2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Trust-Region Neural Moving Horizon Estimation for Robots

Bingheng Wang, Xuyang Chen, and Lin Zhao

***Abstract*— Accurate disturbance estimation is essential for safe robot operations. The recently proposed neural moving horizon estimation (NeuroMHE), which uses a portable neural network to model the MHE's weightings, has shown promise in further pushing the accuracy and efficiency boundary. Currently, NeuroMHE is trained through gradient descent, with its gradient computed recursively using a Kalman filter. This paper proposes a trust-region policy optimization method for training NeuroMHE. We achieve this by providing the second-order derivatives of MHE, referred to as the MHE Hessian. Remarkably, we show that many of the intermediate results used to obtain the gradient, especially the Kalman filter, can be efficiently reused to compute the MHE Hessian. This offers linear computational complexity with respect to the MHE horizon. As a case study, we evaluate the proposed trust region NeuroMHE on real quadrotor flight data for disturbance estimation. Our approach demonstrates highly efficient training in under 5 min using only 100 data points. It outperforms a state-of-the-art neural estimator by up to 68.1% in force estimation accuracy, utilizing only 1.4% of its network parameters. Furthermore, our method showcases enhanced robustness to network initialization compared to the gradient descent counterpart.**

I. Introduction

Disturbances are widespread in robot operations, arising from various sources like system uncertainties [1], aerodynamic influences [2], ground friction forces [3], and hydrodynamic disturbances [4]. Addressing these disturbances is vital in robotic control to prevent significant performance degradation [5], [6]. However, developing a versatile model to capture these disturbances across environments is typically impractical due to their complexity. Therefore, online precise disturbance estimation that adapts to environments is crucial for safe and effective robot operations.

Existing works in disturbance estimation mainly utilize either model-based [7]–[11] or model-free [12]–[14] methods. While model-based estimators are data-efficient, they generally have limited dynamic response range, since the models used are typically limited to specific slowly-varying disturbances [7]–[9]. Another challenge lies in their heavy reliance on manually tuning numerous parameters [10], [11], which demands significant experimental efforts and expert knowledge. In contrast, model-free approaches using neural networks (NN) can achieve high performance across various disturbance scenarios with minimal expert knowledge. Examples include estimating system uncertainties for robotic joints [12] and aerodynamic effects for aerial robots [13], [14]. In practice, these NN-based estimators typically use large network models for training, along with substantial disturbance data and complicated learning curricula.

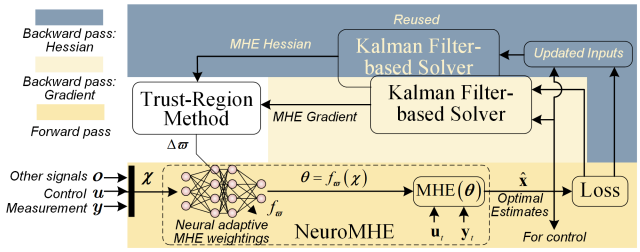


Fig. 1: Learning pipelines of the trust-region NeuroMHE. Currently, NeuroMHE is trained via gradient descent with its gradient computed recursively using a Kalman filter. This paper enhances NeuroMHE training with the second-order trust-region method. Interestingly, we show that the MHE Hessian can be obtained recursively using the same Kalman filter with just minor modifications to its inputs.

To leverage advantages from both model-free and model-based methods, we recently proposed the neural moving horizon estimation (NeuroMHE) [15]. This algorithm fuses a portable neural network into an MHE to realize accurate estimation and fast online adaptation (See the dashed block in Fig. 1). MHE is an optimal state estimator using control theory for dynamic nonlinear optimization online, with a receding horizon strategy. It has consistently demonstrated superior performance in disturbance estimation for both ground [16] and aerial [17], [18] robots. Nevertheless, the parameter tuning remains a challenge for vanilla MHE, especially when dealing with dynamic and nonlinear parameters [19]. In contrast, NeuroMHE tackles this challenge by modeling its parameters using the neural network trained through gradient descent. The gradient includes the derivative of MHE's optimal solution trajectory with respect to (w.r.t) the parameters, computed recursively using a Kalman filter. This approach results in linear computational complexity relative to the MHE horizon, significantly improving training efficiency over state-of-the-art auto-tuning MHE methods [20], [21].

In this paper, we enhance NeuroMHE training by providing second-order derivative information of MHE (i.e., the MHE Hessian). This allows us to employ the trust-region method (TRM), a second-order optimization technique, for training NeuroMHE. The trust-region method offers adaptive step-size updates, faster convergence, and improved robustness to initialization. It has shown efficacy across machine

This work was supported by the Singapore Ministry of Education Tier 1 Academic Research Fund (22-5460-A0001) and Tier 2 AcRF (T2EP20123-0037). (Corresponding author: Lin Zhao.)

These authors are with the Department of Electrical and Computer Engineering, National University of Singapore, 4 Engineering Drive 3, 117583 Singapore, Singapore {wangbingheng, chenxuyang}@u.nus.edu, elezhli@nus.edu.sg

The source code of this work is available at https://github.com/BinghengNUS/TR-NeuroMHE

learning and robotics, including policy optimization [22] and, more recently, synergizing optimal control with inverse reinforcement learning [23]. Fig. 1 outlines the learning pipelines of the trust-region NeuroMHE. Two key ingredients of our algorithm are the gradient and the Hessian trajectories of the MHE's optimal solutions w.r.t the tuning parameters in the backward passes. By differentiating the Karush-Kuhn-Tucker (KKT) conditions of the MHE optimization problem in the forward pass, we can efficiently obtain the MHE gradient trajectory using a Kalman filter, as developed in [15]. Interestingly, we further show that the same Kalman filter can be reused to compute the MHE Hessian trajectory with only minor modifications to its inputs, which are acquired through double differentiation of the KKT conditions. This preserves the linear computational complexity w.r.t the MHE horizon, further ensuring the scalability of our method to high-dimension estimation problem.

We evaluate the proposed trust-region NeuroMHE in estimating complex aerodynamic disturbances using open-source real quadrotor flight data as a case study. Compared to the state-of-the-art estimator NeuroBEM [13], our approach achieves: 1) highly efficient training less than $5\ \mathrm{min}$ with only 100 data points; 2) using only 1.4% of the NeuroBEM's network parameters; and 3) reducing the overall force estimation error by up to 68.1%. Further comparisons with the gradient descent counterpart show that, although computing the MHE Hessian trajectory naturally takes longer than the gradient trajectory, our method significantly reduces both the total training episodes and the overall training time. Additionally, our approach exhibits strong robustness to the initialization of the neural network. In summary, our contributions are threefold:

1) We propose a trust-region policy optimization method for training the recently developed NeuroMHE.
2) We show that much of the computation used to obtain the MHE gradient trajectory, particularly the Kalman filter, can be efficiently reused for computing the MHE Hessian trajectory in a recursive manner.
3) We validate the effectiveness of our approach using a real flight dataset, showing highly efficient training and superior performance compared to the state-of-the-art method in aerodynamic force estimation accuracy.

The rest of this paper is organized as follows. Section II briefly reviews the formulation of NeuroMHE. In Section III, we detail the process of reusing the Kalman filter to compute the MHE Hessian. Section IV develops the proposed trust-region NeuroMHE training method. Simulation results on the real dataset are reported in Section V. We conclude this paper and discuss our future work in Section VI.

II. Preliminary of NeuroMHE

A. Moving Horizon Estimation

MHE is a model-based optimal state estimator. Without loss of generality, we assume that the robotic dynamics model used in MHE takes the following form:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}\left(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}\right), \ \boldsymbol{y} = \boldsymbol{h}\left(\boldsymbol{x}\right) + \boldsymbol{\nu}, \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the system state, $\boldsymbol{u} \in \mathbb{R}^m$ is the control input, $\boldsymbol{f}$ is the system model, $\boldsymbol{w} \in \mathbb{R}^w$ is the process noise, $\boldsymbol{h}$ is the output function, and $\boldsymbol{y} \in \mathbb{R}^l$ denotes the measurement affected by the noise $\boldsymbol{\nu} \in \mathbb{R}^l$. At time t, given the most recent measurements $\mathbf{y}_t = \{\boldsymbol{y}_k\}_{k=t-N}^{t}$ and control inputs $\mathbf{u}_t = \{\boldsymbol{u}_k\}_{k=t-N}^{t-1}$ collected in a moving data window of horizon N, the MHE estimator optimizes over $\mathbf{x} = \{\boldsymbol{x}_k\}_{k=t-N}^{t}$ and $\mathbf{w} = \{\boldsymbol{w}_k\}_{k=t-N}^{t-1}$ at each time step $t \geq N$ by solving the following nonlinear optimization problem online.

$$\min_{\mathbf{x},\mathbf{w}} J = \underbrace{\frac{1}{2}\left\|\boldsymbol{x}_{t-N} - \hat{\boldsymbol{x}}_{t-N}\right\|_{\boldsymbol{P}}^2}_{\text{arrival cost}} + \underbrace{\frac{1}{2}\sum_{k=t-N}^{t}\left\|\boldsymbol{y}_k - \boldsymbol{h}\left(\boldsymbol{x}_k\right)\right\|_{\boldsymbol{R}_k}^2 + \frac{1}{2}\sum_{k=t-N}^{t-1}\left\|\boldsymbol{w}_k\right\|_{\boldsymbol{Q}_k}^2}_{\text{running cost}} \tag{2a}$$

$$\text{s.t. } \boldsymbol{x}_{k+1} = \mathbf{f}\left(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k, \Delta t\right), \tag{2b}$$

where $\boldsymbol{P} \in \mathbb{R}^{n\times n}$, $\boldsymbol{R}_k \in \mathbb{R}^{l\times l}$, and $\boldsymbol{Q}_k \in \mathbb{R}^{w\times w}$ are the positive-definite weighting matrices, $\mathbf{f}$ is the discrete-time model of $\boldsymbol{f}$ with the step-size of Δt for predicting the state, and $\hat{\boldsymbol{x}}_{t-N}$ is the filter priori, chosen as the MHE's optimal estimate $\hat{\boldsymbol{x}}_{t-N|t-1}$ of $\boldsymbol{x}_{t-N}$ obtained at $t-1$ [24]. We denote by $\hat{\mathbf{x}} = \left\{\hat{\boldsymbol{x}}_{k|t}\right\}_{k=t-N}^{t}$ and $\hat{\mathbf{w}} = \left\{\hat{\boldsymbol{w}}_{k|t}\right\}_{k=t-N}^{t-1}$ the MHE's optimal solutions to Problem (2) at time step t.

B. Formulation of NeuroMHE

The weighting matrices in the cost function (2a) are the tuning parameters, which determine the MHE performance. For ease of presentation, we collect them in a vector $\boldsymbol{\theta} = \left[vec\left(\boldsymbol{P}\right), \{vec\left(\boldsymbol{R}_k\right)\}_{k=t-N}^{t}, \{vec\left(\boldsymbol{Q}_k\right)\}_{k=t-N}^{t-1}\right] \in \mathbb{R}^p$ where $vec\left(\cdot\right)$ denotes the vectorization of a given matrix. Tuning $\boldsymbol{\theta}$ typically necessitates expert knowledge of the covariances of the noises entering robotic systems. In practice, identifying these noise covariances is difficult due to their large number and strong coupling. It becomes even more demanding when these values exhibit dynamic behavior.

NeuroMHE tackles the above challenge by modeling $\boldsymbol{\theta}$ with a portable neural network:

$$\boldsymbol{\theta} = \boldsymbol{f}_{\boldsymbol{\varpi}}\left(\boldsymbol{\chi}\right). \tag{3}$$

The neural network can accept various signals as its inputs $\boldsymbol{\chi}$, including measurements $\boldsymbol{y}$, controls $\boldsymbol{u}$, and other external signals $\boldsymbol{o}$, depending on specific applications. Its parameters $\boldsymbol{\varpi}$ are trained using powerful machine learning techniques. Here, $\boldsymbol{\varpi}$ represents the tuning parameters for NeuroMHE, and we parameterize Problem (2) as $\mathrm{NeuroMHE}\left(\boldsymbol{\varpi}\right)$, with the MHE's solution denoted as $\hat{\mathbf{x}}\left(\boldsymbol{\varpi}\right)$. To tune MHE, a high-level optimization problem can be formulated as follows:

$$\min_{\boldsymbol{\varpi}} \ L\left(\hat{\mathbf{x}}\left(\boldsymbol{\varpi}\right)\right) \tag{4a}$$

$$\text{s.t. } \hat{\mathbf{x}}\left(\boldsymbol{\varpi}\right) \text{ generated by } \mathrm{NeuroMHE}\left(\boldsymbol{\varpi}\right). \tag{4b}$$

where $L\left(\hat{\mathbf{x}}\left(\boldsymbol{\varpi}\right)\right)$ is a loss function assessing NeuroMHE's performance. It can be built upon estimation errors when

ground truth disturbance data is available. Alternatively, it can be tailored to penalize tracking errors in applications involving robust trajectory tracking control.

III. Analytical Hessian Trajectory

Our previous work [15] solved Problem (4) using gradient descent, with the gradient computed recursively through a Kalman filter. In this section, our objective is to efficiently reuse the same Kalman filter and much of the computation initially employed to obtain the gradient. We intend to apply these resources in computing the MHE Hessian for training NeuroMHE with second-order optimization techniques.

A. MHE Gradient

The gradient of $L\left(\hat{\mathbf{x}}\left(\boldsymbol{\varpi}\right)\right)$ w.r.t $\boldsymbol{\varpi}$ can be computed using the chain rule:

$$\nabla_{\boldsymbol{\varpi}} L\left(\hat{\mathbf{x}}\left(\boldsymbol{\varpi}\right)\right) = \nabla_{\hat{\mathbf{x}}} L\left(\hat{\mathbf{x}}\right) \nabla_{\boldsymbol{\theta}} \hat{\mathbf{x}}\left(\boldsymbol{\theta}\right) \nabla_{\boldsymbol{\varpi}} \boldsymbol{\theta}\left(\boldsymbol{\varpi}\right). \tag{5}$$

The gradients $\nabla_{\hat{\mathbf{x}}} L$ and $\nabla_{\boldsymbol{\varpi}} \boldsymbol{\theta}$ are straightforward to obtain as both $L\left(\hat{\mathbf{x}}\right)$ and $\boldsymbol{\theta}\left(\boldsymbol{\varpi}\right)$ are explicit functions. The main challenge lies in the computation of $\nabla_{\boldsymbol{\theta}} \hat{\mathbf{x}}$ (the MHE gradient). Recall that $\hat{\mathbf{x}}\left(\boldsymbol{\theta}\right)$ is the MHE's solution, with its dependence on $\boldsymbol{\theta}$ implicitly defined via the KKT conditions. This implicit definition enables us to compute $\nabla_{\boldsymbol{\theta}} \hat{\mathbf{x}}$ by differentiating through the KKT conditions of Problem (2). To proceed, we associate the equality constraints (2b) with the dual variables $\boldsymbol{\lambda} = \{\boldsymbol{\lambda}_k\}_{k=t-N}^{t-1}$ and denote $\boldsymbol{\lambda}^* \in \mathbb{R}^n$ as their optimal values. The corresponding Lagrangian can be written as

$$\mathcal{L} = J + \sum_{k=t-N}^{t-1} \boldsymbol{\lambda}_k^T \left(\boldsymbol{x}_{k+1} - \mathbf{f}\left(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k, \Delta t\right)\right). \tag{6}$$

Then the KKT conditions of Problem (2) at $\hat{\mathbf{x}}$, $\hat{\mathbf{w}}$, and $\boldsymbol{\lambda}^*$ are given by

$$\nabla_{\hat{\boldsymbol{x}}_{t-N|t}} \mathcal{L}\left(\hat{\boldsymbol{x}}_{t-N|t}, \hat{\boldsymbol{x}}_{t-N}, \hat{\boldsymbol{w}}_{t-N|t}, \boldsymbol{\lambda}_{t-N}^*, \boldsymbol{\theta}\right) = \mathbf{0}, \tag{7a}$$

$$\nabla_{\hat{\boldsymbol{x}}_{k|t}} \mathcal{L}\left(\hat{\boldsymbol{x}}_{k|t}, \hat{\boldsymbol{w}}_{k|t}, \boldsymbol{\lambda}_k^*, \boldsymbol{\lambda}_{k-1}^*, \boldsymbol{\theta}\right) = \mathbf{0}, \tag{7b}$$

$$\nabla_{\hat{\boldsymbol{w}}_{k|t}} \mathcal{L}\left(\hat{\boldsymbol{x}}_{k|t}, \hat{\boldsymbol{w}}_{k|t}, \boldsymbol{\lambda}_k^*, \boldsymbol{\theta}\right) = \mathbf{0}, \tag{7c}$$

$$\nabla_{\lambda_k^*} \mathcal{L} = \hat{\boldsymbol{x}}_{k+1|t} - \mathbf{f}\left(\hat{\boldsymbol{x}}_{k|t}, \boldsymbol{u}_k, \hat{\boldsymbol{w}}_{k|t}, \Delta t\right) = \mathbf{0}, \tag{7d}$$

Differentiating the KKT conditions (7) w.r.t $\boldsymbol{\theta}$ yields the following differential KKT conditions.

$$\begin{aligned}\frac{d\nabla_{\hat{x}_{t-N|t}} \mathcal{L}}{d\boldsymbol{\theta}} &= \boldsymbol{L}_{t-N}^{xx} \nabla_{\boldsymbol{\theta}} \hat{\boldsymbol{x}}_{t-N|t} - \boldsymbol{P} \nabla_{\boldsymbol{\theta}} \hat{\boldsymbol{x}}_{t-N} + \boldsymbol{L}_{t-N}^{x\theta} \\ &\quad + \boldsymbol{L}_{t-N}^{xw} \nabla_{\boldsymbol{\theta}} \hat{\boldsymbol{w}}_{t-N|t} - \boldsymbol{F}_{t-N}^T \nabla_{\boldsymbol{\theta}} \boldsymbol{\lambda}_{t-N}^* = \mathbf{0},\end{aligned} \tag{8a}$$

$$\begin{aligned}\frac{d\nabla_{\hat{x}_{k|t}} \mathcal{L}}{d\boldsymbol{\theta}} &= \boldsymbol{L}_k^{xx} \nabla_{\boldsymbol{\theta}} \hat{\boldsymbol{x}}_{k|t} + \boldsymbol{L}_k^{xw} \nabla_{\boldsymbol{\theta}} \hat{\boldsymbol{w}}_{k|t} - \boldsymbol{F}_k^T \nabla_{\boldsymbol{\theta}} \boldsymbol{\lambda}_k^* \\ &\quad + \nabla_{\boldsymbol{\theta}} \boldsymbol{\lambda}_{k-1}^* + \boldsymbol{L}_k^{x\theta} = \mathbf{0},\end{aligned} \tag{8b}$$

$$\begin{aligned}\frac{d\nabla_{\hat{w}_{k|t}} \mathcal{L}}{d\boldsymbol{\theta}} &= \boldsymbol{L}_k^{wx} \nabla_{\boldsymbol{\theta}} \hat{\boldsymbol{x}}_{k|t} + \boldsymbol{L}_k^{ww} \nabla_{\boldsymbol{\theta}} \hat{\boldsymbol{w}}_{k|t} \\ &\quad - \boldsymbol{G}_k^T \nabla_{\boldsymbol{\theta}} \boldsymbol{\lambda}_k^* + \boldsymbol{L}_k^{w\theta} = \mathbf{0},\end{aligned} \tag{8c}$$

$$\frac{d\nabla_{\lambda_k^*} \mathcal{L}}{d\boldsymbol{\theta}} = \nabla_{\boldsymbol{\theta}} \hat{\boldsymbol{x}}_{k+1|t} - \boldsymbol{F}_k \nabla_{\boldsymbol{\theta}} \hat{\boldsymbol{x}}_{k|t} - \boldsymbol{G}_k \nabla_{\boldsymbol{\theta}} \hat{\boldsymbol{w}}_{k|t} = \mathbf{0}, \tag{8d}$$

where $\boldsymbol{F}_k = \nabla_{\hat{\boldsymbol{x}}_{k|t}} \mathbf{f}$, $\boldsymbol{G}_k = \nabla_{\hat{\boldsymbol{w}}_{k|t}} \mathbf{f}$, $\boldsymbol{H}_k = \nabla_{\hat{\boldsymbol{x}}_{k|t}} \boldsymbol{h}$, $\boldsymbol{\lambda}_t^* = \mathbf{0}$, $\boldsymbol{L}_k^{xx} = \frac{\partial^2 \mathcal{L}}{\partial \hat{\boldsymbol{x}}_{k|t}^2}$, $\boldsymbol{L}_k^{xw} = \frac{\partial^2 \mathcal{L}}{\partial \hat{\boldsymbol{x}}_{k|t} \partial \hat{\boldsymbol{w}}_{k|t}}$, $\boldsymbol{L}_k^{x\theta} = \frac{\partial^2 \mathcal{L}}{\partial \hat{\boldsymbol{x}}_{k|t} \partial \boldsymbol{\theta}}$, $\boldsymbol{L}_k^{ww} = \frac{\partial^2 \mathcal{L}}{\partial \hat{\boldsymbol{w}}_{k|t}^2}$, $\boldsymbol{L}_k^{wx} = \frac{\partial^2 \mathcal{L}}{\partial \hat{\boldsymbol{w}}_{k|t} \partial \hat{\boldsymbol{x}}_{k|t}}$, and $\boldsymbol{L}_k^{w\theta} = \frac{\partial^2 \mathcal{L}}{\partial \hat{\boldsymbol{w}}_{k|t} \partial \boldsymbol{\theta}}$.

The above differential KKT conditions (8) can be easily shown to be the same as the KKT conditions of the following linear MHE problem:

$$\begin{aligned}\min_{\mathbf{X},\mathbf{W}} J_2 = &\frac{1}{2} \mathrm{Tr} \left\| \boldsymbol{X}_{t-N} - \hat{\boldsymbol{X}}_{t-N} \right\|_{\boldsymbol{P}}^2 \\ &+ \mathrm{Tr} \sum_{k=t-N}^{t} \left(\frac{1}{2} \boldsymbol{X}_k^T \bar{\boldsymbol{L}}_k^{xx} \boldsymbol{X}_k + \boldsymbol{W}_k{}^T \boldsymbol{L}_k^{wx} \boldsymbol{X}_k \right) \\ &+ \mathrm{Tr} \sum_{k=t-N}^{t-1} \left(\frac{1}{2} \boldsymbol{W}_k^T \boldsymbol{L}_k^{ww} \boldsymbol{W}_k + \left(\boldsymbol{L}_k^{w\theta}\right)^T \boldsymbol{W}_k \right) \\ &+ \mathrm{Tr} \sum_{k=t-N}^{t} \left(\left(\boldsymbol{L}_k^{x\theta}\right)^T \boldsymbol{X}_k \right)\end{aligned} \tag{9a}$$

$$\text{s.t. } \boldsymbol{X}_{k+1} = \boldsymbol{F}_k \boldsymbol{X}_k + \boldsymbol{G}_k \boldsymbol{W}_k, \tag{9b}$$

where $\mathbf{X} = \{\boldsymbol{X}_k\}_{k=t-N}^{t}$, $\mathbf{W} = \{\boldsymbol{W}_k\}_{k=t-N}^{t-1}$, $\bar{\boldsymbol{L}}_k^{xx} = \frac{\partial^2 \bar{\mathcal{L}}}{\partial \hat{\boldsymbol{x}}_{k|t}^2}$ with $\bar{\mathcal{L}}$ obtained by excluding the arrival cost (See (2a)) from $\mathcal{L}$, and $\mathrm{Tr}\left(\cdot\right)$ is the matrix trace. Hence, the optimal solution $\hat{\mathbf{X}}$ to Problem (9) is exactly the gradient $\nabla_{\boldsymbol{\theta}} \hat{\mathbf{x}}$, denoted as $\hat{\mathbf{X}} = \left\{\hat{\boldsymbol{X}}_k\right\}_{k=t-N}^{t} = \left\{\nabla_{\boldsymbol{\theta}} \hat{\boldsymbol{x}}_{k|t}\right\}_{k=t-N}^{t}$.

Note that (9) has a quadratic cost function (9a) with linear dynamics constraints (9b). It is possible to obtain $\hat{\mathbf{X}}$ analytically. In [15], we proposed a recursive method for computing $\hat{\mathbf{X}}$ using a Kalman filter. The details of this method will be presented in Algorithm 1 (See III-C). Next, we will focus on an interesting observation: the Hessian of the loss function can also be efficiently computed using a new auxiliary linear MHE system. Remarkably, this approach leverages the same Kalman filter and much of the computation that was already used to obtain the gradient.

B. MHE Hessian

Let $\mathrm{H}_{\hat{\mathbf{x}}} L$ denote the Hessian of L w.r.t $\hat{\mathbf{x}}$, $\mathrm{H}_{\boldsymbol{\theta}} \hat{\mathbf{x}} := \partial vec\left(\nabla_{\theta} \hat{\mathbf{x}}\right) / \partial \boldsymbol{\theta}$ the MHE Hessian, and $\otimes$ the Kronecker product. The total second-order derivative of $L\left(\hat{\mathbf{x}}\left(\boldsymbol{\varpi}\right)\right)$ w.r.t $\boldsymbol{\varpi}$ can be obtained using the chain rule of matrix calculus:

$$\begin{aligned}H_{\boldsymbol{\varpi}} L\left(\hat{\mathbf{x}}\left(\boldsymbol{\varpi}\right)\right) = &\left(\nabla_{\boldsymbol{\theta}} \hat{\mathbf{x}}\right)^T \mathrm{H}_{\hat{\mathbf{x}}} L \nabla_{\boldsymbol{\theta}} \hat{\mathbf{x}} \left(\nabla_{\boldsymbol{\varpi}} \boldsymbol{\theta}\right)^2 \\ &+ \left(\nabla_{\boldsymbol{\varpi}} \boldsymbol{\theta}\right)^T \mathrm{H}_{\boldsymbol{\theta}} \hat{\mathbf{x}} \left[\left(\nabla_{\hat{\mathbf{x}}} L\right)^T \otimes \boldsymbol{I}\right] \nabla_{\boldsymbol{\varpi}} \boldsymbol{\theta} \\ &+ \left[\left(\nabla_{\hat{\mathbf{x}}} L \nabla_{\boldsymbol{\theta}} \hat{\mathbf{x}}\right) \otimes \boldsymbol{I}\right] \mathrm{H}_{\boldsymbol{\varpi}} \boldsymbol{\theta}.\end{aligned} \tag{10}$$

Note that all the first-order derivatives have been obtained in computing the gradient of $L\left(\hat{\mathbf{x}}\left(\boldsymbol{\varpi}\right)\right)$ w.r.t $\boldsymbol{\varpi}$. Computing the second-order derivatives $\mathrm{H}_{\hat{\mathbf{x}}} L$ and $\mathrm{H}_{\boldsymbol{\varpi}} \boldsymbol{\theta}$ is straightforward, which is explicit differentiation of the loss function and the NN, respectively. The major challenge lies in the computation of $\mathrm{H}_{\boldsymbol{\theta}} \hat{\mathbf{x}}$.

Given the implicit dependence of $\hat{\mathbf{x}}$ on $\boldsymbol{\theta}$ through KKT conditions, we are motivated to differentiate the KKT condi-

tions (7) w.r.t $\boldsymbol{\theta}$ twice. This results in the following second-order differential KKT conditions.

$$H_{\boldsymbol{\theta}}\nabla_{\hat{\boldsymbol{x}}_{t-N|t}}\mathcal{L} = \mathring{\boldsymbol{L}}^{xx}_{t-N}\mathrm{H}_{\boldsymbol{\theta}}\hat{\boldsymbol{x}}_{t-N|t} - \mathring{\boldsymbol{P}}\mathrm{H}_{\boldsymbol{\theta}}\hat{\boldsymbol{x}}_{t-N} + \mathring{\boldsymbol{L}}^{x\theta}_{t-N} + \mathring{\boldsymbol{L}}^{xw}_{t-N}\mathrm{H}_{\boldsymbol{\theta}}\hat{\boldsymbol{w}}_{t-N|t} - \mathring{\boldsymbol{F}}^{T}_{t-N}\mathrm{H}_{\boldsymbol{\theta}}\boldsymbol{\lambda}^{*}_{t-N} = \boldsymbol{0}, \tag{11a}$$

$$H_{\boldsymbol{\theta}}\nabla_{\hat{\boldsymbol{x}}_{k|t}}\mathcal{L} = \mathring{\boldsymbol{L}}^{xx}_{k}\mathrm{H}_{\boldsymbol{\theta}}\hat{\boldsymbol{x}}_{k|t} + \mathring{\boldsymbol{L}}^{xw}_{k}\mathrm{H}_{\boldsymbol{\theta}}\hat{\boldsymbol{w}}_{k|t} - \mathring{\boldsymbol{F}}^{T}_{k}\mathrm{H}_{\boldsymbol{\theta}}\boldsymbol{\lambda}^{*}_{k} + \mathrm{H}_{\boldsymbol{\theta}}\boldsymbol{\lambda}^{*}_{k-1} + \mathring{\boldsymbol{L}}^{x\theta}_{k} = \boldsymbol{0}, \tag{11b}$$

$$H_{\boldsymbol{\theta}}\nabla_{\hat{\boldsymbol{w}}_{k|t}}\mathcal{L} = \mathring{\boldsymbol{L}}^{wx}_{k}\mathrm{H}_{\boldsymbol{\theta}}\hat{\boldsymbol{x}}_{k|t} + \mathring{\boldsymbol{L}}^{ww}_{k}\mathrm{H}_{\boldsymbol{\theta}}\hat{\boldsymbol{w}}_{k|t} - \mathring{\boldsymbol{G}}^{T}_{k}\mathrm{H}_{\boldsymbol{\theta}}\boldsymbol{\lambda}^{*}_{k} + \mathring{\boldsymbol{L}}^{w\theta}_{k} = \boldsymbol{0}, \tag{11c}$$

$$H_{\boldsymbol{\theta}}\nabla_{\boldsymbol{\lambda}^{*}_{k}}\mathcal{L} = \mathrm{H}_{\boldsymbol{\theta}}\hat{\boldsymbol{x}}_{k+1|t} - \mathring{\boldsymbol{F}}_{k}\mathrm{H}_{\boldsymbol{\theta}}\hat{\boldsymbol{x}}_{k|t} - \mathring{\boldsymbol{G}}_{k}\mathrm{H}_{\boldsymbol{\theta}}\hat{\boldsymbol{w}}_{k|t} + \mathring{\boldsymbol{L}}^{\lambda\theta}_{k} = \boldsymbol{0}, \tag{11d}$$

where the coefficient matrices are defined below:

$$\begin{aligned}
\mathring{\boldsymbol{L}}^{xx}_{k} &= \boldsymbol{I}\otimes\boldsymbol{L}^{xx}_{k},\ \mathring{\boldsymbol{F}}_{k} = \boldsymbol{I}\otimes\boldsymbol{F}_{k},\ \mathring{\boldsymbol{G}}_{k} = \boldsymbol{I}\otimes\boldsymbol{G}_{k},\\
\mathring{\boldsymbol{L}}^{xw}_{k} &= \boldsymbol{I}\otimes\boldsymbol{L}^{xw}_{k},\ \mathring{\boldsymbol{L}}^{ww}_{k} = \boldsymbol{I}\otimes\boldsymbol{L}^{ww}_{k},\ \mathring{\boldsymbol{P}} = \boldsymbol{I}\otimes\boldsymbol{P},\\
\mathring{\boldsymbol{L}}^{x\theta}_{t-N} &= \left(\nabla_{\hat{\boldsymbol{x}}_{t-N|t}}\bar{\boldsymbol{f}}_0\right)\hat{\boldsymbol{X}}_{t-N|t} + \left(\nabla_{\boldsymbol{\lambda}^{*}_{t-N}}\bar{\boldsymbol{f}}_0\right)\boldsymbol{\Lambda}^{*}_{t-N}\\
&\quad + \left(\nabla_{\hat{\boldsymbol{w}}_{t-N|t}}\bar{\boldsymbol{f}}_0\right)\hat{\boldsymbol{W}}_{t-N|t} + \nabla_{\boldsymbol{\theta}}\bar{\boldsymbol{f}}_0,\\
\mathring{\boldsymbol{L}}^{x\theta}_{k} &= \left(\nabla_{\hat{\boldsymbol{x}}_{k|t}}\bar{\boldsymbol{f}}_k\right)\hat{\boldsymbol{X}}_{k|t} + \left(\nabla_{\hat{\boldsymbol{w}}_{k|t}}\bar{\boldsymbol{f}}_k\right)\hat{\boldsymbol{W}}_{k|t}\\
&\quad + \left(\nabla_{\boldsymbol{\lambda}^{*}_{k}}\bar{\boldsymbol{f}}_k\right)\boldsymbol{\Lambda}^{*}_{k} + \nabla_{\boldsymbol{\theta}}\bar{\boldsymbol{f}}_k,\\
\mathring{\boldsymbol{L}}^{w\theta}_{k} &= \left(\nabla_{\hat{\boldsymbol{x}}_{k|t}}\bar{\boldsymbol{g}}_k\right)\hat{\boldsymbol{X}}_{k|t} + \left(\nabla_{\hat{\boldsymbol{w}}_{k|t}}\bar{\boldsymbol{g}}_k\right)\hat{\boldsymbol{W}}_{k|t}\\
&\quad + \left(\nabla_{\boldsymbol{\lambda}^{*}_{k}}\bar{\boldsymbol{g}}_k\right)\boldsymbol{\Lambda}^{*}_{k} + \nabla_{\boldsymbol{\theta}}\bar{\boldsymbol{g}}_k,\\
\mathring{\boldsymbol{L}}^{\lambda\theta}_{k} &= \left(\nabla_{\hat{\boldsymbol{x}}_{k|t}}\bar{\boldsymbol{h}}_k\right)\hat{\boldsymbol{X}}_{k|t} + \left(\nabla_{\hat{\boldsymbol{w}}_{k|t}}\bar{\boldsymbol{h}}_k\right)\hat{\boldsymbol{W}}_{k|t},
\end{aligned} \tag{12}$$

with $\bar{\boldsymbol{f}}_0$, $\bar{\boldsymbol{f}}_k$, $\bar{\boldsymbol{g}}_k$, and $\bar{\boldsymbol{h}}_k$ being the left hand sides of (8a)-(8d).

It is interesting to notice that (11) and (8) share exactly the same structure. The only exception lies in (11d) which has an additional term $\mathring{\boldsymbol{L}}^{\lambda\theta}_{k}$, not present in (8d). This term can be considered as a known input to the linear system. As shown in III-A and [15], (8) can represent the KKT conditions of the auxiliary MHE optimization problem (9), which is subject to the linear system (9b) constructed from (8d). Given this observation and concept, we can interpret (11) as the KKT conditions of another auxiliary MHE optimization problem. It is subject to a new linear system constructed from (11d):

$$\mathring{\boldsymbol{X}}_{k+1} = \mathring{\boldsymbol{F}}_{k}\mathring{\boldsymbol{X}}_{k} + \mathring{\boldsymbol{G}}_{k}\mathring{\boldsymbol{W}}_{k} - \mathring{\boldsymbol{L}}^{\lambda\theta}_{k}, \tag{13}$$

where $\mathring{\boldsymbol{X}}_k$ and $\mathring{\boldsymbol{W}}_k$ denote $\mathrm{H}_{\boldsymbol{\theta}}\boldsymbol{x}_k$ and $\mathrm{H}_{\boldsymbol{\theta}}\boldsymbol{w}_k$, respectively. The related cost function can be established directly from (9a) by updating the latter's coefficient matrices, for example, by replacing $\boldsymbol{P}$ with $\mathring{\boldsymbol{P}}$ defined in (12). Consequently, we can efficiently reuse Algorithm 1 to solve for the MHE Hessian $\mathrm{H}_{\boldsymbol{\theta}}\hat{\mathbf{x}}$, which is the optimal solution $\hat{\mathring{\mathbf{X}}}$ to the new auxiliary linear MHE problem. Only the algorithm's inputs need minor modifications as follows:

- Replace $\hat{\boldsymbol{X}}_{t-N}$ with $\hat{\mathring{\boldsymbol{X}}}_{t-N}$ (which is approximated by $\mathrm{H}_{\boldsymbol{\theta}}\hat{\boldsymbol{x}}_{t-N|t-1}$ obtained at $t-1$);
- Update the matrices in (16) using (12). In particular, add $\mathring{\boldsymbol{L}}^{\lambda\theta}_{k}$ to (15a) such that the updated $\boldsymbol{A}_k$ becomes $\mathring{\boldsymbol{A}}_k = \mathring{\boldsymbol{G}}_k\left(\mathring{\boldsymbol{L}}^{ww}_{k}\right)^{-1}\mathring{\boldsymbol{L}}^{w\theta}_{k} + \mathring{\boldsymbol{L}}^{\lambda\theta}_{k}$. This can be derived based on the example used to obtain (15a) (See III-C).

C. Kalman Filter-based Solver

First, a Kalman filter (KF) is solved to obtain the estimates $\left\{\hat{\boldsymbol{X}}^{\mathrm{KF}}_{k|k}\right\}^{t}_{k=t-N}$: the initial condition is given by

$$\hat{\boldsymbol{X}}^{\mathrm{KF}}_{t-N|t-N} = \left(\boldsymbol{I} + \boldsymbol{C}_{t-N}\boldsymbol{S}_{t-N}\right)\hat{\boldsymbol{X}}_{t-N} + \boldsymbol{C}_{t-N}\boldsymbol{T}_{t-N}. \tag{14}$$

where $\boldsymbol{I}$ is an identity matrix, $\boldsymbol{C}_{t-N}$ is defined in (15c) with $\boldsymbol{P}_{t-N} = \boldsymbol{P}^{-1}$, $\boldsymbol{S}_{t-N}$ and $\boldsymbol{T}_{t-N}$ are given in (16), and $\hat{\boldsymbol{X}}_{t-N}$ is approximated by $\nabla_{\boldsymbol{\theta}}\hat{\boldsymbol{x}}_{t-N|t-1}$ obtained at $t-1$. Then, the remaining estimates are generated through the following equations from $k = t-N+1$ to t:

$$\hat{\boldsymbol{X}}_{k|k-1} = \bar{\boldsymbol{F}}_{k-1}\hat{\boldsymbol{X}}^{\mathrm{KF}}_{k-1|k-1} - \boldsymbol{A}_{k-1}, \tag{15a}$$

$$\boldsymbol{P}_{k} = \bar{\boldsymbol{F}}_{k-1}\boldsymbol{C}_{k-1}\bar{\boldsymbol{F}}^{T}_{k-1} + \boldsymbol{B}_{k-1}, \tag{15b}$$

$$\boldsymbol{C}_{k} = \left(\boldsymbol{I} - \boldsymbol{P}_{k}\boldsymbol{S}_{k}\right)^{-1}\boldsymbol{P}_{k}, \tag{15c}$$

$$\hat{\boldsymbol{X}}^{\mathrm{KF}}_{k|k} = \left(\boldsymbol{I} + \boldsymbol{C}_{k}\boldsymbol{S}_{k}\right)\hat{\boldsymbol{X}}_{k|k-1} + \boldsymbol{C}_{k}\boldsymbol{T}_{k}. \tag{15d}$$

where $\boldsymbol{A}_k$, $\boldsymbol{B}_k$, $\boldsymbol{S}_k$, $\boldsymbol{T}_k$, and $\bar{\boldsymbol{F}}_k$ are defined below:

$$\begin{aligned}
\boldsymbol{A}_{k} &= \boldsymbol{G}_{k}(\boldsymbol{L}^{ww}_{k})^{-1}\boldsymbol{L}^{w\theta}_{k},\\
\boldsymbol{B}_{k} &= \boldsymbol{G}_{k}(\boldsymbol{L}^{ww}_{k})^{-1}\boldsymbol{G}^{T}_{k},\\
\boldsymbol{S}_{k} &= \boldsymbol{L}^{xw}_{k}(\boldsymbol{L}^{ww}_{k})^{-1}\boldsymbol{L}^{wx}_{k} - \bar{\boldsymbol{L}}^{xx}_{k},\ \boldsymbol{S}_{t} = -\bar{\boldsymbol{L}}^{xx}_{t},\\
\boldsymbol{T}_{k} &= \boldsymbol{L}^{xw}_{k}(\boldsymbol{L}^{ww}_{k})^{-1}\boldsymbol{L}^{w\theta}_{k} - \boldsymbol{L}^{x\theta}_{k},\ \boldsymbol{T}_{t} = -\boldsymbol{L}^{x\theta}_{t},\\
\bar{\boldsymbol{F}}_{k} &= \boldsymbol{F}_{k} - \boldsymbol{G}_{k}(\boldsymbol{L}^{ww}_{k})^{-1}\boldsymbol{L}^{wx}_{k}.
\end{aligned} \tag{16}$$

Second, the new dual variables $\boldsymbol{\Lambda}^{*} = \{\boldsymbol{\Lambda}^{*}_{k}\}^{t-1}_{k=t-N}$ are computed using the following equation backward in time from $k = t$ to $t-N+1$, starting with $\boldsymbol{\Lambda}^{*}_{t} = \boldsymbol{0}$:

$$\boldsymbol{\Lambda}^{*}_{k-1} = \left(\boldsymbol{I} + \boldsymbol{S}_{k}\boldsymbol{C}_{k}\right)\bar{\boldsymbol{F}}^{T}_{k}\boldsymbol{\Lambda}^{*}_{k} + \boldsymbol{S}_{k}\hat{\boldsymbol{X}}^{\mathrm{KF}}_{k|k} + \boldsymbol{T}_{k}. \tag{17}$$

Third, the optimal estimates $\hat{\mathbf{X}}$ are obtained using

$$\hat{\boldsymbol{X}}_{k|t} = \hat{\boldsymbol{X}}^{\mathrm{KF}}_{k|k} + \boldsymbol{C}_{k}\bar{\boldsymbol{F}}^{T}_{k}\boldsymbol{\Lambda}^{*}_{k}, \tag{18}$$

iteratively from $k = t-N$ to t.

The above recursions are summarized in Algorithm 1.

Algorithm 1: Solving for $\hat{\mathbf{X}}$ using a Kalman filter

Input: $\hat{\boldsymbol{X}}_{t-N}$ and the matrices in (16);
def *Kalman_Filter_based_Gradient_Solver*:
  Set $\hat{\boldsymbol{X}}^{\mathrm{KF}}_{t-N|t-N}$ using (14);
  for $k \leftarrow t-N+1$ **to** t **by** 1 **do**
    Update $\hat{\boldsymbol{X}}^{\mathrm{KF}}_{k|k}$ using the Kalman filter (15);
  end for
  for $k \leftarrow t$ **to** $t-N+1$ **by** -1 **do**
    Update $\boldsymbol{\Lambda}^{*}_{k-1}$ using (17) with $\boldsymbol{\Lambda}^{*}_{t} = \boldsymbol{0}$;
  end for
  for $k \leftarrow t-N$ **to** t **by** 1 **do**
    Update $\hat{\boldsymbol{X}}_{k|t}$ using (18);
  end for
  return $\left\{\hat{\boldsymbol{X}}_{k|t}\right\}^{t}_{k=t-N}$
Output: $\nabla_{\boldsymbol{\theta}}\hat{\mathbf{x}} = \hat{\mathbf{X}}$

In the Kalman filter (15), (15a) is the state predictor, (15b) handles the error covariance prediction, (15c) addresses the error covariance correction, and (15d) is the state corrector. These equations are derived from the differential KKT conditions (8) by induction. For example, we obtain (15a) by following steps: 1) solve for $\hat{\boldsymbol{W}}_{k|t} := \nabla_{\boldsymbol{\theta}} \hat{\boldsymbol{w}}_{k|t}$ from (8c); 2) plug $\hat{\boldsymbol{W}}_{k|t}$ and $\hat{\boldsymbol{X}}_{k|t}$ (obtained from (18)) into (8d); 3) define the resulting equation, excluding the term related to $\boldsymbol{\Lambda}_k^* := \nabla_{\boldsymbol{\theta}} \boldsymbol{\lambda}_k^*$, as $\hat{\boldsymbol{X}}_{k+1|k}$ using $\boldsymbol{A}_k$ and $\bar{\boldsymbol{F}}_k$ given in (16). The detailed theoretical justification of (15) can be found in Appendix-C of [15]. Note that (15) is not presented in the standard form of a Kalman filter for the purpose of a clearer and more inductive proof, as explained in [15]. The Kalman filter provides analytical gradients in a recursive form, significantly enhancing computational efficiency.

IV. Trust-Region Learning Framework

We aim to train the neural network's parameters $\boldsymbol{\varpi}$ using the trust-region method, which addresses Problem (4) by creating a trust region around the current parameters $\boldsymbol{\varpi}_t$. Within this region, it minimizes a locally quadratic approximation of $L\left(\hat{\mathbf{x}}\left(\boldsymbol{\varpi}_t\right)\right)$ to determine a candidate update $\boldsymbol{\mu}$ for $\boldsymbol{\varpi}_t$. The corresponding optimization problem is as follows:

$$\min_{\boldsymbol{\mu}} \operatorname{TRM}(\boldsymbol{\mu}) := L + \boldsymbol{\mu}^T \nabla_{\boldsymbol{\varpi}_t} L + \frac{1}{2} \boldsymbol{\mu}^T \left(H_{\boldsymbol{\varpi}_t} L\right) \boldsymbol{\mu} \quad (19a)$$

$$\text{s.t. } \|\boldsymbol{\mu}\| \leq \Delta_t. \quad (19b)$$

Given predefined thresholds $0 \leq \xi_1 \leq \xi_2 < \xi_3$, $\kappa_1 \in (0,1)$, $\kappa_2 > 1$, and the upper bound of the radius $\bar{\Delta}$, the method iteratively explores the range of trust region by updating its radius $\Delta_t \in \mathbb{R}_+$ with the following law [25]:

$$\Delta_{t+1} = \begin{cases} \kappa_1 \Delta_t & \text{if } \rho_t < \xi_2 \\ \min\left(\kappa_2 \Delta_t, \bar{\Delta}\right) & \text{if } \rho_t > \xi_3 \& \|\boldsymbol{\mu}_t\| = \Delta_t \\ \Delta_t & \text{otherwise} \end{cases} \quad (20)$$

Here, ρ_t denotes the current ratio between the actual and predicted reductions on the loss function

$$\rho_t = \frac{L\left(\hat{\mathbf{x}}\left(\boldsymbol{\varpi}_t\right)\right) - L\left(\hat{\mathbf{x}}\left(\boldsymbol{\varpi}_t + \boldsymbol{\mu}_t\right)\right)}{\operatorname{TRM}(\mathbf{0}) - \operatorname{TRM}\left(\boldsymbol{\mu}_t\right)}, \quad (21)$$

which is used to decide whether to update $\boldsymbol{\varpi}_t$: if $\rho_t > \xi_1$, $\boldsymbol{\varpi}_t$ will be updated using $\boldsymbol{\mu}_t$. We summarize the trust-region policy optimization for training NeuroMHE in Algorithm 2.

V. Numerical Case Study

We numerically validate the performance of trust-region NeuroMHE in estimating complex aerodynamic disturbances on quadrotors using a real flight dataset from various agile flights [13]. We compare it with the gradient descent counterpart and the state-of-the-art estimator NeuroBEM [13] to showcase the following advantages: 1) highly-efficient training; 2) improved robustness to network initialization; and 3) superior force estimation accuracy.

In alignment with NeuroBEM, NeuroMHE is trained using estimation errors. In Algorithm 2, we define the desired states $\mathcal{S}$ using the ground truth disturbance forces $\boldsymbol{F}$ and torques $\boldsymbol{\tau}$ from the real dataset and the quadrotor model. For

Algorithm 2: Trust-region NeuroMHE training

Input: The desired states $\mathcal{S}$ and the measurements $\mathcal{Y}$.

while L_{mean} *not converged* **do**
  for $t \leftarrow 0$ **to** T **do**
    Obtain $\hat{\mathbf{x}}\left(\boldsymbol{\varpi}_t\right)$ by solving $\operatorname{NeuroMHE}\left(\boldsymbol{\varpi}_{\mathrm{t}}\right)$ with the most recent measurements $\mathbf{y}_t \in \mathcal{Y}$;
    Compute $L\left(\hat{\mathbf{x}}\left(\boldsymbol{\varpi}_t\right)\right)$ using $\hat{\mathbf{x}}\left(\boldsymbol{\varpi}_t\right)$ and $\mathbf{s} \in \mathcal{S}$;
    Compute $\nabla_{\boldsymbol{\varpi}_t} L$ using (5) and Algorithm 1;
    Compute $H_{\boldsymbol{\varpi}_t} L$ using (10) and Algorithm 1 with its inputs updated by (12);
    Obtain $\boldsymbol{\mu}_t$ by solving TRM Problem (19);
    Obtain $\hat{\mathbf{x}}\left(\boldsymbol{\varpi}_t + \boldsymbol{\mu}_t\right)$ by solving $\operatorname{NeuroMHE}\left(\boldsymbol{\varpi}_{\mathrm{t}} + \boldsymbol{\mu}_{\mathrm{t}}\right)$ with the same $\mathbf{y}_t$;
    Compute $L\left(\hat{\mathbf{x}}\left(\boldsymbol{\varpi}_t + \boldsymbol{\mu}_t\right)\right)$ using $\hat{\mathbf{x}}\left(\boldsymbol{\varpi}_t + \boldsymbol{\mu}_t\right)$ and the same $\mathbf{s}$;
    Update ρ_t using (21);
    Update Δ_t using (20);
    if $\rho_t > \xi_1$ **then**
      Update $\boldsymbol{\varpi}_{t+1} \leftarrow \boldsymbol{\varpi}_t + \boldsymbol{\mu}_t$;
    else
      $\boldsymbol{\varpi}_{t+1} \leftarrow \boldsymbol{\varpi}_t$;
    end if
  end for
  Compute the mean loss $L_{\text{mean}} = \frac{1}{T} \sum_{t=0}^{T} L\left(\hat{\mathbf{x}}\right)$ for the next episode
end while

NeuroMHE, the measurements $\mathcal{Y}$ are set as the quadrotor's linear $\boldsymbol{v}$ and angular velocities $\boldsymbol{\omega}$. By comparison, the NeuroBEM's inputs also comprise motor speeds, requiring specific sensors and autopilot firmware. To address the state estimation problem (2), we integrate the velocities with the unmeasurable disturbances to create the augmented state $\boldsymbol{x} = [\boldsymbol{v}; \boldsymbol{F}; \boldsymbol{\omega}; \boldsymbol{\tau}] \in \mathbb{R}^{12}$. Assuming no loss of generality, the disturbance dynamics are driven by the process noises $\boldsymbol{w}_f \in \mathbb{R}^3$ and $\boldsymbol{w}_\tau \in \mathbb{R}^3$. The system model for state prediction in MHE is as follows:

$$\dot{\boldsymbol{v}} = m^{-1}\left(-mg\boldsymbol{e}_3 + \boldsymbol{F}\right), \qquad \dot{\boldsymbol{F}} = \boldsymbol{w}_f, \quad (22a)$$

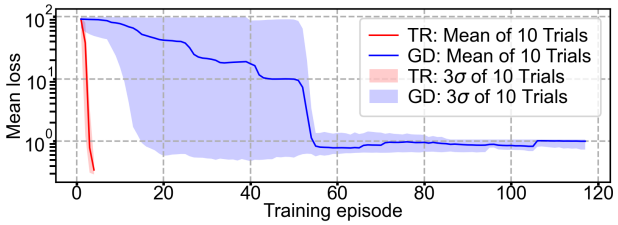
$$\dot{\boldsymbol{\omega}} = \boldsymbol{J}^{-1}\left(-\boldsymbol{\omega}^{\times} \boldsymbol{J}\boldsymbol{\omega} + \boldsymbol{\tau}\right), \qquad \dot{\boldsymbol{\tau}} = \boldsymbol{w}_\tau, \quad (22b)$$

where $g = 9.81\ \mathrm{m/s^2}$, $\boldsymbol{e}_3 = [0; 0; 1]$, $m = 0.772\ \mathrm{kg}$, and $\boldsymbol{J} = \operatorname{diag}(0.0025, 0.0021, 0.0043)\ \mathrm{kgm^2}$ [1].

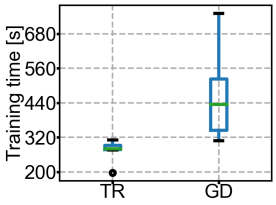
The weighting matrices of NeuroMHE have the following dimensions: $\boldsymbol{P} \in \mathbb{R}^{12\times12}$, $\boldsymbol{R}_k \in \mathbb{R}^{6\times6}$, and $\boldsymbol{Q}_k \in \mathbb{R}^{6\times6}$. Instead of training the neural network to generate all $\boldsymbol{R}_k$ and $\boldsymbol{Q}_k$ over an MHE horizon, we introduce two forgetting factors $\gamma_{1,2} \in (0,1)$ to parameterize the time-varying $\boldsymbol{R}_k$ and $\boldsymbol{Q}_k$ by $\boldsymbol{R}_k = \gamma_1^{t-k} \boldsymbol{R}_t$ and $\boldsymbol{Q}_k = \gamma_2^{t-1-k} \boldsymbol{Q}_{t-1}$. To minimize the network's size, we configure $\boldsymbol{P}$, $\boldsymbol{R}_t$, and $\boldsymbol{Q}_{t-1}$ as diagonal matrices. Finally, the diagonal elements are specified as $P_{\cdot} = \varsigma + p_{\cdot}^2$, $R_{\cdot} = \varsigma + r_{\cdot}^2$, and $Q_{\cdot} = \varsigma + q_{\cdot}^2$, where $\varsigma > 0$, to guarantee the positive definiteness. The gradient (5)

[1]These inertial values are reported at the NeuroBEM's website `https://rpg.ifi.uzh.ch/neuro_bem/Readme.html`.

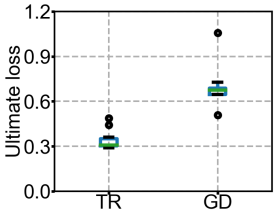
and Hessian (10) are updated accordingly to incorporate the above parameterization. Our neural network has two hidden layers with the Leaky-ReLU activation function. It takes the measurement $\boldsymbol{y} = [\boldsymbol{v}; \boldsymbol{\omega}]$ as inputs, has 8 neurons in each hidden layer, and outputs $\boldsymbol{\Theta} = [p_{1:12}, \gamma_1, r_{1:6}, \gamma_2, q_{1:6}] \in \mathbb{R}^{26}$, resulting in a total of 362 network parameters.



(a) Mean loss in 10 trials.



(b) Overall training time.



(c) Ultimate loss.

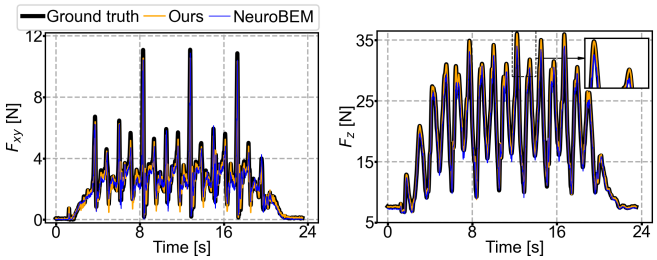
Fig. 2: Comparison of the training performance between the gradient descent (GD) and the trust-region (TR) methods. We randomly initialize the neural network in 10 trials with the Kaiming initialization, a common method for ReLU [26]. For the GD method, we set the learning rate to 1×10^{-4} by balancing between training stability and performance. We carefully select the trials of gradient descent such that the untrained mean loss closely matches that of the TR method, enabling fair comparisons. The ultimate loss shown in Fig. 2c corresponds to the loss value in the last episode.

For training, we select a 0.25-s-long trajectory segment from a Figure-8 flight collected at 400Hz (a total of 100 data points). One training episode amounts to training NeuroMHE over this 0.25-s-long dataset once. For testing, we adopt the same dataset as in [13] to compare NeuroMHE with NeuroBEM across 13 unseen agile trajectories. We implement our algorithm using CasADi [27] and conduct the simulations in a workstation with an Intel Core i7-11700K processor.

TABLE I: Runtime of Algorithm 1 for Different MHE Horizons

Horizon N	10	20	40	60	80
MHE Hessian [ms]	67.5	137.8	253.4	379.3	516.1
MHE Gradient [ms]	1.83	3.74	6.97	12.36	14.93

Before showing the estimation performance of trust-region NeuroMHE, we assess its training efficiency by comparing the CPU runtime of Algorithm 1 and the overall training time with gradient descent. Table I shows that both methods exhibit roughly linear computational complexity w.r.t the horizon, indicating scalability for large MHE problems. Computing the MHE Hessian takes longer than the MHE gradient due to the $\otimes$ operation, which expands the matrix dimension (See (12)). Despite this, the trust-region method significantly reduces the overall training time (under 5 min) by up to 63% (See Fig. 2b) while producing better training results (See Fig. 2c). Given the quadrotor's fast dynamics in agile flights, we set $N = 10$ in the subsequent tests.



(a) Estimation of F_{xy}.

(b) Estimation of F_z.

Fig. 3: Comparison of the force estimation performance between NeuroMHE and NeuroBEM on an aggressive Figure-8 flight test dataset as used in [13].

TABLE II: RMSE Comparisons on the Figure-8 Flight Test Dataset

Method	F_{xy} [N]	F_z [N]	τ_{xy} [Nm]	τ_z [Nm]	F [N]	τ [Nm]
NeuroBEM	0.51	1.08	0.03	0.01	1.20	0.03
Ours	**0.32**	**0.31**	**0.02**	0.01	**0.45**	**0.02**

Fig. 3 compares the performance of trust-region NeuroMHE with NeuroBEM in estimating force on an unseen flight test dataset. Our method maintains high accuracy, even in challenging force spikes where NeuroBEM experiences substantial performance degradation. Table II presents the related quantitative comparison using Root-Mean-Square errors (RMSEs). Our approach significantly outperforms NeuroBEM in the planar and vertical force estimations, reducing the RMSEs by 37.3% and 71.3%, respectively. Further comparisons on the entire test dataset[2] show that our method substantially improves the overall force estimation by up to 68.1% across most trajectories. Note that this superior generalizability is achieved by training a lightweight network with 362 parameters on just 0.25-s-long data. By comparison, NeuroBEM requires 3150-s-long flight data to train a high-capacity neural network with 25k parameters.

VI. Conclusion and Future Work

This paper proposed a trust-region policy optimization method for training NeuroMHE. Our critical insight is that most of the computation initially used to obtain the MHE gradient, especially the Kalman filter, can be efficiently reused for calculating the MHE Hessian in a recursive form. Through extensive simulations using real flight data, we have shown that our method provides highly-efficient training, accurate estimation with fast online adaptation to various challenging scenarios, and improved robustness to network initialization.

Our future work includes conducting a theoretical analysis on the stability of trust-region NeuroMHE, testing its efficacy in solving large scale estimation problems with various activation functions in the neural network, and simultaneously learning optimal controllers and estimators using second-order optimization techniques.

[2]The comparison results on the entire 13 unseen test flight trajectories are available at `https://github.com/BinghengNUS/TR-NeuroMHE`.

REFERENCES

[1] M. L. Corradini and G. Orlando, "Robust tracking control of mobile robots in the presence of uncertainties in the dynamical model," *Journal of robotic systems*, vol. 18, no. 6, pp. 317–323, 2001.

[2] C. Powers, D. Mellinger, A. Kushleyev, B. Kothmann, and V. Kumar, "Influence of aerodynamics and proximity effects in quadrotor flight," in *Experimental Robotics: The 13th International Symposium on Experimental Robotics*. Springer, 2013, pp. 289–302.

[3] E. Hashemi, M. Pirani, A. Khajepour, B. Fidan, A. Kasaiezadeh, S.-K. Chen, and B. Litkouhi, "Integrated estimation structure for the tire friction forces in ground vehicles," in *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE, 2016, pp. 1657–1662.

[4] Z. Chu, F. Wang, T. Lei, and C. Luo, "Path planning based on deep reinforcement learning for autonomous underwater vehicles under ocean current disturbance," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 108–120, 2022.

[5] P. Coelho and U. Nunes, "Path-following control of mobile robots in presence of uncertainties," *IEEE Transactions on Robotics*, vol. 21, no. 2, pp. 252–261, 2005.

[6] B. S. Park, S. J. Yoo, J. B. Park, and Y. H. Choi, "Adaptive neural sliding mode control of nonholonomic wheeled mobile robots with model uncertainty," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 1, pp. 207–214, 2008.

[7] B. Yüksel, C. Secchi, H. H. Bülthoff, and A. Franchi, "A nonlinear force observer for quadrotors and application to physical interactive tasks," in *2014 IEEE/ASME international conference on advanced intelligent mechatronics*. IEEE, 2014, pp. 433–440.

[8] T. Tomić and S. Haddadin, "A unified framework for external wrench estimation, interaction control and collision reflexes for flying robots," in *2014 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2014, pp. 4197–4204.

[9] J. Huang, S. Ri, T. Fukuda, and Y. Wang, "A disturbance observer based sliding mode control for a class of underactuated robotic system with mismatched uncertainties," *IEEE Transactions on Automatic Control*, vol. 64, no. 6, pp. 2480–2487, 2018.

[10] D. Hentzen, T. Stastny, R. Siegwart, and R. Brockers, "Disturbance estimation and rejection for high-precision multirotor position control," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 2797–2804.

[11] C. D. McKinnon and A. P. Schoellig, "Estimating and reacting to forces and torques resulting from common aerodynamic disturbances acting on quadrotors," *Robotics and Autonomous Systems*, vol. 123, p. 103314, 2020.

[12] X. Liu, C. Yang, Z. Chen, M. Wang, and C.-Y. Su, "Neuro-adaptive observer based control of flexible joint robot," *Neurocomputing*, vol. 275, pp. 73–82, 2018.

[13] L. Bauersfeld, E. Kaufmann, P. Foehn, S. Sun, and D. Scaramuzza, "Neurobem: Hybrid aerodynamic quadrotor model," *Proceedings of Robotics: Science and Systems XVII*, p. 42, 2021.

[14] G. Shi, W. Hönig, X. Shi, Y. Yue, and S.-J. Chung, "Neural-swarm2: Planning and control of heterogeneous multirotor swarms using learned interactions," *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 1063–1079, 2021.

[15] B. Wang, Z. Ma, S. Lai, and L. Zhao, "Neural moving horizon estimation for robust flight control," *IEEE Transactions on Robotics*, vol. 40, pp. 639–659, 2024.

[16] T. Kraus, H. J. Ferreau, E. Kayacan, H. Ramon, J. De Baerdemaeker, M. Diehl, and W. Saeys, "Moving horizon estimation and nonlinear model predictive control for autonomous agricultural vehicles," *Computers and electronics in agriculture*, vol. 98, pp. 25–33, 2013.

[17] B. Wang, Z. Ma, S. Lai, L. Zhao, and T. H. Lee, "Differentiable moving horizon estimation for robust flight control," in *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, 2021, pp. 3563–3568.

[18] A. Papadimitriou, H. Jafari, S. S. Mansouri, and G. Nikolakopoulos, "External force estimation and disturbance rejection for micro aerial vehicles," *Expert Systems with Applications*, vol. 200, p. 116883, 2022.

[19] D. G. Robertson, J. H. Lee, and J. B. Rawlings, "A moving horizon-based approach for least-squares estimation," *AIChE Journal*, vol. 42, no. 8, pp. 2209–2224, 1996.

[20] H. N. Esfahani, A. B. Kordabad, and S. Gros, "Reinforcement learning based on mpc/mhe for unmodeled and partially observable dynamics," in *2021 American Control Conference (ACC)*. IEEE, 2021, pp. 2121–2126.

[21] S. Muntwiler, K. P. Wabersich, and M. N. Zeilinger, "Learning-based moving horizon estimation through differentiable convex optimization layers," in *Learning for Dynamics and Control Conference*. PMLR, 2022, pp. 153–165.

[22] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[23] K. Cao and L. Xie, "Trust-region inverse reinforcement learning," *IEEE Transactions on Automatic Control*, 2023.

[24] A. Alessandri, M. Baglietto, G. Battistelli, and V. Zavala, "Advances in moving horizon estimation for nonlinear systems," in *49th IEEE Conference on Decision and Control (CDC)*. IEEE, 2010, pp. 5681–5688.

[25] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer, 1999.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[27] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "Casadi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.