Online Dynamics Learning for Predictive Control with an Application to Aerial Robots

Anonymous Author(s)
Affiliation
Address
email

Abstract: In this work, we consider the task of improving the accuracy of dynamic models for model predictive control (MPC) in an online setting. Even though prediction models can be learned and applied to model-based controllers, these models are often learned offline. In this offline setting, training data is first collected and a prediction model is learned through an elaborated training procedure. After the model is trained to a desired accuracy, it is then deployed in a model predictive controller. However, since the model is learned offline, it does not adapt to disturbances or model errors observed during deployment. To improve the adaptiveness of the model and the controller, we propose an online dynamics learning framework that continually improves the accuracy of the dynamic model during deployment. We adopt knowledge-based neural ordinary differential equations (KNODE) as the dynamic models, and use techniques inspired by transfer learning to continually improve the model accuracy. We demonstrate the efficacy of our framework with a quadrotor robot, and verify the framework in both simulations and physical experiments. Results show that the proposed approach is able to account for disturbances that are possibly time-varying, while maintaining good trajectory tracking performance.

Keywords: Online Learning, Model Learning, Model Predictive Control, Aerial Robotics

1 Introduction

In recent years, model predictive control (MPC) has shown significant potential in robotic applications. As an optimization-based approach that uses prediction models, the MPC framework leverages readily available physics models or accurate data-driven models to allow robotic systems to achieve good closed-loop performance for a variety of tasks. However, the reliance on accurate dynamic models makes it a challenging task for the controller to adapt to system changes or environmental uncertainties. In the case where the robot dynamics change during deployment or in the presence of disturbances, it becomes essential for the controller to update and adapt its dynamic model in order to maintain good performance. Recent advancements in deep learning techniques have demonstrated promising results in using neural networks to model dynamical systems. One advantage of neural networks over physics models is that they alleviate the need for bottom-up construction of dynamics, which often requires expert knowledge or physical intuition. Neural networks are also increasingly faster to optimize, owing to the development of optimization algorithms. This makes the application of neural networks to model predictive controllers more amenable, which allows the controllers to adapt to disturbances or system changes by refining its dynamic model. In this work, we propose a novel online dynamic learning algorithm using a deep learning method, knowledge-based neural ordinary differential equations. Through both simulations and physical experiments, we show that our framework, which comprises of a suite of algorithms, can improve the

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.

adaptiveness of MPC by continually refining its dynamic model of a quadrotor system, and in turn maintains desirable closed-loop performance during deployment.

Related Work. Traditional supervised machine learning paradigms consist of two phases – training and inference. The training phase requires data to be collected in advance and then takes place in an offline fashion. After the model is trained, it is then deployed for inference. Online learning, on the other hand, performs learning with data arriving in a sequential order. Notably, online learning has been used to transfer existing knowledge to assist training [1, 2], predict general time series data [3, 4], and learn inverse dynamics in a derivative free manner [5].

Learning dynamic models has gathered increasing attentions because of the development in scientific machine learning. A spectrum of techniques has emerged ranging from blackbox approaches [6, 7], to methods that require some known structures [8, 9]. In the middle of this spectrum, techniques have been developed to combine physics knowledge with machine learning [10, 11, 12, 13], and hysical priors have also been incorporated into neural networks [14]. A library based on continuous-time deep learning techniques have been developed for scientific machine learning [15].

There are a number of works that use models learned with data within an MPC framework. In [16, 17], Gaussian processes are used to characterize dynamics models for MPC and they are applied to autonomous vehicles and quadrotors. Data-driven models in a control framework is model-based reinforcement learning (MBRL). In [18], the authors use feed-forward networks to learn a dynamic model, which is then integrated into a stochastic predictive control framework. The authors in [19] use MBRL to design low-level controllers using sensors on-board a robot. In [20], the authors use a hybrid model that combines a first-principles model with a neural network. Our work is hugely inspired by [20], but instead of learning the model offline, we devise a framework that allow the dynamic model to be continually improved during deployment. The motivation for online dynamics learning is so that the model is able to account for uncertainty and disturbances that are not captured by the training data. To the authors' best knowledge, there has yet been online learning techniques specifically developed for learning dynamical systems and being applied to model predictive control schemes.

2 Problem Formulation

Given a robot that uses a predictive model-based controller, we aim to use data to continually improve its closed-loop performance in an online setting. Specifically, we seek to continually refine the dynamic model of the robot during deployment. Let the true model of a robot be given by

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}), \tag{1}$$

where the function f is the dynamics of the robot, the vector $\mathbf{x}$ is the state of the robot, and the vector $\mathbf{u}$ is the control input. The output of the dynamics is $\dot{\mathbf{x}}$, the derivative of the state with respect to time. Let the time stamps of a deployment of the robot for some tasks be $T = [t_0, t_1, \cdots, t_N]$. The training samples generated by the true dynamics in (1) during this deployment are $\mathcal{S} = [(\mathbf{x}(t_0), \mathbf{u}(t_0)), (\mathbf{x}(t_1), \mathbf{u}(t_1)), \cdots, (\mathbf{x}(t_N), \mathbf{u}(t_N))]$. We seek to estimate the robot dynamics given by

$$\dot{\mathbf{x}} = \hat{f}(\mathbf{x}, \mathbf{u}). \tag{2}$$

Given some performance metric on $\mathcal{S}$, we seek to improve it by refining $\hat{f}$ using the streams of data collected during deployment.

3 Online Dynamics Learning

3.1 Knowledge-based Neural Ordinary Differential Equations

Neural ordinary differential equations (NODE) was first proposed to approximate continuous-depth residual networks [6]. It has since been used in scientific machine learning to model a wide variety

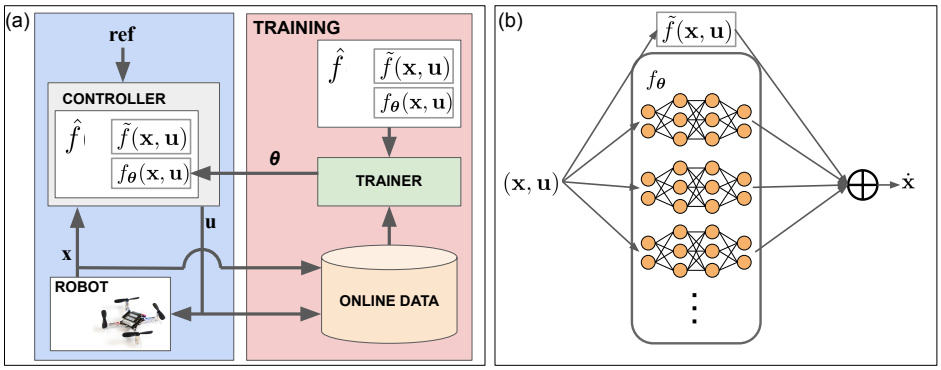

Figure 1: **Framework overview**. (a) The training process runs in parallel with the robot carrying out some tasks. The robot states and control inputs are collected for training. The trained model is used to update the dynamic model in the controller. (b) the neural network consists of a nominal model and repeatedly added neural networks. Each neural network and the nominal model run in parallel with each other, with their output coupled to give the state derivative.

of dynamical systems [11] Knowledge-based neural ordinary differential equations (KNODE) is an extension of NODE that couples physics knowledge with neural networks by leveraging its compatibility with first principles knowledge, and has demonstrated learning continuous-time models with improved generalizability. Three aspects of KNODE make it a suitable candidate for our online learning task. First, it requires less data for training, which means each data collection period during robot deployment can be short, and thus improving the adaptiveness of the MPC. Second, KNODE is a continuous-time dynamic model, which is compatible with many existing MPC frameworks. Third, many robotic systems have readily available physics models that can be used as knowledge. The original KNODE was applied only to non-controlled dynamical systems, but variants of it have been developed to incorporate control inputs to dynamic models [20]. In this work, we use a similar approach as [20], where we concatenate the state and control as $\mathbf{z} = [\mathbf{x}, \mathbf{u}]^T$. During training, the control is simply ignored when computing loss.

Using KNODE model, the dynamics in (2) is expressed as $\hat{f}(\mathbf{z}, t) = M_\psi(\tilde{f}(\mathbf{z}, t), f_\theta(\mathbf{z}, t))$, where $\tilde{f}$ is the physics knowledge, f_θ is a neural network parametrized by θ, and M_ψ is a selection matrix which couples the neural network with knowledge. A loss function is then given by

$$\mathcal{L}(\theta, \psi) = \frac{1}{m-1} \sum_{i=1}^{m-1} \int_{t_i}^{t_{i+1}} \delta(t_s - \tau) \|\hat{\mathbf{x}}(\tau) - \mathbf{x}(\tau)\|^2 d\tau + \mathcal{R}(\theta, \psi), \tag{3}$$

where δ is the Dirac delta function, $t_s \in T$ is any sampling time in T, and $\mathcal{R}$ is the regularization on the neural network and coupling matrix parameters. The estimated state $\hat{\mathbf{x}}(\tau)$ in (3) comes from $\hat{\mathbf{z}}(\tau)$, which is given by

$$\hat{\mathbf{z}}(\tau) = \mathbf{z}(t_i) + \int_{t_i}^{\tau} \hat{f}\left(\mathbf{z}(\omega), \omega\right) d\omega. \tag{4}$$

Intuitively, $\hat{\mathbf{z}}(\tau)$ is the state at $t = \tau$ generated using $\hat{f}$ with the initial condition $\mathbf{z}(t_i)$, and the loss function (3) computes the mean squared error between the one-step-ahead estimates of the states and the ground truth data. The integration in (4) is computed using numerical solvers in practice.

The optimization of the neural network parameters in KNODE can be done with either backpropagation or adjoint sensitivity method, which is a memory efficient alternative to backpropagation. In this work, adjoint sensitivity method is used similar to [6]. In the following sections, the dynamic models are all based on KNODE.

Algorithm 1 Data collection and model updates

1: Initialize the current time, total duration, and the collection interval as t_i, t_N, and t_{col}
2: $t_i \leftarrow 0$
3: OnlineData $\leftarrow$ []
4: **while** $t_i < t_N$ **do**
5: **if** New model is available **then**
6: Controller updates new model
7: **end if**
8: **if** t_i is not 0 and t_i modulo t_{col} is 0 **then**
9: Save OnlineData
10: OnlineData $\leftarrow$ []
11: **end if**
12: Robot updates state using control input
13: Append new robot state and control input to OnlineData
14: $t_i \leftarrow t_i + dt$
15: **end while**

Algorithm 2 Online dynamics learning

1: Initialize the current time and total duration as t_i and t_N
2: $t_i \leftarrow 0$
3: **while** $t_i < t_N$ **do**
4: **while** No new data available **do**
5: Wait
6: **end while**
7: Train a new model with the newest data ▷ new neural network added each iteration
8: Save the trained model
9: **end while**

3.2 Online Data Collection and Learning

The basic features of the proposed online dynamics learning algorithm include the dynamic model update logic and a trainer which runs in parallel with the robot, which are described in Alg. 1 and 2 respectively.

For data collection, the robot state is repeatedly appended to a data array during deployment. This data array is saved and reset (Line 10 and 11 of Alg. 1) at regular time intervals (Line 9 of Alg. 1). A key hyperparameter of data collection is the collection interval t_{col}, which dictates how much data in a batch (measured in seconds) to be collected for training. With a longer collection interval, the adaptiveness of the algorithm will decrease because it not only takes longer to collect data but also takes more time to train a new model. This leads to less frequent model updates for the controller, and therefore less adaptiveness to system/environment changes. With a shorter collection interval, while the adaptiveness improves, a model is more likely to overfit during training due to the smaller training data size.

When new data becomes available, a model is trained as described in Alg. 2. A challenge for online dynamic model learning is how to continually improve the model with newly collected data. When the model gets updated, the closed-loop trajectory of the controller changes accordingly and any new data collected will reflect the updated controller. As a result, the previous dynamic model needs to be preserved, and training the neural network already onboard will not serve the purpose as it alters the previous controller. To tackle this problem, we repeatedly add new neural networks in parallel with the existing model during robot deployment, as illustrated in Fig. 1 (b). Mathematically, given a nominal model $\tilde{f}$ as the first dynamic model estimate, it is recursively augmented by

$$\begin{aligned} \hat{f}^{(i+1)} &= M_{\psi_{(i+1)}}(\hat{f}^{(i)}, f_{\theta_{(i+1)}}), \\ \hat{f}^{(0)} &= \tilde{f}, \end{aligned} \tag{5}$$

where the index $(i+1)$ denotes the $(i+1)$th update to dynamic model. As mentioned earlier, the previous controller need to be preserved, and therefore only the parameters $\theta_{(i+1)}$ and $\psi_{(i+1)}$ are trained for the $(i+1)$th update, while other existing neural networks are frozen.

Also note that after each controller update, the next set of data for training must be entirely produced using the updated controller. If a set of data is collected using a mix of old and new controllers, it must be discarded so that the next set of training data represented the most up-to-date controller. This requirement is illustrated in Alg. 2 Line 4.

3.3 Applying Learned Models in MPC

Inspired by the framework in [20], we apply the learned dynamic model in an MPC framework. Specifically, we solve the following constrained optimization problem in a receding horizon manner,

$$\underset{\substack{x_0,\ldots,x_N,\\ u_0,\ldots,u_{N-1}}}{\text{minimize}} \quad \sum_{i=1}^{N-1} \left(x_i^T Q x_i + u_i^T R u_i\right) + x_N^T P x_N \tag{6a}$$

$$\text{subject to} \quad x_{i+1} = f(x_i, u_i), \quad \forall\, i = 0, \ldots, N-1, \tag{6b}$$

$$x_0 = x(t), \tag{6c}$$

$$x_i \in \mathcal{X}, \quad u_i \in \mathcal{U}, \quad \forall\, i = 0, \ldots, N-1, \tag{6d}$$

$$x_N \in \mathcal{X}_f, \tag{6e}$$

where x_i, u_i are the predicted states and control inputs, N is the horizon and $\mathcal{X}$, $\mathcal{U}$, $\mathcal{X}_f$ are the state, control input and terminal state constraint sets. $f(\cdot,\cdot)$ in (6b) is a discretized version of the learned KNODE model, as described in Section 3.1. This model is used to predict the future states within the horizon. A precise model gives more accurate predictions of the states, which in turn provides more effective control actions. The weighting matrices Q and R are designed to penalize the states and control inputs within the cost function and P is the terminal state cost matrix. $x(t)$ is the state obtained at time step t, which acts as an input to the optimization problem (6). Upon solving (6), the first element in the optimal control sequence u_0^* is applied to the robot as the control action. The robot moves according to this control action and generate new state measurements $x(t+1)$, which will be used to solve (6) at the next time step.

4 Simulations

To demonstrate the efficacy of the framework in real-world robotic applications, we apply it to a quadrotor system and conduct tests in both simulations and physical experiments.

4.1 Dynamics of a Quadrotor System

To apply the KNODE-MPC-Online framework, we first construct a KNODE model, by combining a nominal model derived from physics, with a neural network. For the quadrotor, the nominal model can be derived from its equations of motion,

$$m\ddot{\mathbf{r}} = m\mathbf{g} + \mathbf{R}\eta, \tag{7a}$$

$$\mathbf{I}\dot{\boldsymbol{\omega}} = \boldsymbol{\tau} - \boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega}, \tag{7b}$$

where $\mathbf{r}$ and $\boldsymbol{\omega}$ are the position and angular rates of the quadrotor, η, $\boldsymbol{\tau}$ are the thrust and moments generated by the motors of the quadrotor. $\mathbf{g}$ is the gravity vector and $\mathbf{R}$ is the transformation matrix that maps η to the accelerations. m and $\mathbf{I}$ are the mass and inertia matrix of the quadrotor. Furthermore, by defining the state as $\mathbf{x} := [\mathbf{r}^\top\ \dot{\mathbf{r}}^\top\ \mathbf{q}^\top\ \boldsymbol{\omega}^\top]^\top$ and control input as $\mathbf{u} := [\eta\ \boldsymbol{\tau}^\top]^\top$, where $\mathbf{q}$ denotes the quaternions representing its orientation, the KNODE model for the quadrotor system can then be expressed as

$$\dot{\mathbf{x}} = \hat{f}(\mathbf{x}, \mathbf{u}) := \tilde{f}(\mathbf{x}, \mathbf{u}) + f_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{u}), \tag{8}$$

where $f(\cdot,\cdot)$ is the nominal model obtained from (7) and $f_\theta(\cdot,\cdot)$ is the neural network.

4.2 Simulation Setup

A simulation environment for the quadrotor system is implemented based on the equations of motion given in (7). An explicit Runge-Kutta method (RK45) with a sampling time of 2 milliseconds is used for numerical integration to simulate dynamic responses of the quadrotor. The predictive controller described in Section 3.3 is assumed to have full measurements of the dynamics of the quadrotor. The optimization problem (6) is implemented and solved using CasADi [21], and its solution provides control commands to the quadrotor model to generate the closed-loop responses. In these simulations, we consider circular trajectories, with a range of target speeds of 0.8, 1 and 1.2 m/s and radii of 2, 3 and 4m. Each simulation run lasts 8 seconds. To test the adaptiveness of our framework, we consider two time instances during the simulation where we change the mass of the quadrotor mid-flight. At time = 2 seconds, the mass of the quadrotor is reduced by 50% and at time = 5 seconds, it is increased to 133% of the original mass.

Benchmarks. We verify the performance of our proposed framework, KNODE-MPC-Online, by comparing against two benchmarks. First, we consider a standard nonlinear model predictive control (MPC) framework, where we use a discretized version of the nominal model $\tilde{f}(\mathbf{x}, \mathbf{u})$ as the prediction model in (6b). Comparing against this benchmark provides insights on the role of the neural network, in the presence of unknown residual dynamics. As a second benchmark, we consider the approach taken in [20], where the KNODE model (8) is learned offline. This approach consists of two phases; data is first collected using a nominal controller and a KNODE model is trained using this collected data. The KNODE model is then deployed as the prediction model in (6b). We denote this approach as KNODE-MPC. By comparing our approach to KNODE-MPC, we show the effectiveness of online learning against residual dynamics and uncertainty that are possibly time-varying. In particular, to highlight the adaptive and generalization abilities of our approach, for the KNODE-MPC approach, we collect training data such that it only accounts for the first mass change at time = 2 seconds, but not the second mass change at time = 5 seconds.

4.3 Simulation Results

Figure 2 gives a comparison of our framework, KNODE-MPC-Online, against the benchmarks, MPC and KNODE-MPC. The plotted overall mean squared errors (MSE) are calculated by considering the difference between the time histories of the reference trajectory and the quadrotor position for each simulation run, and along each axis. They are calculated element-wise and indicate the overall trajectory tracking performance for each run. As shown in the color plots, KNODE-MPC-Online provides the best overall performance across different target speeds and radii. Notably, by considering the median across the runs, KNODE-MPC-Online outperforms MPC and KNODE-MPC by 25.5% and 39.5% respectively. KNODE-MPC-Online performs well as it is able to account for the mass changes that occur during mid-flight. On the other hand, KNODE-MPC, with the KNODE model trained offline, is only able to account for the first mass change, but it is unable to account for the second change, as the effects of the second mass change was not observed from its training data. Figure 3 illustrates the trajectories taken by the quadrotor when the three frameworks are deployed. The quadrotor is required to track a circular reference trajectory of radius 3m, shown in red. Evidently, from both the top and side views, the KNODE-MPC-Online framework allows the quadrotor to track the reference trajectory more closely, as compared to the two frameworks.

5 Physical Experiments

5.1 Experimental Setup

The open-source Crazyflie 2.1 quadrotor [22] is used as the platform for our physical experiments. The Crazyflie has a mass of 0.032kg and has a size of 9 cm^2. A computer running on Intel i7 CPU is used as the base station and communication with the Crazyflie is established with the Crazyradio PA and at a nominal rate of 400 Hz. The software architecture is implemented using the CrazyROS position [23]. Position measurements of the quadrotor are obtained with a VICON motion capture

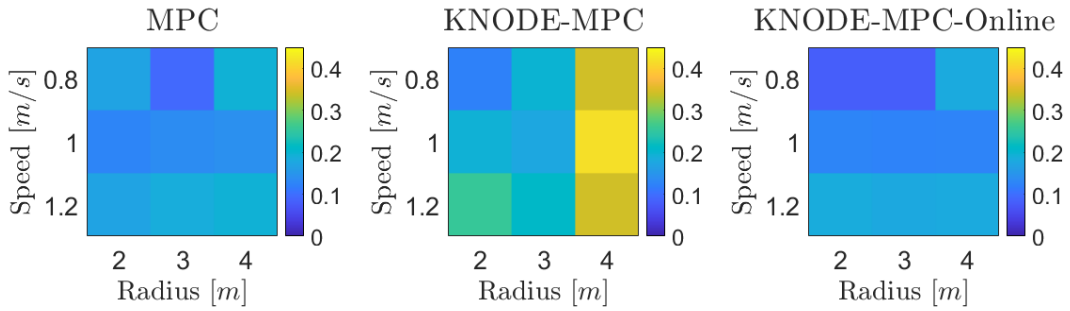

Figure 2: **Performance of KNODE-MPC-Online in simulations.** Color maps for the overall trajectory tracking mean squared errors (MSE) for the two benchmarks, MPC and KNODE-MPC, as well as our approach, KNODE-MPC-Online, are plotted.

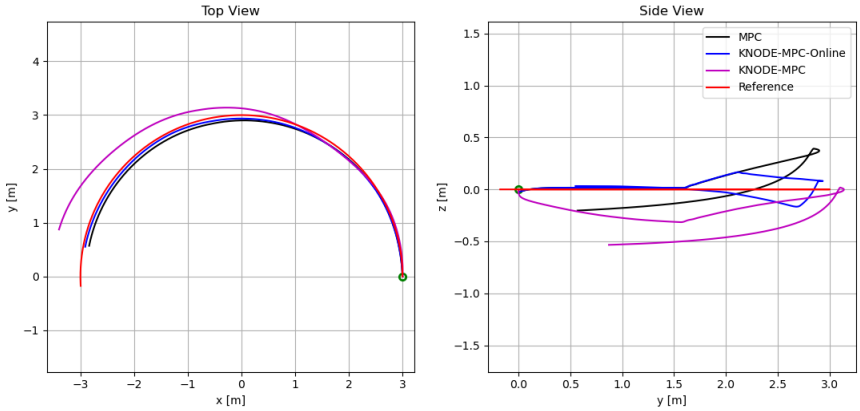

Figure 3: **Trajectory plots** for MPC, KNODE-MPC and KNODE-MPC-Online (ours), from both top and side views. The reference trajectory is shown in red and the quadrotor position is initialized at $(x, y, z) = (3, 0, 0)$.

system which communicates with the base station. Linear velocities are estimated from the positions obtained from VICON, while accelerations and angular velocities are measured from the on-board accelerometers and gyroscope sensors. For these experiments, the Crazyflie is commanded at a target speed of 0.4m/s and to track a circular trajectory of radius 0.5m. Each run has a duration of at least 40 seconds. Similar to the mass changes done in simulations, we test the adaptiveness of the proposed framework by attaching an object of mass 0.003kg (10% of the mass of Crazyflie) mid-flight. We compare our approach with the two benchmarks described in Section 4.2. For the KNODE-MPC approach, training data is collected with the object attached, so that its effects can be accounted for in the model that is trained offline.

5.2 Results and Discussion

Results from the physical experiments are illustrated in Figure 4. The MSE shown is defined in the same way, as described in Section 4.3. They are computed with 30 seconds of flight data, approximately starting from the time when the object is attached. As shown in the figure, our proposed approach, KNODE-MPC-Online, outperforms both benchmarks in terms of the overall MSE. In particular, the overall MSE for KNODE-MPC-Online improves by 44.1% as compared to nominal MPC and by 12.6%, compared to KNODE-MPC. Furthermore, it is observed that results of KNODE-MPC-Online have a smaller variance as compared to the benchmarks. This implies that the approach is consistent in terms of control performance across all runs. Since the nominal MPC framework is unable to account for the mass perturbation mid-flight, the MSEs are observed to be

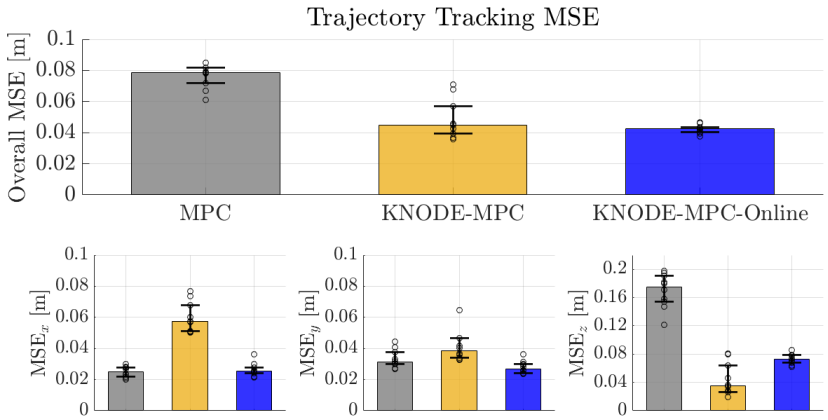

Figure 4: **Performance of KNODE-MPC-Online in physical experiments.** Statistics for trajectory tracking mean squared errors (MSE) for the two benchmarks, nominal MPC, KNODE-MPC and our approach, KNODE-MPC-Online, are shown. The top of the bars denote the median, while the ends of the error bars represent the 25th and 75th percentiles. The top plot depicts the overall MSE and the bottom three plots shows the MSEs in separate axes.

generally larger, especially in the z axis. More notably, from the MSEs for each of the axes, we observe that the offline KNODE-MPC approach has larger errors in the x and y axes and smaller errors in the z axis. This suggests that the KNODE model trained offline is able to compensate for the mass perturbation induced in flight, since its effect is present in the training data. However, it is unable to compensate for errors in the x-y plane, which are likely not reflected in the training data. In contrast, the KNODE-MPC-Online approach achieves more consistent MSE in all three axes and it is also to adapt to uncertainty during flight.

6 Conclusion

In this work, we propose a novel and sample-efficient framework, KNODE-MPC-Online, that learns the dynamics of a quadrotor robot in an online setting. We then apply the learned KNODE model in an MPC scheme and adaptively update the dynamic model during deployment. Results from simulations and real-world experiments show that the proposed framework allows the quadrotor to adapt and compensate for uncertainty and disturbances during flight and improves the closed-loop trajectory tracking performance. Future work includes applying this framework to other robotic applications where dynamic models can be learned to achieve enhanced control performance.

7 Limitations

A fundamental assumption of our framework is the continuous-time nature of system dynamics. This means our framework has limited applicability to stochastic systems. However, there has been variants of NODE which models stochastic differential equations. For future work, we hope to extend our algorithm to incorporate stochastic models to improve its applicability.

Another limitation with the proposed framework is its increasing computational load as the model gets updated. As more neural networks get added to the KNODE model, the latency increases. For future work, we plan to perform network pruning and reduced order modeling on the learnt dynamic models, in order to keep the computation overhead constant.

In our physical experiments, there has been occasional model updates failures due to reading and writing data using ROS1 input-output (I/O) stream. In the future, we may implement the algorithm without ROS to avoid such failure cases.

References

[1] P. Zhao, S. C. Hoi, J. Wang, and B. Li. Online transfer learning. *Artificial Intelligence*, 216:76–102, 2014. ISSN 0004-3702. doi:https://doi.org/10.1016/j.artint.2014.06.003. URL https://www.sciencedirect.com/science/article/pii/S0004370214000800.

[2] S. C. Hoi, D. Sahoo, J. Lu, and P. Zhao. Online learning: A comprehensive survey. *Neurocomputing*, 459:249–289, 2021. ISSN 0925-2312. doi:https://doi.org/10.1016/j.neucom.2021.04.112. URL https://www.sciencedirect.com/science/article/pii/S0925231221006706.

[3] O. Anava, E. Hazan, S. Mannor, and O. Shamir. Online learning for time series prediction, 2013. URL https://arxiv.org/abs/1302.6927.

[4] V. Kuznetsov and M. Mohri. Time series prediction and online learning. In V. Feldman, A. Rakhlin, and O. Shamir, editors, *29th Annual Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, pages 1190–1213, Columbia University, New York, New York, USA, 23–26 Jun 2016. PMLR. URL https://proceedings.mlr.press/v49/kuznetsov16.html.

[5] D. Romeres, M. Zorzi, R. Camoriano, S. Traversaro, and A. Chiuso. Derivative-free online learning of inverse dynamics models. *IEEE Transactions on Control Systems Technology*, 28 (3):816–830, 2020. doi:10.1109/TCST.2019.2891222.

[6] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Inf. Process. Syst.*, pages 6571–6583. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7892-neural-ordinary-differential-equations.pdf.

[7] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott. Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach. *Phys. Rev. Lett.*, 120: 024102, Jan 2018. doi:10.1103/PhysRevLett.120.024102. URL https://link.aps.org/doi/10.1103/PhysRevLett.120.024102.

[8] S. Brunton, J. Proctor, and J. Kutz. Discovering governing equations from data: Sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113:3932–3937, 09 2015. doi:10.1073/pnas.1517384113.

[9] A. A. AlMomani, J. Sun, and E. Bollt. How entropic regression beats the outliers problem in nonlinear system identification. *Chaos*, 30:013107, 01 2020.

[10] A. Wikner, J. Pathak, B. Hunt, M. Girvan, T. Arcomano, I. Szunyogh, A. Pomerance, and E. Ott. Combining machine learning with knowledge-based modeling for scalable forecasting and subgrid-scale closure of large, complex, spatiotemporal systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 30:053111, 05 2020. doi:10.1063/5.0005541.

[11] T. Z. Jiahao, M. A. Hsieh, and E. Forgoston. Knowledge-based learning of nonlinear dynamics and chaos. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31(11):111101, 2021. doi:10.1063/5.0065617.

[12] S. Greydanus, M. Dzamba, and J. Yosinski. Hamiltonian neural networks, 2019. URL https://arxiv.org/abs/1906.01563.

[13] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho. Lagrangian neural networks, 2020. URL https://arxiv.org/abs/2003.04630.

[14] M. Finzi, G. Benton, and A. G. Wilson. Residual pathway priors for soft equivariance constraints. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors,

Advances in Neural Information Processing Systems, volume 34, pages 30037–30049. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/fc394e9935fbd62c8aedc372464e1965-Paper.pdf.

[15] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, and A. Edelman. Universal differential equations for scientific machine learning, 2020. URL https://arxiv.org/abs/2001.04385.

[16] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger. Learning-based model predictive control for autonomous racing. 4(4):3363–3370, 2019.

[17] G. Torrente, E. Kaufmann, P. Föhn, and D. Scaramuzza. Data-driven mpc for quadrotors. 6 (2):3769–3776, 2021.

[18] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE Int. Conf. on Robot. and Automat.*, pages 1714–1721. IEEE, 2017.

[19] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. J. Pister. Low-level control of a quadrotor with deep model-based reinforcement learning. 4(4):4224–4230, 2019. doi:10.1109/LRA.2019.2930489.

[20] K. Y. Chee, T. Z. Jiahao, and M. A. Hsieh. Knode-mpc: A knowledge-based data-driven predictive control framework for aerial robots. *IEEE Robotics and Automation Letters*, 7(2): 2819–2826, 2022. doi:10.1109/LRA.2022.3144787.

[21] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Math. Program. Computation*, 11 (1):1–36, 2019.

[22] Bitcraze. Crazyflie 2.1. URL https://www.bitcraze.io/products/crazyflie-2-1/.

[23] W. Hönig and N. Ayanian. *Flying Multiple UAVs Using ROS*, pages 83–118. Springer Int. Publishing, 2017. ISBN 978-3-319-54927-9. doi:10.1007/978-3-319-54927-9_3.