2024 7th International Conference on Soft Robotics (RoboSoft)
San Diego, CA, USA. April 14-17, 2024

MoLDy: Open-source Library for Data-based Modeling and Nonlinear Model Predictive Control of Soft Robots

Daniel G. Cheney and Marc D. Killpack

Abstract— **Aggressive and accurate control of complex dynamical systems, such as soft robots, is especially challenging due to the difficulty of obtaining an accurate and tractable model for real-time control. Learned dynamic models are incredibly useful because they do not require derivation of an analytical model, they can represent complex, nonlinear behavior directly from data, and they can be evaluated quickly on graphics-processing units (GPUs). In this paper, we present an open-source Python library to further current research in model-based control of soft robot systems. Our library for Modeling of Learned Dynamics (MoLDy), is designed to generate learned forward models of complex systems through a data-driven approach to hyperparameter optimization and learned model training. Included in the MoLDy library, we present an open-source version of NEMPC (Nonlinear Evolutionary Model Predictive Control), a previously published control algorithm validated on soft robots. We demonstrate the ability of MoLDy and NEMPC to accurately perform model-based control on a physical pneumatic continuum joint. We also present a benchmarking study on the effect of the loss metric used in model training on control performance. The results of this paper serve to guide other researchers in creating learned dynamic models of novel systems and using them in closed-loop control tasks.**

I. Introduction

Robotic systems are capable of completing a large variety of unique tasks. However, as the complexity of tasks a robot is designed to complete increases, the dynamics of the robotic system may also become more complex. Especially in the case of soft robotics, dynamic and accurate control is difficult due to inherent compliance, underactuation, and nonlinear fluid dynamics for pneumatically actuated systems.

Model-free control [1] is one solution to dynamic control of soft robots, where, given a simulated or physical system, a controller learns to output control commands that achieve a desired task outcome. However, once trained, model-free controllers generally struggle to be applied to novel tasks. In contrast, model-based control is a control approach in which a controller uses a dynamic or kinematic model, learned or analytical, to generate control commands. Model-based control has several advantages, including a reduced controller search space and the ability to extend the underlying model to novel control tasks.

Deriving an exact analytical model quickly becomes time intensive and error prone, and requires ongoing system identification after the model is derived to accurately reflect changes in the system hardware. In the case of the pneumatic link shown in Fig. 1, multiple links can be combined to make full manipulators, creating significant complexity due to more diverse flow dynamics and configuration dependent inertial effects. With heavy use, components in the soft robot, such as the plastic chambers, degrade, further changing the true dynamics when compared to a previously derived model.

Fig. 1. Graphic of the continuum joint used in this paper, referred to as the "grub". The grub has four pneumatically-actuated plastic chambers that control its motion. Angle position and velocity states are estimated using two HTC Vive trackers.

As an alternative to derived models, learned dynamic models based on neural networks are capable of accurately forward propagating system dynamics with high accuracy. Deep neural networks (DNNs) are effective at representing nonlinear relationships, including unknown system dynamics, rendering them highly effective for modeling soft robot dynamics. Despite the utilization of learned dynamic models in model-based control, past work has required additional compensation measures in control, such as an integrator [2] or a residual model [3] because the learned model alone was inadequate. Additionally, selecting correct learned model parameters, called hyperparameters, is difficult and time consuming, yet critical for accurate models [4].

This work seeks to address the challenge of model-based control with a learned dynamic model for a variety of soft robot platforms, without relying on analytical equations or

Both authors are with the Robotics and Dynamics Laboratory at Brigham Young University

This work was supported by the National Science Foundation under Grant No. 1935312

2024 IEEE 7th International Conference on Soft Robotics (RoboSoft) | 979-8-3503-8181-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/ROBOSOFT60065.2024.10522031

Authorized licensed use limited to: Brigham Young University. Downloaded on December 26,2024 at 17:02:29 UTC from IEEE Xplore. Restrictions apply.

prior knowledge about system dynamics. Our intent is that this approach will be particularly beneficial for researchers in the field of soft robotics through the following contributions:

1) The development and open-source release of MoLDy (**Mo**deling of **L**earned **Dy**namics), a Python library providing a framework to optimize hyperparameters and train dynamic models. Found here: `https://bit.ly/moldy_control`.
2) Open-source release of the Nonlinear Evolutionary Model Predictive Control (NEMPC) algorithm [2], [5] that uses learned models directly in a sampling-based approach on a Graphics Processing Unit (GPU).
3) A case study using MoLDy and NEMPC for model-based control of a simulation and hardware soft pneumatic continuum link (see Fig. 1). Real-time hardware control is compared against a traditional PID controller.

We next outline the remainder of this paper. Section II explores previous work similar to ours in the field of learning for modeling and control of soft robots. In section III-A, we present MoLDy, the open-source library used to create learned dynamic models. We then give a brief overview of the NEMPC control algorithm released within the MoLDy library (Section III-B). Section IV presents the results of using MoLDy and NEMPC for model-based control on the simulation and hardware platforms. We conclude in Section V with a brief summary of the results and contributions made in this paper.

II. Related Work

The related literature presented in this section focuses on machine learning methods applied to soft robotics, as there is an extensive body of work in machine learning for modeling and control of soft robots [6]. Additionally, learned dynamic models have been used in many other fields including fluid dynamics [7] and traditional robotics [8]. In contrast to these other fields, soft robots provide an effective platform to test the accuracy of learned models due to their inherent uncertainty and the difficulty in deriving tractable dynamic models, especially for hardware platforms. We organize our review of the existing literature into three areas: model-free control, model-based control, and current modeling toolboxes.

A. Model-Free Control

As previously discussed, model-free control, where a control policy is learned directly, does not require a model of the system dynamics or kinematics. Reinforcement Learning (RL), a common model-free approach, enables an agent to learn control policies through interactions with the environment, as demonstrated in a soft robotic throwing task [9]. A common application of RL in soft robotics is solving the Inverse Kinematics (IK) problem [1], [10]. Non-RL learned controllers have also been used to learn an IK controller using data gathered from an analytical model [11], [12]. Although model-free controllers can achieve good performance, they are often not generalizable to other control tasks.

B. Model-Based Control

While model-based control can be performed with an analytical or learned model, in this review we focus on model-based control with learned models. Controllers are often learned [13] or optimization-based, such as Model Predictive Control (MPC) [14]. Model-based control applications include stiffness and position control of soft actuators [15], open-loop control using a learned forward kinematics model for a soft manipulator [16], and joint-level control of a large-scale pneumatically actuated soft robot arm [17]. Typically, a model is trained on simulation data generated from an analytical model of the system. However, when applied to physical systems this approach can result in steady-state error during control, primarily due to model error from unaccounted effects present in hardware.

To solve this challenge, previous work has implemented solutions such as using an integrator in control [2], [14] or a simulation model combined with a residual model, which predicts the error between the simulation model and actual hardware dynamics [3]. Another solution is to use a data-driven approach, which relies solely on data from the hardware platform for model training [18], avoiding a simulation model entirely.

C. Modeling Toolboxes

Notable toolboxes like AutoMPC [19] facilitate System ID and MPC tuning, while DeepTime [20] focuses on creating dynamic models from time-series data without offering control functionality. The SOFA toolbox is specifically designed for soft robotics, incorporating modeling and control based on Finite Element Method (FEM) solvers [21]. While these libraries offer a wide range of functionality, no library offers modeling and control methods that resolve the challenges faced for dynamic control of soft robotics, including cases where analytical and computer models are intractable. Furthermore, Finite Element Analysis (FEA) models may fall short in accurately capturing the dynamics of soft robot hardware to enable model-based control. This can be due to unique material properties and platform-specific characteristics, which cannot be fully captured using simulation tools.

III. Methods

A. MoLDy: Open-Source Dynamic Modeling Library

The MoLDy Python library is provided as an open-source toolbox for other researchers to create and evaluate learned dynamic models, especially for use in closed-loop control. To best address the needs described to this point, the MoLDy library adheres to the following design requirements:

- It must be able to generate models without the need for an analytical or computer-based model, enabling modeling and control for complex soft robots.
- It must provide methods to analyze results and compare control performance across different models.
- It must have a simple user interface and generated models must be easily exported for use with other Deep Learning (DL) libraries.

Authorized licensed use limited to: Brigham Young University. Downloaded on December 26,2024 at 17:02:29 UTC from IEEE Xplore. Restrictions apply.

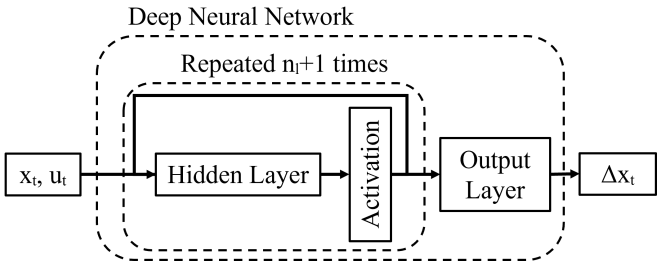

Fig. 2. Neural network architecture design implemented and used in MoLDy, where there is one input layer and n_l is the number of hidden layers.

MoLDy provides the user with several Python classes that can be inherited in order to model a novel system. Additional information on using MoLDy for a novel system can be found in the repository's ReadMe file. MoLDy operates through three key stages: data gathering, hyperparameter optimization, and model training. For novel systems, users begin by collecting a dataset, followed by optimizing neural network hyperparameters based on the specific system characteristics, ultimately concluding with model training using the optimized hyperparameters.

Training data can be generated from an analytical equation, a simulation, or collected from a physical robot. In the simulation environment for this paper, random state/command pairs are generated and forward propagated by one time step using fourth-order Runge-Kutta numerical integration on the analytical model. For hardware systems, data can be collected as per the current system configuration, such as through the use of the Robot Operating System (ROS), and then converted into the required format for model training. The default form for neural network training and validation data is given by:

$$\begin{aligned} x &= [x_t, u_t] \\ y &= \Delta x_t \end{aligned} \tag{1}$$

where x_t represents a vector of system states, u_t is a vector of command inputs and x and y denote the input and output data, respectively, utilized to train the neural network. The state at the next time step (x_{t+1}) is defined as:

$$x_{t+1} = x_t + \Delta x_t. \tag{2}$$

MoLDy provides users with options to normalize data by the infinity norm, normalize to a standard deviation of 1 and a mean of 0, or use raw values. Additionally, the user can specify whether the output data should be in the form of Δx_t or x_{t+1}.

To perform hyperparameter optimization, we use the Ray Tune [22] and Optuna [23] Python libraries. The implementation on top of these libraries is simple, enabling a quick start to generating learned models, while allowing the user to access advanced functionality as required. By default each hyperparameter optimization run uses the validation loss reported from model training as the optimization objective. However, the optimization objective is easily modified to track other metrics. The default optimization algorithm in MoLDy is the Tree Structured Parzen Estimator (TPE), a Bayesian-based optimization technique. There are a wide variety of optimization methods available in Optuna, including grid search, random search, and other advanced algorithms. Early stopping is implemented with the Asynchronous Successive Halving Algorithm (ASHA) Scheduler to terminate under performing trials, reducing computation time. The optimization is population-based and can be executed for any specified number of samples, with each sample representing a learned model trained on a unique set of optimizer-selected hyperparameters. For each sample, a model is trained until the maximum number of training epochs is reached, or until the ASHA scheduler terminates the training. The full optimization results along with models, hyperparameters, and training results for each sample are saved locally. With MoLDy, many hyperparameters can be optimized, including the number of hidden layers, number of hidden nodes per layer, activation function(s), weights optimizer, learning rate, learning rate scheduler, and network weight initialization scheme.

Model training is implemented using PyTorch Lightning [24]. Despite many different architectures that have been proven to be effective in control of learned dynamic models, our experiments demonstrated successful real-time control with a simple Feedforward Neural Network (FNN). The specific architecture is contingent upon the provided set of hyperparameters, whether obtained through hyperparameter optimization or user input. Nevertheless, the general architecture is shown in Fig. 2. Learned models generated with MoLDy take as input the system state at a discrete time step, x_t, along with the corresponding command input, u_t, at the same time step. While a user can choose to have models output the full state of the system, we find that choosing Δx_t as the network output reduces the learning burden on the neural network. Models are saved as ".ckpt" files in PyTorch and are easily exported to other Deep Learning libraries such as TensorFlow, Keras, or Caffe.

B. NEMPC Open-Source Controller

We also present the Nonlinear Evolutionary Model Predictive Control (NEMPC) algorithm as a Python library to make control validation of learned dynamic models easier for other researchers. The version we release was originally published in [2], [5] and later modified in [3], with a few minor changes noted here. NEMPC is a sampling-based controller that leverages a GPU to forecast control trajectories in parallel at each time step. The controller then applies the first step of the lowest-cost trajectory, where the cost is summed over the control horizon of length n. A significant advantage of NEMPC is its capability to use nonlinear models in control, presenting a valuable resource for soft robotics, which often are modeled as nonlinear analytical or learned models. NEMPC also has the advantage of defining flexible cost functions, allowing for nonlinearity, non-convexity, and other variations within the cost function. In this work we use a quadratic cost on the state and command inputs:

Authorized licensed use limited to: Brigham Young University. Downloaded on December 26,2024 at 17:02:29 UTC from IEEE Xplore. Restrictions apply.

$$J = \sum_{i=0}^{n} (\tilde{x}_i^T Q \tilde{x}_i + \tilde{u}_i^T R \tilde{u}_i) \tag{3}$$

with

$$\begin{aligned} \tilde{x_i} &= x_i - x_{i,desired} \\ \tilde{u_i} &= u_i - u_{i,previous} \end{aligned} \tag{4}$$

where J is a scalar number corresponding to the total cost of a specific trajectory, Q is a matrix of weights on the error in system state, and R is a matrix of weights on change in command inputs. The subscript i denotes the time step in the control horizon for each corresponding variable. The state trajectory of the system is represented by x_i, while the desired state of the system at time step i is represented by $x_{i,desired}$. Current command inputs are represented by u_i and the previous input is denoted as $u_{i,previous}$.

Given a control trajectory, we use Integrated Absolute Error (IAE) as a metric for evaluating control performance:

$$\text{IAE} = \sum_{i=0}^{n} |x_i - x_{i,desired}| \tag{5}$$

We choose IAE due to its effectiveness in capturing both the transient error response, as well as the cumulative effect of steady state error. For systems with multiple states of interest, a total IAE measure is obtained by summing the IAE from each relevant state.

IV. Case Study

In this section, we present a case study using MoLDy to model and NEMPC to control a single link continuum robot (see Fig. 1). We show results in simulation and on real hardware. Additionally, we compare the effect of several popular loss metrics for learned models on control performance as an example of MoLDy's capability to test the effects of important parameters related to using learned models for control. We first provide details on the hardware platform, discuss experiment setup, and then present results of modeling dynamics with neural networks and performing joint-level control for both the simulation and hardware systems.

A. Platform Description

The grub is a single-link continuum joint actuated by four pressure valves, connected to blow-molded plastic chambers. The plastic chambers surround an inextensible Kevlar rope that maintains an approximately fixed distance between the end plates. The physical robot stands approximately 30 cm tall and is 19 cm in diameter. We define the state of the robot in the same manner as in [3], denoted as $x = [p, \dot{q}, q]^T$. Here p represents a vector containing the four chamber pressure states, $\dot{q} = [\dot{\theta}, \dot{\phi}]$ and $q = [\theta, \phi]$, where θ and ϕ correspond to decoupled bending angles (see Fig. 3). The control inputs are defined as $u = [p_{cmd}]^T$ where p_{cmd} is an array of four pressure commands.

A pressure differential between p_0 and p_1, as seen in Fig. 4, causes a net torque and bending about the x-axis. When $p_0 > p_1$, a positive θ rotation about the x-axis is induced.

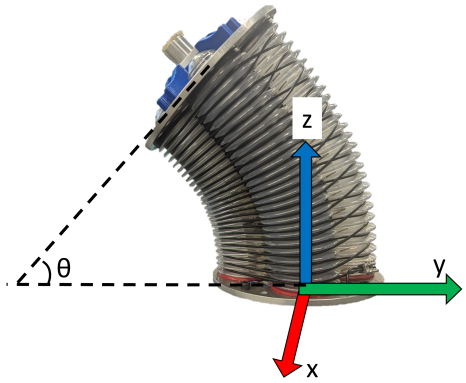

Fig. 3. Rotation conventions on the grub, where the RGB axes correspond to XYZ directions. A positive θ rotation is about the x-axis while a positive ϕ rotation corresponds to a positive rotation about the y-axis.

Similarly, differential pressures between p_2 and p_3 cause a ϕ rotation about the y-axis. Each plastic chamber is rated for pressures up to 400 kPa, but we limit the pressure to 275 kPa for safety reasons, which is sufficient to quickly reach joint limits. The simulation model is based on analytical equations derived in [25], using experimentally determined values for parameters such as damping and stiffness. On the hardware system, we use HTC Vive trackers to estimate the bending angles.

Although the link only has two degrees of freedom, there are significant challenges when modeling such a robot. Nonlinear pressure dynamics arise due to the valves filling and venting the plastic chambers. Additionally, stiffness and damping parameters inherent in the material are difficult to correctly measure. Difficulty in accurately controlling the joint in this paper compared to previous studies [3], [25], comes from recent design changes to reduce parasitic torque and allow more available torque for each joint, which resulted in reduced overall damping and inertia.

B. Experiment Setup

To compare the effect of each loss metric we complete two sets of model training for both the simulation and hardware experiments. For the first set, each model is trained with hyperparameters optimized independently for each loss

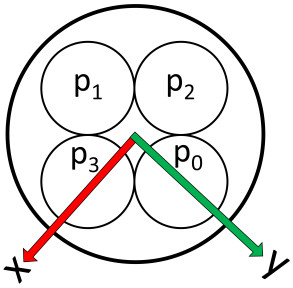

Fig. 4. Cross-section of a grub link with antagonistic pressure chambers causing a net torque.

Authorized licensed use limited to: Brigham Young University. Downloaded on December 26,2024 at 17:02:29 UTC from IEEE Xplore. Restrictions apply.

metric. In contrast, the second set of models is trained using a uniform set of hyperparameters across all loss metrics. This approach ensures that we can distinguish between the effect of the hyperparameters and the effect of the loss metric on control.

For the first set, we performed a hyperparameter optimization with 300 samples per loss metric. The optimized hyperparameters and their ranges are listed in Table I. We limited the number of hidden layers and hidden nodes to ensure models could evaluate fast enough for control. Using these optimized hyperparameters, we trained 20 models for each loss metric to account for the stochastic nature of initialization, data shuffling, and optimizer steps. For the second set, we trained 20 additional models for each loss metric with a uniform set of hyperparameters selected from those obtained during the initial optimization. The dataset is split, with 80% used for training and 20% for validation. To prevent exploding gradients during the training process, input and output data were normalized using the infinity norm. The models were trained for 500 epochs. Model weights were initialized using the Xavier uniform initialization, and all biases were initially set to zero. We used the "ReduceLROnPlateau" learning rate scheduler and the Adam optimizer from the PyTorch library. For each model, the weights corresponding to the epoch with the lowest validation loss were saved and used in subsequent control validation.

After training, we evaluate the control performance of each model. Models are controlled using NEMPC with hand tuned parameters that are kept constant across control trials to ensure fair results between models. Performance is reported as the IAE over a commanded joint angle trajectory that is the same for all trials on both the simulation and hardware systems. On the hardware platform we compare our learned model-based NEMPC approach to a simple differential Proportional-Integral-Derivative (PID) controller. The PID controller calculates pressure differentials from the error between the commanded joint angle and the actual joint angle. Actual pressure commands are calculated from the output of the differential PID controller, u_{diff} by:

$$\begin{aligned} p_0 &= \frac{p_{max} - p_{min}}{2} + u_{diff} \\ p_1 &= \frac{p_{max} - p_{min}}{2} - u_{diff} \end{aligned} \tag{6}$$

The gains of the differential controller were tuned to minimize the IAE control error over the commanded trajectory.

TABLE I

HYPERPARAMETER RANGES FOR MODEL OPTIMIZATION.

Hyperparameter	Possible Values
Hidden Layers	0, 1, 2, 3
Hidden Nodes	16, 32, 64, 128, 256, 512, 1024
Learning Rate	Between 1e-5 and 1e-3
Activation Function	ReLU, TanH, Sigmoid, Leaky ReLU

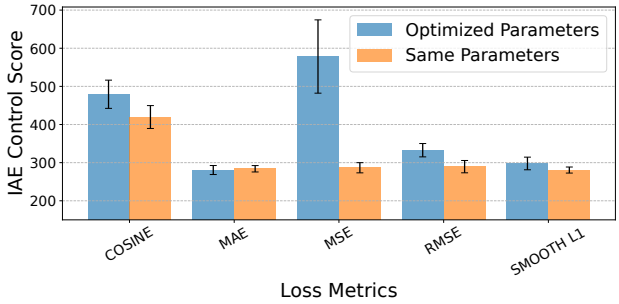

Fig. 5. Average IAE control measure for each of the loss metrics across 20 models are shown for trials performed in the simulation environment.

TABLE II

GRUB SIMULATION OPTIMIZED HYPERPARAMETERS WHERE H. IS ABBREVIATED FOR "HIDDEN", ACT FCN IS "ACTIVATION FUNCTION", AND LR IS THE "LEARNING RATE".

Metric	H. Layers	H. Nodes	Act Fcn	lr
Same	**1**	**1024**	**ReLU**	**9.9e-4**
Cosine	2	1024	ReLU	9.6e-4
MAE	1	1024	ReLU	9.9e-4
MSE	3	256	Tanh	2.9e-4
RMSE	2	1024	ReLU	3.2e-4
Smooth L1	1	1024	ReLU	6.7e-4

C. Experiment Results

1) Grub Simulation: For the grub simulation, model training was conducted with a batch size of 512 on 150,000 random data points generated at 100 Hz. The optimized hyperparameters are shown in Table II. Control was performed in simulation at a rate of 100 Hz with a control horizon of 0.5 seconds. The IAE control performance of models from each loss metric is presented in Fig. 5. Control results on the simulation system with a step reference trajectory are shown in Fig. 6. Generally, the control performance is consistent across loss metrics, with the exception of the optimized MSE models and all Cosine models. We hypothesize that the MSE hyperparameter optimization converged to a local minimum, causing poor control performance due to the selected network architecture being unable to represent the simulated dynamics.

2) Grub Hardware: To collect data on the hardware platform, a command of four random pressures was applied to the system for a random delay between 1 and 5 seconds. This process was repeated until we collected 180,000 data points at 50 Hz, which took approximately one hour. Models are trained with a batch size of 1024 and to predict Δx_t for 0.02 seconds into the future, in order to achieve a longer horizon during real-time control. 40 total models (20 with optimized hyperparameters and 20 with uniform hyperparameters) were trained for each loss metric as before, but only five from each loss metric were selected to be run in hardware control trials based on each model's open-loop prediction performance. We do not include a comparison of optimized and constant hyperparameters for the hardware system because,

Authorized licensed use limited to: Brigham Young University. Downloaded on December 26,2024 at 17:02:29 UTC from IEEE Xplore. Restrictions apply.

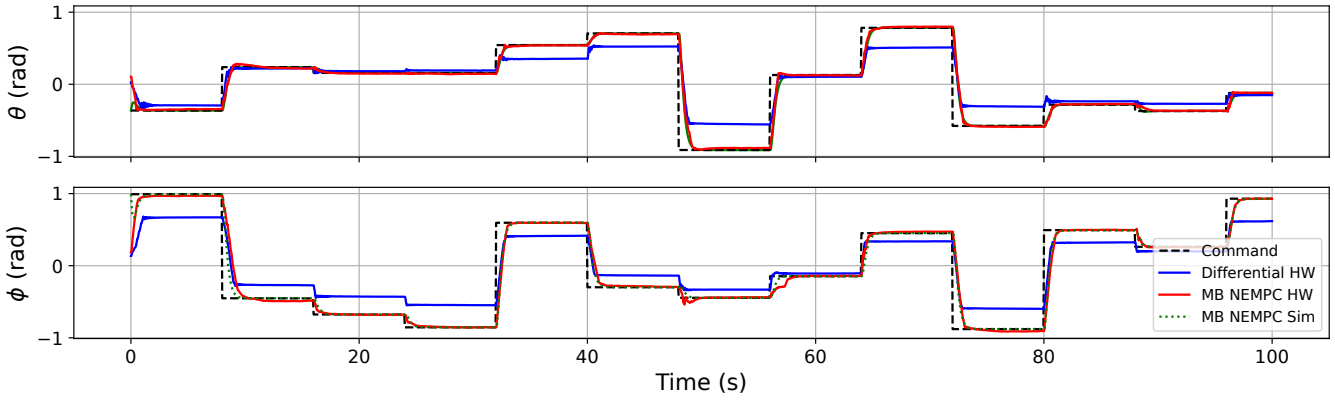

Fig. 6. Subset of the control trajectory used in experiments to test the Model-based NEMPC (MB NEMPC) and differential PID controllers.

for all loss metrics, the optimized hyperparameters showed little variation. Fig. 8 illustrates the open-loop prediction performance of the learned hardware model. Given only the initial state and the trajectory of pressure commands, the hardware model is able to accurately predict the full state of the robot for over 25 seconds. Control was performed in real-time at a rate of 25 Hz with a 1.2 second horizon. Additionally, we apply a first-order filter to the command outputs from NEMPC to reduce control signal noise on the hardware task. We report control results for each loss metric in Fig. 7. The trajectory tracking results for model-based NEMPC and the differential PID controller are compared in Fig. 6. The hardware models also have the ability to track a variety of reference inputs such as sine and ramp waves (Fig. 9). For a video showing the robot performing a portion of the control task as well as an overview of the MoLDy library, see `https://bit.ly/moldy_video`.

The hardware results are similar to the simulation results, including the Cosine metric performing the worst. The higher error in hardware results over simulation results is likely due to various real-world factors, such as sensor noise, non-uniformity of the robot materials, friction, and pressure dynamics. These factors make generalizing a model to the hardware data more difficult. Additionally, we had to use NEMPC parameters that allowed for real-time control on hardware, sacrificing performance for speed. The comparison between the differential PID and model-based NEMPC controllers in Fig. 6 demonstrates the advantages of model-based control, particularly for soft robots. The average control IAE for the differential PID controller was 6905.49 radians, while the average for the Smooth L1 loss models on hardware was 514.86 radians. Even after substantial tuning of the differential controller, incorporating integral and derivative gains, there is persistent steady-state error, highlighting the limitations of simple PID control on a complex system, necessitating a model-based approach as presented in this paper.

V. Conclusion

As shown in the case study, MoLDy can be used to effectively generate a learned dynamic model of complex soft

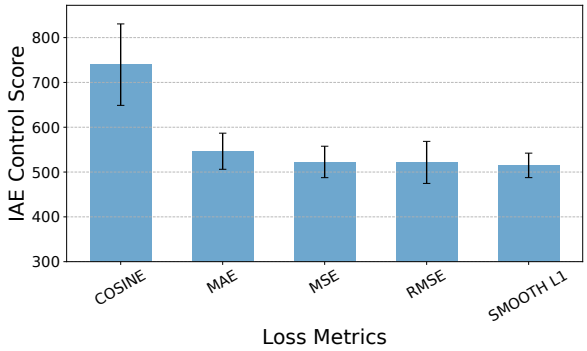

Fig. 7. Average IAE control measure for each of the loss metrics across five models are shown for trials performed on the hardware platform.

robots. Interestingly, we found that the loss metric used in model training had a minimal effect on control performance compared to other hyperparameters, excluding Cosine loss. We conclude that any of the other high-performing loss metrics we compared can be used to successfully create learned dynamic models. This process also demonstrates the ability of the MoLDy toolbox to perform benchmarking studies to explore the effects of varying hyperparameters. The optimization performed by MoLDy was helpful to obtain a set of hyperparameters that performed well across loss metrics. The models produced by MoLDy have good dynamic performance and low steady state error when used in control with NEMPC, without requiring an integrator or pre-training on simulation data.

The versatility of MoLDy is highlighted by its capability to model and control novel systems, such as the soft link, without the need for prior knowledge beyond data collection. The MoLDy toolbox could be used to create learned models of other soft systems, such as sensors or actuators, with an input/output relationship that can be measured empirically.

In this work, we have presented a new library for creating effective learned models for use with model-based control. Additionally, we have released a previously published control algorithm (NEMPC) that can directly use these learned models. These resources will enable other researchers in soft

Authorized licensed use limited to: Brigham Young University. Downloaded on December 26,2024 at 17:02:29 UTC from IEEE Xplore. Restrictions apply.

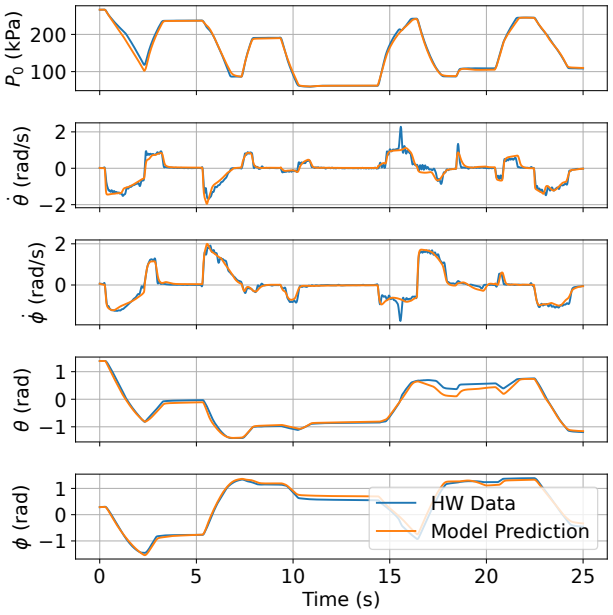

Fig. 8. Open-loop prediction performance for the grub hardware. Only two pressure states are included as the performance was similar for all four pressure states.

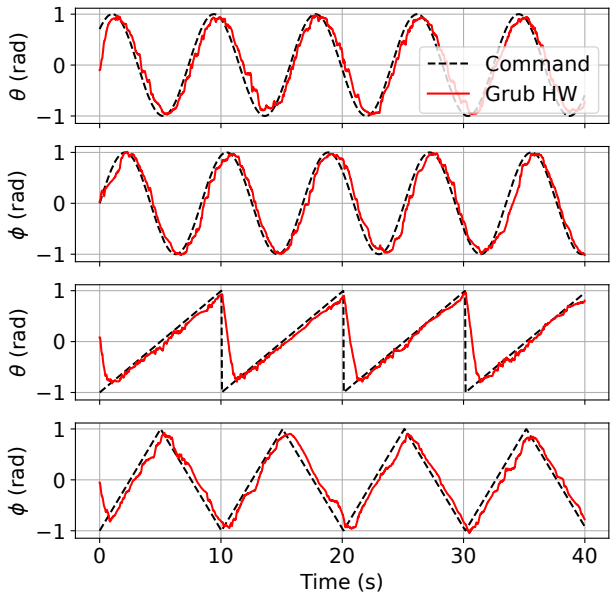

Fig. 9. Grub control performance for sine and ramp command inputs on the hardware platform.

robotics to perform accurate, model-based control of various soft robot platforms without the need to derive analytical equations or hand tune model parameters.

References

[1] H. Zhang, R. Cao, S. Zilberstein, F. Wu, and X. Chen, "Toward effective soft robot control via reinforcement learning," in *International Conference on Intelligent Robotics and Applications*, 2017.

[2] P. Hyatt and M. D. Killpack, "Real-time nonlinear model predictive control of robots using a graphics processing unit," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1468–1475, 2020.

[3] C. C. Johnson, T. Quackenbush, T. Sorensen, D. Wingate, and M. D. Killpack, "Using first principles for deep learning and model-based control of soft robots," *Frontiers in Robotics and AI*, vol. 8, 2021.

[4] J. M. Bern, Y. Schnider, P. Banzet, N. Kumar, and S. Coros, "Soft robot control with a learned differentiable model," in *2020 3rd IEEE International Conference on Soft Robotics (RoboSoft)*, 2020.

[5] P. Hyatt, C. S. Williams, and M. D. Killpack, "Parameterized and GPU-parallelized real-time model predictive control for high degree of freedom robots," 2020. [Online]. Available: https://arxiv.org/abs/2001.04931

[6] D. Kim, S.-H. Kim, T. Kim, B. B. Kang, M. Lee, W. Park, S. Ku, D. Kim, J. Kwon, H. Lee, J. Bae, Y.-L. Park, K.-J. Cho, and S. Jo, "Review of machine learning methods in soft robotics," *PLOS ONE*, vol. 16, no. 2, pp. 1–24, 02 2021.

[7] N. Geneva and N. Zabaras, "Transformers for modeling physical systems," *Neural Networks*, vol. 146, pp. 272–289, feb 2022.

[8] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive Processing*, vol. 12, no. 4, pp. 319–340, Nov 2011.

[9] D. Bianchi, M. Antonelli, C. Laschi, and E. Falotico, "Open-loop control of a soft arm in throwing tasks," in *Proceedings of the 19th International Conference on Informatics in Control, Automation and Robotics - Volume 1: ICINCO,*, INSTICC. SciTePress, 2022.

[10] X. You, Y. Zhang, X. Chen, X. Liu, Z. Wang, H. Jiang, and X. Chen, "Model-free control for soft manipulators based on reinforcement learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 2909–2915.

[11] A. Alkhodary and B. Gur, "Kinematics transformer: Solving the inverse modeling problem of soft robots using transformers," 2022.

[12] A. Centurelli, A. Rizzo, S. Tolu, and E. Falotico, "Open-loop model-free dynamic control of a soft manipulator for tracking tasks," in *2021 20th International Conference on Advanced Robotics (ICAR)*, 2021.

[13] T. G. Thuruthel, E. Falotico, F. Renda, and C. Laschi, "Model-based reinforcemefnt learning for closed-loop dynamic control of soft robotic manipulators," *IEEE Transactions on Robotics*, vol. 35, 2019.

[14] M. T. Gillespie, C. M. Best, E. C. Townsend, D. Wingate, and M. D. Killpack, "Learning nonlinear dynamic models of soft robots for model predictive control with neural networks," in *2018 IEEE International Conference on Soft Robotics (RoboSoft)*, 2018.

[15] T. Luong, K. Kim, S. Seo, J. Jeon, C. Park, M. Doh, J. C. Koo, H. R. Choi, and H. Moon, "Long short term memory model based position-stiffness control of antagonistically driven twisted-coiled polymer actuators using model predictive control," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 4141–4148, 2021.

[16] T. G. Thuruthel, E. Falotico, F. Renda, and C. Laschi, "Learning dynamic models for open loop predictive control of soft robotic manipulators," *Bioinspiration & Biomimetics*, vol. 12, 2017.

[17] P. Hyatt, D. Wingate, and M. D. Killpack, "Model-based control of soft actuators using learned non-linear discrete-time models," *Frontiers in Robotics and AI*, vol. 6, 2019.

[18] T. G. Thuruthel, B. Shih, C. Laschi, and M. T. Tolley, "Soft robot perception using embedded soft sensors and recurrent neural networks," *Science Robotics*, vol. 4, no. 26, 2019.

[19] W. J. J. Edwards, G. Tang, G. Mamakoukas, T. D. Murphey, and K. K. Hauser, "Automatic tuning for data-driven model predictive control," *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7379–7385, 2021.

[20] M. Hoffmann, M. K. Scherer, T. Hempel, A. Mardt, B. de Silva, B. E. Husic, S. Klus, H. Wu, J. N. Kutz, S. Brunton, and F. Noé, "Deeptime: a python library for machine learning dynamical models from time series data," *Machine Learning: Science and Technology*, 2021.

[21] F. Largilliere, V. Verona, E. Coevoet, M. Sanz Lopez, J. Dequidt, and C. Duriez, "Real-time Control of Soft-Robots using Asynchronous Finite Element Modeling," in *ICRA 2015*, May 2015.

[22] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," *arXiv preprint arXiv:1807.05118*, 2018.

[23] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," 2019.

[24] W. Falcon and The PyTorch Lightning team, "PyTorch Lightning," Mar. 2019. [Online]. Available: https://github.com/Lightning-AI/lightning

[25] P. Hyatt, C. C. Johnson, and M. D. Killpack, "Model reference predictive adaptive control for large-scale soft robots," *Frontiers in Robotics and AI*, vol. 7, 2020.

Authorized licensed use limited to: Brigham Young University. Downloaded on December 26,2024 at 17:02:29 UTC from IEEE Xplore. Restrictions apply.