the state.

Network Architectures

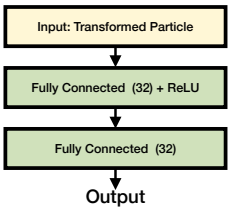
Fig. 2.32 gives details on the neural network architectures used for the dynamics and measurement models within the PF for the Deepmind maze tracking task.



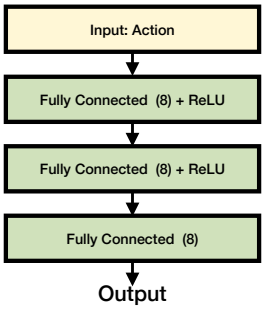
(a) Observation encoder.



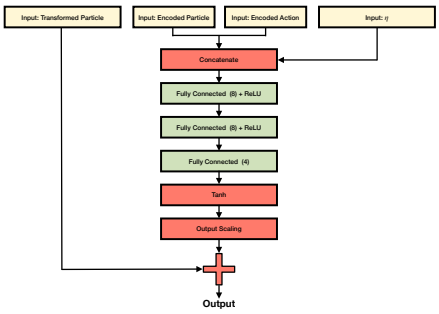
(b) Particle encoder for dynamics Model.



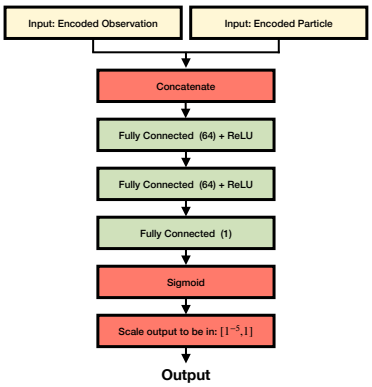
(c) Particle encoder for measurement model.



(d) Action encoder for dynamics Model.



(e) Dynamics model residual network. Output scaling scales the transnational dimensions to be within [-7, 7] and the unit vector representing angles to be within [-1, 1].



(f) Measurement model weights network.

Figure 2.32: Neural architecture for the networks within the dynamics and measurement models for the Deepmind maze tracking task.

House3D Tracking Task

In this section we give additional task specific information and network architecture details for the House3D tracking task.

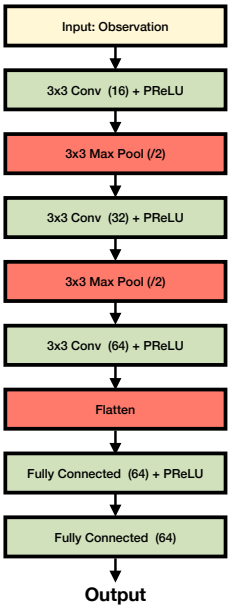
Additional Experiment Details

When training MDPF using the Negative Log-Likelihood (NLL) loss, we set the learning rate (LR) of the neural networks to be 0.0005 and the bandwidth learning rate to 0.00005. When training using MSE loss, we use 0.0001 and 0.00001 for the neural network and bandwidth learning rates respectively. For the discrete PF methods, we use LR = 0.0005 for training the neural network using the MSE loss. We use the same bandwidth for optimizing the bandwidth using the NLL loss when the neural networks are held fixed. For the LSTM, we use 0.001 as the learning rate for the LSTM layers and LR = 0.0005 when optimizing the bandwidth parameter using NLL.

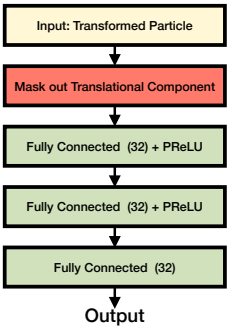
We apply gradient norm clipping, clipping the norm of the gradients to be ≤ 250.0 when training with NLL loss. When training with MSE loss, we disable gradient norm clipping.

Network Architectures

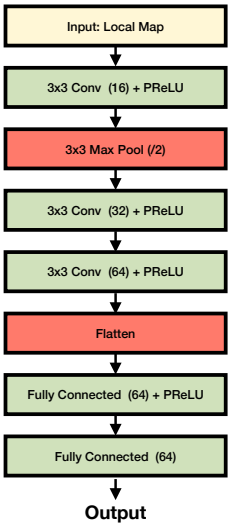
Fig. 2.32 gives details on the neural network architectures used for the dynamics and measurement models within the PF for the House3D tracking task.



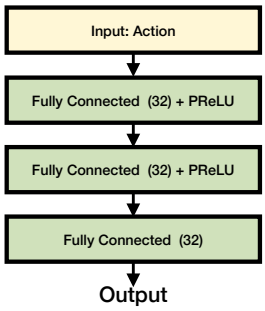
(a) Observation encoder.



(b) Particle encoder for dynamics Model.

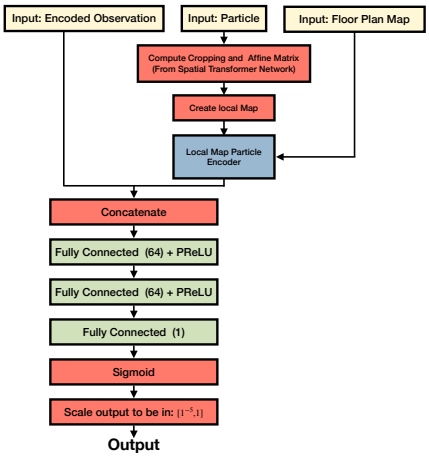


(c) Particle encoder for measurement model.



(d) Action encoder for dynamics Model.

(e) Dynamics model residual network. Output scaling scales the transnational dimensions to be within [-75, 75] and the unit vector representing angles to be within [-0.99, 0.99].



(f) Measurement model weights network.

Figure 2.33: Neural architecture for the networks within the dynamics and measurement models for the House3D tracking task.

LSTM Model Details

We train using LSTM [45] based recurrent neural network models for all tasks as a baseline comparison. For the LSTM, we use the same encoders as the PF methods. Fig. 2.34 shows the LSTM network architecture used for all tasks. We use the same output scaling the PF methods.

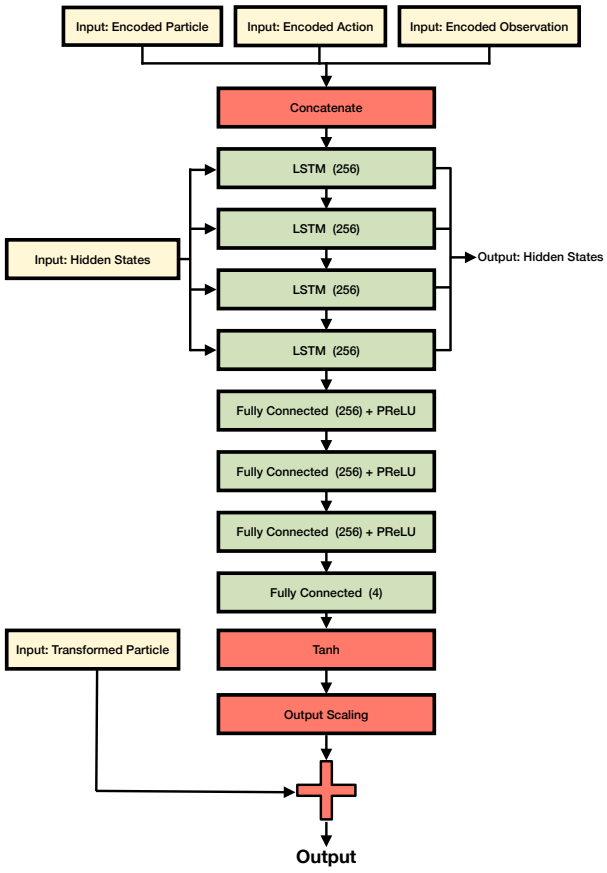


Figure 2.34: LSTM network architecture used for all tasks. The output is the 4D state which is then converted back into the 3D state using the inverse particle transform.

Chapter 3

End-to-end Learnable Particle Smoothers

In this chapter we build on our Mixture Density Particle Filter from chapter 2 to develop the first, to the best of our knowledge, end-to-end differentiable particle smoother which we name Mixture Density Particle Smoother (MDPS). MDPS uses both past and future observations to estimate a more accurate and robust posterior estimate of the state over time. MDPS combines two MDPFs, one running forward in time and one running backward in time, along with a novel combination step yielding an efficient inference algorithm. We then apply our MDPS to the difficult task of global localization [52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62] in real-world city-scale environments where observations are first-person view images from handheld and vehicle mounted cameras and the map is a planimetric representation of the city. MDPS shows dramatic accuracy improvements over MDPF as well as methods specifically tailored towards the global localization task [54, 59, 62].

3.1 Introduction

For challenging state estimation problems arising in domains like vision and robotics, particle-based representations attractively enable temporal reasoning about multiple posterior modes. A number of methods for end-to-end *particle filter* (PF) training have been proposed [59, 63, 7, 5, 6] to tackle such state estimation problems (see chapter 2). Particle smoothers offer the potential for more accurate offline data analysis by propagating information both forward and backward in time, but have classically required human-engineered dynamics and observation models. Extending our recent advances in discriminative training of particle filters (see chapter 2), we develop a framework for low-variance propagation of gradients across long time sequences when training particle smoothers.

In this chapter we focus on the application of global localization of the state of a moving vehicle using a city-scale map is challenging due to the large area, as well as the inherent ambiguity in urban landscapes, where many street intersections and buildings appear similar. Recent work on global localization [52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62] has typically localized each time point independently during training, sometimes followed by temporal post-processing, often with demanding requirements like near-exact external estimation of relative vehicle poses [62].

Learnable PFs are suitable for global localization because they can represent multi-modal posterior densities, propagate uncertainty over time, and learn models of real vehicle dynamics and complex sensors directly from data. However, most learnable PF methods (see chapter 2) have only been applied to simulated environments [63, 7], with only a few preliminary applications to real-world data [59, 5]. Further, particle filters only use past observations to estimate the current state. For offline inference from complete time series, more powerful particle smoothing (PS) algorithms [64, 65, 66, 67, 68, 69] may in principle perform better by integrating past and future data. But to our knowledge, recent advances

in end-to-end differentiable training of PFs have not been generalized to the more complex particle smoothing scenario, requiring error-prone human engineering of particle smoothing dynamics and observation models. Classical work on generative parameter estimation via particle smoothing [24] is limited to parametric models with few parameters.

In this chapter we develop a differentiable particle smoother that scales to complex models defined by deep neural networks. Our "two-filter" smoother integrates particle streams that are propagated forward and backward in time, while incorporating stratification and importance weights in the resampling step to provide low-variance gradient estimates for neural network dynamics and observation models. We apply our resulting *mixture density particle smoother* to the difficult task of city-scale global vehicle localization from real-world videos and maps, where we show significant accuracy improvements over state-of-the-art differentiable particle filter methods as well as methods specifically tailored to the global localization task.

3.2 Reduced Variance Particle Resampling

The stochastic nature of particle filter dynamics causes some particles to drift towards states with low posterior probability. These low-weight particles do not usefully track the true system state, wasting computational resources and reducing the expressiveness of the overall approximate posterior. In the extreme case, only a handful of particles contribute to the posterior density and further divergence of any remaining particles could cause the particle filter to degenerate.

Particle resampling offers a remedy by drawing a new uniformly weighted particle set $\hat{x}_t^{(:)}$ from $x_t^{(:)}$, with each particle duplicated (or not) proportional to its current weight $w_t^{(i)}$. To maintain an unbiased posterior while avoiding degeneracy, resampling redistributes proba-

bility mass to all particles with resampled particles have weight $\hat{w}_t^{(i)} = \frac{1}{N}$. Several methods for resampling exist, each with differing computation complexities and variance reduction properties.

3.2.1 Multinomial Resampling

The simplest and most used resampling method is *multinomial* resampling strategy [65, 70, 71] which chooses particles via discrete sampling with replacement:

$$\hat{x}_t^{(i)} = x_t^{(j)}, \qquad j \sim \text{Cat}(w_t^{(1)}, \ldots, w_t^{(N)}). \tag{3.1}$$

Multinomial resampling may be easily implemented [70] by drawing a continuous Unif$(0, 1)$ variable for each particle, and transforming these draws by the inverse *cumulative distribution function* (CDF) of particle weights. Though multinomial resampling is unbiased and trivial to implement, it introduces additional variance caused by random draws.

3.2.2 Stratified Resampling

Stratified resampling [65, 70, 71] reduces the variance of conventional multinomial resampling, by first partitioning the interval $(0, 1]$ into N sub-intervals $(0, \frac{1}{N}] \cup ... \cup (1 - \frac{1}{N}, 1]$. One uniform variable is then sampled within each sub-interval, before transforming these draws by the inverse CDF of particle weights (The same inverse CDF transformation as multinomial resampling is then used to generate the resampled particle index [70]).

Stratified resampling introduces negligible computational overhead while reducing sampling variance making training more robustly avoid local optima (see Fig. 3.3).

3.2.3 Residual Resampling

While other methods like *residual* resampling [72, 70, 73] have been proposed in the PF literature, this partially-deterministic approach is less robust than stratified resampling in our experiments (see Fig. 3.3).

Residual resampling can be implemented via 2 stages [70]. In the first stage, the minimum number of times each particle should be resampled is deterministically computed using the particle weights to produce a resampled particle set that is smaller than the desired resampled particle set size. Assuming N particles are to be resampled to produce a resampled particle set with M particles then the number of times a given particle is resampled $n_t^{[i]}$ is:

$$n_t^{[i]} = \lfloor N \cdot w_t^{[i]} \rfloor \tag{3.2}$$

Since $\sum_{i=1}^{N} n_t^{[i]} \neq M$, we must draw $R = M - \sum_{i=1}^{N} n_t^{[i]}$ additional particles to produce a resampled particle set of size M. In stage two, these remaining R particles are sampled via multinomial resampling but with modified weights to account for the particles resampled in stage one:

$$\hat{x}_t^{(i)} = x_t^{(j)}, \qquad j \sim \text{Cat}(\bar{w}_t^{(1)}, \ldots, \bar{w}_t^{(N)}), \qquad \bar{w}_t^{(k)} = \frac{(N \cdot w_t^{[i]}) - \lfloor N \cdot w_t^{[i]} \rfloor}{M - R} \tag{3.3}$$

Unfortunately residual resampling is not easily parallelizable onto modern GPU hardware and is thus much slower than multinomial or stratified resampling.

3.2.4 Integration into Resampling from Mixture Distributions

Reduced variance resampling methods such can be easily used to sample from mixture distributions. In the case of resampling from a continuous Gaussian mixture, in which all

components share a common standard deviation β, this resampling can equivalently be expressed

$$\hat{x}_t^{(i)} = \mu_t^{(i)} + \beta\eta_t^{(i)}, \qquad \text{where } \eta_t^{(i)} \sim N(0, I), \qquad \mu_t^{(i)} = x_t^{(j)} \tag{3.4}$$

where $\mu_t^{(i)}$ is a component mean selected with probability proportional to the component weights:

$$\mu_t^{(i)} = x_t^{(j)}, \qquad j \sim \text{Cat}(w_t^{(1)}, \ldots, w_t^{(N)}) \tag{3.5}$$

For our mixture density PF (see chapter 2) and our mixture density PS developed in this chapter, reduced variance resampling methods can be easily used during the particle resampling to boost performance. We incorporate stratified resampling as it is more stable and much faster than residual resampling while offering a performance improvement over multinational resampling.

3.3 From Filtering to Smoothing

Particle smoothers extend PFs to estimate the state posteriors $p(x_t|y_{1:T})$ given a full T-step sequence of observations. (To simplify equations, we do not explicitly condition on actions $a_{1:T}$ in the following two sections). Particle smoothers continue to approximate posteriors via a collection of particles $\overleftrightarrow{x}_t^{(1:N)}$ with associated weights $\overleftrightarrow{w}_t^{(1:N)}$, where we use bi-directional overhead arrows to denote smoothed posteriors. Classical particle smoothing algorithms, which are non-differentiable and typically assume human-engineered dynamics and likelihoods, fall into two broad categories.

3.3.1 Forward-Filtering, Backward Smoothing

Forward-Filtering, Backward Smoothing (FFBS) algorithms [66, 67] compute $p(x_t|y_{1:T})$ by factoring into forward filtering and backward smoothing components:

$$p(x_t|y_{1:T}) = \int p(x_t, x_{t+1}|y_{1:T})dx_{t+1} = \underbrace{p(x_t|y_{1:t})}_{\text{forward filtering}} \underbrace{\int \frac{p(x_{t+1}|y_{1:T})p(x_{t+1}|x_t)}{\int p(x_{t+1}|x_t)p(x_t|y_{1:t})}dx_{t+1}}_{\text{backward smoothing}}. \quad (3.6)$$

A natural algorithm emerges from Eq. (3.6), where a conventional PF first approximates $p(x_t|y_{1:t})$ for all times via particles $\overrightarrow{x}_t^{(1:N)}$ with weights $\overrightarrow{w}_t^{(1:N)}$. A backward smoother then recursively reweights the "forward" particles to account for future data, but does *not* change particle locations:

$$\overleftrightarrow{w}_t^{(i)} \propto \overrightarrow{w}_t^{(i)} \left(\sum_{j=1}^{N} \overleftrightarrow{w}_{t+1}^{(j)} \frac{p(\overrightarrow{x}_{t+1}^{(j)}|\overrightarrow{x}_t^{(i)})}{\sum_{k=1}^{N} \overrightarrow{w}_t^{(k)} p(\overrightarrow{x}_{t+1}^{(j)}|\overrightarrow{x}_t^{(k)})} \right). \quad (3.7)$$

Because FFBS sets $\overleftrightarrow{x}_t^{(i)} = \overrightarrow{x}_t^{(i)}$, it is only effective when filtered state posteriors $p(x_t|y_{1:t})$ substantially overlap with smoothed posteriors $p(x_t|y_{1:T})$ [67]; performance deteriorates when future data is highly informative. FFBS also requires explicit evaluation, not just simulation, of the state transition dynamics $p(x_{t+1}|x_t)$, which is not tractable for dynamics parameterized as in Eq. (2.7).

3.3.2 Two Filter Smoothing

Two Filter Smoothing (TFS) algorithms [64, 66, 67] instead express the smoothed posterior as a normalized product of distinct forward-time and backward-time filters:

$$p(x_t|y_{1:T}) = \frac{p(x_t|y_{1:t})p(y_{t+1:T}|x_t)}{p(y_{t+1:T}|y_{1:t})} \propto p(x_t|y_{1:t})p(y_{t+1:T}|x_t). \quad (3.8)$$

Here $p(x_t|y_{1:t})$ may be approximated by a standard PF, and $p(y_{t+1:T}|x_t)$ is the so-called *backward information filter* [67, 66] defined as

$$p(y_{t:T}|x_t) = \int p(y_{t+1:T}|x_{t+1})p(x_{t+1}|x_t)p(y_t|x_t)dx_{t+1}. \tag{3.9}$$

Because $p(y_{t:T}|x_t)$ is a likelihood function rather than a probability density in x_t, and it is possible that $\int p(y_{t:T}|x_t)dx_t = \infty$. This is not an issue when $p(y_{t:T}|x_t)$ is computed analytically as in Kalman smoothers for Gaussian models [74], but particle-based methods can only hope to approximate finite measures. Bresler [64] addresses this issue via an *auxiliary* probability measure $\gamma_t(x_t)$:

$$p(y_{t:T}|x_t) \propto \frac{\tilde{p}(x_t|y_{t:T})}{\gamma_t(x_t)}, \qquad \tilde{p}(x_{t:T}|y_{t:T}) \propto \gamma_t(x_t)p(y_t|x_t)\prod_{s=t+1}^{T} p(x_s|x_{s-1})p(y_s|x_s). \tag{3.10}$$

From Eqs. (3.8,3.10), the smoothed posterior is a reweighted product of forward and backward filters:

$$p(x_t|y_{1:T}) \propto \frac{\overbrace{p(x_t|y_{1:t})}^{\text{forward filtering}}\ \overbrace{\tilde{p}(x_t|y_{t+1:T})}^{\text{backward filtering}}}{\gamma_t(x_t)}. \tag{3.11}$$

This suggests an algorithm where two PFs are run on the sequence independently, one forward and one backward in time, to compute forward particles $\{\overrightarrow{x}_t^{(1:N)}, \overrightarrow{w}_t^{(1:N)}\}$ and backward particles $\{\overleftarrow{x}_t^{(1:N)}, \overleftarrow{w}_t^{(1:N)}\}$. Because continuously sampled forward and backward particle sets will not exactly align, classic TFS integrates these two filters by rewriting Eq. (3.11) as follows:

$$p(x_t|y_{1:T}) \propto \frac{p(y_t|x_t)\tilde{p}(x_t|y_{t+1:T})\int p(x_t|x_{t-1})p(x_{t-1}|y_{1:t-1})dx_{t-1}}{\gamma_t(x_t)}. \tag{3.12}$$

This yields a particle re-weighting approach where backward filter particles $\overleftarrow{x}_t^{(1:N)}$ are re-weighted using the forward filter particle set, to produce the final smoothed particle weights

$\overleftrightarrow{w}_t^{(1:N)}$:

$$\overleftrightarrow{w}_t^{(i)} \propto \overleftarrow{w}_t^{(i)} \sum_{j=1}^{N} \overrightarrow{w}_{t-1}^{(j)} \frac{p(\overleftarrow{x}_t^{(i)} | \overrightarrow{x}_{t-1}^{(j)})}{\gamma_t(\overleftarrow{x}_t^{(i)})}, \qquad \sum_{i=1}^{N} \overleftrightarrow{w}_t^{(i)} = 1. \tag{3.13}$$

Conventional TFS set $\overleftrightarrow{x}_t^{(1:N)} = \overleftarrow{x}_t^{(1:N)}$, which similar to FFBS, makes performance heavily dependent on significant overlap in support between $p(x_t|y_{t+1:T})$ and $p(x_t|y_{1:T})$. Like FFBS, TFS also restrictively requires evaluation (not just simulation) of the state transition dynamics.

3.4 Mixture Density Particle Smoothers

Our novel *Mixture Density Particle Smoother* (MDPS, Fig. 3.1) can be seen as a differentiable TFS, where the forward and backward filters of Eq. (3.11) are defined as MDPFs (chapter 2.5). Using discriminative differentiable particle filters (MDPFs) within the TFS frameworks, and replacing Eq. (3.12) with an importance-weighted integration of forward and backward particles, enables an effective and end-to-end differentiable particle smoother.

We begin by rewriting Eq. (3.11) as

$$p(x_t|y_{1:T}) \propto \frac{p(y_t|x_t)p(x_t|y_{1:t-1})\tilde{p}(x_t|y_{t+1:T})}{\gamma_t(x_t)}, \tag{3.14}$$

where the forward and backward filters no longer condition on the current observation. This allows for functionally identical MDPFs to be used for both directions, simplifying implementation. MDPFs parameterize state posteriors as continuous kernel density mixtures:

$$p(x_t|y_{1:t-1}) = \sum_{i=1}^{N} \overrightarrow{w}_t^{(i)} K(x_t - \overrightarrow{x}_t^{(i)}; \overrightarrow{\beta}), \qquad p(x_t|y_{t+1:T}) = \sum_{i=1}^{N} \overleftarrow{w}_t^{(i)} K(x_t - \overleftarrow{x}_t^{(i)}; \overleftarrow{\beta}). \tag{3.15}$$

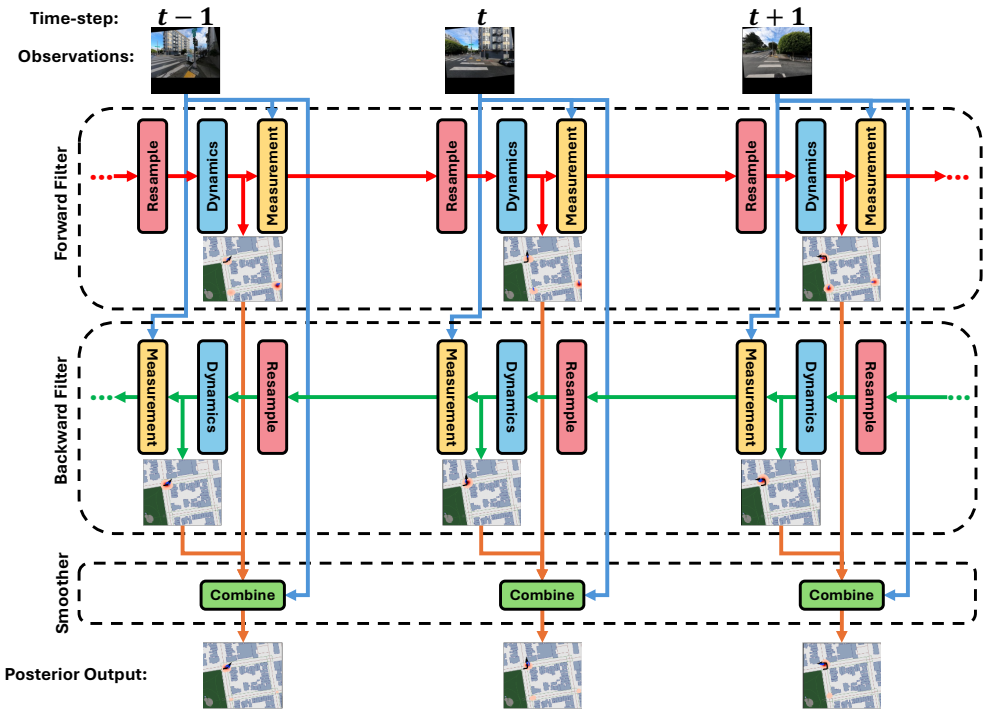


Figure 3.1: Our mixture density particle smoother method showing the forward and backward particle filters, which are integrated (via learned neural networks) to produce a smoothed mixture posterior.

Unlike discrete probability measures, these continuous mixture distributions can be combined via direct multiplication to give a smoothed posterior mixture containing N^2 components, one for each pair of forward and backward particles. For this product integration of forward and backward filters, the normalizing constants for all pairs of kernels must be explicitly computed to correctly account for the degree to which hypotheses represented by forward and backward particles are consistent. These normalization constants are tractable for some simple kernels including Gaussians [75], but more complex for the other cases such as von Mises kernels of orientation angles [76, 77].

Direct mixture multiplication eliminates the need to evaluate the dynamics model, as in classic TFS, but introduces significant overhead due to the quadratic scaling of the number of mixture components. To address this issue, our MDPS uses importance sampling where the smoothed posterior is defined by $M \ll N^2$ particles drawn from a mixture of the filter

posteriors:

$$\overleftrightarrow{x}_t^{(i)} \sim q(x_t) = \frac{1}{2}p(x_t|y_{1:t-1}) + \frac{1}{2}p(x_t|y_{t+1:T}), \qquad i = 1, \ldots, M. \tag{3.16}$$

By construction, this proposal will include regions of the state space that lie within the support of *either* $p(x_t|y_{1:t-1})$ or $p(x_t|y_{t+1:T})$, improving robustness. Our experiments set $M = 2N$, drawing N particles from each of the filtered and smoothed posteriors. Given true dynamics and likelihood models, importance sampling may correct for the fact that smoothed particles are drawn from a mixture rather than a product of filtered densities, as well as incorporate the local observation:

$$\overleftrightarrow{w}_t^{(i)} \propto \frac{p(y_t|\overleftrightarrow{x}_t^{(i)})p(\overleftrightarrow{x}_t^{(i)}|y_{1:t-1})\tilde{p}(\overleftrightarrow{x}_t^{(i)}|y_{t+1:T})}{\gamma_t(\overleftrightarrow{x}_t^{(i)})q(\overleftrightarrow{x}_t^{(i)})}, \qquad \sum_{i=1}^{M} \overleftrightarrow{w}_t^{(i)} = 1. \tag{3.17}$$

To more easily train a discriminative PS, rather than estimating each term in Eq. (3.17) separately, we directly parameterize their product via a feed-forward neural network $l(\cdot)$:

$$\overleftrightarrow{w}_t^{(i)} \propto \frac{l(\overleftrightarrow{x}_t^{(i)}; y_t, p(\overleftrightarrow{x}_t^{(i)}|y_{1:t-1}), \tilde{p}(\overleftrightarrow{x}_t^{(i)}|y_{t+1:T}))}{q(\overleftrightarrow{x}_t^{(i)})}, \qquad \sum_{i=1}^{M} \overleftrightarrow{w}_t^{(i)} = 1. \tag{3.18}$$

The posterior weight network $l(\cdot)$ scores particles based on agreement with y_t, as well as consistency with the forward and backward filters, and implicitly accounts for the auxiliary distribution $\gamma_t(\cdot)$. To allow state prediction and compute the training loss, the smoothed posterior is approximated as:

$$p(x_t|y_{1:T}) \approx m(x_t|\overleftrightarrow{x}_t^{(:)}, \overleftrightarrow{w}_t^{(:)}, \overleftrightarrow{\beta}) = \sum_{i=1}^{N} \overleftrightarrow{w}_t^{(i)} K(x_t - \overleftrightarrow{x}_t^{(i)}; \overleftrightarrow{\beta}), \tag{3.19}$$

where $\overleftrightarrow{\beta}$ is a learned, dimension-specific bandwidth parameter. The full MDPS algorithm can be seen in Fig. 3.2.

Mixture Density Particle Smoothing:

Given observations $y_{1:T}$ and actions $a_{1:T}$ with N being the number of particles

1. Compute forward filter particle sets using Mixture Density Particle Filtering with $y_{1:T}$ and $a_{1:T}$:

$$\{\overrightarrow{x}^{(:)}_{1:T}, \overrightarrow{\widetilde{w}}^{(:)}_{1:T}, \overrightarrow{w}^{(:)}_{1:T}, \overrightarrow{\beta}\}$$

2. Compute backward filter particle sets using Mixture Density Particle Filtering with $y_{T:1}$ and $a_{T:1}$ (with time reversed):

$$\{\overleftarrow{x}^{(:)}_{1:T}, \overleftarrow{\widetilde{w}}^{(:)}_{1:T}, \overleftarrow{w}^{(:)}_{1:T}, \overleftarrow{\beta}\}$$

3. For $t = 1, ..., T$ **and** $i = 1, ..., N$

   (a) Define:

$$q(x) = \frac{1}{2} m(x; \overrightarrow{x}^{(:)}_{1:T}, \overrightarrow{\widetilde{w}}^{(:)}_{1:T}, \overrightarrow{\beta}) + \frac{1}{2} m(x; \overleftarrow{x}^{(:)}_{1:T}, \overleftarrow{\widetilde{w}}^{(:)}_{1:T}, \overleftarrow{\beta})$$

   (b) Sample particles:

$$\overleftrightarrow{x}^{(i)}_t \sim q(x)$$

   (c) Compute weights

$$\overleftrightarrow{w}^{(i)}_t = \frac{l\Big(\overleftrightarrow{x}^{(i)}_t; y_t, m(\overleftrightarrow{x}^{(i)}_t; \overrightarrow{x}^{(:)}_{1:T}, \overrightarrow{\widetilde{w}}^{(:)}_{1:T}, \overrightarrow{\beta}), m(\overleftrightarrow{x}^{(i)}_t; \overleftarrow{x}^{(:)}_{1:T}, \overleftarrow{\widetilde{w}}^{(:)}_{1:T}, \overleftarrow{\beta})\Big)}{q(\overleftrightarrow{x}^{(i)}_t)}$$

4. **Output:** $\{\overleftrightarrow{x}^{(:)}_{1:T}, \overleftrightarrow{w}^{(:)}_{1:T}, \overleftrightarrow{\beta}\}$

Figure 3.2: The Mixture Density Particle Smoother algorithm

3.4.1 Training Loss and Gradient Computation

We discriminatively train our MDPS by minimizing the negative log-likelihood of the true state sequence:

$$\mathcal{L} = \frac{1}{T} \sum_{t \in T} -\log(m(x_t | \overleftrightarrow{x}^{(:)}_t, \overleftrightarrow{w}^{(:)}_t, \overleftrightarrow{\beta})). \tag{3.20}$$

During training, the IWSG estimator of Eq. (2.16) provides unbiased estimates of the gradients of the forward and backward resampling steps. We may similarly estimate gradients of the mixture resampling (3.16) that produces smoothed particles, enabling the first end-to-end differentiable PS:

$$\nabla_\phi \overleftrightarrow{w}_t^{(i)} \propto \frac{\nabla_\phi l(\overleftrightarrow{x}_t^{(i)}; y_t, p(\overleftrightarrow{x}_t^{(i)}|y_{1:t-1}), \tilde{p}(\overleftrightarrow{x}_t^{(i)}|y_{t+1:T}))}{q(\overleftrightarrow{x}_t^{(i)})}. \tag{3.21}$$

3.4.2

Training Details Because the smoother weights of Eq. (3.18) cannot be effectively learned when filter parameters are random, we train MDPS via a three-stage procedure. In stage 1, the forward and backward PFs are trained separately (sharing only parameters for the encoders, see Fig. 3.1) to individually predict the state. In stage 2, the PFs are frozen and the particle smoother measurement model $l(\cdot)$ of Eq. (3.18) is trained. In stage 3, all models are unfrozen and trained jointly to minimize the loss in the MDPS output state posterior predictions of the true states. The forward MDPF, backward MDPF, and MDPS posterior each have separate kernel bandwidths ($\overrightarrow{\beta}$, $\overleftarrow{\beta}$, $\overleftrightarrow{\beta}$); that are jointly learned with the dynamics and measurement models. We randomly resample a stochastic subset of the training sequences for each step, and adapt learning rates via the Adam [78] optimizer.

3.4.3 Computational Requirements

At training time, to allow unbiased gradient propagation, MDPS computes importance weights for each particle during resampling. For N particles and T time-steps, this requires $\mathcal{O}(TN^2)$ operations. At inference time, importance weighting is not needed as the particle weights can simply be set as uniform, and resampling only requires $\mathcal{O}(TN)$ operations. All phases of our MDPS scale linearly with N at test time, in contrast with other differentiable

relaxations such as OT-PF [63], which requires $\mathcal{O}(TN^2)$ operations for both training and inference.

3.5 Experiments

We evaluate our mixture density particle smoother on the synthetic bearings-only tracking task from chapter 2.6.1, as well as on real-world city-scale global localization. For all tasks, we estimate the MDPF/MDPS posterior distributions of a 3D (translation and angle) state $x_t = (x, y, \theta)$, using Gaussian kernels for the position dimensions, and von Mises kernels for the angular dimensions of the state posterior mixtures.

3.5.1 Bearings Only Tracking Task

To allow comparison to prior discriminative PFs as well as our mixture density particle filter from chapter 2, we use the same bearings-only tracking task as chapter 2.6.1 where the 3D state of a variable-velocity synthetic vehicle is tracked via noisy bearings from a fixed-position radar. 85% of observations are the true bearing plus von Mises noise, while 15% are uniform outliers:

$$y_t \sim \alpha \cdot \text{Uniform}(-\pi, \pi) + (1 - \alpha) \cdot \text{VonMises}(\psi(x_t), \kappa),$$

where $\psi(x_t)$ is the true bearing and $\alpha = 0.15$.

Train and evaluation sequences have length $T = 50$. Unlike in chapter 2, we do not use truncated BPTT [47] since we find that careful initialization of the bandwidth renders truncated BPTT unnecessary. Filtering particles are initialized as the true state with added Gaussian noise, while MDPF-Backward (and the MDPS backwards filter) are initialized with uni-

formly sampled particles to mimic datasets where often only the starting state is known. More details can be found in the Appendix.

We compare our MDPS methods to several existing differentiable particle filter baselines as well as our MDPS, but no differentiable particle smoother baseline exists. Instead, we implement the classic FFBS [66, 67] algorithm (chapter 3.3), which assumes known dynamics and measurement models. Since FFBS is not differentiable, we learn the dynamics model using the dataset true state pairs $\{x_{t-1}, x_t\}$ outside of the FFBS algorithm. In order to simulate from and evaluate the state transition dynamics, as needed by the FFBS, we parameterize the dynamics model to output a mean and use a fixed bandwidth parameter (tuned on validation data) to propose new particles. We also use the true observation likelihood as the measurement model, instead of a learned approximation; this boosts FFBS performance.

Results

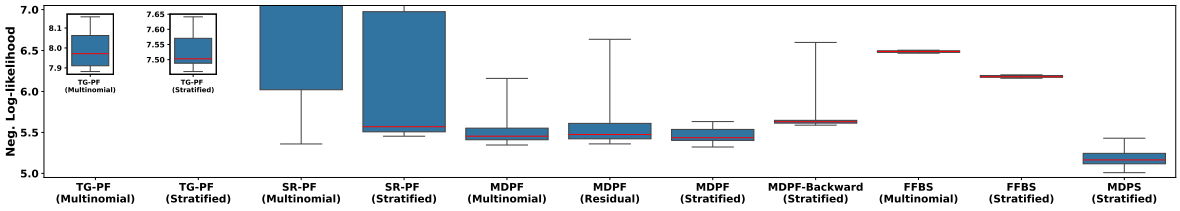


Figure 3.3: Box plots showing median (red line), quartiles (blue box), and range (whiskers) over 11 training runs for Bearings-Only tracking. We boost the robustness of the top-performing MDPF by incorporating variance-reduced stratified resampling; residual resampling is both slower and less effective. Stratified resampling provides larger advantages for the less-sophisticated TG-PF [5] and SR-PF [7] gradient estimators, but these baselines remain inferior to MDPF. Our MDPS substantially improves on all PFs by incorporating both past and future observations when computing posteriors. Classic FFBS particle smoothers [66, 67] have poor performance, even when provided the true likelihoods (rather than a learned approximation), showing the effectiveness of our end-to-end learning of particle proposals and weights. Forward PFs are initialized with noisy samples of the true state, while MDPF-Backward (the backwards-time PF component of MDPS) is initialized by sampling uniformly from the state space.

In Fig. 3.3 we show statistics of performance over 11 training and evaluation runs for each

method. We compare to TG-PF [5], SR-PF [7], the classical FFBS [66, 67], and MDPF but not OT-PF [63] as it is prohibitively slow to train and yields poor results when compared to other methods as seen in chapter 2.6. Interestingly, MDPF outperforms SR-PF and TG-PF even when the initial particle set is drawn uniformly from the state space as in MDPF-Backward.

By incorporating more temporal data, MDPS substantially outperforms MDPF. Even when unfairly provided the true observation likelihood, FFBS performs poorly since particles are simply re-weighted (not moved) by the backward smoother. This inflexibility, and lack of end-to-end learning, makes FFBS less robust to inaccuracies in the forward particle filter.

Comparing Resampling Methods

We are the first to compare resampling variants in the context of modern discriminative PFs. Stratified resampling substantially improves TG-PF and SR-PF performance, but only modestly improves the worst-performing MDPF runs. This may be because even with basic multinomial resampling, the lower-variance MDPF gradients dramatically outperform all TG-PF and SR-PF variants. Residual resampling performs worse than stratified resampling, and is also much slower since it cannot be easily parallelized on GPUs, so we do not consider it for other datasets.

3.5.2 City Scale Global Localization Task

Our global localization task is adopted from Sarlin et al. [62], where we wish to estimate the 3D state (position and heading) of a subject (person/bike/vehicle) as it moves through real-world city-scale environments. Observations are gravity-aligned first-person images, actions are noisy odometry, and a 2D planimetric map is provided to help localize globally. We use

the Mapillary Geo-Localization [62] and KITTI [79] datasets to compare our MDPS method to MDPF as well as other methods specifically designed for the global localization task, which are not sequential Monte Carlo methods.

Our global localization task is distinct from local localization systems, which aim to track subject positions relative to their starting position, instead of in relation to the global map origin. Visual SLAM systems [13] almost exclusively solve the local localization task, using the starting position as the origin of their estimated map. If a map is provided, then just the localization part of Visual SLAM can be run, but detailed visual or 3D maps of the environment are needed. These have prohibitive memory requirements at city-scales, and need constant updating as the visual appearance of the environment changes (e.g., with the weather/seasons) [62]. Hybrid place recognition with localization also requires detailed visual or 3D maps [80]. In our experiments, we instead seek to use planimetric maps for global localization, which are compact and robust to environment changes. It is not obvious how to apply SLAM/Hybrid place recognition systems to this type of map.

Existing Works in Global Localization

Several existing methods have been proposed for the global localization task and can broadly be categorized into 3 groups:

Retrieval methods [52, 53, 54, 55, 56, 57, 58] rely on latent vector similarity where map patches and the observation are encoded into a common latent state space before doing a vector similarity search. These methods are trained using a contrastive loss [53, 81] that forces the observation encoding to be similar to its corresponding map patch encoding, while being dissimilar to other patch and observation encodings. Accuracy depends on map patch extraction density and patch similarity; if similar patches are mapped to near-identical encodings, performance suffers. Zhou et al. [59] extend this framework using a

non-differentiable PF, where a retrieval-based measurement model is trained outside the PF framework, and non-learned dynamics are fixed to actions with added Gaussian noise.

Refinement methods [60, 61] refine an initial position estimate via expensive optimization, by maximizing the alignment of features extracted from the observation and map. Due to the extreme non-linearity of this objective, refinement methods require an accurate initial estimate to converge to the correct solution, if at all. This prevents their use for city-scale global localization where the state of the system is unknown at runtime.

Dense Search [62] extracts *birds-eye-view* (BEV) features from the observed images via geometric projection (see Fig. 3.4), before applying a dense search to align BEV features with extracted map features. Heuristic alignment probabilities may be produced by tracking alignment values during search, and applying a softmax operator. Higher discretization density boosts accuracy, but requires significantly more memory and compute. Temporal information can be used by warping probability volumes onto the current time-step, but this requires near-exact relative poses which Sarlin et al. [62] determine via an expensive, black-box Visual-Inertial SLAM system [13] adding significantly to the computation requirements.

Applying baselines to city-scale environments. Due to memory constraints, dense search over the whole map is not possible (approx. 809 GB is needed for $T = 100$ length sequence at 0.5m per pixel resolution). We therefore offer 2 methods for applying this dense search at city scales. *Ground Truth (GT) Cheat Method:* using the ground truth state, we extract a small region from the map in which we do dense search. This greatly reduces the search space, saving memory but also greatly (and *artificially*) improves performance. *Sliding Window Method:* At $t = 1$ a small region extracted around the true state is densely searched. At subsequent time-steps, the best alignment from the dense search of the previous time-step is propagated using a_t and used as the center of a new small region which is then searched. Retrieval methods tend to fail when applied to large environments with Zhou et al. [59] even limiting the search to patches on known road networks. To address this, we limit

the search space of Retrieval like in Dense Search using the GT Cheat and Sliding Window techniques, considering map patches within small regions.

3.5.3 Mapillary Geo-Localization (MGL) Dataset

In the Mapillary Geo-Localization (MGL) [62] dataset, images sequences are captured from handheld or vehicle-mounted cameras as a subject (person/bike/car) roams around various European and U.S. cities. Observations are set as 90° Field-of-View images in various viewing directions, with actions being noisy odometry. A planimetric map of the environment is also provided via the OpenStreetMap platform [82] at 0.5 meter/pixel resolution. We generate custom training, validation, and test splits to create longer sequences with $T = 100$ steps. For particle-based methods, we use stratified resampling and set the initial particle sets to be the true state with added noise. More details are in the Appendix.

Implementation Detail

For MDPF and our MDPS, we set the dynamics model to a multi-layer perceptron (MLP) network. The measurement model incorporates BEV features [62] and map features as illustrated in Fig. 3.4. The smoother measurement model incorporates additional inputs via an extra MLP. Memory constraints prevent Dense Search in city-scale environments, so we consider two methods to limit the search space. A sliding window limits the search space to a $256m \times 256m$ area that is recursively re-centered around the best position estimate at $t-1$, propagated to t using a_t. We can also limit the search space to a $256m \times 256m$ area containing the true state, though this *artificially* increases performance. We similarly limit the search space for Retrieval, which performs poorly in large environments; Zhou et al. [59] even limit the vector search to known road networks.

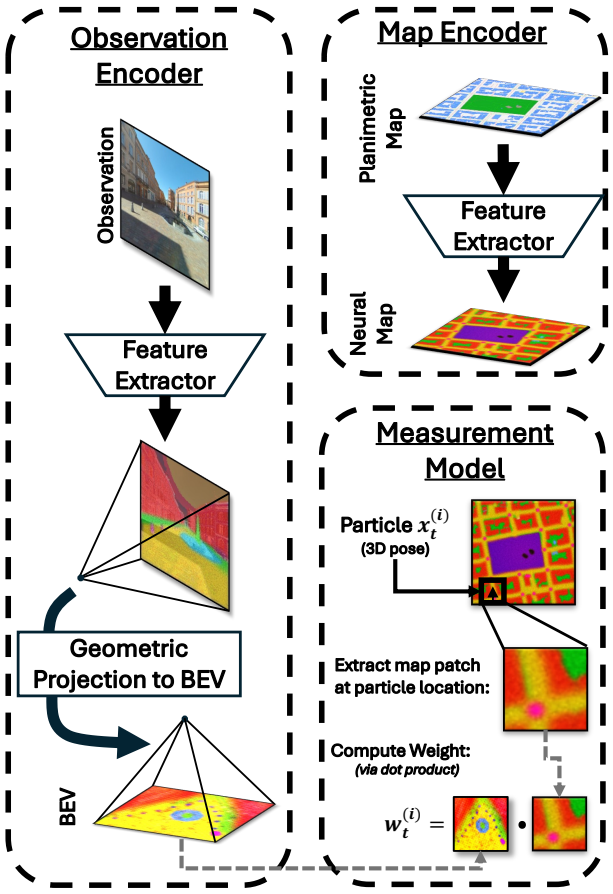


Figure 3.4: Feature encoders and measurement model used for global localization. First-person camera views are encoded into a Birds-Eye-View (BEV) feature map by extracting features before applying a geometric projection [62]. Map features are extracted via a feed-forward encoder, and un-normalized particle weights are computed as an inner product between BEV features and features of a local map extracted from the global map at the particle location.

Top 3 Mode Finding via Non-Maximal Suppression

Due to multi-modality of the posterior estimate, simply extracting the top mode to evaluate errors is not a good gauge of performance. Instead we to extract the top-3 modes from the posterior density for evaluation. This can be easily achieved via a non-maximal suppression scheme where modes are extracted before particles around those modes are deleted. Specifically after extracting the top mode, the distance of all particles to that top mode is calculated. Particle within some threshold of the top mode are deleted from the particle set and the weights of all remaining particles are re-normalized to admit a valid probability density after deletion. The next top mode is then extracted and the deletion process repeated until a total of 3 modes are extracted. In our implementation, we delete particles that are within 5m and 30° of the top mode.

For methods that admit a discrete probability volume (such as Dense Search and Retrieval), we suppress the values of all probability cells within 5m and 30° of the top mode during the deletion step. Extracting the top mode can be simply achieved by finding the maximum value within the discrete probability volume.

Results

We compare our MDPS to MDPF (see chapter 2.5), Dense Search [62], Retrieval [54] implemented as detailed by [62], and Retrieval (PF) [59], reporting the position and rotation recall in Fig. 3.5. Due to ambiguity in large city environments (e.g., intersections can look very similar), estimated state posteriors can be multi-modal (see Fig. 3.6), and thus simply reporting accuracy using the highest probability mode does not fully characterize performance. We thus also extract the top-three modes using non-maximal suppression (see Appendix), and report accuracy of the best mode. Interestingly, MDPF and MDPS give dramatic improvements over baselines engineered specifically for this task, highlighting the usefulness of

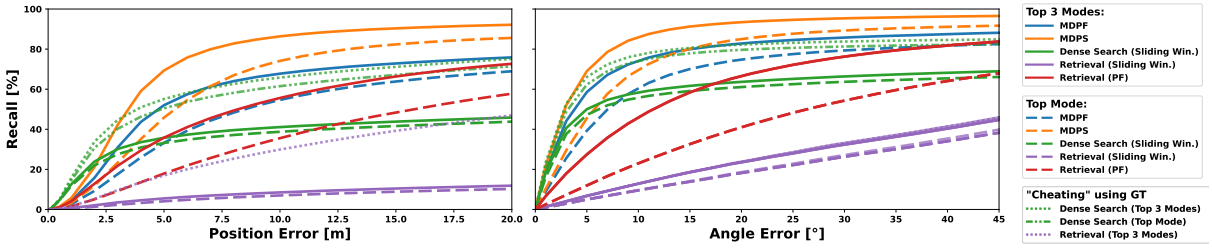


Figure 3.5: Position and angle error recall using the MGL [62] dataset. Recall is computed with the top posterior mode as well as with the best of the top-3 posterior modes, extracted via non-maximal suppression. As expected, Retrieval [54] methods do poorly due to their lack of discrimination power between neighboring map patches. Dense search [62] does better by using fine map details during localization, but it requires a ground truth hint ("Cheating" with GT, which artificially improves performance) to work well at city-scale environments. Retrieval (PF) [59] uses unlearned state dynamics, which proves useful, but still suffers from the poor discriminative ability of retrieval. In contrast, MDPF uses end-to-end learned dynamics and measurement models, allowing for good performance but suffering from only using past information when estimating posterior densities. Our MDPS is able to learn similar strong dynamics and measurement models as MDPF, and also incorporates future as well as past information to achieve a more accurate posterior density and thus higher recall.

end-to-end training. MDPS outperforms MDPF by using the full sequence of data to reduce mode variance, and discard incorrect modes as illustrated in Fig. 3.6.

Informative dynamics models boost performance, as demonstrated by the MDPS and MDPF results in Fig. 3.5. We visualize the learned dynamics of the MDPS forward filter in Fig. 3.7. Good dynamics models keep particles densely concentrated in high-probability regions, while also including diversity to account for sometimes-noisy actions. This enables learning of more discriminative measurement models, since training encourages the weights model to disambiguate nearby particles.

Number of Particles Ablation Study

A key hyper-parameter for particle filters and smoothers is the number of particles to use at inference time. Using more particles increases performance as shown in Fig. 3.8 but performance can quickly plateau. As seen in Fig. 3.6, effective models tend to concentrate

Figure 3.6: Example trajectories from the MGL dataset with observations shown in the top row. We show the current true state and state history (black arrow and black line), the estimated posterior density of the current state (red cloud, with darker being higher probability) and the top 3 extracted modes (blue arrows) for the MDPS as well as its forward and backward MDPFs. Due to ambiguity at early time-steps, MDPF is unable to resolve the correct intersection, and instead places probability mass at multiple intersections. By fusing both forward and backward filters, our MDPS resolves this ambiguity with probability mass focused on the correct intersection. Furthermore, MDPS provides a tighter posterior density than either MDPF-Forward or MDPF-Backward.

particles densely in likely regions of the state space. By using more particles, the particle density of these likely regions is increased but with diminishing returns. Each additional particle within the dense regions will vary only slightly from its neighbors, minimally adding to the particle set diversity. Further using more particles increases computation and memory requirements making smarter models which are more particle efficient, such as our MDPS, attractive for real world deployment

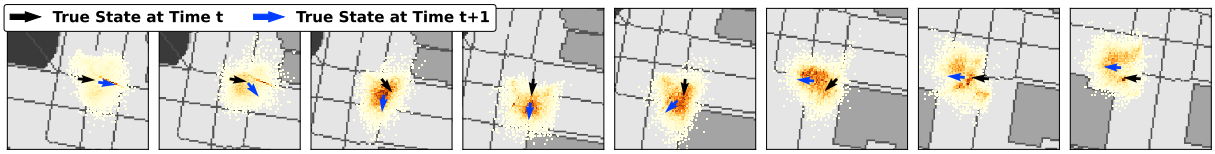


Figure 3.7: Learned dynamics from the forward filter of MDPS trained on the MGL dataset. Density cloud illustrates density of particles after applying dynamics while marginalizing actions. MDPS clearly learns informative, non-linear dynamics models which aid in state posterior estimation.

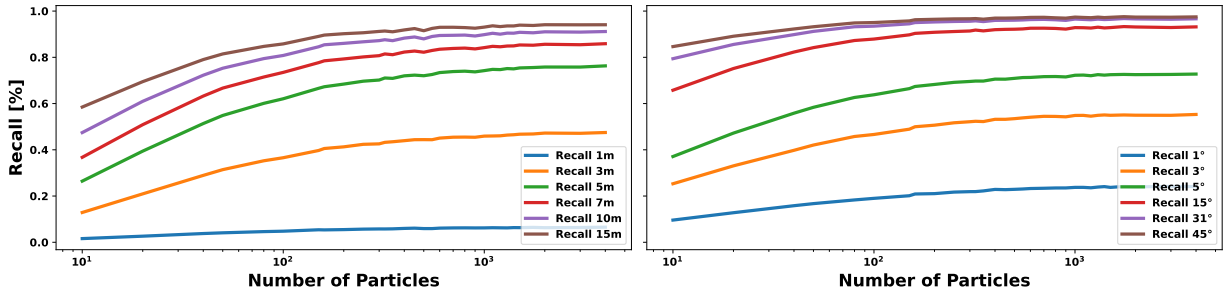


Figure 3.8: Recall of position and angle for MDPS using the MGL dataset [62] with varying numbers of particles at inference time. Here we specify the number of particles N used for the forward and backward filters of MDPS. The final MDPS posterior density is defined by $2N$ particles as described in chapter. 3.4. Interestingly, performance plateaus quickly as we increase the number of particles implying MDPS's ability to use particles smartly and efficiently, allowing for fewer particles to be used, lowering the computation and memory requirements neeeded for effective models.

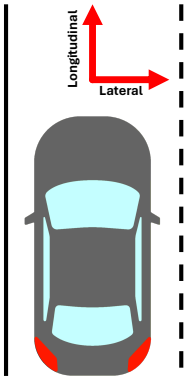


Figure 3.9: Lateral and longitudinal errors are in the vehicle frame of reference.

3.5.4 KITTI Dataset

We also evaluate our MDPS method for the global localization task using the KITTI [79] dataset, where observations are forward-facing camera images from a moving vehicle. We augment this datatset with noisy odometry computed from the ground truth states and use the default *Train*, *Test1*, and *Test2* splits for training, validation, and evaluation respectively. Like MGL, a planimetric map of the environment is provided via the OpenStreetMap platform [82] at 0.5 meter/pixel resolution. Due to the small size of the KITTI dataset, we pre-train all methods using MGL before refining on KITTI, using the same network

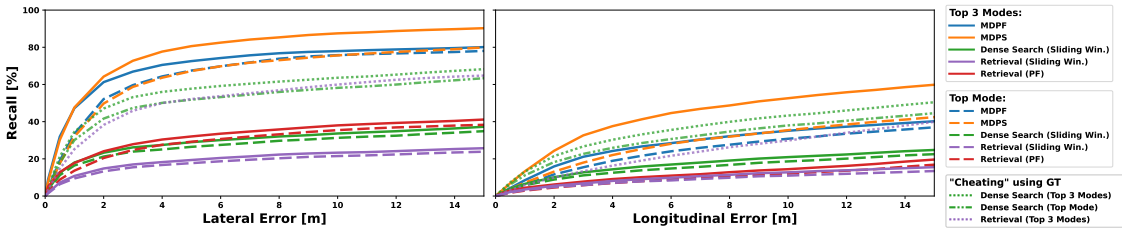


Figure 3.10: Position recall versus error for the KITTI [79] dataset. Recall is computed with the top posterior mode as well as with the best of the top-3 posterior modes. Longitudinal localization performance is poor for all methods due to lack of visual features. MDPF offers dramatic improvements for lateral error over Retrieval [54], Retrieval (PF) [59] and Dense Search [62] baselines, even when these baselines are constrained to operate around the ground truth state ("Cheating" with GT). For longitudinal recall, methods using "Cheating" with GT have good performance because they are *artificially* constrained to be near the true state, and thus have significantly less position ambiguity along the roadway. MDPS offers further improvements over MDPF as it maintains a more diverse set of posterior modes, instead of prematurely collapsing to incorrect modes.

architectures as was used for the MGL dataset. See Appendix for details.

Results

Due to the forward-facing camera, the observation images lack visual features for useful localizing along the roadway, therefore we decouple the position error into lateral and longitudinal errors (see Fig. 3.9 when reporting recalls in Fig. 3.10). Understandably, all methods have larger longitudinal error than lateral error. MDPF and MDPS offer similar Top 3 mode performance for small lateral errors (under 2 meters) while significantly outperforming all other methods. When the lateral error is greater than 2 meters, MDPS sees a performance gain as it maintains a more diverse set of posterior modes, whereas MDPF prematurely collapses the posterior density to incorrect modes.

3.6 Limitations

Like all particle-based methods, our MDPS suffers from the *curse of dimensionality* [83] where particle sparsity increases as the dimension of the state space increases, reducing expressiveness of state posteriors. More effective use of particles via smarter dynamics and measurement models, as enabled by end-to-end MDPS training, can reduce but not eliminate these challenges. Regardless, application of MDPS to higher dimensional problems would require an increase in the number of particles used, increasing computation and memory demands.

MDPS, like all smoothing algorithms, is strictly an offline method as it requires the full observation sequence to be present at inference time. Unlike filtering algorithms it cannot be run in real time, processing observations as they are observed by the sensors. As such its use in real time applications is limited.

3.7 Discussion

We have developed a fully-differentiable, two-filter particle smoother (mixture density particle smoother) that robustly learns more accurate models than classic particle smoothers, whose components are often determined via heuristics. MDPS successfully incorporates temporal information from the past as well as the future to produce more accurate state posteriors than state-of-the-art differentiable particle filters. When applied to city-scale global localization from real imagery, our learned MDPS dramatically improves on search and retrieval-based baselines that were specifically engineered for the global localization task.

Appendix

3.A Additional Experiment Results

In this section we give additional experiment results for the global localization task on the MGL [62] and KITTI [79] datasets.

3.A.1 MGL Dataset Additional Results

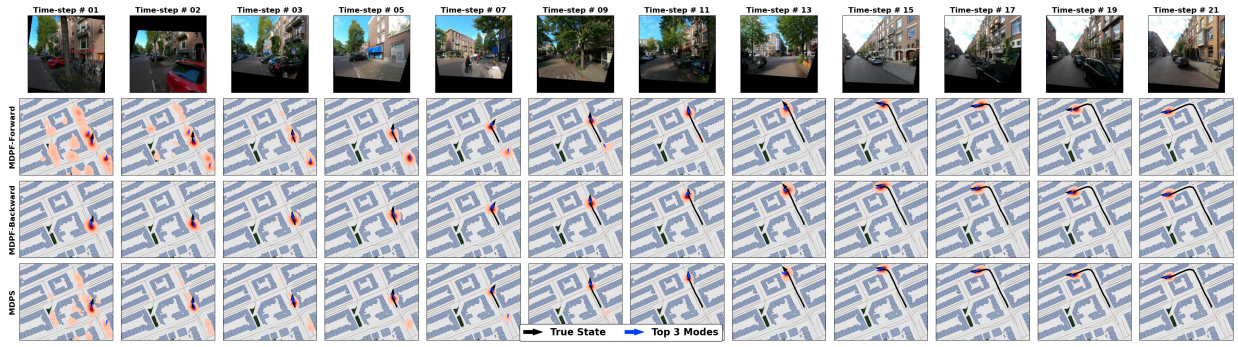


Figure 3.11: Additional example trajectories from the MGL dataset with observations shown in the top row. We show the current true state and state history (black arrow and black line), the estimated posterior density of the current state (red cloud, with darker being higher probability) and the the top 3 extracted modes (blue arrows) for each method. By using the full sequence of observations and actions when computing state posteriors, MDPS is able to estimate a more accurate and tighter posterior than the forward or backward MDPFs.

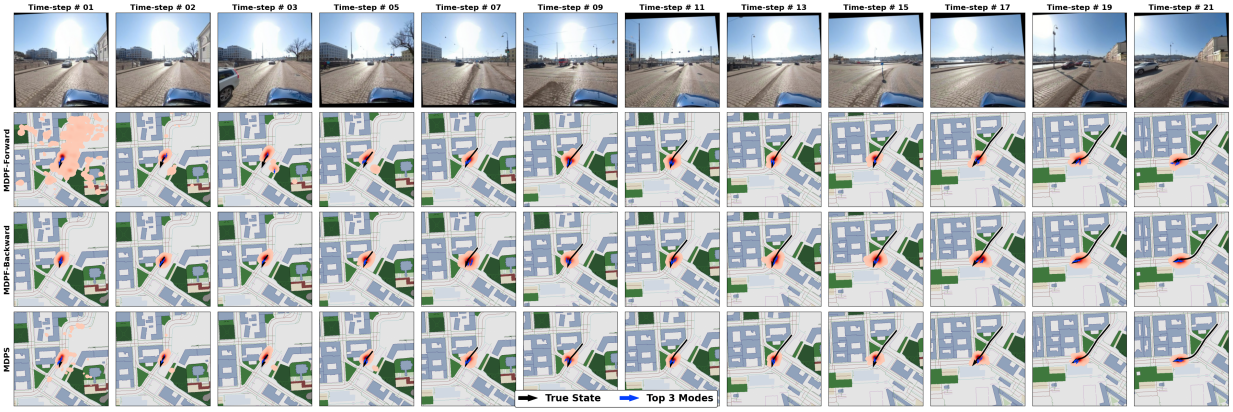


Figure 3.12: Additional example trajectories from the MGL dataset with observations shown in the top row. We show the current true state and state history (black arrow and black line), the estimated posterior density of the current state (red cloud, with darker being higher probability) and the the top 3 extracted modes (blue arrows) for each method. By using the full sequence of observations and actions when computing state posteriors, MDPS is able to estimate a more accurate and tighter posterior than the forward or backward MDPFs.

3.A.2 Kitti Dataset Additional Results

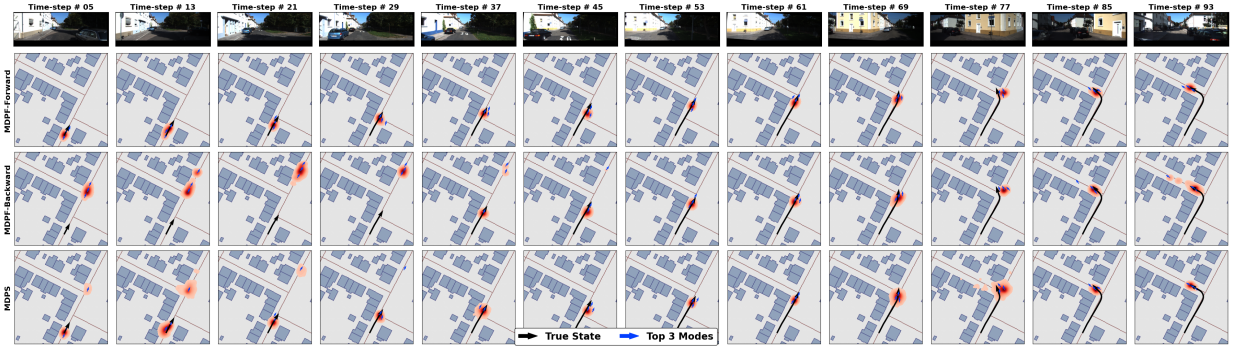


Figure 3.13: Additional example trajectories from the KITTI dataset with observations shown in the top row. We show the current true state and state history (black arrow and black line), the estimated posterior density of the current state (red cloud, with darker being higher probability) and the the top 3 extracted modes (blue arrows) for each method.

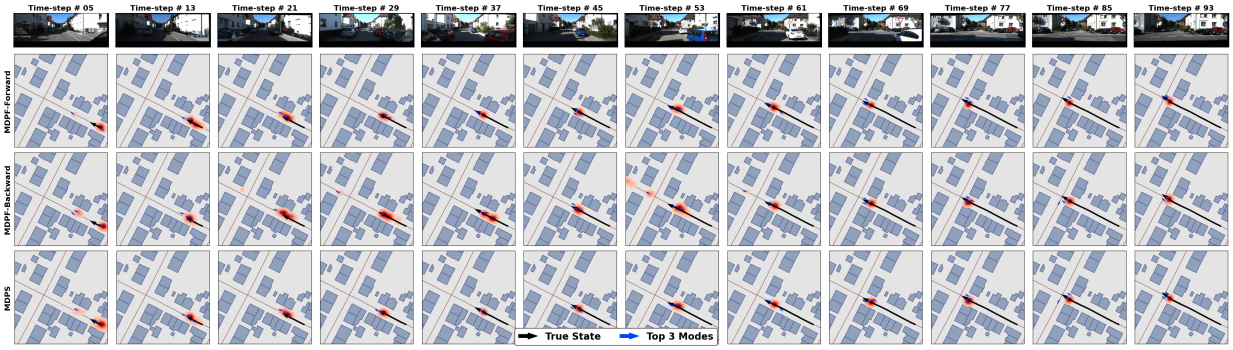


Figure 3.14: Additional example trajectories from the KITTI dataset with observations shown in the top row. We show the current true state and state history (black arrow and black line), the estimated posterior density of the current state (red cloud, with darker being higher probability) and the the top 3 extracted modes (blue arrows) for each method.

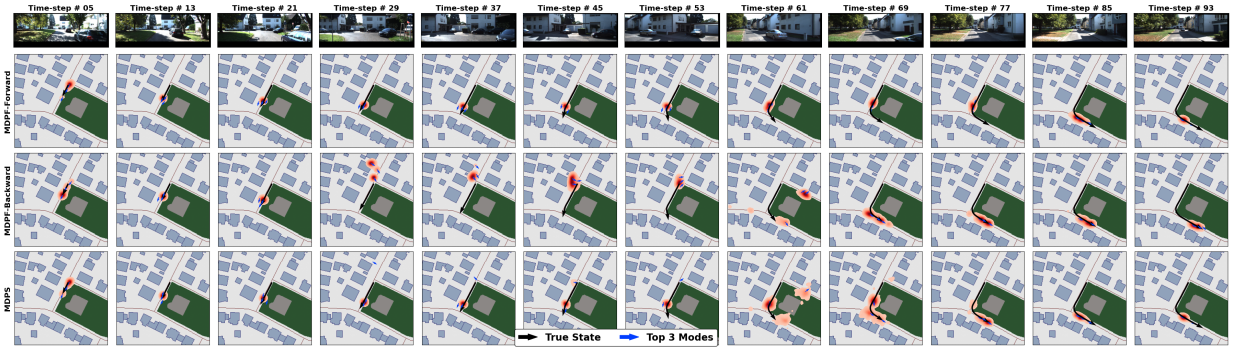


Figure 3.15: Additional example trajectories from the KITTI dataset with observations shown in the top row. We show the current true state and state history (black arrow and black line), the estimated posterior density of the current state (red cloud, with darker being higher probability) and the the top 3 extracted modes (blue arrows) for each method.

3.B Additional Experiment Details

3.B.1 Bearings Only Tracking Task

The Bearings Only Tracking Task (see chapters 2.6.1 and 3.5.1) aims to track the state of a vehicle as it navigates a simple environment. No actions are provided for this task and the observations are given as noisy bearings to a radar station:

$$y_t \sim \alpha \cdot \text{Uniform}(-\pi, \pi) + (1 - \alpha) \cdot \text{VonMises}(\psi(x_t), \kappa),$$

where $\psi(x_t)$ is the true bearing with $\alpha = 0.15$ and $\kappa = 50$. The velocity of the vehicle varies over time, changing randomly when selecting a target new way point with 1m/s or 2m/s being equally likely. During training, ground truth states are provided every 4 time-steps however dense true states are given at evaluation time. All sequences are of length $T = 50$ and we use 5000, 1000 and 5000 sequences for training, validation and evaluation respectively.

For all methods we use 50 particle during training and evaluation. For forward-in-time running PF methods, we initialize the particle set as the true state with small Gaussian noise ($\sigma = 0.01$) on the x-y components of the state. For the angle components of the initial particles we add Von Mises noise (with concentration $\kappa = 100$) to the true state. For backward-in-time running PF methods we set the initial particle set as random samples drawn uniformly from the state space.

Learning rates are varied throughout the training stages ranging from 0.001 to 0.000001 though we find that all methods are robust to learning rate selection when using the Adam [78] optimizer, with sensible learning rate effecting convergence speed but not performance.

For SR-PF [7] we set $\lambda = 0.1$.

Model Architectures

All methods (baselines and ours) for the Bearings Only Tracking Task use the same dynamics and measurement model architectures which are described below.

Dynamics Models. We parameterize the dynamics model as a residual neural network as shown in figs. 3.16 and 3.17. To maintain position in-variance, we mask out the position components of particles when input into the dynamics model. We also transform the angle component of the state into a vector representation $T(\theta) = (\sin(\theta), \cos(\theta))$ before applying dynamics. Afterwards we transform the angle component of the state back into an angle representation $T(u, v) = \text{atan2}(u, v) = \theta$

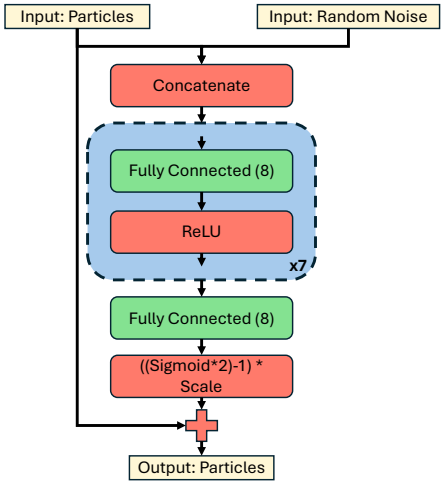


Figure 3.16: Dynamics model used for the Bearings Only Tracking Task. The output scaling scales the position components of the residual to be within $[-5, 5]$ and $-2, 2$ for the positional and angle (in vector representation) components respectively.

Measurement Models. Figure 3.18 shows the feed-forward neural network architecture for the measurement models used for the Bearings Only Tracking Task.

MDPS Forward Backward Combination. For the Bearings Only Tracking Task, the MDPS smoothed measurement model is very similar to the measurement model using for MDPF but with additional inputs. The MDPS smoothed measurement model network architecture is shown in Fig. 3.19.

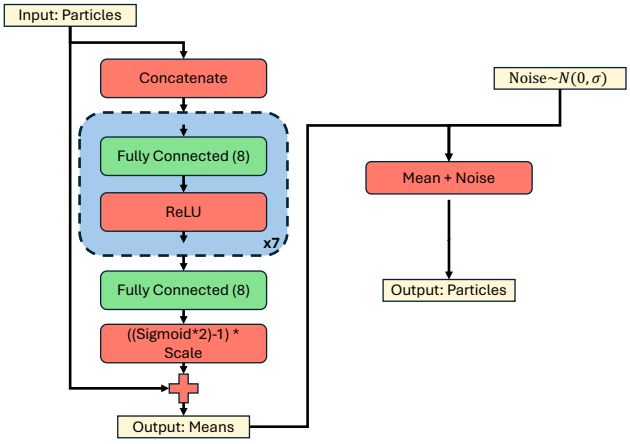


Figure 3.17: Dynamics model used for FFBS in the Bearings Only Tracking Task. The output scaling scales the position components of the residual to be within $[-5, 5]$ and $-2, 2$ for the positional and angle (in vector representation) components respectively. The dynamics model outputs a mean. Using this mean along with a hand tuned standard deviation allows for simulation from the dynamics model as well as evaluating state transition probabilities as required for FFBS. Of note: the angle dimension of the state is approximated by a Normal distribution with bound variance to avoid issues with angular discontinuities. The hand-tuned standard deviations values used are $[1.0, 1.0, 1.25]$.

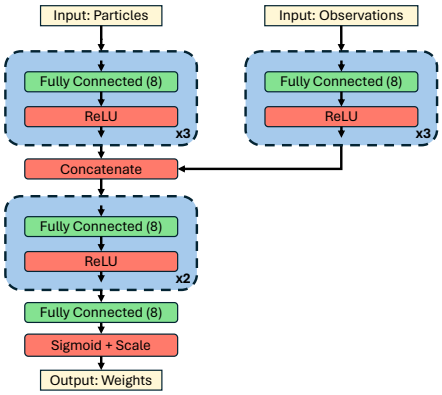


Figure 3.18: Particle filter measurement model used for the Bearings Only Tracking Task. The output scaling scales the weights to be within $[0.00001, 1]$

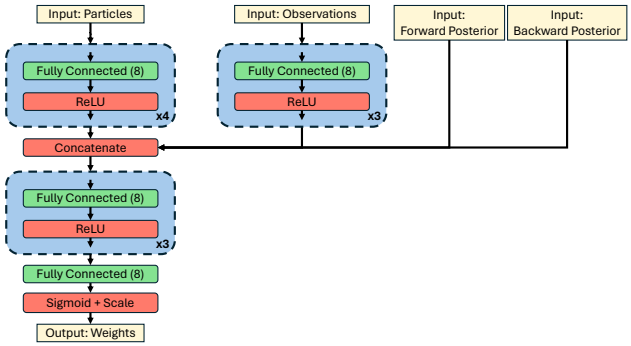


Figure 3.19: Measurement model used for the Bearings Only Tracking Task when computing the smoothed particle weights for MDPS. The output scaling scales the weights to be within $[0.00001, 1]$

3.B.2 Global Localization Task with Mapillary Geo-Location Dataset and KITTI Datasets

In this section we give more information about the experiments conducted with the Mapillary Geo-Location Dataset (MGL)[62] and KITTI [79] datasets.

For all particle filter methods (including ones internal to MDPS) we use 250 particle during training and evaluation and initialize the filters using 1000 particles. For PF and smoother methods, we initialize the particle set as the true state with Gaussian noise ($\sigma = 50$ meters) on the x-y components of the state. For the angle components of the initial particles we add Von Mises noise to the true state.

Initial learning rates are varied throughout the training stages ranging from 0.01 to 0.000001 though we find that all methods are robust to learning rate selection when using the Adam [78] optimizer, with sensible learning rates effecting convergence speed. For comparison methods, we use the learning rates as specified by the method authors or select them via a brief hyper-parameter search if they are not stated.

Mapillary Geo-Location Dataset Additional Details

In the MGL dataset, observations are images captured by various types of handheld or vehicle mounted cameras. Some cameras capture 360° images which require additional processing before being used as observations. These 360° images are cropped to a 90° Field-of-View in random viewing directions, with the same viewing direction being used for the whole observation sequence. All images are then gravity aligned to produce the observation sequence. A planimetric map of the environment is also provided via the OpenStreetMap platform [82] at 0.5 meter/pixel resolution, and all observation images are publicly available under a CC-BY-SA license via the Mapillary platform. All KITTI data is published under the CC-BY-NC-SA licence.

Unfortunately the creators of the MGL dataset trained their methods on single observations from the dataset and did not use sequences during training [62]. As such, observations from sequences are scattered amongst the training and testing splits, preventing effecting training of methods that require longer uninterrupted sequence data such as MDPF and MDPS. We therefore create custom train, validation and evaluation splits of the MGL dataset in order to accommodate longer sequences for training and evaluation.

Due to data integrity and corruption issues we exclude all sequences from the "Vilnius" portion of the dataset.

MDPF/MDPS Model Architectures

Dynamics Models. The network architecture of the dynamics model used for the MGL and KITTI is shown in Fig. 3.20 and is similar to that used in the Bearings Only Tracking task, using the same angle to vector particle transformation. Note for this dynamics model we do not mask out any component of the state.

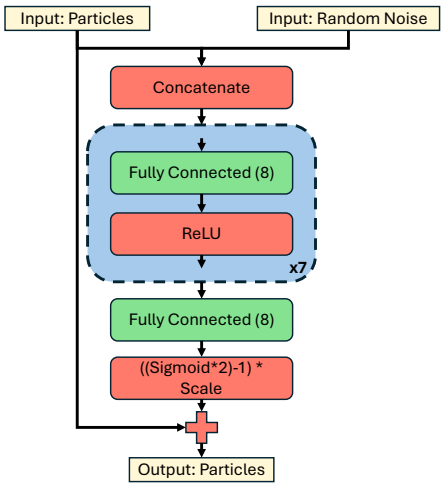


Figure 3.20: Network architecture for the dynamics model used for the Bearings Only Tracking Task. The output scaling scales the position components of the residual to be within $[-128, 128]$ and $-2, 2$ for the positional and angle (in vector representation) components respectively.

Measurement Models. The measurement model used for MDPF and MDPS uses the Birds-Eye-View (BEV) feature encoder and the map encoder from the official Dense Search implementation released by Sarlin et al. [62]. As shown in Fig. 3.1, a Birds-Eye-View (BEV) feature map is estimated from the observation using a geometric projection [62] where columns of the observation image are considered polar rays with features binned into course depth planes projected away from the camera focal plane. This gives a top-down representation of the local area in polar coordinates (bearing and course distance of an image feature from the camera center). The polar representation of the scene is then sampled into top-down Cartesian coordinates to yield the final BEV feature map. We refer the reader to Sarlin et al. [62] for more details. To compute particle weights, we compute the alignment, via a dot-product, between the BEV feature map and a local region from the neural map, cropped and rotated at the current particles location. For $l(\cdot)$ from Eqn. 3.18 we use fully-connected layers to produce the final smoothed particle weights from the BEV-map dot-product alignment and the forward and backward filter posterior densities. The map feature encoder is a U-Net based architecture with a VGG-19 [84] backbone and the BEV feature encoder is based on a multi-head U-Net with a ResNet-101 [85] backbone as well as

a differentiable but un-learned geometric projection. We refer the reader to Sarlin et al. [62] for more details about the encoders. To derive the un-normalized particle log-weights, the BEV encoding is compared, via dot-product, to a local map patch extracted at a specific particle to compute an alignment value. This is akin to the dense search described in Sarlin et al. [62] but at only a single location determined by the particle.

MDPS Forward Backward Combination. The MDPS smoother measurement model differs from the measurement models of MDPF as it requires additional inputs as described in Eqn 3.18. We implement this model as a 4-layered, 64-wide fully-connected feed-forward neural network with PReLU [86] activation's shown in Fig. 3.21. Importantly all computed un-normalized weights are bound to be within $[0.0001, 1]$ using a Sigmoid function with an offset. Input into this network is the BEV-map feature alignment computed in the same way as the MDPF measurement model as well as the posterior probability values from the forward and backward filters for the current smoothed particle.

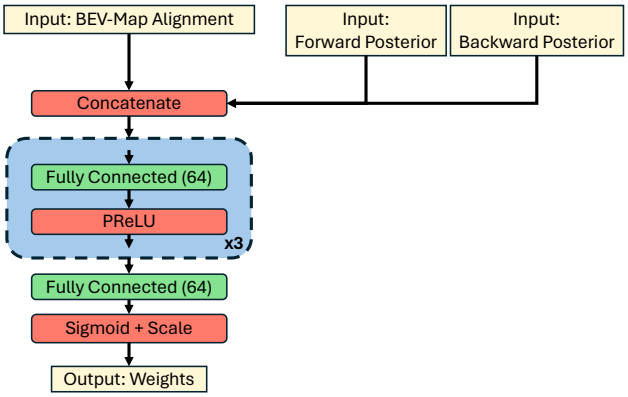


Figure 3.21: Network architecture for the MDPS smoothed measurement model used for the MGL and KITTI datasets.

MDPS Training Procedure Details

As stated in chapter 3.4, effective training of MDPS requires training in stages. Importantly due to VRAM constraints, we are unable to train large map and observation encoders on

long sequences. Therefore we train on short sequences before freezing the encoders, training the rest of the models on longer sequences.

Training procedure for MDPS on the MGL dataset:

1. Train forward and backward MDPFs individually via independent loss functions, sharing map and observation encoders, on short sequences from the dataset. Here we hold the output posterior bandwidths fixed to prevent converging to poor local optima where the bandwidth is widened while the dynamics and measurement models are not informative.

2. Freeze all MDPF models, unfreeze the MDPF output posterior bandwidths and train on long sequences.

3. Freeze all MDPF models (including bandwidths) and train the MDPS measurement model and output posterior bandwidth on long sequences.

4. Unfreeze all models except the map and BEV encoders and train MDPS on long sequences.

Due to the small size of the KITTI dataset along with pre-training using the MGL dataset, a special constrained training procedure is required to prevent immediate over-fitting to the training split. Instead of jointly refining all components simultaneously, we fine-tuning the forward and backward MDPFs before freezing those models and fine-tuning the MDPS smoothed measurement model:

1. Train the forward and backward MDPFs individually via independent loss functions, sharing map and observation encoders, on short sequences from the dataset. We hold the output posterior bandwidths fixed.

2. Freeze BEV and map encoders, training the forward and backward MDPFs individually via independent loss functions on long sequences.

3. Freeze all MDPF models, unfreeze the MDPF output posterior bandwidths and train on long sequences.

4. Freeze all MDPF models (including bandwidths) and train the MDPS smoothed measurement model and output posterior bandwidth on long sequences.

5. Freeze all models except for the MDPS output posterior bandwidth and train on long sequences.

Baseline Implementation Details

Retrieval. The Retrieval [54] baseline as described by Samano et al. [54] encodes individual patches from the environment global map into the a latent space. This is inefficient if the map patches are densely sampled and is prohibitive to run for large environments. Instead a dense feature map can be predicted from the global map in one forward CNN pass to generate dense map encoding for map patches roughly sized according to the CNN receptive field [62]. We adopt this dense encoding approach for Retrieval, implementing the Retrieval method as specified by Sarlin et al. [62].

Dense Search. For Dense Search we use the official implementation released by Sarlin et al. [62] which differs from the text description present in the official paper. In the paper description a location prior is computed from the provided map which estimates regions of the state space that are likely to be occupied (e.g. the prior says that rivers and inside buildings are unlikely to be occupied by a car). The prior is then multiplied with the observation likelihood (probability volume computed via dense search) to produce the final state posterior. In the official implementation this prior is disabled making the state posterior simply the observation likelihood. Further we use VGG-19 [84] as our map encoder backbone.

Retrieval (PF). Zhou et al. [59] embeds standard Retrieval methods within a non-differentiable particle filter where the dynamics are set as Gaussian Noise:

$$x_t^{(i)} \sim \mathcal{N}(x_{t-1}^{(i)} + a_t, \gamma) \tag{3.22}$$

In our experiments we set γ as $2.5m$ for the x-y state position components and 15° for the angular state components, chosen via a brief hyper-parameter search. The measurement model is defined as

$$w_t^{(i)} = \exp\left(\frac{-d_t^{(i)}}{2\sigma^2}\right) \tag{3.23}$$

where $d_t^{(i)}$ is the alignment of the observation latent encoding with the map patch encoding at the current particles location $x_t^{(i)}$,computed via the Retrieval method. After a brief hyper-parameter search we set $\sigma = 2$ in our experiments.

Chapter 4

End-to-end Learnable Particle Belief Propagation

In this chapter we develop an end-to-end differentiable belief propagation algorithm which we call Mixture Density Belief Propagation (MDBP) for inference in continuous probabilistic graphical models. MDBP is applicable to sequential state estimation tasks for which the state can be decomposed into a set of smaller dimensional sub-states, connected via local pairwise interactions, inducing a pairwise tree-structured undirected graphical model. For such tasks, MDBP is able to effectively learn complex measurement models and pairwise constraint functions as well as the interesting dynamics which governs how the system evolves with time. We apply MDBP to several articulated pose estimation problems and showcase improved performance over previously proposed differentiable belief propagation algorithms as well as recent models tailored to these specific tasks.

4.1 Introduction

Probabilistic graphical models (PGM) [87] (see Fig. 4.1) decompose high dimensional multivariate distributions into a set of local interactions between smaller marginalized distributions among a subset of variables. This decomposition creates conditional independence between subsets of variables, simplifying the learning and inference problems. In fact, certain problems may be intractable in their original form, but decomposing into probabilistic graphical models can make these problems tractable. Fig. 4.1 shows some example probabilistic graphical models. Note that for simplicity we do not draw all observation nodes, but instead assume each latent node is connected to an observation node.

In chapters 2 and 3 we focused on *directed* graphical models which are useful for encoding causal processes. In this chapter we will instead tackle *undirected* graphical models or *Markov Random Fields* (MRFs) [87] where local interactions encode symmetric dependencies instead of causal ones. We will then extend our methods to graphs containing both directed and undirected edges, specifically for problems in which the system evolves over time.

Articulated Human Mesh Recovery (HMR) is one such task that benefits from PGM decomposition. In HMR, we wish to estimate the 3D mesh of a human subject from RGB video frames [88, 89, 90, 91, 92, 93]. Due to advances in human mesh modeling [94, 95, 96], HMR has been simplified from needing to estimate the raw 3D mesh vertices parameters (which can number in the thousands) to only needing to estimate certain human skeleton joint angles (such as knees and elbows) as well as a body shape parameter vector, significantly reducing the dimensionality of the problem we need to solve. Further only the skeleton joint angles vary over time (as the subject moves around), with the body shape parameters being constant for each subject. Interestingly, joint angles are mainly influenced by local interaction to neighboring joints. For example the left ankle is heavily influenced by the left knee but not the right wrist. These local interactions naturally decomposes the HMR into

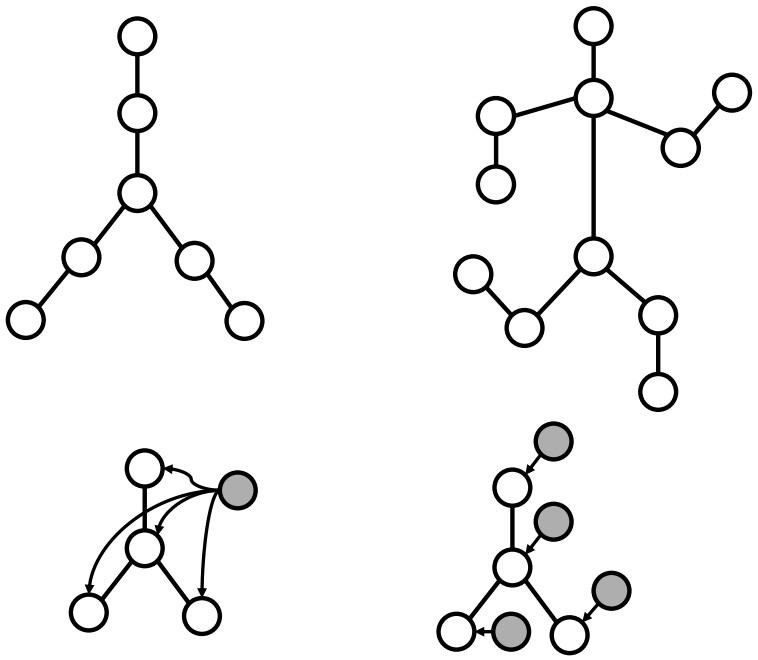


Figure 4.1: Examples probabilistic graphical models. *top Right:* A simplified example formulation of the Human Mesh Recover (HMR) task as a probabilistic graphical model. *Bottom:* Observations (gray nodes) can be shared across multiple nodes (left) or each nodes can can use different observations (right).

a pairwise undirected graphical model (see Fig. 4.1) where nodes represent individual joint angles and edges encode interactions between neighboring joints in the kinematic tree.

Inference on probabilistic graphical models has been well explored with belief propagation (BP) emerging as a natural and effective method which uses the model structure to efficiently and accurately perform inference [97, 87]. Many of these methods are non-differentiable [98, 99, 75, 100, 101, 102] and thus rely on human experts to develop suitable pairwise functions, which encode knowledge of the local interactions between pairs of state variables, and develop useful observation models, limiting performance on complex and hard to model problems. Some methods blend neural techniques and belief propagation by approximating the BP algorithm via neural networks [103, 104, 105, 106]. These methods learn an approximation of the BP algorithm and thus can be viewed as fancy neural architectures instead of a faithful implementation of the BP algorithm with learned models. Finally using learned models within the classical BP framework has been attempted, though with limited success [107].

In this chapter we propose a novel end-to-end learnable belief propagation algorithm for inference on pairwise tree-structured graphical models. Our *Mixture Density Belief Propagation* (MDBP) algorithm is able to learn the complex pairwise interactions that are expressed in the graphical model structure as well as the interesting dynamics of the overall state as it evolves over time. Our MDBP shows improvements over previously proposed differentiable belief propagation methods on articulated pose estimation tasks and shows qualitative improvements on the human mesh recovery (HMR) task when compared to recent leading models specifically designed for HMR.

4.2 Belief Propagation

An undirected graph G is defined as a set of nodes $V = 1, ..., K$ and a set of corresponding edges E. For pairwise undirected graphs,which we consider in this chapter, each edge in E is defined as a connection between 2 nodes. Further for simplicity we assume there are no self edges in G. Since each edge is comprised of 2 nodes, we can define Γ_d to be the set of neighbors of node d in G, specifically the set of nodes s such that (s, d) is an edge in G. In undirected graphs the edges are symmetrical and do not encode any ordering between the nodes, therefore the edge (s, d) can also be written as (d, s).

4.2.1 Pairwise Markov Random Fields

(Pairwise) Markov Random Fields MRFs associate every node $d \in V$ with a hidden random variable x_d. We denote $X = \{x_1, ..., x_{|V|}\}$ as the set of all hidden random variables in the MRF. Each node $d \in V$ is also associated with a positive unary potential function $\phi_d(x_d; y) > 0$ which scores the a particular assignment of x_d given some observed observation y. For each local interaction, i.e. edge, between nodes s and d, there exists a positive and

pairwise potential function $\phi_{s,d}(x_s, x_d) > 0$ which encodes soft local constraints between nodes. Observations y are optional for the unary potentials; being dropped if unavailable. The joint posterior distribution of a pairwise MRF can be defined as:

$$p(X|y) \propto \Big(\prod_{s \in V} \phi_s(x_s; y) \Big) \cdot \Big(\prod_{(s,d) \in E} \phi_{s,d}(x_s, x_d; y) \Big) \tag{4.1}$$

with proportionality due to the unknown normalization constant, set such that $p(X|y)$ integrates to 1.

4.2.2 Belief Propagation

For undirected tree structured graphs, exact node marginal distributions $p(x_d|y)$ for each node can be computed using the *Belief Propagation* algorithm [87, 97]. Of note, the node marginal $p(x_d|y)$ is distinctly different to the unary potential $\phi_d(x_d; y)$ due to the local interactions between nodes via the pairwise potential functions. Belief propagation is a message passing algorithm where positive messages $m_{s \to d}(x_d) > 0$ are passed between all neighbors $s \in \Gamma_d$ for all nodes $d \in V$:

$$m_{s \to d}(x_d) \propto \int \phi_{s,d}(x_s, x_d) \phi_s(x_s; y) \prod_{z \in \Gamma_s \setminus d} m_{z \to s}(x_s) dx_s \tag{4.2}$$

Figure 4.2 illustrates this message passing. Messages summarize the all information that x_s collected regarding the variable x_d at any point. The message passing integral of eqn. (4.2) contains the pre-message which represents all the available knowledge about x_s before sending a message to node d:

$$M_{s \to d}(x_s) = \phi_s(x_s; y) \prod_{z \in \Gamma_s \setminus d} m_{z \to s}(x_s) \tag{4.3}$$

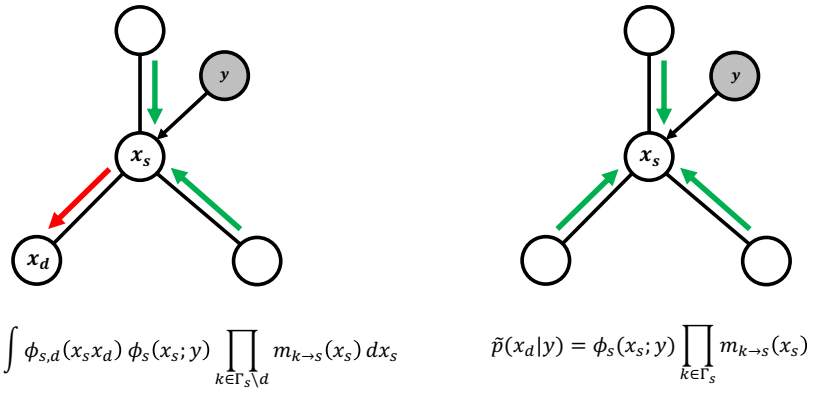


Figure 4.2: *Left:* A new outgoing message (red) is computed by combining all incoming messages (green) with the unary and pairwise potential functions. *Right:* An approximate marginal is computed by combining all incoming messages (green) with the unary potential function.

This pre-message can then be combined along with the pairwise potential function to propagate information to the neighboring nodes.

An approximation of x_d's marginal (or belief) can be produced by combining all incoming messages x_d has received from its neighbors with its unary potential function (see Fig. 4.2):

$$p(x_d|y) \approx \tilde{p}(x_d|y) \propto \phi_d(x_d) \prod_{s\in\Gamma_d} m_{s\to d}(x_d) \tag{4.4}$$

For tree structured graphs, $\tilde{p}(x_d|y)$ converges to the true marginals $p(x_d|y)$ once messages from each node have propagated to every other node in the graph [97, 87]. This can be achieved via a simple message passing schedule shown in Fig. 4.3 where a root node is arbitrarily chosen and messages are passed from the leaf nodes to the root node. Messages are then sent back from the root node to the leaf nodes, ensuring that each node has received information about all other nodes.

Message passing within BP by definition involves only local interactions between nodes. As

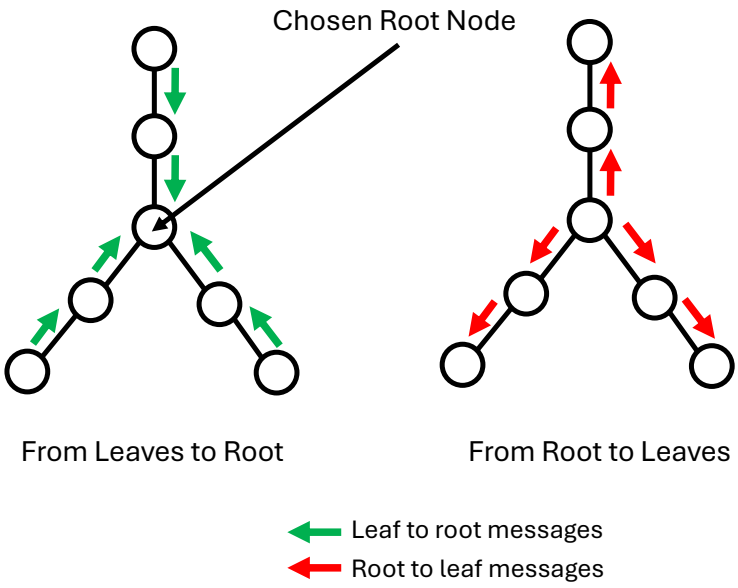


Figure 4.3: Example message passing schedule where messages are passed from leaf nodes to an arbitrarily chosen root node and then back to the leaf nodes, ensuring that messages from all nodes have been propagated to all other nodes.

such BP can be applied to graphs that contain cycles (known as Loopy BP), however the resulting beliefs will not converge to the true marginals [97, 87]. Further there is no guarantee that messages under loopy BP will converge at all and indeed they may oscillate between several modes, though methods have been developed to mitigate this issue [108, 109, 87]. Interestingly loopy BP has shown good performance on a range of tasks making it a useful algorithm despite only having partial theoretical justification [97, 110].

4.2.3 Discrete Belief Propagation

For MRFs where nodes are discrete random variables, a discrete Belief Propagation technique can be applied [97, 87]. The so called sum-product algorithm [111] converts integrals in eqn. (4.2) to sums. Denoting $\mathcal{X}_s$ as the domain of node x_s (which is discrete in this case) we

can define messages simply as:

$$m_{s\to d}(x_d) = \sum_{x_s \in \mathcal{X}_s} \phi_{s,d}(x_s, x_d)\phi_s(x_s; y) \prod_{z \in \Gamma_s \setminus d} m_{z\to s}(x_s) \tag{4.5}$$

The approximate marginals for each node is expressed as in eqn. (4.4), with the vector representing the marginal appropriately normalized such that $\sum_{x_d \in \mathcal{X}_d} p(x_d|y) = 1$.

In sum-product BP, messages can be computed via matrix-vector operations. Interestingly, general-purpose computing on graphics processing units aided by developer friendly tools such as CUDA [112] has enabled extremely fast matrix multiplication, making sum-product BP very efficient. Furthermore specific implementations of sum-product BP where all messages are passed in parallel are particularly amenable to GPU acceleration. In parallel message passing [97] all messages are initializes to be uniform and then in each round of message passing computes all messages simultaneously [87].

4.3 Belief Propagation in Continuous Spaces

In chapter 4.2, we introduced belief propagation and its implementation when the state space of nodes is discrete. For problems with continuous state spaces, the sum-product BP algorithm can be applied by discretizating the continuous space into a discrete one. This however can be limiting since accuracy and efficient are determined by the discretization density, with higher density leading to better accuracy at the cost of potentially much higher memory and computation costs. These costs also grow exponentially as the dimensionality of the state space is increased. Further for continuous spaces with a large (and possibly infinite) range, adequate discretizations may be intractable. These deficiencies in applying discrete BP algorithms to continuous state spaces motivates the development of belief propagation methods that use the continuous space directly.

4.3.1 Gaussian Belief Propagation

For MRFs where nodes are set as continuous random variables, a closed form BP algorithm is possible when the marginals are defined to be Gaussian [87, 113]. In Gaussian BP, the unary unary and pairwise potential functions are assumed to be Gaussian or can be linearized using similar methods to the Extended Kalman Filter [13, 14], though success of Gaussian BP is determined by accuracy of the linear approximation. Gaussian BP can be seen as a generalization of Kalman Smoothing [74] to undirected tree structured graphs and indeed Gaussian BP can be applied to the same problems Kalman Smoothing can be applied.

Like Kalman smoothing, Gaussian BP expresses the messages and node marginals as a unimodal Gaussian distributions and thus cannot faithfully capture uncertainty in domains where multiple modes are inherent; such as in the global localization task shown in chapter 3.5. In such tasks, the Gaussian node marginals may collapse to one of the (possibly incorrect) true modes or may exhibit large uncertainty over the state space via an overly wide variance.

4.3.2 Nonparametric Belief Propagation

Nonparametric Belief Propagation (NBP) [98, 99] is a particle based method similar to particle filtering [1, 2, 3, 4] and smoothing [64, 65, 66, 67, 68, 69]. Like particle filters and smoother, NBP represents the node marginals as a collection of weighted particles, enabling multi-modal node marginal estimates. NBP can be applied to problems with continuous state spaces, focusing particles in likely regions to produce accurate node marginals.

In NBP, messages are represented via a kernal density estimate (KDE) [21] parameterized by a set of N particles, weights and a shared bandwidth parameter similar to the KDE used

for the posterior densities in mixture density particle filters (see chapter 2.5):

$$m_{s\to d}(x_d) = \sum_{i=1}^{N} w^{(i)} K(x_d - x_d^{(i)}, \beta) \tag{4.6}$$

In Sudderth et al. [98], only Gaussian kernels are explored, however the authors acknowledge that certain other kernel function may also be used.

The message update integral, eqn. (4.2), is then approximated via stochastic integration in two stages. In stage 1, samples are drawn from the pre-message (eqn. 4.3) $\hat{x}_{s\to d}^{(i)} \sim M_{s\to d}(x_s)$ using an expensive Gibbs sampling technique [114]. This Gibbs sampling avoids the need to directly multiply the mixtures representing the incoming messages to draw samples [98, 99].

In stage 2, the message update integral (eqn. 4.2), is approximated via a collection of samples $x_{s\to d}^{(i)}$ by propagating the pre-message samples $\hat{x}_{s\to d}^{(i)}$ using the pairwise potential function, computing the outgoing message $m_{s\to d}(x_s)$:

$$x_{s\to d}^{(i)} \sim \phi_{s,d}(x_s = \hat{x}_{s\to d}^{(i)}, x_d) \tag{4.7}$$

Finally the message mixture can be created by computing an appropriate bandwidth β to the particle set. Sudderth et al. [98] assign this bandwidth using the "Rule Of Thumb" method [21]. Messages are propagated though the graph until all the messages have converged.

The node marginals as shown in eqn. (4.4) can also be approximated via a collection of weighted particles at any point using all the currently available incomoing message for each node. Interestingly, the node marginals have a similar form to the pre-message and thus the same Gibbs sampling technique can be used to draw unbiased samples from the product of mixtures. Once the node marginal samples $x_s^{(:)}$ for node s are drawn, a bandwidth can again be computed.

Drawbacks of Nonparametric Belief Propagation

Gibbs sampling is computationally expensive. Techniques proposing alternatives methods for sampling from products of mixtures which are computationally more efficient[75, 100], though the success of these techniques is dependent on the input mixture distribution shape and thus these methods may not be suitable for all problems.

Further, NBP is not differentiable due to the expensive Gibbs sampling procedure. Computing gradients for particles drawn using Gibbs sampling with respect to the sampling distribution is not possible, limiting the end-to-end differentiability of NBP which prevents using neural networks to approximate the complex unary and pairwise potential functions.

4.3.3 Pull Message Passing Nonparametric Belief Propagation

Pull message passing nonparametric belief propagation (PMPNBP) [102] is an alternative nonparametric BP algorithm to that proposed by Sudderth et al. [98]. Like NBP, messages are represented via a collection of weighted samples, however instead of drawing samples from pre-messages using an expensive Gibbs sampler for each message, PMPNBP assumes an approximate marginal distribution $\tilde{p}_d(x_d)$, represented as a collection of particles and weights, for each node exists and draws samples from that marginal $x_d^{(i)} \sim \tilde{p}_d(x_d)$ via discrete resampling with replacement, similar to the resampling step of particle filters. In the first round, samples can be drawn from some initial distribution such as the uniform distribution. These samples are then propagated to neighboring nodes and weights computed to produce messages. At a high level, PMPNBP is referred to as a "pull" method since messages weigh samples at each node by pulling information from neighboring sending nodes. In contrast NBP is a "push" method since each message generates samples via the pre-message before propagating (a.k.a pushing) those samples via the pairwise potential function to the receiving node.

Like classical BP, message passing in PMPNBP is repeated for several iterations until the messages converge. Once messages converge, each node has a new estimate of the belief and thus new particles can be drawn, or *resampled*, and new messages computed using the new particles. This resampling and message-passing is repeated for several rounds with the number of rounds being a tuned hyper-parameter. The particle resampling concentrates particles into high probability parts of the state space, allowing the marginal estimate to improve with each round.

PMPNBP, like NBP, is not differentiable due to the particle resampling at the beginning of each PMPNBP round. This resampling is a discrete resampling with replacement from the current marginal estimate and is thus not differentiable. This prevents end-to-end learning limiting the application of neural networks to model the unary and pairwise potential function.

4.3.4 Particle Belief Propagation

Particle belief propagation (PBP) [101] is a particle based algorithm that can be effectively applied to continuous spaces. In PBP particles $x_d^{(1)}, ..., x_d^{(N)}$ are sampled for each node $d \in V$ from some arbitrary sampling distribution $q_d(x_d)$. Messages are then computed by re-writing the message generation eqn. (4.2) as an importance-sampling corrected expectation:

$$m_{s \to d}(x_d) = E_{x_s \sim q_s(x_s)}\Big[\frac{\phi_{s,d}(x_s, x_d)\phi_s(x_s; y)}{q_s(x_s)} \prod_{z \in \Gamma_s \setminus d} m_{z \to s}(x_s)\Big] \tag{4.8}$$

Using the drawn samples, this expectation can be represented via a finite sample estimate with importance weighting, where messages are simply weights assigned to each of the drawn

particles for each node:

$$m_{s\to d}(x_d^{(j)}) = w_{s\to d}^{(j)} = \sum_{i=1}^{N} \Big[\frac{\phi_{s,d}(x_s^{(i)}, x_d^{(j)}) \phi_s(x_s^{(i)}; y)}{q_s(x_s^{(i)})} \prod_{z \in \Gamma_s \setminus d} m_{z\to s}(x_s^{(i)}) \Big] \tag{4.9}$$

Equation (4.9) specifies a belief propagation algorithm where messages are defined in terms of particles drawn for each node. Each message shares the same set of particles and simply assigns the particles different weights using importance weighting. This is in contrast to nonparametric BP methods where different particles are drawn for each message. Sharing the particles eliminates the need to smooth the particle set via a kernel density estimate as all incoming messages to a node are defined in terms of the exact same particle set and thus multiplication of messages can be exactly computed by simply multiplying message weights for each of the particles.

Once messages have converged, the marginal probability can be estimate by computing weights for the drawn particles using the unary potential function as well as all incoming messages:

$$\tilde{p}(x_d^{(j)}|y) = w_d^{(j)} = \phi_d(x_d^{(j)}) \prod_{s \in \Gamma_d} m_{s\to d}(x_d^{(j)}) = \phi_d(x_d^{(j)}) \prod_{s \in \Gamma_d} w_{s\to d}^{(j)} \tag{4.10}$$

with $w_d^{(j)}$ being normalized such that $\sum_{j=1}^{N} w_d^{(j)} = 1$

In PBP, messages are iteratively computed from one node to another until all messages converge. PBP, like PMPNBP, is a multi-round algorithm where after messages have converged, new particles for each node are sampled from the sampling distribution $q_s(x_s)$. If $q_s(x_s)$ is set to be the current estimate of the node true marginals, $q_s(x_s) = \tilde{p}(x_s|y)$, then successive rounds of message-convergence particle-resampling tend to concentrate particles around high probability regions of the state space, improving the estimated marginal at each round. Indeed Ihler and McAllester [101] shows empirically that using the node marginals

is a good choice for $q_s(x_s)$.

Drawbacks Particle Belief Propagation

Like PMPNBP, particle BP contains a particle resampling step. During resampling, particles are drawn from the current best estimate of the marginals for each node. In Ihler and McAllester [101], this resampling is performed via discrete resampling with replacement which is non-differentiable as discussed in chapter 2.2. Again this non-differentiable resampling step limits the use of neural networks within this BP algorithm.

4.4 Towards End-to-End Learnable Belief Propagation

Merging deep learning approaches with belief propagation has has been explored in the past and has led to some interesting deep learning based approximations to the belief propagation algorithm [115, 103, 104, 105, 106, 116]. These methods are not faithful blends of BP with neural networks but instead emulate the BP message passing algorithms using neural models. Indeed methods based on Graph Neural Networks (GNNs) [117] have shown good results on a variety of tasks where belief propagation methods are typically applied [105, 106, 104] though these methods cannot be seen as an implementation of classical BP. Unfortunately most works apply only to discrete state spaces [115, 104, 103, 116, 105] or simply produce point estimates for each node without any measure of uncertainty [106], which is not ideal when the true marginals may contain multiple modes.

4.4.1 Differentiable Nonparametric Belief Propagation

Differentiable Nonparametric Belief Propagation (DNBP) [107] blends nonparametric belief propagation with neural networks to create a learnable BP algorithm for continuous state spaces. Building on pull message passing nonparametric belief propagation of chapter 4.3.3, DNBP specifies the unary and pairwise potential functions as learned neural networks (constrained to output only positive values), propagating the gradient through PMPNBPs message passing steps. DNBP borrows from Truncated-Gradient Particle Filter [5] (see chapter 2.3) and simply truncates the gradients at each particle resampling step within the PMPNBP algorithm to create a learnable algorithm, though biasing the gradients significantly.

Further the PMPNBP algorithm requires simulation from and evaluation of the pairwise potential function $\phi_{s,d}(x_s, x_d)$ when computing message weights. This limits to use of neural networks to very few architectures that are able to both score as well as draw samples. Interestingly Opipari et al. [107] models the pairwise potential as a pair of neural networks networks $\{g_{s,d}(x_s, x_d), \tilde{g}_{s,d}(x_s, x_d)\}$ where $g_{s,d}(x_s, x_d)$ is able to score particle pairs and $\tilde{g}_{s,d}(x_s, x_d)$ able to generate samples. These are two distinct networks thus there is no guarantees that both are modeling the same distribution and in fact DNBP simply assumes that both networks will learn to model the same underlying distribution when trained without adding any additional constraints to enforce this.

4.5 Mixture Density Belief Propagation

Our novel *Mixture Density Belief Propagation* (MDBP) algorithm, Fig. 4.4, can be seen as differentiable variant of Particle Belief Propagation [101] for application in continuous state spaces. In PBP, estimates of node marginals are represented as a collection of N particles

$\{x_d^{(1)}, ..., x_d^{(N)}\}$ and their associated message weights $\{w_d^{(1)}, ..., w_d^{(N)}\}$ as shown in equation 4.10. During the resampling, PBP draws new particles from the node marginals using discrete resampling with replacement which is non-differentiable (see chapter 2.2.1). MDBP sidesteps this issue by instead adopting the same Importance Weighted Sample Gradient (IWSG) estimator that was developed in chapter 2.4.2.

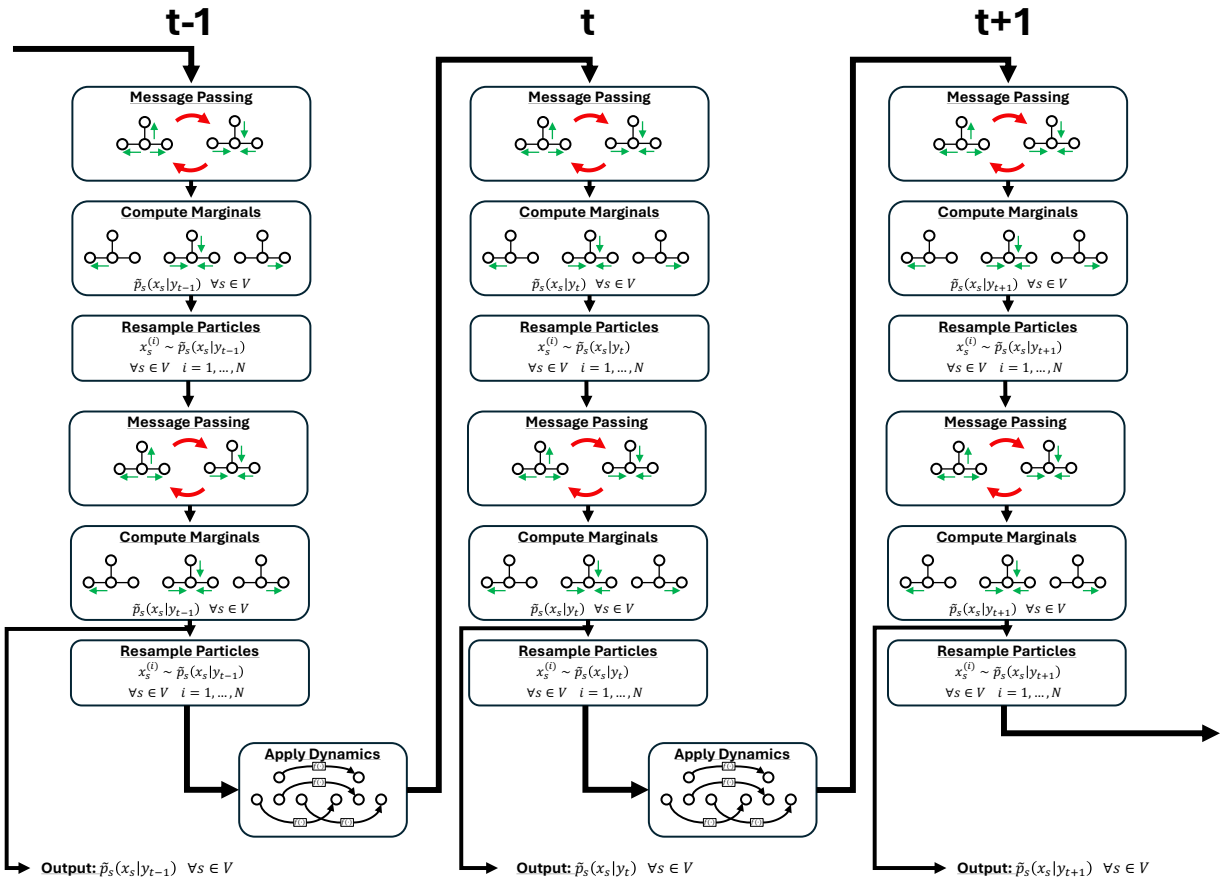


Figure 4.4: Flow diagram for Mixture Density Belief Propagation. Two rounds of MDBP message passing are shown in this flow diagram.

4.5.1 Differentiable Resampling from the Node Marginals

Instead of discrete resampling with replacement as is done in PBP, MDBP first smooths particles using a kernel density estimate to achieve a continuous valued marginal with the

new marginal estimate being defined as a mixture distribution:

$$\tilde{p}(x_d^{(j)}|y) = \sum_{i=1}^{N} w_d^{(i)} \cdot K(x_d - x_d^{(i)}; \beta). \tag{4.11}$$

Here $K(\cdot)$ is an arbitrary kernel function suitably chosen to match the state estimation task with β being a set bandwidth parameter shared across all mixture components. Like in MDPF and MDPS, we let β be a learned bandwidth parameter. Letting $\theta = x_d^{(:)} \cup w_d^{(:)} \cup \{\beta\}$, sampling new particles from the marginal at each BP round now amounts to drawing samples from this continuous mixture distribution with resampling weights (and their gradients) determined using our IWSG estimator:

$$\hat{w}_d^{(i)} = \frac{\tilde{p}(x_d^{(i)}|y,\theta)|_{\theta=\theta_0}}{\tilde{p}(x_d^{(i)}|y,\theta_0)} = 1, \qquad \nabla_\theta \hat{w}_d^{(i)} = \frac{\nabla_\theta \tilde{p}(x_d^{(i)}|y,\theta)|_{\theta=\theta_0}}{\tilde{p}(x_d^{(i)}|y,\theta_0)} \tag{4.12}$$

Using IWSG, the newly resampled weights $\hat{w}_d^{(:)}$ are given weight one because they are sampled from a mixture with the current parameters θ_0. Gradients account for how importance weights will change as the parameters θ deviate from θ_0. Modifying eqn. (4.9) to account for the resampled weight we get the following message passing equation:

$$m_{s\to d}(x_d^{(j)}) = w_{s\to d}^{(j)} = \sum_{i=1}^{N} \left[\hat{w}_s^{(i)} \frac{\phi_{s,d}(x_s^{(i)}, x_d^{(j)}) \phi_s(x_s^{(i)}; y)}{q_s(x_s^{(i)})} \prod_{z \in \Gamma_s \setminus d} m_{z\to s}(x_s^{(i)}) \right] \tag{4.13}$$

During inference, $\hat{w}_s^{(i)} = 1$ and we exactly recover the message passing equation of PBP, eqn. 4.9. When differentiating through the message passing algorithm of MDBP, gradients propagate back through the particle resampling step via the resampled importance weights $\hat{w}_s^{(i)}$ computed via IWSG.

Making this minor change to PBP resampling, we produce our novel end-to-end differentiable belief propagation method, *Mixture Density Belief Propagation*. MDBP only requires that the unary potentials $\phi_s(x_s)$ and the pairwise potentials $\phi_{s,d}(x_s, x_d)$ can be evaluated and

thus can be parameterized via neural networks with almost no constraints on the network architectures, other than the outputs must be positive. This enables use of a wide range of advanced network architectures such as Vision Transformers [118]. We show out MDBP message passing algorithm in Fig. 4.5.

4.5.2 Application to Dynamic Systems

MDBP message passing computes estimates of marginal probabilities given an observation y as well as the (learned) unary and pairwise potential functions. These marginals are parameterized by a set of particles and weights, smoothed using a bandwidth parameter to produce a mixture distribution. As formulated above, MDBP can be readily applied to pairwise, tree-structured undirected graphical models (Fig. 4.6) where the system is static (does not evolve through time) and for which the observation is constant. However MDBP can also be applied to dynamical systems where the state at each node evolves through time via some unknown system dynamics. To model these systems, the pairwise, tree-structured undirected graphical model can be augmented with directed edges, see Fig. 4.6, which capture the causal dynamical nature of the system as it changes through time. These directed edges capture systems dynamics at each node and can be modeled via a (learned) dynamics model similar to that of particle filters.

To enable inference on graph structures with system dynamics shown in Fig. 4.6, we adapt MDBP by incorporating node-specific dynamics models into the MDBP algorithm. Given an observation y_t for time $t = 1, ..., T$, MDBP message passing can be run (until convergence, for several rounds) to produce an estimates of node marginals $\tilde{p}_t(x_d|y_t)$ for the current time t. To propagate this marginal to the next time $t+1$, we simply draw N particles $\hat{x}_{t,d}^{(i)} \sim \tilde{p}_t(x_d|y_t)$, $i = 1..., N$, for each node and assign their weights $\hat{w}_{t,d}^{(i)}$ using our IWSG method from chapter 2.4.2, yielding uniformly weighted samples from the node marginals. These particles are

Mixture Density Belief Propagation Message Passing

Given:

- Unary potential function $\phi_d(x_d; y)$ for all nodes $d \in V$
- Pairwise potential function $\phi_{s,d}(x_s, x_d)$ for all edges $(s, d) \in E$
- Observation y
- Initial set of N particles $x_{0,d}^{(i)} \sim \tilde{p}_0(x_d|y) \quad i = 1, ..., N$ and uniform weight $w_{0,d}^{(i)} = 1$

def MDBPMessagePassing(inputs):

1. For $r = 1, ..., R$ rounds

   (a) Draw N particles (with uniform weights) for each node:

   i. If $r \neq 1$ Then
   Draw from the previous estimate of the posterior:

$$x_{r,d}^{(i)} \sim \tilde{p}_{r-1}(x_d|y), \qquad w_{r,d}^{(i)} \text{ computed via IWSG,} \qquad \begin{matrix} \forall d \in V \\ i = 1, ..., N \end{matrix}$$

   ii. Else
   Use the initial set of particles and assign uniform weights:

$$x_{1,d}^{(i)} = x_{0,d}^{(i)}, \qquad w_{1,d}^{(i)} = w_{0,d}^{(i)}, \qquad \begin{matrix} \forall d \in V \\ i = 1, ..., N \end{matrix}$$

   (b) Using some message schedule, pass messages (i.e. compute message weights) from nodes to neighboring nodes until all messages converge:

$$w_{r,s\to d}^{(i)} = \sum_{j=1}^{N} \Big[\hat{w}_{r,s}^{(i)} \frac{\phi_{s,d}(x_{r,s}^{(j)}, x_{r,d}^{(i)}) \phi_s(x_{r,s}^{(j)}; y)}{\tilde{p}_{r-1}(x_{r,s}^{(j)})} \prod_{z \in \Gamma_s \setminus d} w_{r,z\to s}^{(j)} \Big], \qquad \begin{matrix} \forall (s,d) \in E \\ i = 1, ..., N \end{matrix}$$

   (c) Compute particle weights for the node marginals for each of the particles for all nodes:

$$w_{r,d}^{(i)} = \phi_d(x_{r,d}^{(i)}|y) \prod_{s \in \Gamma_d} w_{r,s\to d}^{(i)}, \qquad \text{Normalized s.t. } \sum_{i=1}^{N} w_{r,s\to d}^{(i)} = 1$$

   (d) Estimate new bandwidth parameter $\beta_d \quad \forall d \in V$

   (e) Construct new marginal probability estimate for each node:

$$\tilde{p}_r(x_d|y) = \sum_{i=1}^{N} w_{r,d}^{(i)} \cdot K(x_{r,d} - x_{r,d}^{(i)}; \beta_d), \qquad \forall d \in V$$

2. **Output:** $\Big\{ \{x_{R,d}^{(:)}, w_{R,d}^{(:)}, \beta_d\} | \quad \forall d \in V \Big\}$

Figure 4.5: The Mixture Density Belief Propagation message passing algorithm.

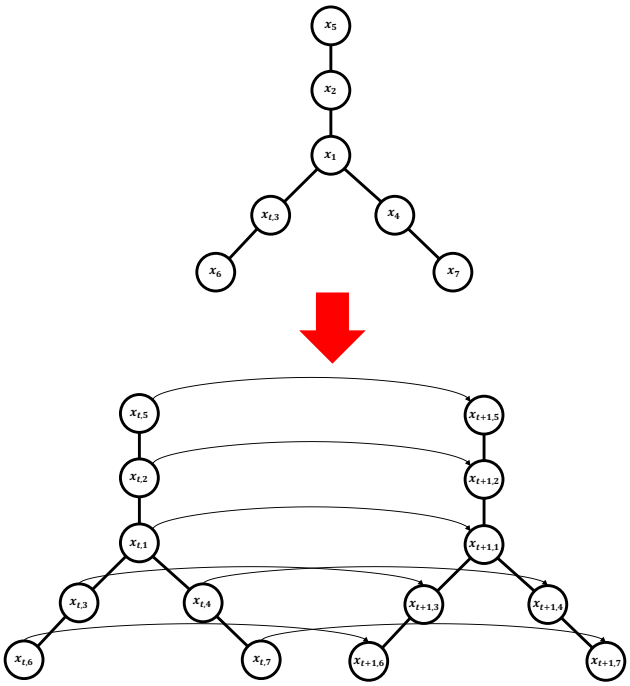


Figure 4.6: *Top:*Example probabilistic graphical model for static (non time evolving) problems. *Bottom:* Example probabilistic graphical model incorporating node dynamics for time evolving systems.

then propagated through time via a (possibly learned) dynamics model similar to that used by particle filters:

$$\check{x}_{t+1,d}^{(i)} \sim f_d(\hat{x}_{t,d}^{(i)}) \qquad \check{w}_{t+1,d}^{(i)} = \hat{w}_{t,d}^{(i)} \tag{4.14}$$

These propagated particles and weights, $\check{x}_{t+1,d}^{(i)}$ and $\check{w}_{t+1,d}^{(i)}$ respectively, along with the observation y_{t+1} can be used as input into MDBP message passing for timestep $t+1$ to compute the node marginals estimates for $t+1$. The complete MDBP algorithm is shown in Figs. 4.5 and 4.7.

4.5.3 Decoupling Resampling and Marginals Bandwidths

As explored in MDPF (chapter 2) and MDPS (chapter 3), decoupling the distributions used for resampling and the final node marginals can have added benefits. In MDBP, particle

Mixture Density Belief Propagation

Given:

- Observation sequence $y_1, ..., y_T$
- Initial estimate node marginals $\tilde{p}_0(x_d|y)$ for all nodes $d \in V$

def MDBP(inputs):

1. For $t = 1, ..., T$

   (a) Draw N particles (with uniform weights) for each node from the previous estimate of the posterior:

   $$\hat{x}^{(i)}_{t-1,d} \sim \tilde{p}_{t-1}(x_d|y_{t-1}), \qquad \hat{w}^{(i)}_{t-1,d} \text{ computed via IWSG,} \qquad \begin{matrix} \forall d \in V \\ i = 1, ..., N \end{matrix}$$

   (b) Apply node specific dynamics models:

   $$\check{x}^{(i)}_{t,d} \sim f_d(\hat{x}^{(i)}_{t-1,d}) \qquad \check{w}^{(i)}_{t,d} = \hat{w}^{(i)}_{t-1,d} \qquad \begin{matrix} \forall d \in V \\ i = 1, ..., N \end{matrix}$$

   (c) Run MDBP message passing (figure 4.5 to produce node marginals for timestep t

   $$\tilde{p}_t(x_d|y_t) = \text{MDBPMessagePassing}(y_t, \check{x}^{(:)}_{t,d}, \check{w}^{(:)}_{t,d})$$

2. **Output:** $\left\{ \tilde{p}_t(x_d|y_t) | \quad \forall d \in V, \quad t = 1, ..., T \right\}$

Figure 4.7: The Mixture Density Belief Propagation algorithm.

resampling has two competing goals; concentrating particles to high probability regions of the state space (exploiting known information about the marginals) and exploring new regions in the hopes of achieving a better marginal estimate. These goals may not align well with the node marginals which only seeking to exploit all available information to achieve the most accurate estimate possible. As such decoupling the node marginal estimates and the the resampling distributions can be beneficial. We achieve this decoupling in MDBP by using separate learned bandwidth parameters for resampling ($\hat{\beta}$) and marginal estimation (β).

4.5.4 Drawing Full Graph Samples

Like all BP algorithms, MDBP allows for drawing samples from the full graph. To do this we can select a root node x_{root} and draw an initial sample z_{root} from its marginal distribution. We then draw samples from the neighbors $s \in \Gamma_{\text{root}}$ of the selected root node, conditioned on the root nodes drawn sample. To sample from the neighbors $s \in \Gamma_{\text{root}}$ we first re-weight the particles $x_s^{(:)}$ of s using:

$$\acute{w}_s^{(i)} = \phi_s(x_s^{(i)}) \cdot \phi_{s,\text{root}}(x_s^{(i)}, z_{\text{root}}) \cdot \prod_{k \in \Gamma_s \setminus \text{root}} m_{k \to s}(x_s^{(i)}) \qquad i = 1, ..., N \tag{4.15}$$

These new weights can then be used along with a learned bandwidth parameter $\acute{\beta}$ to construct a mixture distribution from which samples can be drawn for node s:

$$z_s \sim \sum_{i=1}^{N} \acute{w}_s^{(i)} K(x_s^{(i)}, \acute{\beta}) \tag{4.16}$$

For this conditional sampling, it is expected that conditioning on a drawn sample from a neighboring node will reduce the variance of the distribution used to sample from the current node. Therefore it could be beneficial to use a new learned bandwidth parameter $\acute{\beta}$ instead of either the resampling or marginal bandwidths. In our experiments however we simply set

$\acute{\beta} = \hat{\beta}$, i.e. the resampling bandwidth. This process is repeated from the the root out until all nodes have drawn samples conditioned on their neighbor. These samples then represent a full graph sample $z = \{z_s | \forall s \in V\}$.

4.5.5 Training Procedure

To train our MDBP method, we optimize a Negative Log-Likelihood loss of the node marginals given the ground truth for all nodes at each timestep:

$$\mathcal{L} = \frac{1}{|V| \cdot T} \sum_{t=1,\ldots,T} \sum_{s \in V} \tilde{p}(x_{t,s}|y_t) \tag{4.17}$$

Like mixture density particle filters and smoothers, MDBP is sensitive to the bandwidth parameter. To prevent MDBP from optimizing to a local optima where the bandwidth parameter is overly wide, we train MDBP in 3 stages. In stage 1, we freeze the bandwidth and train the unary, pairwise, observation and dynamics models. In stage 2 we freeze all models and optimze the bandwidth parameters. In stage 3 we jointly optimize all models and bandwidths. This 3 stage approach prevents the bandwidth from becoming large while the other models are still unlearned. For all three stages we optimize using the same negative log-likelihood loss.

4.6 Experiments

We evaluate our mixture density belief propagation algorithm on the on the synthetic bearings-only tracking task from chapter 2.6.1, as well as on two articulated pose tracking tasks.

4.6.1 Bearings Only Tracking Task

To compare our MDBP algorithm to MDPF (chapter 2) and MDPS (chapter 3), we use the same bearings-only tracking task as chapter 2.6.1 where the 3D state of a variable-velocity synthetic vehicle is tracked via noisy bearings from a fixed-position radar. 85% of observations are the true bearing plus von Mises noise, while 15% are uniform outliers:

$$y_t \sim \alpha \cdot \text{Uniform}(-\pi, \pi) + (1 - \alpha) \cdot \text{VonMises}(\psi(x_t), \kappa),$$

where $\psi(x_t)$ is the true bearing and $\alpha = 0.15$.

Train and evaluation sequences have length $T = 50$ and we use the same experimental setup as described in chapter 3.5.1. We compare our MDBP method to MDPF, MDPS as well as the classic Forward Filter Backward Smoother (FFBS) [66, 67] as implemented in chapter 3.5.1.

Applying Mixture Density Belief Propagation

To apply MDBP to the bearings-only tracking task, we model this sequential state estimation problem as a Markov random field, shown in Fig. 4.8, where each node represents the state of the system at each timestep. Dynamics are encoded in the pairwise potentials therefore we do not require a separate dynamics model. Applying MDBP to this task amounts to doing inference on a probabilistic graphical model that does *not* evolve with time but instead is static with each node having a separate observation.

For the bearings only task, we initialize the first node in the MRF chain with the true state to mimic MDPF initialization with the other nodes randomly initialized. We run 2 rounds of MDBP message passing before returning the estimate marginals for each of the nodes. Further, each nodes unary potential is parameterized via the same neural network

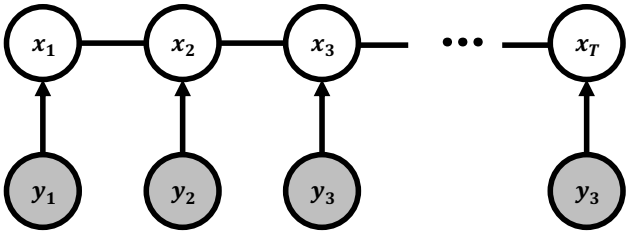


Figure 4.8: Probabilistic graphical model structure used for Mixture Density Belief Propagation for the Bearings-Only Tracking Task.

(sharing weights), conditioned on that nodes observation. Similarly each edges uses the sample pairwise potential network, applied to different pairs of nodes.

Results

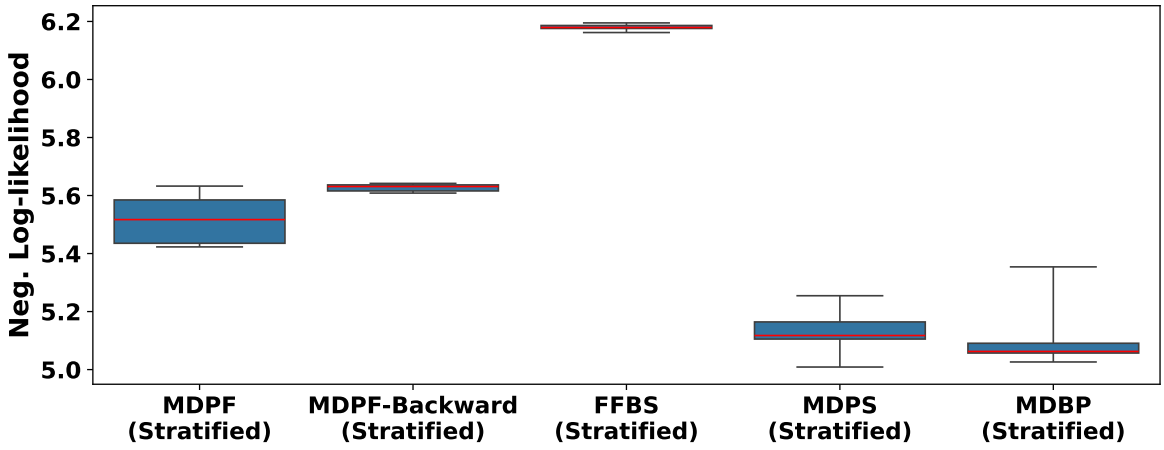


Figure 4.9: Box plots showing median (red line), quartiles (blue box), and range (whiskers) over 5 training runs for Bearings-Only tracking. As expected, MDBP shows substantial accuracy improvements when compared to MDPF as it incorporates both past and future observations when computing posteriors. MDBP also shows mild gains when compared to MDPS. This is expected since particles can be refined in subsequent messages passing iterations of MDBP.

In Fig. 4.9 we show statistics of performance over 5 training and evaluation runs for each method. We compare to MDPF (chapter 2), MDPS (chapter 3) and FFBS [66, 67] as implemented in chapter 3.5.1. As expected MDBP outperforms MDPF since it incorporates temporal information from both the future and the past. Even when unfairly provided the

true observation likelihood, FFBS performs poorly since particles are simply re-weighted (not moved) by the backward smoother.

Interestingly, MDBP shows mild gains over MDPS. This is because particles modeling the node marginals in MDBP are refined over several rounds of MDBP message passing leading to a better estimate. We do not expect large gains over MDPS since MDPS already incorporates past and future information when producing posteriors, using all available information and leaving little left to exploit to improved performance.

4.6.2 Articulated Spider Pose Estimation Task

To characterize MDBPs performance on a more complex dynamical systems we use the synthetic articulated spider pose estimation task from Opipari et al. [107]. As depicted in Fig. 4.10, the spider is comprised of 6 arms, 3 with constant length and 3 which can extend and contract. The spider can be modeled as an undirected pairwise markov random field with the dynamics added via directed edges between nodes at different timesteps. These 6 arms are connected to form an articulated spider consisting of 7 nodes, with the central node controlling the x-y position of the spider as a whole. In this task we wish to estimate the x-y position each node through time given image observations $y_1, ..., y_T$. These observations may contain various amounts of clutter which is designed to both confuse tracking algorithms as well as obscure the synthetic spider being tracked. For more details on this tracking task please see Opipari et al. [107].

In this task, we compare our MDBP to Differentiable Nonparametric Belief Propagation (DNBP) [107]. DNBP is not end-to-end differentiable and instead adopts similar gradient truncation as the truncated gradient particle filter [5], setting the gradient to zero at each particle resampling and (substantially) biasing the gradients. This inhibits DNBP to learn good dynamics models, and indeed for this task, Opipari et al. [107] does not seek to learn

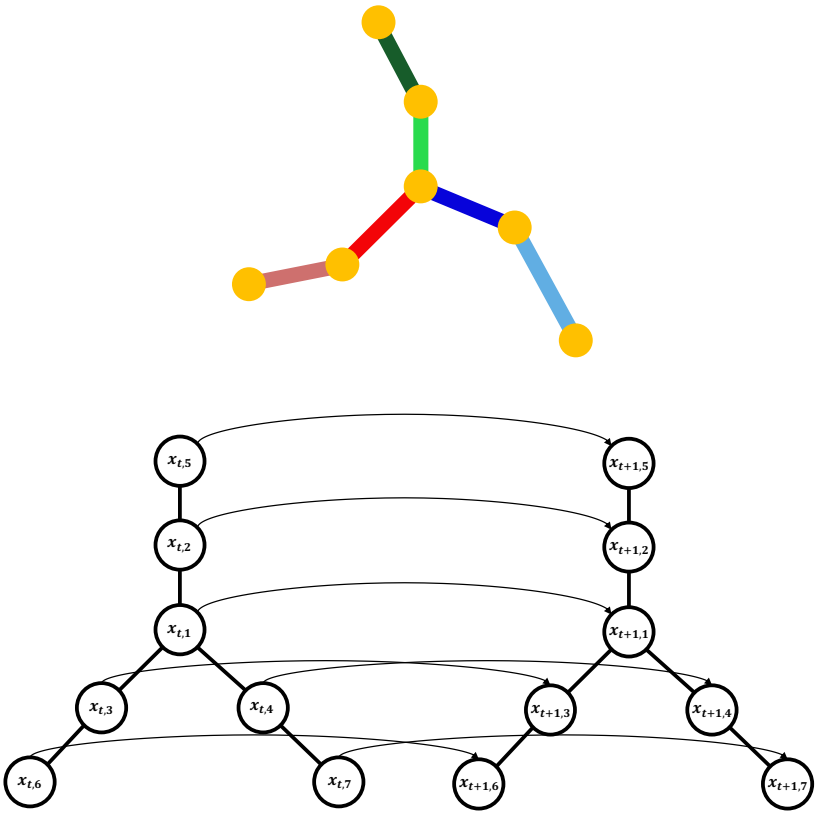


Figure 4.10: *Top:* Skeleton structure of the articulated spider. *Bottom:* Graphical model structure of the articulated spider pose estimation task.

useful dynamics models but instead uses a hand-crafted function. In contrast our MDBP is end-to-end differentiable, producing unbiased gradients allowing it to learn complex system dynamics effectively.

Implementation Details

We use the implementation in Opipari et al. [107] for DNBP and simply retrain the models using the provided dataset. DNBP specifies the unary and potential functions as simple Feed-Forward neural networks (constrained to be positive) with each node and edge using separate weights. Observations are encoded via separate observation encoders for each node and each nodes unary potential network is conditioned on that specific nodes observation encoding. In Opipari et al. [107] the observation encoders parameterized as Convolutional

Neural Networks.

For MDBP, we encode the image observations using a Vision Transformer (ViT) [118] to produce low dimensional feature vector observation encodings. Unary potentials are parametrized as simply Feed-Forward neural networks which score particles for each node conditioned on the encoded observation. Pairwise potentials are parametrized as simply Feed-Forward neural networks as well. Both potential functions are constrained to output positive values. All nodes share the same ViT observation encoder but do not share the unary potentials, with each node using a separate network. Pairwise potentials are also not shared, with each edge using a separate network. Finally we use node-specific dynamics, parameterized via neural networks using the reparameterization trick (see chapter 2.3) where separate dynamics models are independently applied to each node. This dynamics network is a residual network, simply adjusting the particle locations when applied.

Due to deficiencies in DNBP, a slow and cumbersome training procedure to ensure effective models are learned must be used. This procedure requires re-computing the observation encodings multiple times for each round of message passing limiting the use of large powerful network architectures. To achieve a fair comparison with models using a ViT observation encoder, we also compare to DNBP with a pre-trained observation encoder. Here DNBP is given a pre-trained ViT with observations encodings shared for all of the nodes. DNBP does not need to train the observation encoder and thus the encodings can be can be cached. We also increase the number of parameters of DNBPs other networks to match that of MDBP.

We initialize MDBP with 250 particles on the first frame before reducing to 50 particles for each subsequent frame. DNBP in contrast uses 100 particles for all frames but computes the final node marginals using 300 particles by combining and weighting particles from neighboring nodes. Both models are trained using the Negative Log-Likelihood of the true

state for each node at each timestep:

$$\mathcal{L}_{NLL} = \frac{1}{T] \cdot |V|} \sum_{d \in V} \sum_{t=1}^{T} -\log(\tilde{p}_t(x_{t,d} | x_{t,d}^{(:)}, w_{t,d}^{(:)}, \beta, y_t)) \tag{4.18}$$

Results

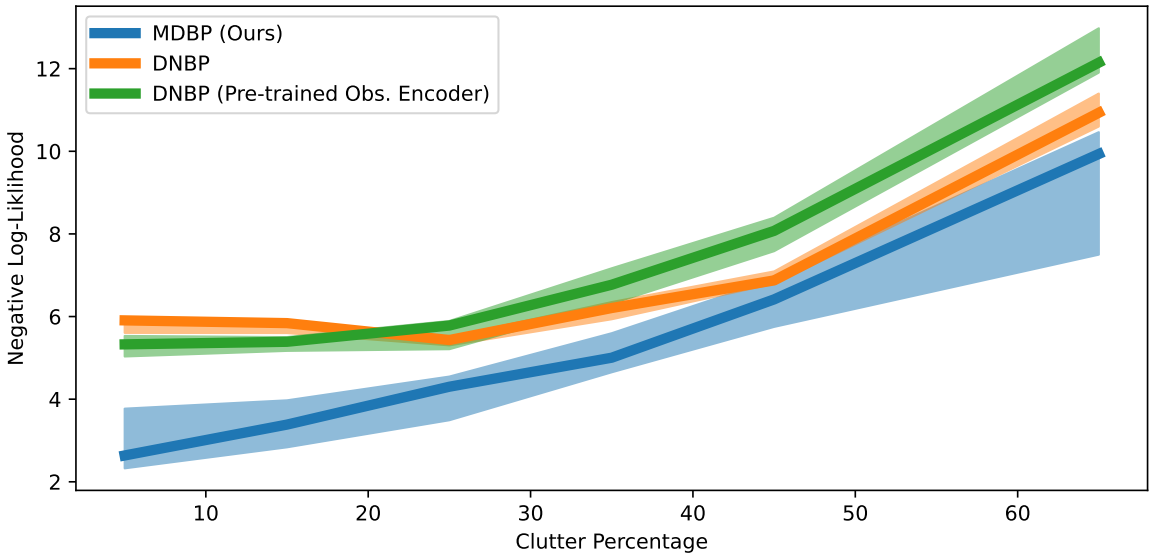


Figure 4.11: Negative Log-Likelihood (NLL) curves showing median (solid line) and range (shaded region) of NLL values over 5 training and evaluation runs for sequences as the percentage of static clutter is increased. Interestingly DNBP with a pre-trained ViT observation encoder and larger unary and pairwise networks performs worse than DNBP with smaller networks, highlighting the negative effects of gradient truncation where substantial bias may interfere with training. MDBP shows substantial improvements of DNBP due to its ability to learn complex system dynamics via end-to-end learning.

In Fig. 4.11 we show statistics of performance over 5 training and evaluation runs for each method. As the amount of clutter in the observation images in increased, we unexpectedly see a drop in performance for all methods. Comparing DNBP as implemented in Opipari et al. [107] with our implementation of DNBP with larger networks for the unary and pairwise potential we see that increasing the network sizes does not improve performance. This is caused by possibly substantial bias in the gradients from truncation preventing effective learning as the number of parameters is increased. MDBP shows major improvements over

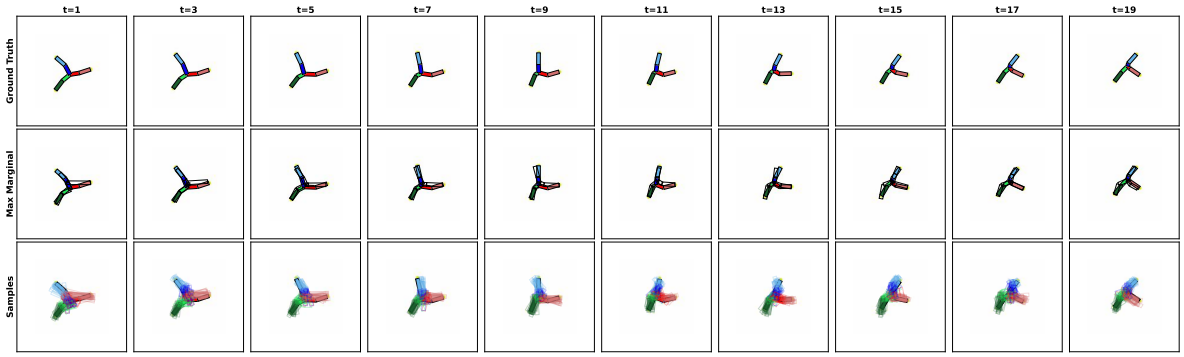


Figure 4.12: Example rendering on the articulated spider pose estimation task using mixture density belief propagation. *Top:* The ground truth articulated spider. *Middle:* Maximum marginal of each node, achieved by finding an approximate peak in the node marginal distributions for each node separately. *Bottom:* 10 samples of the full articulated spider skeleton from MDBP.

DNBP due to its ability to learn complex dynamics from end-to-end training, even when using much fewer particles.

Figure 4.12 shows an example rendered sequence for MDBP. We plot the maximum marginal of each node, achieved by finding an approximate peak in the node marginal distributions for each node separately. We also plot 10 samples of the full spider skeleton using the sampling method described in chapter 4.5.4. Figure 4.13 shows an example sequence with clutter. MDBP is able to successfully track each of the spider nodes even when the observations contain clutter meant to confuse the MDBP algorithm.

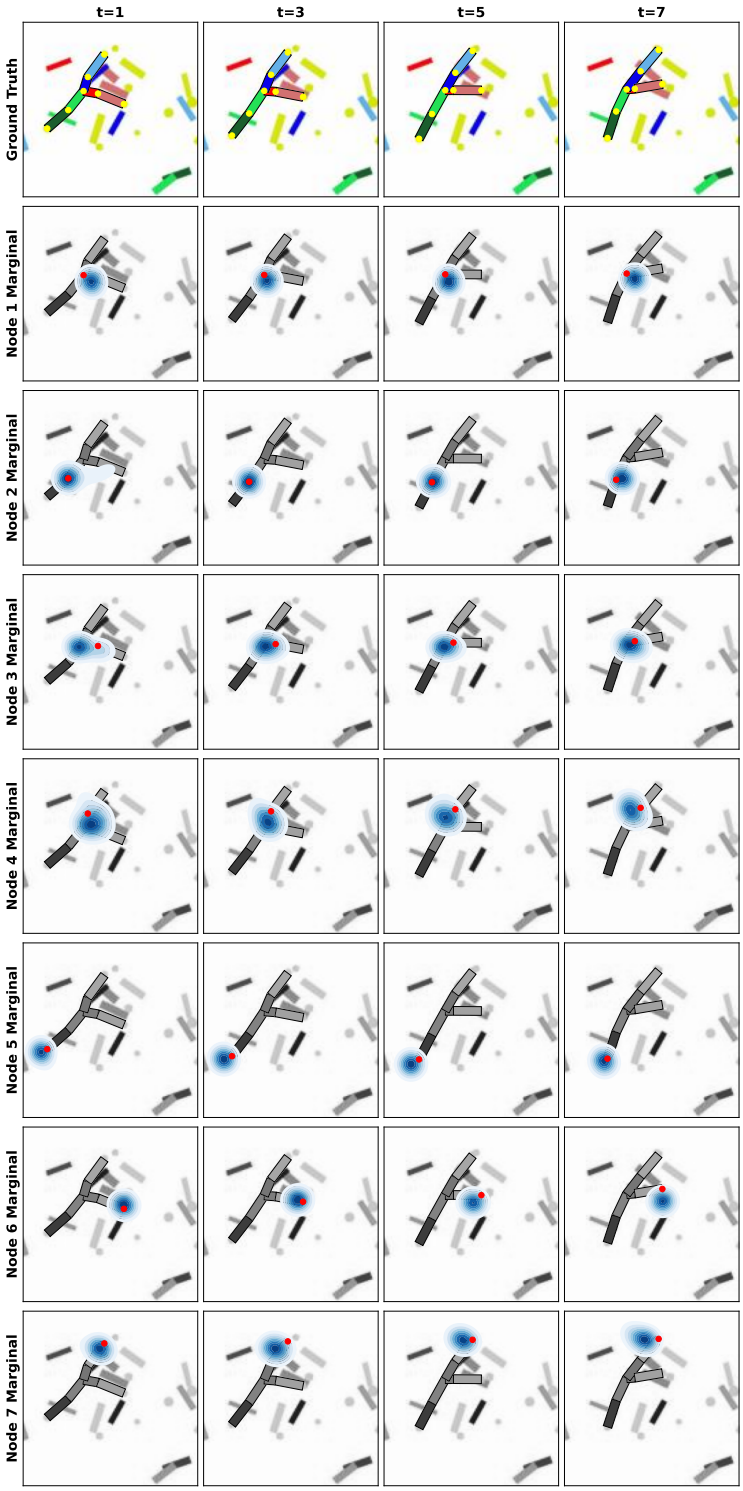


Figure 4.13: Example rendering on the articulated spider pose estimation task using mixture density belief propagation. This example showcases the ability of our mixture density belief propagation to accurately estimate the node marginals even when the observations are cluttered (15 percent clutter). *Top:* The ground truth articulated spider. *Rows 2-8:* The marginal distributions (blue clouds) for each of the 7 spider nodes (red dots).

4.6.3 Human Mesh Recovery from Video

Human Mesh Recovery (HMR) is the task of recovering a human mesh from video sequences of humans doing various tasks such as dancing or gymnastics. In HMR we wish to estimate the 3D mesh of a human subject as the subject moves through the environment [88, 89, 90, 91, 92, 93]. This poses an interesting challenge as much of the 3D mesh information is lost in the 2D video sequence. Advances in human body modeling have led to a great simplification of HMR. Instead of estimating the raw 3D position of each vertex in the human mesh (which could range in the thousands), recent advanced human body models such as SMPL [94], SMPL-X [95] and SUPR [96] can create these raw 3D meshes using only a few parameters as input. Specifically in the case of SMPL, we only need to pass in the rotation of 23 key body joints as well as a 10 dimensional body shape vector for SMPL to construct the final 3D mesh. This reduces the HMR problem to only needing to estimate a relatively few number of parameters.

With the advent of deep learning, regressing human meshes from video via a large neural network has become popular [88, 89, 90, 91, 92, 93]. Methods that regress meshes from video frames individually have been quite successful, especially as the size of these models have grown [88, 90, 119, 120, 121, 122]. To incorporate temporal information, several methods deploy a temporal encoder, which ingests features from large single frame HMR predictors, to produce better estimates of the human mesh through time [89, 123, 93, 92]. A key limitation of these methods is they do not model uncertainty, often only returning a single point wise estimate for the SMPL parameters. Further these methods do not exploit the natural graphical model structure inherent in the joint angle skeleton whereby joints are heavily influenced by neighboring joints and less so by farther away joints.

Our 3D human mesh recovery task for articulated human bodies is adapted from Goel et al. [90]. We use the Human3.6M dataset [124] and estimate the 23 body joint angle and the

10-dimensional body shape parameters used by the SMPL [94] model, generating the final 3D mesh via the SMPL body model. Since Human3.6M only gives ground truth estimate of the 3D positions of several keypoints, tracked using motion capture, Goel et al. [90] estimates ground truth SMPL parameters via an expensive but accurate regression method that minimizes a re-projection loss [125] when projecting the SMPL generated mesh onto the tracked keypoints. We split the Human3.6M dataset into training (60%), validation (10%) and test (30%) sets and train on sequences of length 5.

We qualitatively compare our MDBP method to HMR2.0 [90], a leading human mesh reconstruction method, on a variety of sequences from Human3.6M. Note though HMR and HMR2.0 share similar names, HMR refers to the human mesh recovery from video task whereas HMR2.0 refers to a specific method for solving this task. HMR2.0 uses a large Vision Transformer (ViT)[118] to encode observations (images) into a lower dimensional latent space before using linear projections to regress the final SMPL parameters. HMR2.0 estimates SMPL parameters individually for each image, though it has shown improved performance when compared to other methods, including ones that use temporal information, due to its use of a large ViT encoder[90].

Implementation Details

We implement HMR via MDBP by modeling the human skeleton joints via a pairwise undirected graphical model as shown in Fig. 4.14 where observations are shared for all nodes. We encode image observations using the same ViT implemented in HMR2.0 and do not seek to fine-time this observation encoder. Instead we rely on the unary potentials, pairwise potentials and dynamics models to improve HMR performance. Like in the articulated spider tracking tasks, we parameterize these models via Feed-Forward neural networks, we however we supply the pairwise potential with the encoded observation (i.e. conditioning it on the observation) to help with scoring pairs of particles from adjacent nodes. Since the SMPL

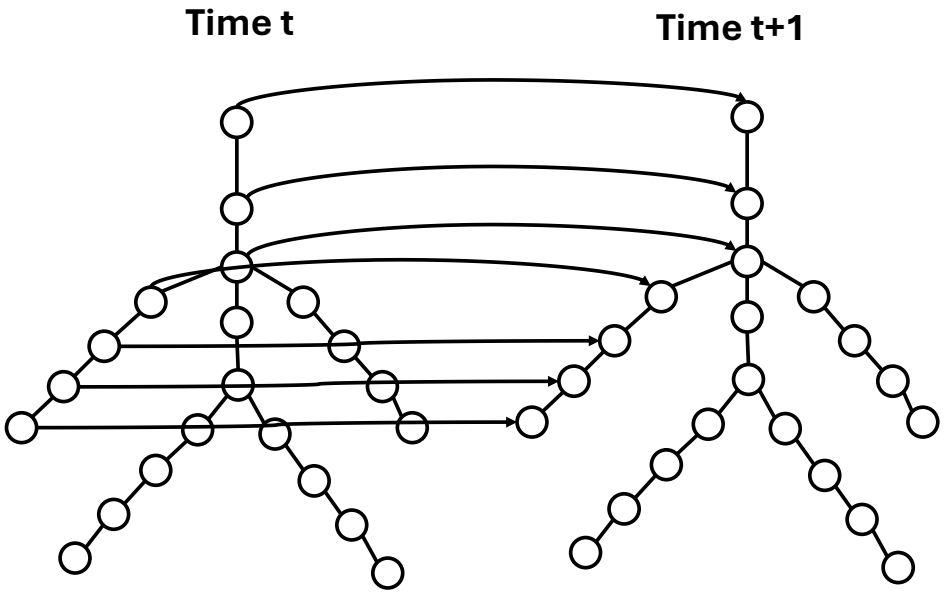


Figure 4.14: Graphical model structure for the human mesh reconstruction task. For clarity only a subset of the directed edges have been drawn. Each node shares the same observation (not drawn).

body shape parameters are intended to be static, they encode the general body shape of the human subject and are not influenced by the subjects pose or motion, we simply use the HMR2.0 estimate to set these parameters. MDBP therefore seeks to only estimate the joint angle values for the 23 SMPL joints.

We use HMR2.0 to initialize our MDBP algorithm on the first frame by creating a particle cloud centered at the HMR2.0 prediction. After the first frame, these particles are propagated through time with the learned node-specific system dynamics. In this task, we set the number of particles for MDBP to be 50. Due to computation constraints, and because the amount of human motion between video frames is typically small, our MDBP implementation only uses 1 round of message passing within each time step.

4.6.4 Results

Figures 4.15, 4.16 and 4.17 qualitatively compares MDBP to the HMR2.0 [90]. To render full body meshes, we extract the global max of the node marginals for each of the nodes (i.e. SMPL joints) and use that as input to the SMPL model. Of note, we plot the meshes using the ground truth global orientation parameter, which controls the overall rotation of the human body mesh, in order to aid comparing produced body meshes with the ground truth.

Since HMR2.0 estimates the human mesh for each video frame individually, it often poorly estimates the angles for certain joint, specifically the feet and hands, resulting in non-human like poses. These poor angle estimates occur when joints are occluded or when those joints blend with the background, as is the case with some of the feet, confusing HMR2.0. Interestingly, MDBP does not suffer from such errors. Though early timesteps of several sequences do have large errors in the extracted joint angles, this results from a poor initialization from HMR2.0. As subsequent frames are ingested by MDBP, temporal information can be incorporated to correct these errors resulting in more realistic poses. Further due to the learned dynamics, MDBP produces smoother body pose trajectories by initializing the MDBP message passing at time t with propagated particles from $t-1$, utilize knowledge of how human body poses evolve over time. Moreover, MDBP is a particle based algorithm and thus maintains multiple particles for each joint. If certain particles diverge from the true pose then they will be down-weighted in message passing rounds of MDBP and replaced with more accurate particles during resampling. Using particles allows MDBP to maintain good pose hypothesis while exploring for better ones, preventing poor pose esitmates.

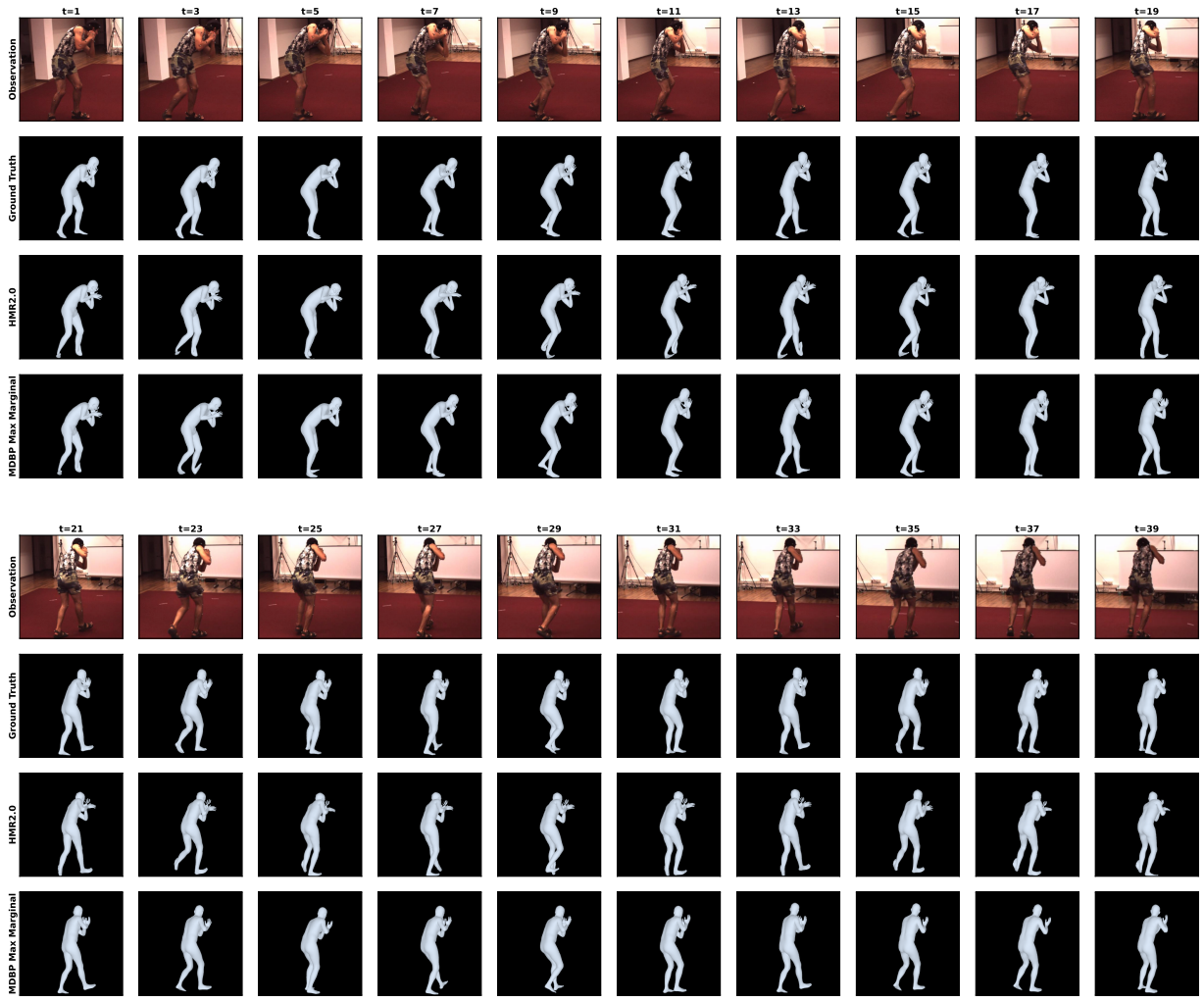


Figure 4.15: Qualitative results on the human mesh recovery task. HMR2.0 fails to correctly estimate the feet joint angles resulting in unrealistic body meshes. Though initialized poorly with HMR2.0, MDBP is able to correct these poor feet joint angles as it ingests additional observations, highlighting the strength of using temporal information.

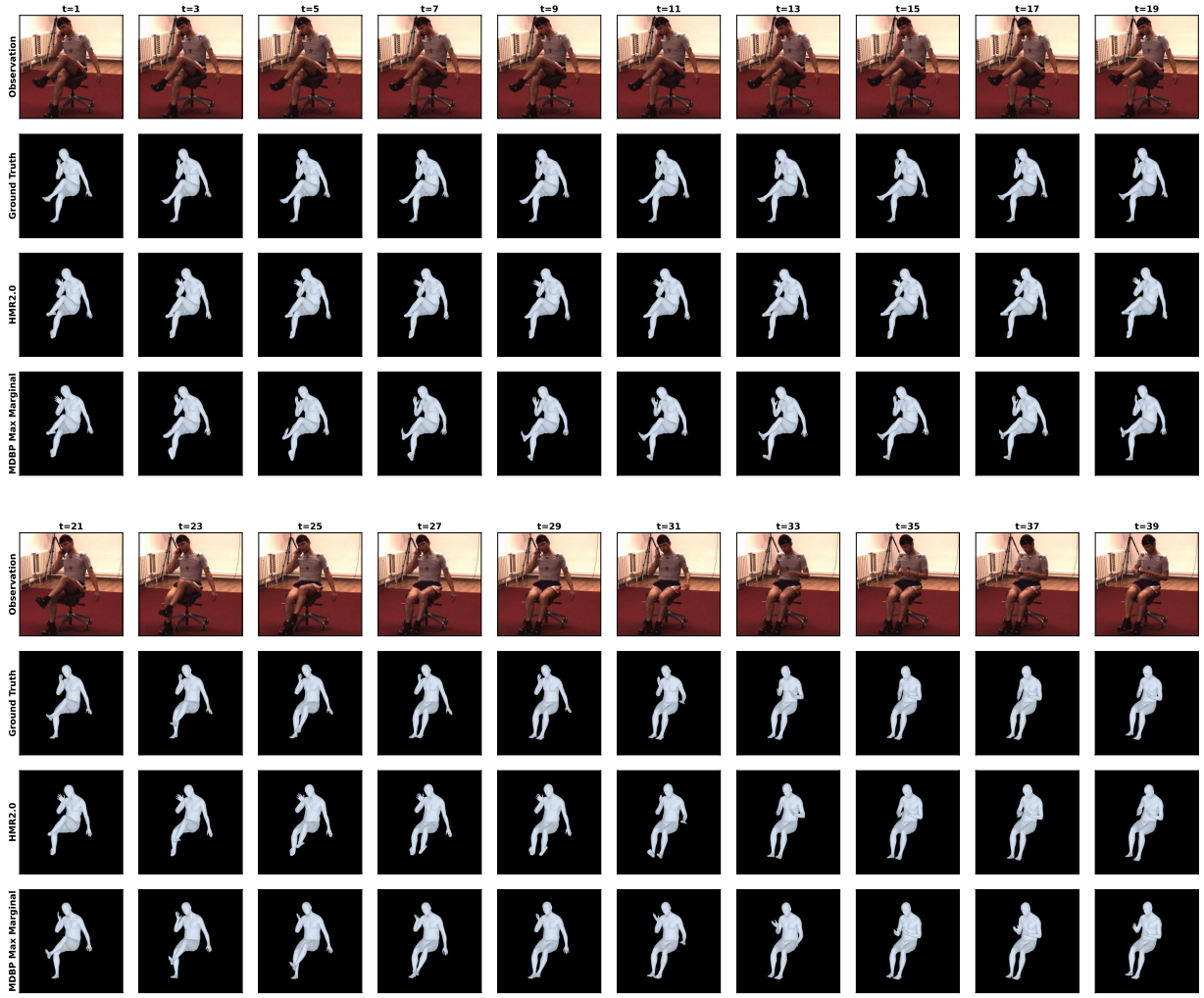


Figure 4.16: Qualitative results on the human mesh recovery task. HMR2.0 fails to correctly estimate the feet joint angles resulting in unrealistic body meshes. Further, HMR2.0 exhibits rapid pose changes since it does not have any knowledge of smooth dynamics (see rapid correction of the feet at $t = 29$ to $t = 31$). Though initialized poorly with HMR2.0, MDBP is able to correct these poor feet joint angles as it ingests additional observations and smoothly evolves the body pose though time via the learned dynamics.

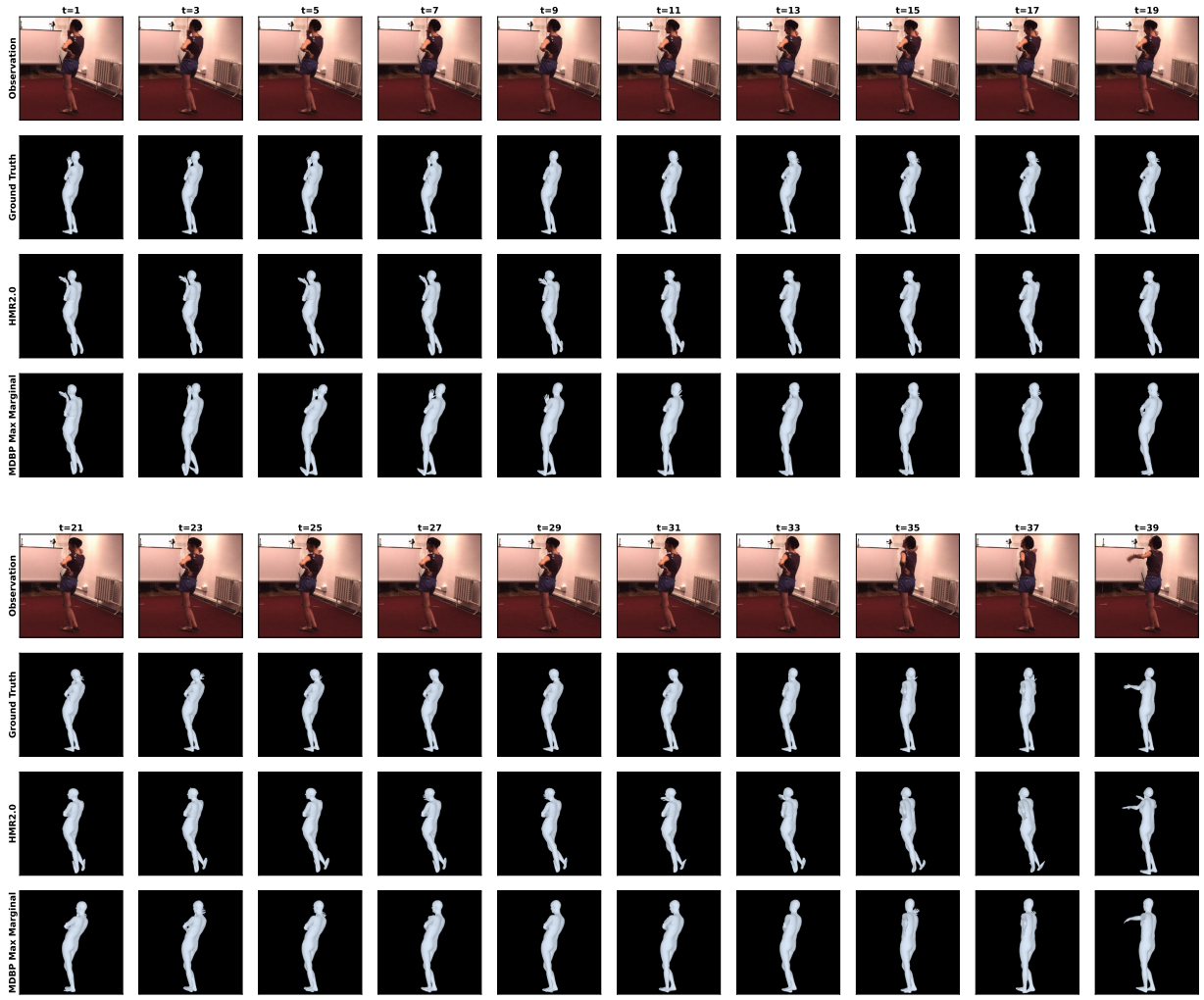


Figure 4.17: Qualitative results on the human mesh recovery task. HMR2.0 fails to correctly estimate the feet joint angles resulting in unrealistic body meshes. HMR2.0 also exhibits rapid pose changes since it does not have any knowledge of smooth dynamics (see rapid changes in the feet from $t = 27$ to $t = 35$ and rapid changes in hand placement from $t = 33$ to $t = 35$). Though initialized poorly with HMR2.0, MDBP is able to correct poor joint angles as it ingests additional observations. Further MDBP exhibits only smooth pose changes due to using learned temporal dynamics to propagate the previous pose particles to the current timestep instead of re-regressing each new observation alone (like HMR2.0).

4.7 Discussion

We have developed a fully-differentiable, end-to-end learnable belief propagation (mixture density belief propagation) method for applications in dynamical systems with continuous state spaces. MDBP effectively learns useful node dynamics as well as robust unary and pairwise potential functions, showing success on a range of complicated tasks such as articulated pose estimation from video sequences. When applied to the task of human mesh recovery, MDBP produces more consistent and robust human body poses compared to other leading methods that do not model state dynamics.

Chapter 5

Conclusion and Future Directions

Previous chapters developed particle based end-to-end differentiable inference methods for a variety of graphical model structures. In this chapter we review the main contributions in this thesis and highlight some possible areas for future work.

5.1 Summary of Methods and Contributions

State estimation for dynamic systems is a challenging yet prevalent task that has applications in many domains from computer vision to robotics and more. In the age of deep learning, solving these tasks using neural networks has grown popular, and blending classical inference methods with the latest neural architectures has shown to be especially fruitful. In this dissertation we developed several inference algorithms that blend classical particle-based methods with neural networks for various state estimation tasks which we outline below.

In chapter 2 we developed an end-to-end differentiable and learnable particle filter algorithm, *Mixture Density Particle Filter*, in the process creating an unbiased stable method for differentiable resampling from mixture distributions which we named *Importance Weighted Sam-*

ple Gradients (IWSG). We compared IWSG to Implicit Reparameterized Gradients [10] and showcase fundamental flaws in reparameterization-based differentiable resampling methods. We then showcased our mixture density particle filters superior performance when compared to other other learnable particle filter methods on a variety of localization tasks.

In chapter 3 we extended ideas from our mixture density particle filter to produce an end-to-end differentiable and learnable particle smoother algorithm, which uses observations from both the past and the future when estimating state posteriors. We name our method the *Mixture Density Particle Smoother* and, to the best of our knowledge, it is the first end-to-end differentiable particle smoother algorithm to be developed. We compare our particle smoothing method to a variety of localization tasks including the difficult global localization task in city-scale environments where our smoother significantly outperformed existing methods tailored to this specific task.

Finally in chapter 4, we extend ideas from our mixture density particle filter and smoother to produced an end-to-end differentiable and learnable particle belief propagation algorithm that can perform accurate inference on dynamical Markov random fields. We name our method the *Mixture Density Belief Propagation* and apply it to the difficult task of articulated pose estimation. When compared to existing differentiable belief propagation methods, our mixture density belief propagation shows superior performance. Further when tested on the difficult task of human mesh recovery from video sequences, our method produces smoother body trajectories and more accurate body meshes when compared to existing leading methods.

5.2 Suggestions for Future Work

5.2.1 Mixture Density Belief Propagation for Loopy Graphs

In chapter 4 we applied our mixture density belief propagation algorithm to tree-structured Markov random fields. Though this was effective, not all tasks can be expressed as an tree-structured pairwise graphical model. Some tasks such as depth-from-stereo [126] or semantic segmentation [127] are more faithfully modeled via a graph containing loops and using tree-structured graphs makes conditional independence assumptions that may be too simplistic. Though applying belief propagation to these loopy graphs has only partial theoretical justifications, using BP for inference has been shown to have good empirical performance when applied to loopy tasks [97, 87], though the number of message passing rounds is greater than what is required for tree-structured graphs.

Direct application of our mixture density belief propagation algorithm to loopy graphs requires additional advances. MDBP is a learned method, computing gradients by differentiating through the various message passing and resampling rounds. Since belief propagation on loopy graphs requires possibly many more rounds of message passing for messages to converge, differentiating through these many rounds may be intractable due to the memory required to store the long gradient tape. Applying the implicit function theorem [128] could be an interesting way to mitigate the need to store such a long gradient tape as gradients can be directly computed using only the converged messages without the need to differentiate through the whole message passing algorithm, though this requires additional development in the context of particle-based message representations.

5.2.2 Mixture Density Belief Propagation Multi-Person Tracking

Mixture density belief propagation is a particle based BP algorithm and thus is able to model node marginals with multiple modes effectively. In chapter 4 we applied our mixture density belief propagation algorithm to the human mesh recovery task where we used video sequences containing only 1 subject. Theoretically, our MDBP algorithm should be able to correctly model videos with multiple subjects by maintaining particles for each subject within the node marginals. This should allow for robustness when subjects cross paths, helping to maintain tracking and correct human meshes. Applying our MDBP to problems in which multiple persons exist, as opposed to just potentially multi-modal uncertainty in the pose of a single person, would be an interesting topic to explore.

5.2.3 Mixture Density Particle Filter Simultaneous Localization and Mapping using 3D Gaussian Splatting

Visual Simultaneous Localization and Mapping (V-SLAM) is the task of localizing a camera as it moves through an environment while simultaneously mapping the environment [13, 129, 130, 131, 132]. V-SLAM can be decomposed into two parts. The tracking front-end performs tracking, localizing the camera using the current estimate of the map. Common methods for tracking front-ends include filters [133, 134], sliding window bundle adjustment [132], or more recently directly regressing via neural networks [131, 135, 136, 137]. After tracking via the tracking front-end, the map is then refined using the current localized camera estimate as well as the latest observation in the mapping backend. A recent trend is representing the mapping backend using 3D Gaussian Splats [135, 136, 137, 138], which produces maps capable of novel view synthesis which are useful for tasks such as robotics and AR/VR. Unfortunately, most Gaussian splatting V-SLAM methods only produce point estimates for the camera position without modeling uncertainty [135, 136, 137].

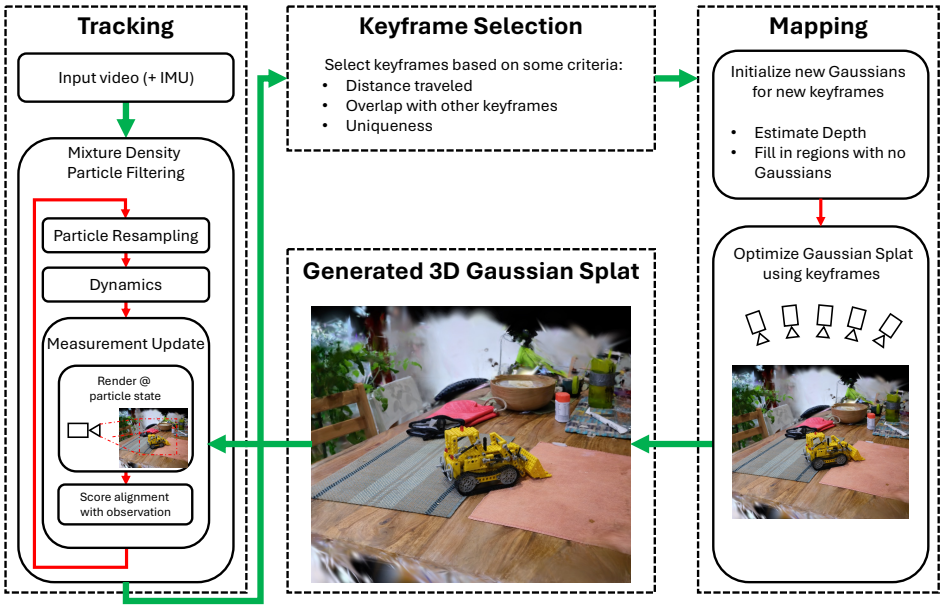


Figure 5.1: Proposed framework for 3D Gaussian Splatting based simultaneousness localization and mapping using Mixture Density Particle Filters.

An interesting idea would be to use our mixture density particle filtering and smoothing methods from chapters 2 and 3 for the tracking front-end along with Gaussian splatting [138] for the map representation. Since MDPF/S use particles to estimate the camera localization state posteriors, they give a natural method to construct measurement models using Gaussian splatting. At each measurement weighting step, we can query the current Gaussian splat map to render a novel view images for all the current particles. We can then weight these particles by how well the rendered image aligns the observation, down-weighting particles with poor alignment. We can then use the latest camera estimate (i.e. the weighted particle set) to refine the Gaussian splat backend by sampling particles from the posterior estimate, with probability equal to their weight, as the camera poses during Gaussian splatting training. This would yield a V-SLAM algorithm capable of uncertainty estimation with a mapping backend capable of novel view synthesis while exploring higher dimensional application for MDPF/S.

A proposed V-SLAM using MDPF and Gaussian splatting is shown in Fig. 5.1.

Bibliography

[1] Neil J Gordon, David J Salmond, and Adrian FM Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEE proceedings F-Radar and Signal Processing*, 1993.

[2] K. Kanazawa, D. Koller, and S. Russell. Stochastic simulation algorithms for dynamic probabilistic networks. In *UAI 11*, pages 346–351. Morgan Kaufmann, 1995.

[3] A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, New York, 2001.

[4] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Trans. Signal Proc.*, 50 (2):174–188, February 2002.

[5] Rico Jonschkowski, Divyam Rastogi, and Oliver Brock. Differentiable particle filters: End-to-end learning with algorithmic priors. *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

[6] Adam Ścibior, Vaden Masrani, and Frank Wood. Differentiable particle filtering without modifying the forward pass. *International Conference on Probabilistic Programming (PROBPROG)*, 2021.

[7] Peter Karkus, David Hsu, and Wee Sun Lee. Particle filter networks with application to visual localization. *Conference on Robot Learning (CORL)*, 2018.

[8] Adrien Corenflos, James Thornton, George Deligiannidis, and Arnaud Doucet. Differentiable particle filtering via entropy-regularized optimal transport. *International Conference on Machine Learning (ICML)*, 2021.

[9] Alex Graves. Stochastic backpropagation through mixture density distributions. *arXiv preprint arXiv:1607.05690*, 2016.

[10] Mikhail Figurnov, Shakir Mohamed, and Andriy Mnih. Implicit reparameterization gradients. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018.

[11] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 1960.

[12] T. Kailath, A. H. Sayed, and B. Hassibi. *Linear Estimation*. Prentice Hall, New Jersey, 2000.

[13] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN 0262201623.

[14] S. J. Julier and J. K. Uhlmann. Unscented filtering and nonlinear estimation. *Proc. IEEE*, 92(3):401–422, March 2004.

[15] Víctor Elvira, Luca Martino, and Christian Robert. Rethinking the effective sample size. *International Statistical Review*, 2022.

[16] Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In *ICML*, page II–1791–1799, 2014.

[17] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *International Conference on Learning Representations (ICLR)*, 2014.

[18] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, volume 32, pages 1278–1286, 2014.

[19] N. Oudjane and C. Musso. Progressive correction for regularized particle filters. *International Conference on Information Fusion*, 2000.

[20] C. Musso, N. Oudjane, and F. Le Gland. Improving regularized particle filters. In A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, pages 247–271. Springer-Verlag, 2001.

[21] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, 1986.

[22] Adrian W Bowman. An alternative method of cross-validation for the smoothing of density estimates. *Biometrika*, 1984.

[23] M Chris Jones, James S Marron, and Simon J Sheather. A brief survey of bandwidth selection for density estimation. *Journal of the American Statistical Association*, 1996.

[24] Nikolas Kantas, Arnaud Doucet, Sumeetpal S Singh, Jan Maciejowski, and Nicolas Chopin. On particle methods for parameter estimation in state-space models. *Statistical Science*, 30(3):328–351, 2015.

[25] Tuomas Haarnoja, Anurag Ajay, Sergey Levine, and Pieter Abbeel. Backprop kf: Learning discriminative deterministic state estimators. *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[26] Minyoung Kim and Vladimir Pavlovic. Conditional state space models for discriminative motion estimation. *IEEE International Conference on Computer Vision*, 2007.

[27] John D. Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *International Conference on Machine Learning (ICML)*, 2001.

[28] Benson Limketkai, Dieter Fox, and Lin Liao. Crf-filters: Discriminative particle filters for sequential state estimation. *IEEE International Conference on Robotics and Automation (ICRA)*, 2007.

[29] Rob Hess and Alan Fern. Discriminatively trained particle filters for complex multi-object tracking. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[30] Michael Zhu, Kevin P. Murphy, and Rico Jonschkowski. Towards differentiable resampling. *ArXiv*, abs/2004.11938, 2020.

[31] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 1990.

[32] Jakob Foerster, Gregory Farquhar, Maruan Al-Shedivat, Tim Rocktäschel, Eric Xing, and Shimon Whiteson. DiCE: The infinitely differentiable Monte Carlo estimator. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 1529–1538, 2018.

[33] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *International Conference on Learning Representations (ICLR)*, 2016.

[34] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *International Conference on Learning Representations (ICLR)*, 2016.

[35] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in Neural Information Processing Systems (NeurIPS)*, 2013.

[36] Martin Jankowiak and Fritz Obermeyer. Pathwise derivatives beyond the reparameterization trick. *International Conference on Machine Learning (ICML)*, 2018.

[37] Martin Jankowiak and Theofanis Karaletsos. Pathwise derivatives for multivariate distributions. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019.

[38] V. A. Epanechnikov. Non-parametric estimation of a multivariate probability density. *Theory of Probability & Its Applications*, 1969.

[39] Ludger Rüschendorf. *Copulas, Sklar's Theorem, and Distributional Transform.* Springer Berlin Heidelberg, 2013.

[40] D. Alspach and H. Sorenson. Nonlinear bayesian estimation using Gaussian sum approximations. *IEEE Transactions on Automatic Control*, 1972.

[41] Mike Klaas, Nando de Freitas, and Arnaud Doucet. Toward practical N^2 Monte Carlo: The marginal particle filter. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 308–315, 2005.

[42] Jinlin Lai, Justin Domke, and Daniel Sheldon. Variational marginal particle filters. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2022.

[43] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[44] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 28, 2015.

[45] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[46] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *International Conference on Machine Learning (ICML)*, 2013.

[47] Herbert Jaeger. Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach. *GMD-Forschungszentrum Informationstechnik Bonn*, 2002.

[48] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

[49] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[50] Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. Building generalizable agents with a realistic and rich 3d environment. *arXiv preprint arXiv:1801.02209*, 2018.

[51] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014.

[52] Yujiao Shi, Xin Yu, Dylan Campbell, and Hongdong Li. Where am i looking at? joint location and orientation estimation by cross-view matching. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[53] Sixing Hu, Mengdan Feng, Rang M. H. Nguyen, and Gim Hee Lee. Cvm-net: Cross-view matching network for image-based ground-to-aerial geo-localization. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[54] Noe Samano, Mengjie Zhou, and Andrew Calway. You Are Here: Geolocation by Embedding Maps and Images. In *European Conference on Computer Vision (ECCV)*, 2020.

[55] Sijie Zhu, Taojiannan Yang, and Chen Chen. Vigor: Cross-view image geo-localization beyond one-to-one retrieval. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3640–3649, 2021.

[56] Yujiao Shi, Liu Liu, Xin Yu, and Hongdong Li. Spatial-aware feature aggregation for image based cross-view geo-localization. In *Advances in Neural Information Processing Systems (Neurips)*, 2019.

[57] Yujiao Shi, Xin Yu, Liu Liu, Tong Zhang, and Hongdong Li. Optimal feature transport for cross-view image geo-localization. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2020.

[58] Zimin Xia, Olaf Booij, Marco Manfredi, and Julian FP Kooij. Visual cross-view metric localization with dense uncertainty estimates. In *European Conference on Computer Vision (ECCV)*, 2022.

[59] Mengjie Zhou, Xieyuanli Chen, Noe Samano, Cyrill Stachniss, and Andrew Calway. Efficient localisation using images and openstreetmaps. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[60] Yujiao Shi and Hongdong Li. Beyond cross-view image retrieval: Highly accurate vehicle localization using satellite image. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[61] Paul-Edouard Sarlin, Ajaykumar Unagar, Måns Larsson, Hugo Germain, Carl Toft, Victor Larsson, Marc Pollefeys, Vincent Lepetit, Lars Hammarstrand, Fredrik Kahl, and Torsten Sattler. Back to the Feature: Learning Robust Camera Localization from Pixels to Pose. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[62] Paul-Edouard Sarlin, Daniel DeTone, Tsun-Yi Yang, Armen Avetisyan, Julian Straub, Tomasz Malisiewicz, Samuel Rota Bulo, Richard Newcombe, Peter Kontschieder, and Vasileios Balntas. OrienterNet: Visual Localization in 2D Public Maps with Neural Matching. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[63] Adrien Corenflos, James Thornton, George Deligiannidis, and Arnaud Doucet. Differentiable particle filtering via entropy-regularized optimal transport. *International Conference on Machine Learning (ICML)*, 2021.

[64] Yoram Bresler. Two-filter formulae for discrete-time non-linear bayesian smoothing. *International Journal of Control*, 1986.

[65] Genshiro Kitagawa. Monte carlo filter and smoother for non-gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, 1996.

[66] Arnaud Doucet, Adam M Johansen, et al. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of nonlinear filtering*, 2009.

[67] Mike Klaas, Mark Briers, Nando de Freitas, A. Doucet, Simon Maskell, and Dustin Lang. Fast particle smoothing: if i had a million particles. *International Conference on Machine Learning (ICML)*, 2006.

[68] Mark Briers, Arnaud Doucet, and Simon Maskell. Smoothing algorithms for state-space models. *Annals of the Institute of Statistical Mathematics*, 2010.

[69] Herbert E Rauch, F Tung, and Charlotte T Striebel. Maximum likelihood estimates of linear dynamic systems. *American Institute of Aeronautics and Astronautics journal*, 1965.

[70] Randal Douc and Olivier Cappé. Comparison of resampling schemes for particle filtering. In *ISPA 2005. Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis, 2005.*, 2005.

[71] Tiancheng Li, Miodrag Bolic, and Petar M. Djuric. Resampling methods for particle filtering: Classification, implementation, and strategies. *IEEE Signal Processing Magazine*, 2015.

[72] Jun S Liu and Rong Chen. Sequential monte carlo methods for dynamic systems. *Journal of the American statistical association*, 1998.

[73] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 1994.

[74] B.D.O. Anderson and J.B. Moore. *Optimal Filtering*. Prentice-Hall, 1979.

[75] Alexander Ihler, Erik Sudderth, William Freeman, and Alan Willsky. Efficient multi-scale sampling from products of gaussian mixtures. In *Advances in Neural Information Processing Systems*, 2003.

[76] K. V. Mardia and P. J. Zemroch. Algorithm as 86: The von mises distribution function. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 1975.

[77] Geoffrey W. Hill. Algorithm 518: Incomplete bessel function i0. the von mises distribution [s14]. *ACM Trans. Math. Softw.*, 1977.

[78] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[79] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[80] Yuwei Wang, Yuanying Qiu, Peitao Cheng, and Junyu Zhang. Hybrid cnn-transformer features for visual place recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, 2023.

[81] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[82] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.

[83] R. Bellman, R.E. Bellman, and Rand Corporation. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957.

[84] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

[85] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[86] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[87] Kevin P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023.

[88] Angjoo Kanazawa, Michael J. Black, David W. Jacobs, and Jitendra Malik. End-to-end recovery of human shape and pose. *Computer Vision and Pattern Recognition (CVPR)*, 2018.

[89] Angjoo Kanazawa, Jason Y. Zhang, Panna Felsen, and Jitendra Malik. Learning 3d human dynamics from video. *Computer Vision and Pattern Regognition (CVPR)*, 2019.

[90] Shubham Goel, Georgios Pavlakos, Jathushan Rajasegaran, Angjoo Kanazawa*, and Jitendra Malik*. Humans in 4D: Reconstructing and tracking humans with transformers. *International Conference on Computer Vision (ICCV)*, 2023.

[91] Jathushan Rajasegaran, Georgios Pavlakos, Angjoo Kanazawa, and Jitendra Malik. Tracking people by predicting 3D appearance, location & pose. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[92] Zhengyi Luo, S. Alireza Golestaneh, and Kris M. Kitani. 3d human motion estimation via motion compression and refinement. *Proceedings of the Asian Conference on Computer Vision (ACCV)*, 2020.

[93] Hongsuk Choi, Gyeongsik Moon, Ju Yong Chang, and Kyoung Mu Lee. Beyond static features for temporally consistent 3d human pose and shape from a video. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[94] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. SMPL: A skinned multi-person linear model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, 2015.

[95] Georgios Pavlakos, Vasileios Choutas, Nima Ghorbani, Timo Bolkart, Ahmed A. A. Osman, Dimitrios Tzionas, and Michael J. Black. Expressive body capture: 3D hands, face, and body from a single image. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[96] Ahmed A A Osman, Timo Bolkart, Dimitrios Tzionas, and Michael J. Black. SUPR: A sparse unified part-based human body model. *European Conference on Computer Vision (ECCV)*, 2022.

[97] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., 1988.

[98] Erik Sudderth, Alexander Ihler, William Freeman, and Alan Willsky. Nonparametric belief propagation. In *Computer Vision and Pattern Recognition (CVPR)*, 2003.

[99] Erik Sudderth, Alexander Ihler, Michael Isard, William Freeman, and Alan Willsky. Nonparametric belief propagation. *Communications of the ACM*, 2010.

[100] Alexander Ihler, John Fisher III, Randolph Moses, and Alan Willsky. Nonparametric belief propagation for self-calibration in sensor networks. *IEEE Journal of Selected Areas in Communication*, 2005.

[101] Alexander Ihler and David McAllester. Particle belief propagation. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009.

[102] Karthik Desingh, Shiyang Lu, Anthony Opipari, and Odest Chadwicke Jenkins. Efficient nonparametric belief propagation for pose estimation and manipulation of articulated objects. *Science Robotics*, 2019.

[103] Jonathan Kuck, Shuvam Chakraborty, Hao Tang, Rachel Luo, Jiaming Song, Ashish Sabharwal, and Stefano Ermon. Belief propagation neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[104] Victor Garcia Satorras and Max Welling. Neural enhanced belief propagation on factor graphs. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2021.

[105] KiJung Yoon, Renjie Liao, Yuwen Xiong, Lisa Zhang, Ethan Fetaya, Raquel Urtasun, Richard Zemel, and Xaq Pitkow. Inference in probabilistic graphical models by graph neural networks. In *Asilomar Conference on Signals, Systems, and Computers*, 2019.

[106] Zhen Zhang, Fan Wu, and Wee Sun Lee. Factor graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[107] Anthony Opipari, Jana Pavlasek, Chao Chen, Shoutian Wang, Karthik Desingh, and Odest Chadwicke Jenkins. Dnbp: Differentiable nonparametric belief propagation. *ACM/JMS Journal of Data Science*, 2024.

[108] Gal Elidan, Ian McGraw, and Daphne Koller. Residual belief propagation: informed scheduling for asynchronous message passing. In *ncertainty in Artificial Intelligence (UAI)*, 2006.

[109] Charles Sutton and Andrew McCallum. Improved dynamic schedules for belief propagation. In *Uncertainty in Artificial Intelligence (UAI)*, 2007.

[110] Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. Loopy belief propagation for approximate inference: an empirical study. *Uncertainty in Artificial Intelligence (UAI)*, 1999.

[111] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 2001.

[112] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Association for Computing Machinery (ACM)*, 2008.

[113] Joseph Ortiz, Talfan Evans, and Andrew J. Davison. A visual introduction to gaussian belief propagation. *arXiv preprint arXiv:2107.02308*, 2021.

[114] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 1984.

[115] Mohammed Haroon Dupty and Wee Sun Lee. Neuralizing efficient higher-order belief propagation, 2020. URL `https://arxiv.org/abs/2010.09283`.

[116] Jonathan Tompson, Arjun Jain, Yann LeCun, and Christoph Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[117] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 2020.

[118] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

[119] Hongwen Zhang, Yating Tian, Xinchi Zhou, Wanli Ouyang, Yebin Liu, Limin Wang, and Zhenan Sun. Pymaf: 3d human pose and shape regression with pyramidal mesh alignment feedback loop. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2021.

[120] Hongwen Zhang, Yating Tian, Yuxiang Zhang, Mengcheng Li, Liang An, Zhenan Sun, and Yebin Liu. Pymaf-x: Towards well-aligned full-body model regression from monocular images. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2023.

[121] Muhammed Kocabas, Chun-Hao P. Huang, Otmar Hilliges, and Michael J. Black. PARE: Part attention regressor for 3D human body estimation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2021.

[122] Riza Alp Güler and Iasonas Kokkinos. Holopose: Holistic 3d human reconstruction in-the-wild. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[123] Muhammed Kocabas, Nikos Athanasiou, and Michael J. Black. Vibe: Video inference for human body pose and shape estimation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[124] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2014.

[125] Nikos Kolotouros, Georgios Pavlakos, Dinesh Jayaraman, and Kostas Daniilidis. Probabilistic modeling for human mesh recovery. In *IEEE International Conference on Computer Vision (ICCV)*, 2021.

[126] Tongfan Guan, Chen Wang, and Yun-Hui Liu. Neural markov random field for stereo matching. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[127] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. *Advances in Neural Information Processing Systems (NeurIPS)*, 2011.

[128] Steven George Krantz and Harold R Parks. *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media, 2002.

[129] Ali Tourani, Hriday Bavle, Jose Luis Sanchez-Lopez, and Holger Voos. Visual slam: What are the current trends and what to expect? *arXiv preprint arXiv:2210.10491*, 2022.

[130] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 2015.

[131] Zachary Teed and Jia Deng. DROID-SLAM: Deep Visual SLAM for Monocular, Stereo, and RGB-D Cameras. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[132] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 2017.

[133] Ke Sun, Kartik Mohta, Bernd Pfrommer, Michael Watterson, Sikang Liu, Yash Mulgaonkar, Camillo J. Taylor, and Vijay Kumar. Robust stereo visual inertial odometry for fast autonomous flight. *IEEE Robotics and Automation Letters (ICRA)*, 2018.

[134] D. Hahnel, W. Burgard, D. Fox, and S. Thrun. An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2003.

[135] Chi Yan, Delin Qu, Dan Xu, Bin Zhao, Zhigang Wang, Dong Wang, and Xuelong Li. Gs-slam: Dense visual slam with 3d gaussian splatting. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[136] Hidenobu Matsuki, Riku Murai, Paul H. J. Kelly, and Andrew J. Davison. Gaussian Splatting SLAM. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[137] Erik Sandström, Keisuke Tateno, Michael Oechsle, Michael Niemeyer, Luc Van Gool, Martin R Oswald, and Federico Tombari. Splat-slam: Globally optimized rgb-only slam with 3d gaussians. *arXiv preprint arXiv:2405.16544*, 2024.

[138] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 2023.