STEADY: Simultaneous State Estimation and Dynamics Learning from Indirect Observations

Jiayi Wei[1], Jarrett Holtz[1], Isil Dillig[1], and Joydeep Biswas[1]

***Abstract*— Accurate kinodynamic models play a crucial role in many robotics applications such as off-road navigation and high-speed driving. Many state-of-the-art approaches for learning stochastic kinodynamic models, however, require precise measurements of robot states as labeled input/output examples, which can be hard to obtain in outdoor settings due to limited sensor capabilities and the absence of ground truth. In this work, we propose a new technique for learning neural stochastic kinodynamic models from noisy and indirect observations by performing *simultaneous state estimation and dynamics learning*. The proposed technique iteratively improves the kinodynamic model in an expectation-maximization loop, where the E Step samples posterior state trajectories using particle filtering, and the M Step updates the dynamics to be more consistent with the sampled trajectories via stochastic gradient ascent. We evaluate our approach on both simulation and real-world benchmarks and compare it with several baseline techniques. Our approach not only achieves significantly higher accuracy but is also more robust to observation noise, thereby showing promise for boosting the performance of many other robotics applications.**

I. Introduction

Stochastic kinodynamic models are widely used in robotics applications such as state estimation, motion planning, and model-predictive control [1]. Such models are often written by hand and rely on large prediction uncertainty to compensate for model inaccuracy. However, in many applications such as off-road navigation and high-speed robotics, more accurate models can lead to better downstream performance. For example, a learned neural kinodynamic model can better capture nonlinear effects like friction and saturation compared to a simple hand-written model. Prior work has proposed using supervised learning to learn ODEs for system dynamics [2], inertial and visual-inertial kinodynamic models [3], [4], [5], and kinodynamic steering function [6]. However, such supervised learning approaches require accurate measurements of robot states as labeled input/output data, which may not be available in many robotics applications due to limited sensing capabilities, such as when trying to learn the kinodynamic response of an off-road vehicle navigating at high speed in a dense forest.

In this work, we propose a new technique for learning stochastic kinodynamic models in settings where accurate state measurements are unavailable and we only have access to *noisy, indirect* observations. At first glance, this appears to be a chicken-and-egg problem because learning the dynamics requires estimating the state trajectories from observations, whereas state estimation in turn requires access to an accurate kinodynamic model. We solve this apparent predicament by *simultaneously* estimating the robot states while also learning the dynamics. Furthermore, our approach is data efficient and can be applied in settings where it is only feasible to collect training data from a few minutes of observation.

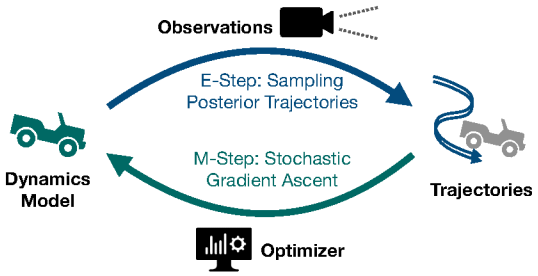

Fig. 1. The high-level workflow of STEADY.

At the heart of our approach lies an expectation-maximization (EM) [7] loop that iteratively improves the neural kinodynamic model as well as our estimation of the robot states. Because EM methods work well for optimization problems with unobserved latent variables, this approach is a good fit for our setting wherein we cannot directly observe the robot states due to limited sensing capabilities. Given a fixed set of observations, our algorithm, *simultaneous STate Estimation And DYnamics learning* (STEADY), alternates between an E Step and an M Step, as illustrated in Figure 1. At a high-level, the E Step estimates a posterior probability density over trajectories by combining both the current kinodynamic model and the imperfect sensor data. Then, the M Step updates the kinodynamic model using stochastic gradient ascent so that it becomes more consistent with the trajectories estimated using the E Step.

Performing the E Step in our setting is challenging because the posterior is defined on a high-dimensional time-series distribution with no analytical solution, making the inference problem computationally intractable. To make matters worse, the E Step needs to be very fast because converging to a good model often requires thousands of EM iterations. To address these challenges, STEADY uses a *particle filtering* approach [8] that approximates the trajectory distribution in linear time and efficiently handles the underlying neural kinodynamic model representation.

Performing the M Step also requires special care because it can be very expensive to fully optimize the neural kinodynamic model in *every* M Step, and the approximations

This work is supported in part by NSF (CCF-2019844) and ARO (W911NF-21-20217).

[1]Computer Science Department, University of Texas at Austin, USA. {jiayi, jaholtz, isil, joydeepb}@cs.utexas.edu

made in the E Step also introduce Monte Carlo errors into the optimization objective. To mitigate these issues, we apply stochastic gradient ascent and only take a single gradient step in each M Step. We also apply a momentum-based optimizer like ADAM [9] to aggregate the gradient across multiple M Steps in order to dampen the effect of the Monte Carlo errors.

We evaluate our proposed algorithm on both simulated and real-world datasets. We compare STEADY with several baseline approaches, including a recently developed learning technique based on stochastic variational inference and a hybrid approach that first performs state estimation and then applies supervised learning. Our main results show that STEADY consistently outperforms all other baselines and achieves performance that is similar to directly learning from the ground truth trajectories.

To summarize, we make the following contributions:

- We introduce *simultaneous state estimation and dynamics learning* as a new way to learn stochastic kinodynamic models for robotics applications and formulate the corresponding mathematical objective.
- We present STEADY, an expectation-maximization-style algorithm for solving the above problem by combining particle filtering and stochastic gradient ascent.
- We provide a working, reusable implementation of STEADY and experimentally demonstrate its superior effectiveness over other baseline approaches.

II. Problem Formulation

Given a system with state x and control inputs u, we are interested in identifying the discrete-time forward kinodynamic function f along with the zero-mean error distribution Q_ϵ such that the state dynamics is given by

$$x_{t+1} = f(x_t, u_t) + \epsilon_t, \quad \epsilon_t \sim Q_\epsilon(\epsilon | x_t) \ . \tag{1}$$

Modeling $\langle f, Q_\epsilon \rangle$ is a challenging problem: while physical analysis can be used for simple kinematic systems such as skid-steer robots [10], the physics of vehicle to ground surface interactions (*e.g.,* for high-speed off-road driving) is poorly understood and must be learned directly from empirically observed data. Given a dataset $\langle x_{1:T}, u_{1:T-1} \rangle$ of a sequence of controls $u_{1:T-1}$ and the corresponding trajectory $x_{1:T}$, $\langle f, Q_\epsilon \rangle$ can be learned via supervised learning [3]. The trajectories in such supervised learning settings are either gathered using motion capture or using simultaneous localization and mapping (SLAM). However, collecting accurate trajectories $x_{1:T}$ in challenging settings is infeasible when motion capture is unavailable or when observations are sparse or noisy such that SLAM-estimated trajectories are of insufficient accuracy for learning $\langle f, Q_\epsilon \rangle$.

Motivated by this problem, this work focuses on the more realistic setting where we cannot directly observe the states and have to instead rely on a *noisy and indirect* observation sequence $y_{1:T} = (y_1, \ldots, y_T)$, which we assume are generated via some *known* observation model

$$y_t \sim Q_y(y | \ x_t) \ , \tag{2}$$

where each observation y_t carries only partial information about x_t and can even be missing for certain time steps. As a result, each observation y_t alone is generally not sufficient for uniquely determining the state x_t, and there will, in fact, be an infinite continuum of trajectories that are probabilistically compatible with the observation sequence $y_{1:T}$.

A common method for learning the dynamics under such noisy observations is to formulate it as a *maximum a posteriori (MAP) estimation* problem, where the goal is to simultaneously find a *single* trajectory $x_{1:T}$ and a dynamical model $\langle f, Q_\epsilon \rangle$ such that they maximize the posterior density

$$\begin{aligned} &P(x_{1:T} | y_{1:T}, u_{1:T-1}) \propto P(x_{1:T}, y_{1:T} | u_{1:T-1}) \\ &= \underbrace{Q_{x_1}(x_1) \prod_{t=1}^{T-1} Q_\epsilon(x_{t+1} - f(x_t, u_t) \ | x_t)}_{P(x_{1:T} | u_{1:T-1})} \underbrace{\prod_{t=1}^{T} Q_y(y_t | x_t)}_{P(y_{1:T} | x_{1:T})} \end{aligned} \tag{3}$$

where Q_{x_1} is the distribution of the initial state. However, this is an ill-formed objective for our setting since we are also learning the error distribution Q_ϵ.[1] To avoid this problem, we instead consider *all* possible trajectories that are consistent with the observations (rather than a single trajectory) and find a dynamical model that maximizes the following *marginalized* observation likelihood [11].

$$P(y_{1:T} | u_{1:T-1}) = \int P(x_{1:T}, y_{1:T} | u_{1:T-1}) \, \mathrm{d}x_{1:T} \ . \tag{4}$$

This is a difficult optimization objective as it requires integrating the joint density over the trajectory space and has no analytical form for nonlinear dynamical systems. Prior works based on stochastic variational inference address this challenge by optimizing a tractable lower bound of (4) [12], [13]. Such a lower bound is obtained by simultaneously training an *inference network* in additional to the dynamical model. The inference network is trained to fit the posterior trajectory distribution $P(x_{1:T} | u_{1:T-1}, y_{1:T})$ and can thus predict the state trajectories from just observations and controls. However, as we will show in our evaluation, when the observation model Q_y is complex, predicting the posterior distribution becomes a highly challenging task, and the inference network often fails to learn an adequate mapping, causing an inaccurate kinodynamic model to be learned. Thus, we explore an alternative approach that does not require training an inference network.

III. The STEADY Algorithm

We now present our approach, STEADY, for iteratively maximizing the optimization objective (4).

A. Algorithm Overview

STEADY is an instance of the *generalized Monte Carlo expectation-maximization* algorithm [14], where the E Step

[1] A simple way to see this is to consider the trivial solution $x_t \equiv 0$ and the corresponding (completely static) dynamics f. By reducing the variance of Q_ϵ toward zero, this solution allows us to drive the Q_ϵ term in (3) to an arbitrarily large value.

Algorithm 1 The STEADY Algorithm

Input: control sequence $u_{1:T-1}$, observation sequence $y_{1:T}$, observation model Q_y, and initial state distribution Q_{x_1}.
Output: θ, the weights of the kinodynamic model $\langle f, Q_\epsilon \rangle$.

1: Randomly initialize θ such that $f \approx \text{identity}$ and $Q_\epsilon(\cdot)$ is sufficiently large.
2: **while** $\langle f, Q_\epsilon \rangle$ has not overfitted **do**
3: Run particle filtering forward in time using Q_y and Q_{x_1} to generate N particles at each time step.
4: Trace back M trajectories $\{x^i_{1:T}\}_{i=1}^M$ s.t. (5) holds.
5: Estimate $\tilde{\Phi}(\theta)$ using the trajectories according to (6).
6: Take a gradient step w.r.t. θ to maximize $\tilde{\Phi}(\theta)$.
7: **end while**

relies on a Monte-Carlo objective and the M Step makes incremental improvements to this objective. We summarize STEADY in Algorithm 1 and elaborate on its key details in later subsections.

At a high level, STEADY takes as input the control and observation sequence and outputs the learned kinodynamic model, $\langle f, Q_\epsilon \rangle$, which is represented as a neural network with parameters θ. The parameters θ are initialized such that f is approximately the identity function and Q_ϵ has large uncertainty (line 1 in Algorithm 1). This initialization strategy ensures that the initial dynamics is stable and that it can be corrected from observation data. Then, the algorithm enters a loop (lines 2–6) in which the network parameters θ are iteratively updated to be more consistent with the observation. Specifically, lines 3-4 constitute the E Step of the algorithm and estimate a posterior probability density over trajectories through particle filtering (explained in more detail later). Then, the M Step in lines 5-6 uses the trajectories estimated in the E Step to update the neural network parameters θ by taking a gradient step.

In more detail, the E Step of STEADY samples M trajectories $x^i_{1:T}$ from the posterior:

$$x^i_{1:T} \sim P(x_{1:T} |\ u_{1:T-1}, y_{1:T}), \quad i = 1 \dots M\ . \tag{5}$$

Note that samples $x^i_{1:T}$ are complete state trajectories. While we could, in principle, use factor graphs to approximate (5) (by computing the full covariance around the MAP estimate of $x_{1:T}$), doing so would be both prohibitively expensive [15] and also inaccurate when the posterior is multi-modal. Instead, we choose to estimate the posterior using particle filtering[2], as described in more detail in Section III-C.2.

Given the trajectories obtained from the E Step, the M Step of STEADY (lines 5-6 in Algorithm 1) applies stochastic gradient ascent to adjust the model parameters θ in order to improve the following objective:

$$\tilde{\Phi}(\theta) = \frac{1}{M} \sum_{i=1}^{M} \log P_\theta(x^i_{1:T}, y_{1:T} | u_{1:T-1})\ , \tag{6}$$

where we write P_θ to emphasize that the model can affect $P_\theta(x^i_{1:T}, y_{1:T}|u_{1:T-1})$ via Eq. (3). To reduce the impact of sampling errors in the E Step, we do not run this optimization to convergence but instead only take a single gradient step. This design choice also reduces the computational overhead of training the neural network in each M Step.

B. Improvement Guarantees

We now briefly explain why our proposed algorithm maximizes the objective (4), borrowing standard assumptions/results from the Monte Carlo expectation-maximization literature (*e.g.,* see [14]).

Denote the value of θ in the E Step as θ_E. Since $x^i_{1:T}$ are sampled from $P_{\theta_E}(x^i_{1:T}|y_{1:T})$, we can view the M Step objective $\tilde{\Phi}(\theta)$ in (6) as the Monte Carlo estimate of another objective $\Phi(\theta, \theta_E)$, defined as

$$\begin{aligned}\Phi(\theta, \theta_E) &= \int \log P_\theta(x_{1:T}, y_{1:T}) P_{\theta_E}(x_{1:T}|y_{1:T}) \mathrm{d}x_{1:T} \\ &= \mathop{\mathbb{E}}_{x^i_{1:T}} \left[\log P_\theta(x^i_{1:T}, y_{1:T}|u_{1:T-1})\right] \approx \tilde{\Phi}(\theta)\end{aligned} \tag{7}$$

Taking the difference between $\Phi(\theta, \theta_E)$ and $\Phi(\theta_E, \theta_E)$, we then have

$$\begin{aligned}&\Phi(\theta, \theta_E) - \Phi(\theta_E, \theta_E) \\ &= \int \log \frac{P_\theta(x_{1:T}|y_{1:T}) P_\theta(y_{1:T})}{P_{\theta_E}(x_{1:T}|y_{1:T}) P_{\theta_E}(y_{1:T})} P_{\theta_E}(x_{1:T}|y_{1:T}) \mathrm{d}x \\ &= \underbrace{\int \log \frac{P_\theta(x_{1:T}|y_{1:T})}{P_{\theta_E}(x_{1:T}|y_{1:T})} P_{\theta_E}(x_{1:T}|y_{1:T}) \mathrm{d}x}_{\text{(negative KL divergence)}} \\ &\qquad + \log P_\theta(y_{1:T}) - \log P_{\theta_E}(y_{1:T}) \\ &\qquad \le \log P_\theta(y_{1:T}) - \log P_{\theta_E}(y_{1:T})\ .\end{aligned} \tag{8}$$

This shows that any improvement on $\Phi(\theta, \theta_E)$ will be a lower bound of the improvement on $\log P_\theta(y_{1:T})$. Thus, when our Monte Carlo estimate $\tilde{\Phi}(\theta)$ closely approximates $\Phi(\theta, \theta_E)$[3], optimizing $\tilde{\Phi}(\theta)$ also optimizes $P_\theta(y_{1:T})$, which is the objective stated in (4).

C. Implementation

In this section, we describe the key design choices behind our implementation of the STEADY algorithm.

1) Kinodynamic Model Architecture: In our implementation, we represent the kinodynamic model $\langle f, Q_\epsilon \rangle$ using the neural architecture shown in Figure 2. This neural network takes as input the current state and control in the local reference frame and predicts the state derivative $\dot{x}$ (in the same reference frame) using a diagonal normal distribution

[2]Compared to more recently developed nonlinear smoothing techniques such as *Forward Filtering/Back Simulation* [16] or *Particle Gibbs with Ancestor Sampling* [17], we use particle filtering since it has a better asymptotic time complexity and can be implemented efficiently when using the neural kinodynamic model.

[3]To ensure that the Monte Carlo estimate is a good approximation of $\Phi(\theta, \theta_E)$, recall that STEADY takes a single optimization step using a momentum-based optimizer, which ensures that most Monte Carlo errors can cancel out across M Steps.

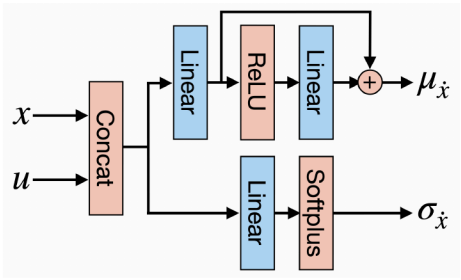

Fig. 2. Architecture of the dynamics network. x and u are the current state and control, and $\mu_{\dot{x}}$ and $\sigma_{\dot{x}}$ are the mean and variance of the predicted state derivatives. All inputs/outputs are defined in local reference frames.

with mean $\mu_{\dot{x}}$ and variance $\sigma_{\dot{x}}$. We compute the mean $\mu_{\dot{x}}$ using a fully connected network with 64 ReLU activation units and one hidden layer. We also add a skip connection to make it easier for the network to learn the linear part of the dynamics [18]. We compute the diagonal elements of the covariance matrix using a simple linear layer followed by the softplus activation $\mathrm{softplus}(x) = \log(1 + \exp(x))$ to ensure that the outputs are always positive.

2) Sampling Posterior Trajectories with Particle Filtering: To sample posterior trajectories using a particle filter, we first run the particle filter forward in time to obtain a fixed number of particles for each time step. This process involves alternating between the prediction and resampling step. In each prediction step, new particles for the next time step are sampled from $P(x_{t+1}|x_t, u_t)$ by forward-simulating the dynamical model. In each resampling step, we use the observation model $P(y_{t+1}|x_{t+1})$ to replicate those particles that are more consistent with the new observation y_{t+1} and discard those that are less so. Once we have obtained the particles at the last time step, we randomly pick one of them and trace its ancestral lineage back to the first time step to obtain a complete state trajectory.[4] When the number of particles is sufficiently large, this approach gives unbiased samples from the posterior distribution [8]. In each E Step, we run the particle filter once to sample N particles at each time step and reuse the particles to sample M trajectories. To support efficient computation with the neural network, we store all particles using the structure-of-arrays pattern and perform all operations in batch.

3) Accelerating Learning by Flattening the E Step posterior: Since the kinodynamic model is very inaccurate in the early stages of training, we found that directly using the E Step posterior $P(x_{1:T}|u_{1:T-1}, y_{1:T})$ can cause all sampled trajectories to concentrate around a single incorrect trajectory, slowing down the learning process or even trapping the optimizer in local optima. To mitigate this problem, we take inspiration from simulated annealing and modify the E Step to instead sample with a modified observation model

$$y_t \sim Q_y(y|\ x_t)^{w_{\mathrm{obs}}}\ ,$$

where the parameter $w_{\mathrm{obs}} \in [0, 1]$ controls the importance of the observations. During the course of the training, we linearly increase w_{obs} from 0 to 1. As shown in our evaluation in Section IV-C, this modification consistently improves the training outcome.

4) Code availability: Our implementation is written in Julia [19] and is built on top of the Flux machine learning library [20]. We publish our code on Github.[5]

IV. Evaluation

In this section, we first describe our experimental setup and then present the results to answer: (1) Can STEADY achieve better performance compared to relevant baselines? (2) How do various factors and design choices affect the performance of STEADY?

A. Experimental Setup

Before presenting our empirical results, we first discuss the benchmarks and baselines used in the evaluation as well as other relevant experimental details.

1) Benchmarks: We perform our evaluation on the following datasets, which consist of both simulated and real-world data:

- **Hov** is a dataset that we create by simulating a hovercraft moving on a 2D plane with nonlinear air resistance. We control the hovercraft by adjusting the thrust of its propellers and simulate 16, 32, and 32 trajectories for training, validation, and testing, respectively. Each trajectory is 10 seconds long.
- **Hov+** is a data-rich variant of Hov that contains 10 times more training trajectories.
- **Truck** is a real-world dataset obtained by tele-operating a scale 1/5, four-wheel drive, Ackermann steering vehicle that we call the AlphaTruck. We divide the collected data into 10-second trajectories and use about 2 minutes of data each for training, validation, and testing.

For all three datasets, we represent the robot state using 6 variables (i.e, 3 variables for the 2D pose and the other 3 for the velocities). We use simulated observations to control the effects of observation noise. We generate these observations from the ground-truth states using a landmark-based, bearing-only observation model. For each benchmark, we randomly place 4 landmarks and measured their relative angles w.r.t. the robot's pose at every time step. We also add a small amount of Gaussian noise ($\sigma = 5°$) to the measured angles to simulate sensor noise.

2) Baseline approaches: We compare STEADY with the following baselines:

- **Hand:** This baseline involves hand-written kinodynamic models customized for each of the three datasets. Specifically, for the Hov datasets, the hand-written model is the ground-truth dynamics with two modifications: (1) it

[4]Although particle filtering is prone to the path degeneration issue (where most trajectories share the same ancestor due to excessive resampling), this issue tends to get better as the dynamical model becomes more consistent with the observations as training progresses. Empirically, we only observe severe path degeneration in early stages of training, at which time even a very noisy gradient of $\tilde{\Phi}(\theta)$ is sufficient to push the parameters toward the right direction.

[5]Repository: `https://github.com/MrVPlusOne/STEADY`

does account for air friction; (2) it compensates for model inaccuracy by using a disturbance variance that is twice the ground truth. For the Truck dataset, the hand-written model is the classical kinematic model that assumes bounded forward acceleration and no sliding along the lateral direction.

- **FitHand:** This baseline first performs state estimation using particle filtering with the hand-written model. It then applies supervised learning to fit an updated model to the estimated trajectories. This method can be viewed as a simplified version of STEADY that performs both state estimation and learning but without embedding them in the EM loop.
- **FitTV:** This baseline performs supervised learning without relying on *any* motion model. In particular, it first estimates the most likely pose at each time step using only the observation model and then applies total variation regularization [21], [22] to estimate the (de-noised) velocities. This baseline is relevant for exploring the impact of an explicit motion model on performing state estimation.
- **SVI:** This baseline learns a kinodynamic model using stochastic variational inference, as described in [12]. To adapt this approach to our setting, we replace the GRU-like dynamics network proposed in [12] with the neural architecture described in Section III-C.1.
- **FitTruth:** As an upper bound on the learning performance, we also report the results of applying supervised learning directly on ground truth states.

3) Hyperparameters: In the following experiments, we use at most 40,000 EM steps, leveraging the validation set to early-stop the training (according to the estimated marginal observation density $P(y_{1:T})$). In our E Step implementation, we use $N = 20,000$ particles and sample $M = 10$ trajectories. When the training data contains multiple trajectories, we only use the subset from a single trajectory in each E Step. In our M Step implementation, we use the ADAM optimizer with a fixed learning rate of 10^{-4} and all other parameters set to their default values. We also normalize the loss by dividing it by the length of the trajectory.

B. Main Results

In this section, we report the results of our empirical comparison between STEADY and the five baselines described earlier. Specifically, we evaluate the (learned) model's performance in terms of state estimation (Table I) and forward prediction error (Table II). Both metrics measure the root-mean-square error (RMSE) between the poses obtained by the model and the ground truth.[6] We run each baseline 3 times and report the best performance on the test set.

As shown in Table I and II, STEADY consistently outperforms all other baselines (excluding FitTruth) on all datasets under both metrics. Furthermore, STEADY in many cases achieves a performance level that is very similar to FitTruth. Comparing the performance of Hand and FitHand, we see that learning from the data generated by a hand-written model can lead to performance improvement; however, the performance of FitHand is still considerably worse than that of STEADY. Comparing FitTV and FitHand, we observe that FitHand performs better than FitTV, which suggests that learning with an inaccurate motion model is still better than without one. Finally, the SVI approach is more data-hungry and only gets close to STEADY's performance on the data-rich Hov+ dataset.

TABLE I
STATE ESTIMATION PERFORMANCE COMPARISONS

	Location Error (m)			**Angular Error (rad)**		
	Hov	Hov+	Truck	Hov	Hov+	Truck
Hand	1.168	1.145	0.554	0.574	0.532	0.103
FitHand	0.505	0.491	0.446	0.040	0.037	0.087
FitTV	1.381	1.355	0.504	0.144	0.092	0.108
SVI	0.637	0.366	0.432	0.050	0.038	0.090
STEADY	**0.375**	**0.346**	**0.258**	**0.033**	**0.031**	**0.063**
FitTruth	*0.368*	*0.314*	*0.238*	*0.035*	*0.030*	*0.063*

TABLE II
FORWARD PREDICTION PERFORMANCE COMPARISONS

	Location Error (m)			**Angular Error (rad)**		
	Hov	Hov+	Truck	Hov	Hov+	Truck
Hand	0.058	0.059	0.472	0.025	0.024	0.151
FitHand	0.050	0.051	0.373	0.033	0.030	0.107
FitTV	0.131	0.147	0.461	0.089	0.087	0.213
SVI	0.088	0.046	0.229	0.036	0.025	0.080
STEADY	**0.042**	**0.038**	**0.200**	**0.013**	**0.013**	**0.079**
FitTruth	*0.031*	*0.027*	*0.174*	*0.017*	*0.012*	*0.085*

To qualitatively validate our results, we also visualize the posterior velocities in Figure 3 for FitHand and STEADY. As shown in this figure, STEADY is able to closely track the overall trends in the lateral motion (show in orange), whereas FitHand's lateral prediction is much more uncertain and resembles white noise. This is consistent with our intuition that using a flawed hand-written model (which assumes no lateral sliding) can introduce biases to the training data and subsequently lead to worse performance.

C. Impact of Observation Noise

In this section, we perform an evaluation to assess the impact of observation noise on both STEADY and the baselines. Additionally, we also consider an ablation of STEADY called STEADY- that does not utilize the E Step flattening modification proposed in Section III-C.3. To perform this evaluation, we vary the magnitude of the angular noise in the Truck dataset between 1.25 and 20 degrees. We run each approach 3 times (except for SVI, which we only run once due to its much longer training time) and report the forward prediction error in Figure 4. Overall, these results follow the

[6]We compute the state estimation error using the posterior trajectories obtained from particle filtering and reduce the observation frequency from 10Hz (at training time) to 1Hz (at test time) to make the effect of the kinodynamic model more salient. We compute the forward prediction error by taking each state from the ground truth trajectories as the initial state, running the learned model forward for 10 time steps (using the recorded controls only), and measuring the RMSE between the predicted final pose and the ground truth.

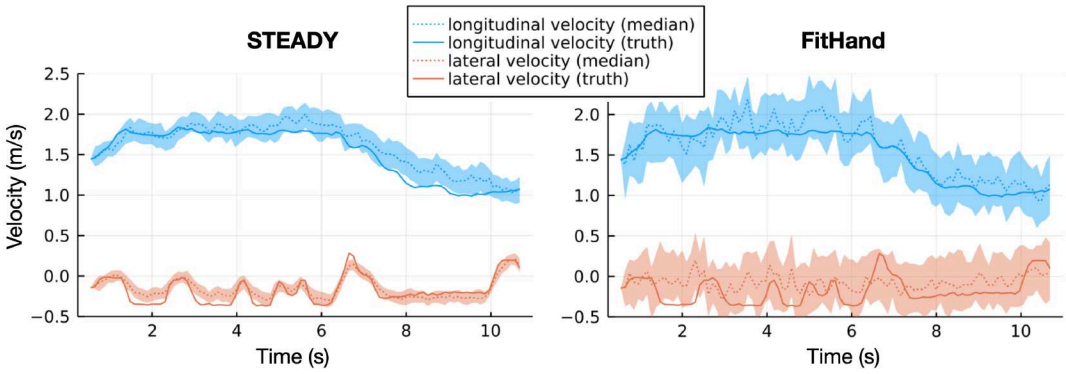

Fig. 3. Posterior velocity visualization. We show the posterior quality of STEADY (left) and FitHand (right) on one test trajectory from the Truck dataset. Both the longitudinal and lateral velocities are measured in the robot's local reference.

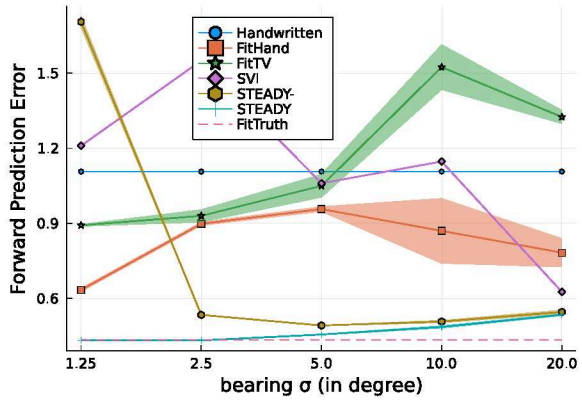

Fig. 4. How observation noise affects forward prediction error. All learning-based approaches are trained on the Truck dataset using the bearing-only observation model.

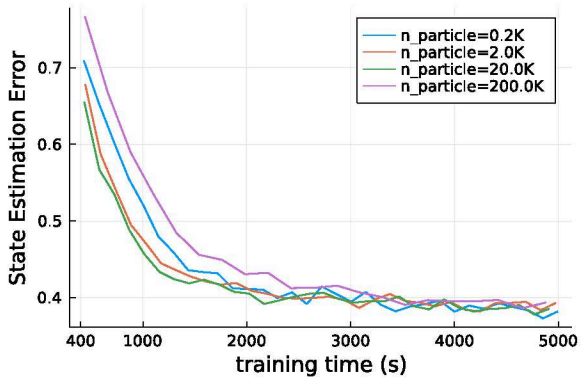

Fig. 5. How the number of particles in the E Step affects the learning speed and accuracy of STEADY. Using 20K particles leads to fastest training in terms of running time, but all four settings eventually converge to similar levels of performance.

expected relationship between noise and model performance: in general, all approaches (except Hand and FitTruth, which do not use the observations for training) tend to get worse as the magnitude of noise increases. However, the performance impact on STEADY is relatively small, demonstrating its robustness to observation noise, and it is also the only approach whose performance converges to that of FitTruth on low-noise settings. We can also see that STEADY- performs significantly worse than STEADY when the noise is low, suggesting that the E Step flattening modification is crucial for stable training under a sharp posterior.

D. Impact of Particle Quantity

In this section, we report the results of an experiment that is designed to evaluate the impact of the particles used in the E Step, which is one of the most important hyper-parameters underlying the STEADY implementation. To this end, we vary the number of particles used for the Truck dataset from 200 to $200,000$ and plot the validation set performance against training time in Figure 5.

The results of this experiment demonstrate the known trade-off between state estimation accuracy and computational cost. With more particles, the estimated gradient in the M Step has higher quality and hence leads to faster learning in terms of training steps. However, more particles also incur a longer running time. Overall, we find that the sweet spot in our setting lies somewhere between 2000 and $20,000$ particles. Another interesting finding from this experiment is that, no matter how many particles are used, all configurations eventually converge to a similar performance before starting to overfit, suggesting that overfitting might be the dominating factor here (compared to Monte Carlo error).

V. Related Work

In this section, we survey robotics literature on learning dynamical models and then review techniques for learning stochastic dynamical systems in a more general context.

Dynamics Learning in Robotics. Most recent publications on robot dynamics learning target different settings than the one we consider here. For example, [3] aims to learn inverse kinematic models in a deterministic setting and assumes the training data consist of directly observed, accurate robot states, whereas we focus on learning forward dynamical models from noisy and indirect observations. [23] focuses on learning the interaction between multiple objects and also assumes the underlying dynamics are deterministic. That work also differs from ours in how the state space is represented: while their learned dynamics operate in

opaque, high-dimensional latent spaces, ours are defined in terms of low-dimensional, physically grounded states. An alternative method to handle partial state observability is by applying the Koopman Operator Theory, as done in [24] to learn deterministic dynamical models of soft-body robots. However, this approach is only applicable when there is little noise in the observations. Finally, [25] learns stochastic dynamical models for hierarchical planning of manipulation tasks, and, similar to [23], it also performs learning in a high-dimensional latent state space.

Learning Stochastic Dynamical Systems. Outside the field of robotics, there is a rich literature on data-driven identification of stochastic dynamical systems. Prior approaches can be divided into two main categories: those based on expectation-maximization (EM) v.s. those based on stochastic variational inference (SVI). The EM category includes the pioneer work of [26], which parameterizes the dynamics as radial basis function approximators and uses Extended Kalman Smoothing for the E Step. More recent work includes [27], which implements the E Step using particle smoothing, and [11], which focuses on learning partial differential equations to model the geophysical flow dynamics. As for the SVI category, one representative approach is the one proposed in [12], upon which our SVI baseline is based. More recent work [13] also applies SVI to learning chaotic dynamical systems in more data-rich settings.

VI. Conclusion and Future Work

We introduced the simultaneous state estimation and dynamics learning problem for learning kinodynamic models from noisy and indirect data. Our key idea is to iteratively improve both the state estimation and the learned kinodynamic model using the expectation-maximization framework. Our approach is simple to implement, works for real-world robots, and outperforms existing approaches.

In the future, we plan to extend our approach to more challenging settings such as simultaneous SLAM and dynamics learning. We also plan to explore ways to learn interpretable dynamical models by combining ideas from program synthesis and symbolic regression.

References

[1] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[2] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in neural information processing systems*, vol. 31, 2018.

[3] X. Xiao, J. Biswas, and P. Stone, "Learning inverse kinodynamics for accurate high-speed off-road navigation on unstructured terrain," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 6054–6060, 2021.

[4] H. Karnan, K. S. Sikand, P. Atreya, S. Rabiee, X. Xiao, G. Warnell, P. Stone, and J. Biswas, "VI-IKD: High-Speed Accurate Off-Road Navigation using Learned Visual-Inertial Inverse Kinodynamics," in *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, 2022.

[5] P. Atreya, H. Karnan, K. S. Sikand, X. Xiao, S. Rabiee, and J. Biswas, "High-Speed Accurate Robot Control using Learned Forward Kinodynamics and Non-linear Least Squares Optimization," in *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, 2022.

[6] P. Atreya and J. Biswas, "State supervised steering function for sampling-based kinodynamic planning," in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2022.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[8] A. Doucet, A. M. Johansen, *et al.*, "A tutorial on particle filtering and smoothing: Fifteen years later," *Handbook of nonlinear filtering*, vol. 12, no. 656-704, p. 3, 2009.

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[10] S. Rabiee and J. Biswas, "A friction-based kinematic model for skid-steer wheeled mobile robots," in *Robotics and Automation (ICRA), IEEE International Conference on*, pp. 8563–8569, IEEE, 2019.

[11] M. Bocquet, J. Brajard, A. Carrassi, and L. Bertino, "Bayesian inference of chaotic dynamics by merging data assimilation, machine learning and expectation-maximization," *Foundations of Data Science*, vol. 2, no. 1, pp. 55–80, 2020.

[12] R. Krishnan, U. Shalit, and D. Sontag, "Structured inference networks for nonlinear state space models," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, 2017.

[13] D. Nguyen, S. Ouala, L. Drumetz, and R. Fablet, "Variational deep learning for the identification and reconstruction of chaotic and stochastic dynamical systems from noisy and partial observations," *arXiv preprint arXiv:2009.02296*, 2020.

[14] R. M. Neal and G. E. Hinton, "A view of the em algorithm that justifies incremental, sparse, and other variants," in *Learning in graphical models*, pp. 355–368, Springer, 1998.

[15] M. Kaess and F. Dellaert, "Covariance recovery from a square root information matrix for data association," *Robotics and autonomous systems*, vol. 57, no. 12, pp. 1198–1210, 2009.

[16] F. Lindsten and T. B. Schön, "Backward simulation methods for monte carlo statistical inference," *Foundations and Trends® in Machine Learning*, vol. 6, no. 1, pp. 1–143, 2013.

[17] F. Lindsten, M. I. Jordan, and T. B. Schon, "Particle gibbs with ancestor sampling," *Journal of Machine Learning Research*, vol. 15, pp. 2145–2184, 2014.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[19] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017.

[20] M. Innes, E. Saba, K. Fischer, D. Gandhi, M. C. Rudilosso, N. M. Joy, T. Karmali, A. Pal, and V. Shah, "Fashionable modelling with flux," *CoRR*, vol. abs/1811.01457, 2018.

[21] R. Chartrand, "Numerical differentiation of noisy, nonsmooth data," *International Scholarly Research Notices*, vol. 2011, 2011.

[22] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the national academy of sciences*, vol. 113, no. 15, pp. 3932–3937, 2016.

[23] Y. Li, J. Wu, J.-Y. Zhu, J. B. Tenenbaum, A. Torralba, and R. Tedrake, "Propagation networks for model-based control under partial observation," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 1205–1211, IEEE, 2019.

[24] D. Bruder, C. D. Remy, and R. Vasudevan, "Nonlinear system identification of soft robot dynamics using koopman operator theory," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 6244–6250, IEEE, 2019.

[25] K. Fang, Y. Zhu, A. Garg, S. Savarese, and L. Fei-Fei, "Dynamics learning with cascaded variational inference for multi-step manipulation," *arXiv preprint arXiv:1910.13395*, 2019.

[26] Z. Ghahramani and S. Roweis, "Learning nonlinear dynamical systems using an em algorithm," *Advances in neural information processing systems*, vol. 11, 1998.

[27] T. B. Schön, A. Wills, and B. Ninness, "System identification of nonlinear state-space models," *Automatica*, vol. 47, no. 1, pp. 39–49, 2011.