Executive Summary of the Thesis

Improved Learning of Automata Models for Cyber-Physical Systems

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Simone Leone

Advisor: Prof. Matteo Giovanni Rossi

Co-advisor: Dr. Livia Lestingi

Academic year: 2023-2024

1. Introduction

As technology advances, Cyber-Physical Systems (CPS) are increasingly significant in fields like manufacturing, transportation, healthcare, and energy. CPS integrates computational algorithms with physical processes, involving interactions between digital components (e.g., sensors, and actuators) and physical elements (e.g., machinery, and vehicles). A major challenge in CPS is accurately modeling and controlling their often nonlinear and complex behaviors. Automata learning, especially through methods like L^*_{SHA}, constructs models of CPS by synthesizing Stochastic Hybrid Automata (SHA) using predefined Ordinary Differential Equations (ODEs). While this approach improves modeling, it is constrained by its reliance on fixed ODEs, which can limit its application to unknown systems. To overcome this limitation, the thesis introduces Sparse Identification of Nonlinear Dynamics (SINDy), which derives parsimonious and interpretable models from observational data without needing prior knowledge of governing equations. Integrating SINDy with L^*_{SHA} provides a robust framework for managing CPS complexity. Lastly, the thesis highlights the need for clean data to ensure accurate modeling. It suggests future work on combining SINDy with advanced machine learning techniques to enhance model accuracy and scalability further.

1.1. Objectives

The goal of this thesis is to enhance CPS modeling by incorporating SINDy in L^*_{SHA}, removing the reliance on predefined ODEs and improving adaptability to unknown system dynamics. The PySINDy library is employed to apply SINDy to real-world CPS data. This approach enables the learning mechanism to evolve autonomously, refining models from observational data. Different case studies will validate the proposed methodology, highlighting its effectiveness in known and less-understood systems. The potential for further refinement using machine learning techniques like Artificial Neural Networks (ANNs) and Reinforcement Learning (RL) is discussed to tackle complex, noisy data scenarios.

1.2. Foundations of SHA learning and SINDy

SHA models combine continuous and discrete dynamics with probabilistic behavior, making them particularly suitable for modeling CPS with inherent randomness and uncertainty. These automata consist of discrete states, con-

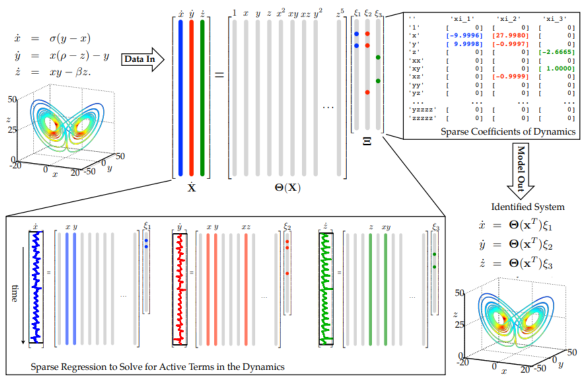
Data In
Model Out
Sparse Coefficients of Dynamics
Identified System
Sparse Regression to Solve for Active Terms in the Dynamics
time

Figure 1: Visual representation of SINDY [4].

tinuous dynamics governed by differential equations, and transitions between states. In this context, the L^*_{SHA} algorithm, designed to learn the SHA that models a CPS, enhances the traditional L^* learning framework by iteratively refining the model through interactions with the environment and analysis of observed data [2]. A crucial advancement in modeling nonlinear systems is the SINDy method, which discovers the governing differential equations of a system directly from data through sparse regression [3]. The PySINDy library, a Python implementation of SINDy, further enhances this process by offering a suite of tools for model discovery. PySINDy simplifies the application of SINDy by providing functionalities for numerical differentiation, feature library construction, and sparse regression optimization, making it a powerful resource for identifying the dynamics of complex systems from data [4]. In Figure 1 there is a visual representation of the sparse regression process where a three-dimensional system is learned.

2. Model Training and Integration

This section outlines the process of training PySINDy models and integrating them into the L^*_{SHA} algorithm. The goal is to identify system dynamics through differential equations and ensure these models are robust and accurate for use in verification tasks. In Figure 2 there is a graphical representation of the workflow followed and can help provide an overall view during the reading.

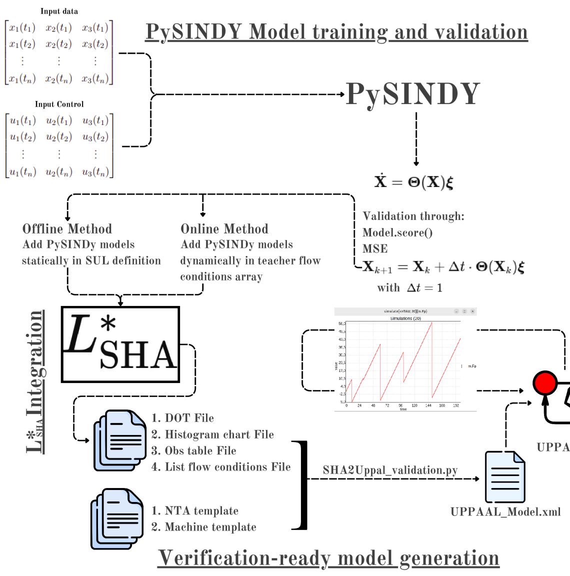
Input data
Input Control
PySINDY Model training and validation
PySINDY
$\dot{\mathbf{X}} = \mathbf{\Theta}(\mathbf{X})\boldsymbol{\xi}$
Offline Method
Add PySINDy models statically in SUL definition
Online Method
Add PySINDy models dynamically in teacher flow conditions array
Validation through:
Model.score()
MSE
$\mathbf{X}_{k+1} = \mathbf{X}_k + \Delta t \cdot \mathbf{\Theta}(\mathbf{X}_k)\boldsymbol{\xi}$
with $\Delta t = 1$
L^*_{SHA}
L^*_{SHA} Integration
1. DOT File
2. Histogram chart File
3. Obs table File
4. List flow conditions File
1. NTA template
2. Machine template
SHA2Uppal_validation.py
UPPAAL_Model.xml
UPPAAL
Verification-ready model generation

Figure 2: Workflow visualization.

2.1. Training PySINDy Models

The SINDy method, implemented through PySINDy, involves data preparation, model construction, function optimization, and validation. The data should be formatted as a matrix where each row corresponds to an observation (a time step) and each column represents a variable (a feature). Here is an example matrix:

$$\texttt{data} = \begin{bmatrix} x_1(t_1) & x_2(t_1) & x_3(t_1) \\ x_1(t_2) & x_2(t_2) & x_3(t_2) \\ \vdots & \vdots & \vdots \\ x_1(t_n) & x_2(t_n) & x_3(t_n) \end{bmatrix}$$

Data must then be preprocessed, including normalization and splitting. The model is constructed using a library of standard candidate functions, and sparse regression techniques select significant terms. The most important hyperparameters are optimized using grid search. PySINDy typically finds ODEs that describe the continuous evolution of a system over time. However, to validate the model and compare it with actual data, these continuous equations need to be discretized. This operation is necessary also because, in the next phase, we will integrate PySINDy into the entire learning process. The ODEs found by PySINDy must form the set of flow conditions from which identification must be made: a representation that allows the calculation of new values at each time instant is required. The Euler method is then used for discretizing the continuous differential equa-

tions [4]. Alternatively, it is possible to train the PySINDy model in discrete time by setting the *discrete_time* parameter.
Below is an example of training a PySINDy model with two state variables x_1,x_2 and there are different methods of training depending on the presence or absence of control input:

```python
# Assume X is the matrix of state
    variables and t is the time vector
# Define the feature library and
    optimization method
feature_library = ps.PolynomialLibrary(
    degree=2)
optimizer = ps.STLSQ(threshold=0.1)
# Initialize the PySINDy model
model = ps.SINDy(feature_library=
    feature_library, optimizer=optimizer
    , discrete_time = "True")
# Fit the model to the data
model.fit(X, t=t)
# If input control is present, U is the
    control input matrix
model.fit(X, t=t, u=U)
```

2.2. Integration with L^*_{SHA}

Integrating PySINDy models into the L^*_{SHA} algorithm requires training separate models for each state of the system's state space to ensure comprehensive coverage. For example, distinct models should be trained for low-energy and high-energy states. The flow conditions from these models populate the flow array of the System Under Learning (SUL), which the L^*_{SHA} algorithm uses to work effectively with these models. During learning rounds, proximity methods such as Fast Derivative Dynamic Time Warping (Fast DDTW) are used to compare models and select the most accurate one based on the alignment of derivatives with observed data. This involves querying the teacher-learner framework to refine the model selection, leading to the generation of an SHA that provides a structured representation of the system's dynamics. There are two methods for populating the flow conditions array: the *offline* method and the *online* one. In the *offline* method PySINDy models are trained before the learning process begins. The trained models provide a static set of flow conditions that are directly integrated into the SUL. This method requires pre-labeled data and domain expertise for accurate training. In the *online* method PySINDy is incorporated directly into the identification phase between the teacher and learner. This method automates the identification of flow conditions, reducing the need for pre-labeled data, and allows for the dynamic determination of governing equations during the learning process. While initially suboptimal, models improve over time through iterative training. The SUL definition includes an *enable flag* to switch between the offline and online methods. If that flag is set to *true*, then the learning process starts with an empty flow conditions array (online method). At each learning step, in the teacher's definition, the process fits one of the pre-configured PySINDy models, which are designed to be as general as possible. Therefore, no code changes are required; you simply need to provide the data to the L*SHA algorithm as usual. If the flag is set to *false*, the implementation requires training the PySINDy models (one for each state describing the system's behavior) on the data formatted as before (offline method). Subsequently, these models need to be added to the flow conditions array in the teacher definition.
The choice of method depends on the system's complexity and the available data. The offline method is suitable for systems where there is some prior knowledge, such as known possible states of the system. In contrast, the online method is advantageous for systems where only data is available without prior knowledge or understanding of the system's origin. Each method has unique benefits and limitations that impact the model discovery process.

2.3. Verification ready model generation

The goal is to generate models that can be utilized in verification tools like UPPAAL for the analysis of SHA. The process starts with converting the learned automata, represented by a DOT file, into a SHA model, which is then refined into an UPPAAL-compatible model. A key step involves adapting templates for Network Timed Automata (NTA), tailored to the specific case study. The *SHA to UPPAAL* function manages these adaptations by modifying the template based on the model's requirements and available data. Additionally, a standard SHA machine template for each case study is automatically compiled with the information representing the SHA under analysis and is dynamically added to the NTA. The conversion process begins with the DOT file and incorporates additional outputs like histogram charts, observation tables, and flow condition models. A custom script performs this transformation in two stages: converting DOT to SHA, and then SHA to an UPPAAL model, ensuring accurate representation for simulation and verification. Finally, the UPPAAL model is generated by integrating learned distributions, flow conditions, and test traces into the NTA template, resulting in a model ready for simulation and analysis.

3. Application of the approaching practices

In this section, we delve into the practical application and validation of the methods discussed in Section 2. The implementation was tested and validated across various simulated and real-world case studies. Selected case studies are:

1. **Thermostat and Human-Robot Interaction (HRI)** – These are categorized as *known* systems, where the underlying dynamics are well-understood.
2. **auto-twin Pizza Line** – This is classified as a *semi-known* system, indicating that there is partial knowledge of the system dynamics.
3. **Energy MADE** – This is considered an *unknown* system, with no prior information about the governing equations.

While the Thermostat and auto-twin Pizza Line case studies were used to implement and test the effectiveness of the identification method, the HRI and Energy MADE cases were fully explored, with complete integration and validation of the method.

3.1. Known system: Thermostat

This section investigates a known CPS involving a room with a thermostat, where temperature dynamics are governed by various operational states such as window status and heating system activity. The system's behavior is modeled using ODEs with random parameters assumed to follow normal distributions. In this case study, PySINDy was employed to uncover the mathematical relationship between the operational states of the windows and the temperature dynamics within the room. The final model demonstrated a linear relationship between the rate of temperature change and the current temperature as described by Equation 1. It accurately reflects the system's behavior as shown in Figure 3, with a Model Score[1] of 0.99.

$$T[k+1] = T[k](1 - 0.010) + 0.695 \qquad (1)$$

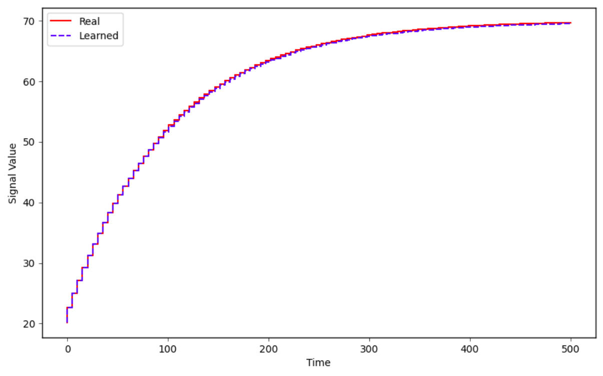

Figure 3: Thermostat final PySINDy model prediction.

This model was further validated through simulations in the UPPAAL environment, where it was incorporated as state invariants within the system model. The results showed in Figure 4 demonstrate that the equations derived by PySINDy provided a reliable approximation of the system's dynamics, reinforcing the effectiveness of PySINDy in uncovering and understanding internal system dynamics.

3.2. Known System: HRI

The HRI system focuses on estimating fatigue levels of individuals across different activities, from walking to working. Two states of heart rate are distinguished: active engagement (the busy model) and restful periods (the idle model). This dual-state approach helps in understanding the relationship between activity levels and fatigue. To model the busy state, a PySINDy model was trained using a dataset generated from predefined initial conditions and system states. The PySINDy approach identified the governing ODE for this system, as described by Equation 2. The resulting model showed high predictive accuracy with a Model Score of 1.00 and produced effective predictions as depicted in

[1]The `model.score()` function computes R^2, indicating the model's fit; R^2 close to 1 implies a good fit, close to 0 implies a poor fit.

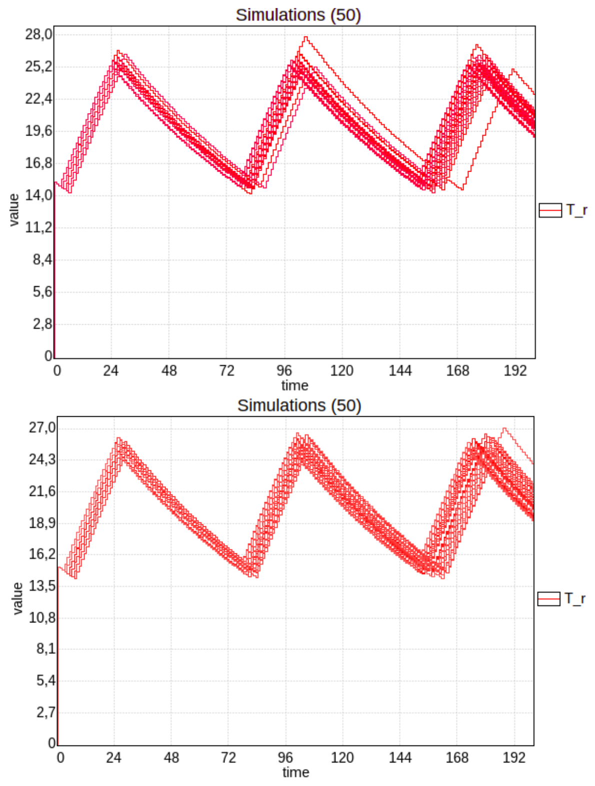

Figure 4: Thermostat Original (up) and Thermostat PySINDy (down).

Figure 5.

$$B[k+1] = B[k](1 - 0.004) + 0.004 \quad (2)$$

Validation in UPPAAL confirmed the reliability of the PySINDy model by comparing simulations of the HRI system dynamics with the original system. The results demonstrate that the model accurately replicates the system behavior, equivalent to what was observed in the Thermostat case study. An application of the second approach (i.e. integration of PySINDy into the learning process) was also explored, showing that the online learning method can converge to models similar to offline methods, but with some suboptimal performance. Finally, the model was converted into a UPPAAL model and validated against experimental data. The UPPAAL model accurately predicted the system's behavior, affirming the reliability of the PySINDy approach as shown in Figure 6.

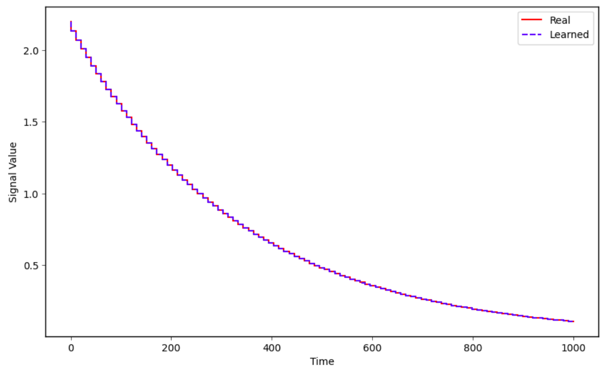

Figure 5: HRI final PySINDy model prediction.

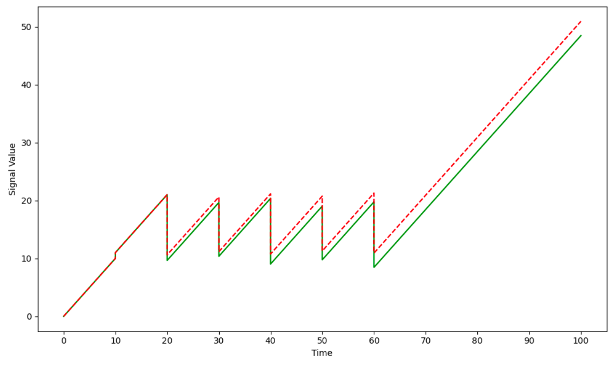

Figure 6: HRI UPPAAL validation.

3.3. Semi-known system: auto-twin Pizza Line

The auto-twin Pizza Line[2] is a simulation model replicating pizza production and packaging processes. It includes various stages, such as baking, freezing, packing, boxing, palletizing, storage, market distribution, and box refurbishing, reflecting real-world manufacturing. The production plan outlines the schedule and specifications for pizzas, including dimensions, weight, and packaging. Item flow is managed by sources and sinks, with decision points for quality control. The objective of the oven component responsible for baking pizzas was to predict power dissipation based on the pizza volume and the ambient temperature. Sensors tracked pizzas entering and exiting the oven, and timestamps were analyzed to calculate the pizza volume inside. Ambient temperature data was also gathered. To model power dissipation relative to pizza volume and temperature, the volume data was scaled by a 10^{-7} factor to align with the temperature data, enabling its effective inclusion in the PySINDy model. The PySINDy model was trained using the scaled pizza volume data

[2]The auto-twin is an international project that has developed a data-driven method that is based on a process mining approach for Automated Digital Twin generation, operations, and maintenance in circular value chains.

along with temperature data. This model accurately predicted power dissipation, with Equation 3 providing the basis for these predictions. The high accuracy is reflected in the Model Score of 0.91 and the close alignment between predicted and actual power data, as shown in Figure 7.

$$\begin{aligned} P[k+1] = {} & 5.220 + 0.617 \cdot P[k] - 0.077 \cdot T \\ & - 3.405 \cdot V + 0.363 \cdot P[k] \cdot V \\ & + 0.071 \cdot T \cdot V - 1.428 \cdot V^2 \end{aligned} \tag{3}$$

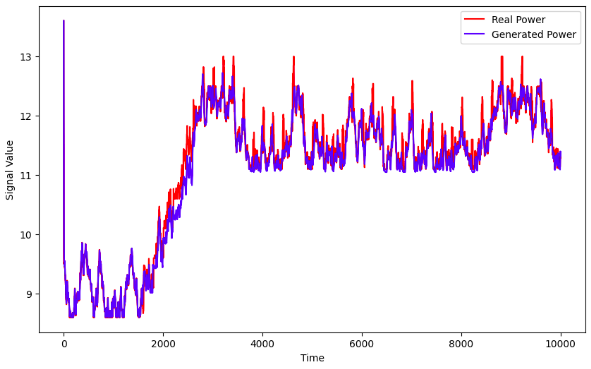

Figure 7: auto-twin Pizza Line final PySINDy model prediction with scaled volume.

Further simulations on 15 days of data show the same results as the one in Figure 7.

3.4. Unknown system: Energy MADE

In this section, we apply the identification method to the Energy MADE system, a vertical machining center used in industrial applications. The goal was to develop a model using lab simulation data without prior knowledge of the system's dynamics. Data from a Famup MC 70 E vertical machining center at the MADE competence center was stored in Siemens Mindsphere. Key signals analyzed were spindle power (x, y, z-axis sum) and spindle speed. Preprocessing included adjusting spindle speed, filtering negative power values, and combining 15 days of data.
Initial polynomial regression identified a significant scale difference between power and speed, which, once corrected, improved prediction accuracy with an MSE of 0.0184. Only for the polynomial regressor, to address data noise, a Kalman filter was applied, smoothing the data. The relationship between power and speed was refined before applying the PySINDy model. The final PySINDy model, employing normalization and regularization techniques, was tested on various sequences and achieved an MSE of 0.0821. While it was slightly less accurate than the polynomial regressor, it demonstrated robustness in handling noisy data. The model is illustrated by Equation 4 and the results are shown in Figure 8.

$$\begin{aligned} P[k+1] = {} & 0.417 + 1.450 \cdot P[k] - 0.163 \cdot S[k] \\ & - 0.036 \cdot P[k]^2 - 0.142 \cdot P[k] \cdot S[k] \\ & + 0.023 \cdot S[k]^2 \end{aligned} \tag{4}$$

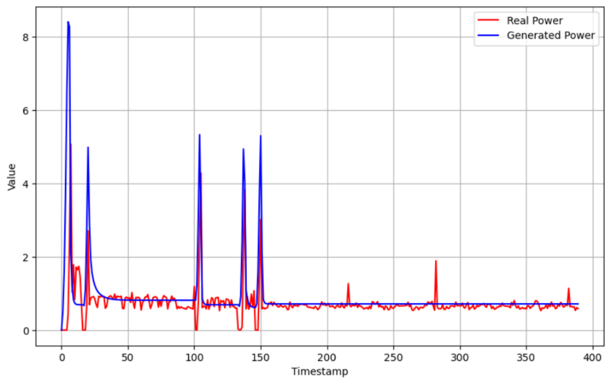

Figure 8: Final PySINDy Model Predictions for Energy MADE

Subsequently, the SHA to UPPAAL function was employed to generate verification-ready models. The SHA and corresponding UPPAAL models for Energy MADE were generated ensuring they are suitable for rigorous evaluation against real-world scenarios. We then compared PySINDy model predictions with actual data using UPPAAL, generating specific speed signals for prediction. The comparison in Figure 9 highlights the differences between actual power signals (red) and SHA predictions (green). The initial MSE of 0.309 indicates that the model captures overall trends but has discrepancies due to isolated and negligible peaks. After removing these peaks, the MSE improves to about 0.1, aligning more closely with local test results.
Despite minor discrepancies, the model effectively predicts operational changes and captures system behavior. Differences between local and

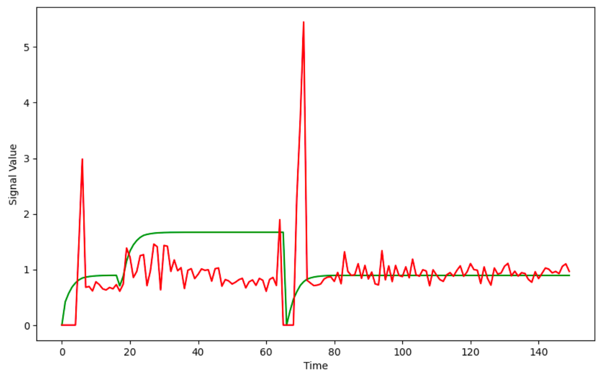

Figure 9: Energy MADE UPPAAL signal validation.

integrated simulations arise from the broader application of PySINDy and discretization methods.

4. Future improvements

One approach to enhancing the modeling framework for complex systems like Energy MADE is the application of advanced machine learning techniques. Techniques such as ANNs and RL can boost the accuracy and reliability of models derived from SINDy. ANNs are effective at capturing complex patterns and nonlinear relationships, improving data preprocessing and model validation for SINDy. RL optimizes decision-making by interacting with the environment and refining control policies based on SINDy's models. Integrating ANNs and RL with SINDy, as demonstrated by extensions like GSINDy [1] and SINDy-RL [5], addresses challenges like noisy data and complex dynamics, leading to more accurate and robust models. Future work will further explore these integrations to advance system identification and control strategies.

5. Conclusions

The thesis demonstrates significant advancements in CPS modeling through the integration of SINDy with the L^*_{SHA} algorithm. By employing the PySINDy library, we successfully incorporated sparse identification of ODEs into the learning process of SHA, allowing for dynamic behaviors to be modeled directly from observational data without relying on predefined ODEs. The thesis presents two distinct approaches for incorporating PySINDy, each tailored to different case study requirements, providing flexibility in application. Comprehensive validation across multiple case studies showcases the effectiveness of these methods. The results affirm that the proposed approach enhances modeling accuracy and adaptability, even in complex scenarios with noisy data. However, challenges remain when dealing with systems characterized by intricate internal dynamics and noisy data, as observed in the Energy MADE case study. While the results are promising, they highlight the need for improved data quality. Future work will explore the combination of SINDy with advanced machine learning methods to further enhance modeling capabilities, system control, and data quality.

References

[1] Dynamics Lab. Modified-sindy: Neural network and sparse identification of nonlinear dynamics integrated algorithm for digital twin identification. `https://github.com/dynamicslab/modified-SINDy`, 2023. Accessed: 2024-08-20.

[2] Livia Lestingi, Marcello M. Bersani, and Matteo Rossi. Model-driven development of service robot applications dealing with uncertain human behavior. *IEEE Intelligent Systems*, 2022.

[3] David G. Schaeffer. Sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 2016.

[4] David G. Schaeffer. *PySINDy Documentation*, 2021.

[5] Nicholas Zolman, Urban Fasel, J. Nathan Kutz, and Steven L. Brunton. Sindy-rl: Interpretable and efficient model-based reinforcement learning. *ArXiv*, March 2024.