

Learning Time Delay Systems with Applications on Intelligent Vehicles

by

Xunbi Ji

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Mechanical Engineering)
in the University of Michigan
2025

Doctoral Committee:

Professor Gábor Orosz, Chair
Doctor Sergei Avedisov
Professor Dimitri Breda
Professor Tulga Ersal
Professor Alex Gorodetsky

Xunbi Ji

xunbij@umich.edu

ORCID iD: 0000-0002-7024-2959

© Xunbi Ji 2025

DEDICATION

To my parents.

ACKNOWLEDGEMENTS

As I reflect on the past five years of my PhD journey, I would describe it as a relatively smooth ride. I will always cherish the memories of the big games at the stadium, the white and freezing winters, the vibrant blossoms of spring, the lively musical summers, and the breathtaking colors of fall in Ann Arbor, Michigan. Of course, I also vividly recall moments of breakdowns and self-doubt. Through it all, I have been fortunate to have the support of many people, and I want to take this opportunity to express my heartfelt gratitude to them.

First of all, I would like to thank my advisor, Professor Gábor Orosz. This journey all started with a class I took with him in my second semester of graduate school. He has been a great mentor and a role model for me. He is always encouraging and supportive, and rigorous in our work. I also want to thank my committee members, who have provided me with valuable feedback and guidance throughout my research. I had interactions with all of them, they are my coauthors, mentors, instructors and friends.

My family and friends have been my biggest support. Without the financial support from my parents, I would not have come to Michigan for graduate study. Special thanks to my husband Tamás, who has seen and shared both my ups and downs throughout this trip. I also want to thank Qingyun Chinese Music Ensemble, which has provided me with a sense of community and belonging. I enjoyed every performance and rehearsal with them.

I greatly acknowledge the support through the J. Robert Beyster Computational Innovation Graduate Fellowship.

TABLE OF CONTENTS

DEDICATION	ii
ACKNOWLEDGEMENTS	iii
LIST OF FIGURES	vi
LIST OF TABLES	xii
LIST OF APPENDICES	xiii
LIST OF ACRONYMS	xiv
ABSTRACT	xv

CHAPTER

1 Introduction	1
1.1 Rich Dynamics in Time Delay Systems	1
1.2 Neural Networks and Dynamical Systems	2
1.3 Control of Intelligent Vehicles with Time Delays	5
1.4 Dissertation Contributions and Outline	7
1.4.1 Contributions	7
1.4.2 Outline	7
2 Neural Delay Differential Equations with Trainable Delays	9
2.1 Time Delay Neural Networks with Trainable Delays	10
2.1.1 Conventional TDNN	10
2.1.2 Constructing TrTDNNs	12
2.2 Loss Function Design	14
2.2.1 Derivative Loss	15
2.2.2 Simulation Loss	16
2.3 Training NDDEs	16
2.4 Comparisons between NDDEs of Different Architectures	19
2.5 Evaluation of the Trained NDDEs	22
2.5.1 Spectrum Examination	23
2.5.2 Bifurcation Analysis	24
2.5.3 Maximal Lyapunov Exponent	24

2.6	Alternative Method for Learning DDEs	27
2.7	Summary	30
3	Examples of Learning Time Delay Systems from Trajectory Data	31
3.1	Simple Nonlinear Autonomous Systems	31
3.2	Delayed Logistic Equation	35
3.3	Mackey-Glass Dynamics	38
3.4	Ikeda Dynamics	45
3.5	Climate Model with Seasonal Excitation	46
3.6	Predator-Prey Model	49
3.7	Double-Delayed Rössler System	51
3.8	Discussion	56
3.9	Summary	58
4	Learning the Longitudinal Dynamics of a Connected Automated Vehicle	60
4.1	Delayed Longitudinal Dynamics	60
4.2	Learning the Delayed Longitudinal Dynamics	64
4.3	Summary	67
5	Learning the Lateral Dynamics of a Teleoperated Vehicle	69
5.1	Latency in Teleoperated Driving System	69
5.2	Stability of Lateral Vehicle Dynamics under Latency	71
5.2.1	Effect of constant latency	71
5.2.2	Effect of time-varying latency	83
5.3	Learning the Delayed Lateral Dynamics	95
5.4	Summary	100
6	Mitigating the Effect of Delay with Act-and-Wait	102
6.1	Act-and-wait Algorithm	104
6.2	Act-and-wait Steering Control	104
6.3	Summary	110
7	Conclusion	111
	APPENDICES	114
	BIBLIOGRAPHY	128

LIST OF FIGURES

FIGURE

1.1	Simulation data of Ikeda equation (1.1) with $\beta = 8$, $\gamma = 2$. (a) A never-repeating trajectory generated by $\tau = 1$. (b) A periodic trajectory generated by $\tau = 0.5$.	2
1.2	Relationships between dynamical systems and neural networks without and with time delay. The input of time delay neural networks on the right involves the past states and inputs.	3
1.3	Overview of modeling time delay systems with neural networks and differential equations	4
1.4	Latency components in the control loop of teleoperated driving.	5
1.5	The delay in drive-by-wire steering command leads to unstable behaviors in the experiments at MCity.	6
2.1	(a) Time delay neural network with trainable delay for predicting state derivative at time t (The actual inputs to the network move along the history vector, depending on the trainable delays); (b) use of TrTDNN as feed-forward mapping during training; (c) use of TrTDNN for simulation: gray color means inputs from data and the yellow color means the inputs obtained from simulation.	12
2.2	Comparison between two loss functions. The derivative loss function $\mathcal{L}_x$ takes in the data directly without simulation and compares the output (state derivatives) to the ones in data (which may be calculated from numerical differentiation). The simulation loss function $\mathcal{L}_x$ compares the simulated state to the data.	15
2.3	Schematic diagrams of network architectures: (a) deep neural network with one delayed input; (b) fully-connected shallow network with two trainable delayed inputs; (c) decoupled shallow network with two trainable delayed inputs; (d) conventional TDNN with a window of fixed delayed inputs.	19
2.4	Evolution of the loss (left) and delays (middle) over the iterations during training, and the equivalent approximation F of the nonlinearities given by trained networks (right): (a)-(c) fully-connected network; (d)-(f) decoupled network.	20
2.5	Static mapping and simulation results for the trained networks: (a), (b) performance for a trajectory with $c_{tr} = 1.4$ in the training dataset; (c), (d) performance on testing dataset with $c_{ts} = 1.3$.	21
2.6	Spectrum approximation of (2.32) around its equilibrium $x(t) \equiv 0$.	23
2.7	Qualitative changes in behavior when varying the time delay. (a)-(c) trajectories generated by the Mackey Glass equation for delay values $\tau = 0.2, 0.5, 1$, respectively. (d) Bifurcation diagram summarizing the change of steady state solution and periodic orbits as delay increases.	25

3.1	Convergence of learned delays in 100 runs: (a) 1 attempt; (b) 5 attempts; (c) 10 attempts; (d) 20 attempts. The color indicates the training loss.	32
3.2	Normalized RMSE and learned delays along training iterations for system (3.1) (a)–(b), and system (3.2) (c)–(d). While only first 700 iterations are shown in the figure, the actual training stops later based on the validation loss criterion.	33
3.3	Performance of TrTDNN and conventional TDNN on the training data for system (3.1) (a), and for system (3.2) (b).	34
3.4	Performance of TrTDNN and conventional TDNN on the test data for system (3.1) (a), and for system (3.2) (b).	34
3.5	(a) Approximation the nonlinearities in (3.2). Solid curves represent the ground truth while the dashed curves are extracted from the TrTDNN. (b) Approximation of the spectrum in (3.2). Black crosses show the ground truth while the green crosses are from the trained network.	35
3.6	Training the NDDE (3.5) (2 hidden layers and 10 neurons in each layer, i.e., $l_1 = l_2 = 10$) with two different datasets of delayed logistic equation.	37
3.7	Training datasets generated by the Mackey-Glass system: each dataset contains one trajectory generated for a different delay value using the initial history $x(\vartheta) \equiv 0.3, \vartheta \in [-\tau_{\max}, 0]$ sampled with time step $\Delta t = 0.1$. (a)–(c) has the same length $T_{\text{tr}} = 15$, while (d) has the length $T_{\text{tr}} = 30$ since the richer dynamics takes more time to present.	39
3.8	Learning the Mackey-Glass dynamics. (a) Training loss along the iterations. (b) Learned delays along the iterations. (c) Derivative prediction on test data. (d) Simulation prediction on test data.	40
3.9	Results when the networks are trained on the limit cycle dataset $\tau = 0.5$ (left) and the chaotic dataset $\tau = 1$ (right). (a)–(b) Learning the delays from the training data. (c)–(f) Comparison of network performances on test data. (g)–(h) Absolute error between $\dot{x}$ and $\hat{\dot{x}}$ in state space where the grey dots indicate training data.	42
3.10	Maximal Lyapunov exponents computed for the Mackey-Glass system and the trained NDDE. Panel (a) shows the MLEs as a function of the truncation time T for delay $\tau = 1$ and history $x_0 \equiv 0.9$. Panels (b) and (c) show the exponents at $T = 10^4$ for delays $\tau = 1$ and $\tau = 1.5$, respectively, while considering the initial histories $x_0 \equiv c, c \in [0.1, 0.9]$ in the testing data.	43
3.11	Bifurcation diagrams of the NDDE learned from data at $\tau = 0.2, 0.5, 1, 1.5$ compared to the ground truth. The dashed line indicates the underlying delay in the training data, and the shaded area indicates where the original Mackey-Glass system exhibits chaotic behavior.	44
3.12	Training performance under two different levels of white noise. Both networks have the same initial delay 0.8 and are trained on chaotic data using weighted derivative loss.	46
3.13	Training and testing performance of the Ikeda system. (a) Derivative prediction on test data. (b) Simulation prediction on test data. (c) Bifurcation diagram.	47

3.14	Training and testing performance of the climate model. (a) Training and testing datasets obtained from a single chaotic trajectory. (b) Training loss along the iterations. (c) Learned delays along the iterations. (d) Derivative prediction on test data. (e) Simulation prediction on test data. (f) Maximal Lyapunov exponents of the climate model and the learned NDDE while using $M = 10$	48
3.15	Training performance of the predator-prey model.	50
3.16	Evaluation of learning the delay from noisy data. (a) success ratio of learning the delay for different levels of noise; (b) distribution of 100 learned delays. All the training runs are initialized with different weights and biases but with the same initial delay 3.	50
3.17	Trajectory reconstruction for the Rössler model (3.15) showing the true trajectory (gray), E-SINDy trajectory when using library with polynomial degree 2 (red), E-SINDy trajectory when using library with polynomial degree 4 (green), and NDDE trajectory (blue). The true delay values are $(\tau_1, \tau_2) = (1, 2)$. The optimized delay values obtained through PS optimization for E-SINDy are $(\hat{\tau}_1, \hat{\tau}_2) = (1.0001, 2.0000)$ when using polynomial library of order 2 and $(\hat{\tau}_1, \hat{\tau}_2) = (1.0004, 2.0000)$ when using polynomial library of order 4. The NDDE approach yields $(\hat{\tau}_1, \hat{\tau}_2) = (0.2309, 1.9064)$ when using simplified input and 5 neurons in each hidden layer.	53
3.18	Learned delays and particle filter errors along the iterations when applying E-SINDy with polynomial library of order 2 to the Rössler model (3.15).	53
3.19	The learned delays along the iterations for the two-delay case when applying the NDDE approach to the Rössler model (3.15). (a) Simplified input and 5 neurons in each hidden layer, trained on one trajectory. (b) Simplified input and 20 neurons in each hidden layer, trained on ten trajectories.	54
4.1	Car-following scenario of a connected automated vehicle following a connected human-driven vehicle.	61
4.2	Stability charts under different delays. (a) $\tau = 0.5$ s. (b) $\tau = 0.6$ s. (c) The curve of the fastest-convergence gains intersects the oscillatory stability boundaries around $\tau_{cr} \approx 2.52 \tau$	64
4.3	(a) Experimental setup where a connected automated vehicle (CAV) follows CHVs based on the data transmitted through vehicle-to-vehicle (V2V) connectivity. (b) Nonlinearities used in the model.	65
4.4	NDDE training results on dataset 500 and dataset 233. Panels (a) and (d) show the training and validation loss; panels (b) and (e) display the evolution of the time delay; (c) and (f) depict the simulation of learned NDDE (black) together with the simulation given by the nominal model (light blue) and the experimental data (red). Note that panels (a), (b), (d) and (e) only show the first 600 iterations of training, but the actual training stops later based on the validation loss criterion. The RMSE of the quantities plotted in panels (c) and (f) are collected in Table 4.1.	66
4.5	Distributions of learned delay in 100 runs for the two datasets.	68
5.1	Components in a teleoperated driving (ToD) control loop.	70

5.2	Kinematic vehicle model with respect to point R.	71
5.3	(a) Stability boundaries for different dimensionless delays $\bar{\tau}$. (b) Performance curves for different convergence rates ρ values when $\bar{\tau} = 0.5$. (c) The fastest convergence rate ρ_{fast} as a function of $\bar{\tau}$. (d) The control gains which give ρ_{fast} are plotted as the black dashed-dotted curve. This curve intersects the stability boundary of $\bar{\tau}_{\text{cr}} = 1.2616$ (grey solid curve) when $\bar{\tau} = 0.5$	73
5.4	Rewrite the kinematic model with respect to the path.	75
5.5	(a)-(b) Stability charts for curvatures $\kappa_* = 0 \text{ m}^{-1}$ and $\kappa_* = 0.2 \text{ m}^{-1}$. (c)-(d) Simulations when following a straight path ($\kappa_* = 0 \text{ m}^{-1}$) under different scaled delays. (e)-(f) Simulations when following a circular path ($\kappa_* = 0.2 \text{ m}^{-1}$) under different scaled delays.	79
5.6	Performance boundaries and stability boundaries for $\bar{\tau} = 0.5$ ($l = 2.73 \text{ m}$, $v_* = 3 \text{ m/s}$ and $\tau = 0.46 \text{ s}$). The stars indicate gains of fastest convergence. Different panels correspond to different curvatures: (a) $\kappa_* = 0$, (b) $\kappa_* = 0.125 \text{ m}^{-1}$, and (c) $\kappa_* = 0.2 \text{ m}^{-1}$	80
5.7	(a) Fastest convergence rates for different curvatures as a function of the dimensionless delay. (b) Fastest convergence gains for different curvatures.	81
5.8	Stability chart, characteristic roots and simulations of following a circular path (radius 5 m) under $\bar{\tau}$ and $\bar{\tau}_{\text{cr}}$ while using the same control gains. (a) The fastest-convergence gains obtained for $\bar{\tau}$ are located on the stability boundary for $\bar{\tau}_{\text{cr}}$. (b) The leading characteristic roots for the same gains under different dimensionless delays. (c) Simulation of lateral error while using the two different delays. . . .	81
5.9	(a) Critical dimensionless delay as a function of the dimensionless delay. For the straight path ($\kappa_* = 0 \text{ m}^{-1}$), the slope is constant. (b) Robustness coefficients as a function of the dimensionless delay.	82
5.10	Two examples of stochastic time-varying latency under different communication network conditions. Data 1 and Data 2 are obtained from communication network simulations with different packet loss rates in [54]. These are compared to their constant latency approximations (black lines) and periodic latency approximations (red and blue curves).	84
5.11	Comparing the stability boundaries and the characteristic multipliers under constant and periodic sawtooth latency profiles, when the gains are chosen based on the fastest convergence under the constant average of the periodic sawtooth latency for $v = 3 \text{ m/s}$, $\kappa_* = 0.125 \text{ m}^{-1}$. Case A (panels (a)-(c)) and Case B (panels (d)-(f)) differ in the value τ_{max} . (g) Difference in the magnitude of the largest characteristic multiplier when considering constant vs. periodic latency.	87
5.12	(a)-(c) Three different sawtooth latency profiles with same τ_{min} but different τ_{max} . (d)-(l) Comparing stability charts for periodic sawtooth latency profiles and the corresponding average constant latency for different curvatures and longitudinal velocities when $l = 2.73 \text{ m}$	88

5.13	Stability and simulations under latency profiles for $v = 3$ m/s, $\kappa_* = 0.125$ m ⁻¹ . (a) Three delay profiles share the same average latency $\tau_{\text{ave}} = 0.5$ s. (b) Stability charts for three latency profiles. Two gain combinations marked as P and Q are selected for simulations. (c)-(e) Simulations with gains P under different latency profiles. (f)-(h) Simulations with gains Q under different latency profiles. Dotted sections of the trajectories indicate that the steering angle exceeds 45 degrees.	90
5.14	Path reference used for simulations. (a) Reference path (brown curve) of the rear axle center point with the vehicle depicted as moving along the path (b) Curvature plan for a left turn path. (c) Two velocity plans defined along the path. (d) Three types of latency profiles with $\tau_{\text{ave}} = 0.46$ s.	91
5.15	Stability charts and simulations under different latency profiles and velocity profiles. The stability charts correspond to constant curvature $\kappa_* = \kappa_{\text{max}} = 0.2$ m ⁻¹ and a constant velocity $v_* = v_{\text{max}} = 3$ m/s or $v_* = v_{\text{max}} = 6$ m/s as indicated. (a)-(c) Stability charts and simulations under constant latency $\tau_{\text{ave}} = 0.46$ s. (d)-(f) Stability charts and simulations under one-tooth latency profile. (g)-(i) Stability charts and simulations under two-tooth latency.	92
5.16	The lateral and yaw errors along the path under different latency profiles. . . .	93
5.17	Simulation results for different latency profiles and velocity plans. (a) Two time-varying latency profiles both have $\tau_{\text{ave}} = 0.46$ s. (b) Reference path (blue solid curve) and the simulation of the ToD without any latency. (c) The curvature function along the reference path with $ \kappa_{\text{max}} = 0.1245$ m ⁻¹ . (d) Two velocity plans along the given path with $v_{\text{max}} = 3$ m/s and $v_{\text{max}} = 6$ m/s. (e) Stability chart for $\kappa_* = 0.1245$ m ⁻¹ , $\tau = 0.46$ s, $v_* = 3$ m/s $\Rightarrow \bar{\tau} = 0.5$, and $\tau = 0.46$ s, $v_* = 6$ m/s $\Rightarrow \bar{\tau} = 1$. (f)-(i) Simulation results for the different velocity plans, with latency profiles $\tau_1(t)$ (red) and $\tau_2(t)$ (green) respectively.	94
5.18	(a) Kinematic vehicle model. (b) Structure of the NDDE. The trainable parameters are indicated in red. As the trainable delay in the NDDE updates, the delayed input to the NDDE changes accordingly.	96
5.19	TELECARLA setup used for collecting experimental data.	97
5.20	Experimental data of a lane-change maneuver with a human remote operator in the loop. Different network latencies $\Delta\tau$ were introduced in the experiments and three runs of data were collected under each condition.	98
5.21	Simulations of trained NDDE are compared to the experimental data on the left and the stability charts of the learned remote human operator are shown on the right. (a)-(b) a run of data collected under $\Delta\tau = 0$ s. (c)-(d) a run of data collected under $\Delta\tau = 0.4$ s.	99
5.22	Stability charts extracted from the TELECARLA experiments.	100
5.23	The overall latency in the loop estimated from individual trajectories for different values of the communication latency $\Delta\tau$	100

6.1	(a) The framework of ToD with delays. (b) Using act-and-wait strategy on top of existing remote operator to stabilize the system under large delay. Top: Simulations of a vehicle following a straight line $y = 0$ under different delays. The nominal control gains chosen for delay $\tau = 0.5$ s (black) lose stability when the delay increases to 1 s (grey). After applying the act-and-wait signal on top the nominal controller (without changing the control gains), the system becomes stable again. Bottom: The periodic act-and-wait signal used in the simulation. .	103
6.2	(a) Stability boundaries for different values of the dimensionless delay $\bar{\tau}$ when $a = 1$. (b) Stability boundaries for different values of the act-wait ratio a when $\bar{\tau} = 1$. (c) The control gains which give the dead-beat (DB) control are plotted as a red dashed-dotted curve for $a = 1$. The DB curve intersects the stability boundary of $\bar{\tau}_{cr} = 0.6731$ (red solid curve) at $\bar{\tau} = 0.5$. (d) The DB curve (red dashed-dotted curve) for $a = 0.7$ intersects the stability boundary of $\bar{\tau}_{cr} = 0.6865$ (red solid curve) at $\bar{\tau} = 0.5$	106
6.3	Robustness coefficient $\bar{\tau}_{cr}/\bar{\tau}$ as a function of the act-wait ratio a	107
6.4	(a) Stability boundaries and selected control gains for scenarios with and without act-and-wait algorithm shown by red and black colors, respectively. (b) and (c) Simulation results for different choices of control gains with and without act-and-wait algorithms.	108
6.5	Simulations when using the act-and-wait algorithm with different amplifications of the control gains. (a)-(b) No amplification of the control gains. (c)-(d) Amplification with coefficient 1.2. (d)-(e) Amplification with coefficient 1.4.	109
D.1	Illustration of discretization of a DDE in history.	126

LIST OF TABLES

TABLE

2.1	Performances of different networks on training and testing trajectories. The normalized root-mean-square deviations are calculated based on $\dot{x}$ for the performance of static mapping and on x for the performance of neural network simulation.	22
3.1	Training and testing results for the two systems	35
3.2	Delays and maximal Lyapunov exponents of the original Mackey-Glass system and the trained NDDEs for the four data sets in Figure 3.7 generated for initial history $x_0 \equiv 0.3$, while using mesh size $M = 10$ and truncation time $T = 10^4$	41
3.3	Performance of E-SINDy and P-SINDy (with collocation mesh size 5) and different polynomial orders for the Rössler system (3.15), spelling out the absolute error between the true and optimized delays, the RMSEs for training and testing phases, the number of SINDy calls and the CPU time while employing particle swarm (PS).	55
3.4	Performance of NDDE for the Rössler system (3.15), spelling out the absolute error between the true and optimized delays and the errors in training and testing phases. The last row (Simplified*) is trained on a larger dataset, where 10 trajectories are generated from 10 different constant histories.	55
3.5	RMSE of trained NDDEs with different size on testing data from delayed logistic equation (3.4). All the NDDEs have two hidden layers while the number of hidden neurons ($l_1 = l_2$) in each hidden layer is indicated in the first column.	56
3.6	RMSE of trained NDDEs with different size on testing data from Mackey-Glass equation (3.6). All the NDDEs have two hidden layers while the number of hidden neurons ($l_1 = l_2$) in each hidden layer is indicated in the first column.	57
3.7	RMSE of trained NDDEs with different size on testing data from climate model. All the NDDEs have two hidden layers while the number of hidden neurons ($l_1 = l_2$) in each hidden layer is indicated in the first column.	57
4.1	Testing results for the two datasets.	68

LIST OF APPENDICES

A Training Algorithms Used in the Examples	114
B Analytical Derivation of the Gradients	118
C MATLAB Codes of NDDE Solver for Simulation Loss Training Algorithms	120
D Creating NODE from TrTDNN	124

LIST OF ACRONYMS

TDNN	time delay neural network
TrTDNN	time delay neural network with trainable delays
GD	gradient descent
LM	Levenberg-Marquardt
ADAM	adaptive moment estimation
DDE	delay differential equation
NDDE	neural delay differential equation
ODE	ordinary differential equation
NODE	neural ordinary differential equation
MLE	maximal Lyapunov exponent
ToD	teleoperated driving
CAV	connected automated vehicle
CHV	connected human-driven vehicle
V2V	vehicle-to-vehicle
SINDy	sparse identification of nonlinear dynamics

ABSTRACT

Time delay systems widely exist in many research domains and they exhibit complex dynamic behaviors which may be challenging to model from first principles. A scalar system with one single delay can produce chaotic dynamics (generating never-repeating trajectories, which are sensitive to initial conditions), and the system behaviors may change qualitatively by only varying the delay. Therefore, it is important to utilize data and prior knowledge about delays when modeling time-delayed systems.

The abundance of data in the last decade has been fostering the development of data-driven methods. Many methods, including machine learning with neural networks, require a significant amount of data while the models may still have poor performance outside the training data despite employing thousands of or even millions of parameters. In contrast, this dissertation aims at using simple neural network structures with trainable time delays (TrTDNN) to reproduce complex delayed dynamics. With explicit delays, the interpretability and generalizability of data-driven models can be improved.

In this dissertation, we propose a gradient-based training framework for learning the time delays together with nonlinearities using a continuous-time perspective. Multiple loss functions, training algorithms are provided and tested on various systems to learn the meaningful time delays. The trainable delays significantly simplify the structure of conventional time delay neural networks (TDNNs) by reducing the number of delayed inputs. This gives many advantages such as faster simulation, better interpretability and generalizability. Moreover, having trainable sparse delay lowers the chance of the network converging to undesired local minima.

By using time delay neural networks with trainable delays (TrTDNNs) in continuous time as neural delay differential equations (NDDEs), many tools developed for analyzing delay differential equations (DDEs) can be applied directly to evaluate the trained networks and further improve the data-driven methods. We propose to examine the maximal Lyapunov exponent (MLE) of the trained networks as an indicator of the richness in the training data dynamics. It is demonstrated that the richness of the data plays a crucial role in the generalizability of the trained model. The proposed method for learning the time delay systems are applied on various systems such as the regulation of hematopoiesis in physiology and the

climate model describing the irregular phenomenon of the El Niño Southern Oscillation.

Another important contribution of this dissertation is that we apply the proposed methods on learning the delayed dynamics of intelligent vehicles from experimental data to obtain simple but informative models for control design. In the domain of automotive engineering, time delays are very common and important in safety and performance. This especially holds for intelligent vehicles which may be connected, automated, or driven remotely. In addition to the actuation delays present in the lower-level actuators of vehicles, there are also delays originating from perception and processing time of the planning and control unit. Furthermore, communication latencies may occur if the vehicle is transmitting and receiving information over a wireless network. Substantial delays can greatly compromise control performance, and may lead to unsafe behaviors.

In the dissertation, the effects of these time delays on vehicle stability and performance are studied both analytically and numerically. We provide the closed-form solution for the control gains which leads to the fastest convergence under constant time delays. We also show how robust the fastest-convergence gains are against the variations of the time delays. The results are extended to periodic time delays and the numerical simulations are provided for the stochastic time-varying delays as well.

Since many control strategies designed for time delay systems require an accurate estimation on the delays, learning the delays and vehicle models from the trajectory data may be necessary. This dissertation provides examples of learning delays in both longitudinal and lateral dynamics, using the developed data-driven method. This enables the evaluations of the driving performance under delays and the possibility of improving the controller. In the end, we present the act-and-wait method to mitigate the negative effect in steering with the knowledge of the time delay.

CHAPTER 1

Introduction

Time delays appear in many real-world applications including cryptography [71, 106], epidemiology [128, 87], ecology [94], climate models [63], machining processes [85], vehicle dynamics [76, 3], and transportation systems [93, 1]. On one hand, these systems often have complex dynamics which are hard to model from first principles, and challenging to capture and generalize from data due to the mixture of nonlinearities and delays. Even with one delay in the state, the dynamical system becomes infinite dimensional (the initial condition is a continuous function instead of a point). A scalar system with one time delay could exhibit chaotic behavior. On the other hand, the delays in dynamical systems are meaningful and provide valuable insights. Many control strategies designed for time delay systems require the knowledge of the delays in the states or/and inputs to achieve better performance. Therefore, it is important to model the time delay systems with the awareness of the delays and with the help of the data.

In this chapter, we first briefly demonstrate the rich dynamics that time delay systems may exhibit. Then we motivate the use of neural networks to learn the dynamics of time delay systems. We also discuss the sources of time delays in vehicle dynamics and their potential effects. In the end, we summarize the main contributions and outline the structure of the dissertation.

1.1 Rich Dynamics in Time Delay Systems

Time delay systems are dynamical systems that involve time delays in their states or inputs. Those delays are usually sparse and have clear physical meanings. For instance, the time delay in a predator-prey system can represent the time needed for the predator to mature and the prey to reproduce. In a climate model describing the depth of the ocean layer where the temperature changes drastically, there are time delays associated with the positive and negative feedbacks.

Time delays in the system can lead to rich dynamics. For example, the Ikeda system [49] is a scalar system and it has only one time delay associated with the state. The nonlinearity in the system is a squared sin function, as described by

$$\dot{x}(t) = \beta \sin^2(x(t - \tau)) - \gamma x(t). \quad (1.1)$$

This system exhibits chaotic behavior for certain parameters, as shown in Figure 1.1 (a). Moreover, the system behavior may change qualitatively with the change of the delay value τ , see the periodic trajectory in Figure 1.1 (b).

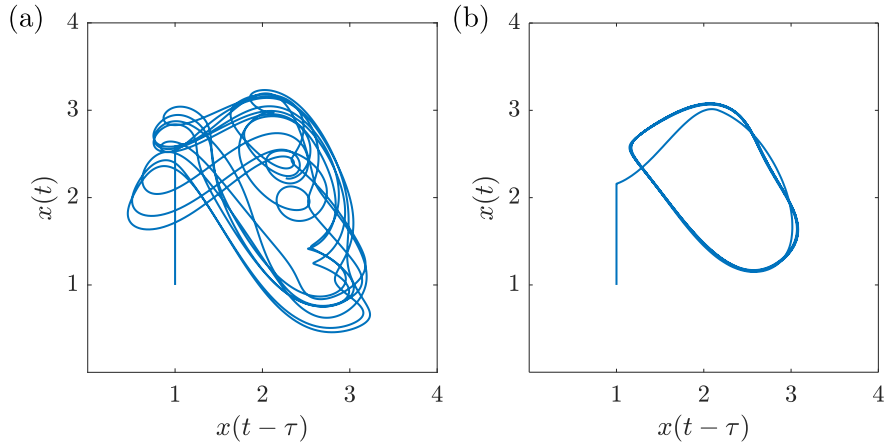


Figure 1.1: Simulation data of Ikeda equation (1.1) with $\beta = 8$, $\gamma = 2$. (a) A never-repeating trajectory generated by $\tau = 1$. (b) A periodic trajectory generated by $\tau = 0.5$.

Without the knowledge of the time delay, one may use a high-dimensional model to approximate the dynamics of the system from the data on the left, which can be redundant and inefficient. Although the data on the right may be reproduced using a low-dimensional model, such a model is typically not generalizable to reproduce the data on the left. Including the correct delayed information into the data-driven models is very helpful when modeling the time delay systems from the data. We aim at using trainable delays in data-driven models, such as neural networks, to capture the dynamics and learn the time delays.

1.2 Neural Networks and Dynamical Systems

In recent years, data-driven methods have been applied to dynamical systems, and among those methods, neural networks have shown their power in function approximation [96, 12]. However, they are also among the most criticized methods since they involve a tremendous number of parameters, which makes their analysis difficult [16]. Moreover, generalizability

of the trained network can be an issue if “big data” are not available. That is, neural networks can have poor performance outside the domain of the training data. Recent works made improvement by embedding prior knowledge gained from first principle models into the network design; see [55, 122, 66, 37, 72] for instance.

Different from the classification and regression problems, which normally aim to obtain maps between individual samples, learning dynamical systems involves sequential data referred to as trajectories. In order to learn the underlying dynamics, neural networks should consider the time dependency in the data. In discrete time, recurrent neural networks such as long short-term memory (LSTM) networks [45] or residual neural networks (ResNet) [44], which consist of many layers, can be used to represent the information flow going through time. In continuous time, neural ordinary differential equation (NODE) [20, 114, 104] directly models the underlying dynamics using a neural network. These networks capture the right-hand side of differential equations, that is, their input is typically the state of a dynamical system, while their output is the time derivative of the state, as shown in Figure 1.2.

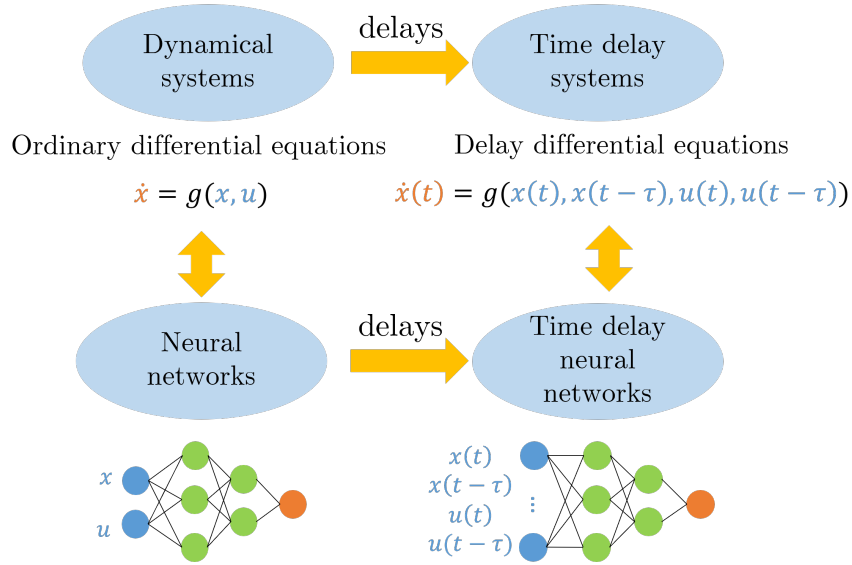


Figure 1.2: Relationships between dynamical systems and neural networks without and with time delay. The input of time delay neural networks on the right involves the past states and inputs.

Adding time delays to neural networks as additional parameters can increase their capability of approximating the dynamics while keeping the network structure simple. The theoretical background of time delay systems has a rich history [22, 40, 82, 69, 51, 29, 10]. Incorporating time delays into the data-driven models can help to interpret the models [56, 57] and generalize for multiple scenarios without requiring additional data [58].

In this dissertation, we take the form of time delay neural networks (TDNNs) to explicitly

model the delays and use neural networks in continuous time to approximate the right-hand side of delay differential equations (DDEs). The use of TDNNs in continuous time results in neural delay differential equations (NDDEs) [132, 133], which are used to address the problem of intersecting trajectories, as the corresponding dynamical system has infinite dimensional state space. It was also shown that NDDEs can be used to capture the latent dynamics of complex systems [41, 42]. Figure 1.3 shows the overview of the learning framework. Instead of deriving the relationship between the state derivative and the past states using first principles, one may use flexible function approximators (i.e., TDNN) to learn the relationship from the data directly. In addition, it is usually helpful to convert the delay differential equation (DDE) into an ordinary differential equation (ODE) via discretization for the purpose of analysis and simulation. The same technique can be applied to convert the neural delay differential equation (NDDE) into a NODE.

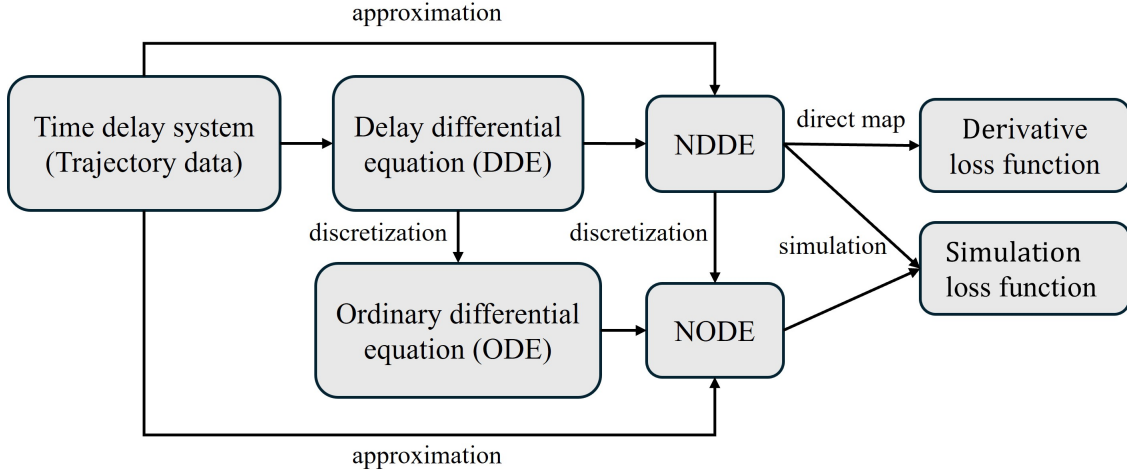


Figure 1.3: Overview of modeling time delay systems with neural networks and differential equations

We propose to use the time delay neural network with trainable delays (TrTDNN), a time delay neural network with trainable delays, which reduces the size of the network and further simplifies the simulation and evaluation of the NDDE. We put forward a gradient-based training framework for NDDEs, enabling the simultaneous learning of time delays and the nonlinearities from trajectory data. The training algorithms proposed for this framework in the dissertation can be categorized by the choice of loss functions, as shown on the right of Figure 1.3. If the loss function involves network simulations, one may simulate the NDDE directly or simulate an equivalent NODE (for which the training algorithms are well-developed and intensively studied). If the loss function only contains a direct map between the data and the state derivatives, the conversion between the NDDE and the NODE is not

necessary. We demonstrate that using such framework, complex dynamics can be learned via neural networks of simple structure. Additionally, we can evaluate the trained NDDE with existing tools developed for DDEs. It is shown that the nonlinearities are well-captured and the networks can generalize well at different delay values if the networks are trained with dynamically rich data.

1.3 Control of Intelligent Vehicles with Time Delays

In the operation of conventional vehicles, time delays appear due to the human driver's reaction time and the actuation time of lower-level actuators, such as delays in the power-train. Intelligent vehicles, that is, vehicles with different levels of autonomy, connected vehicles and teleoperated vehicles, are under development for replacing or assisting human drivers. These blooming technologies may remove the reaction delay of the human driver, but in the meantime other sources of delays are introduced.

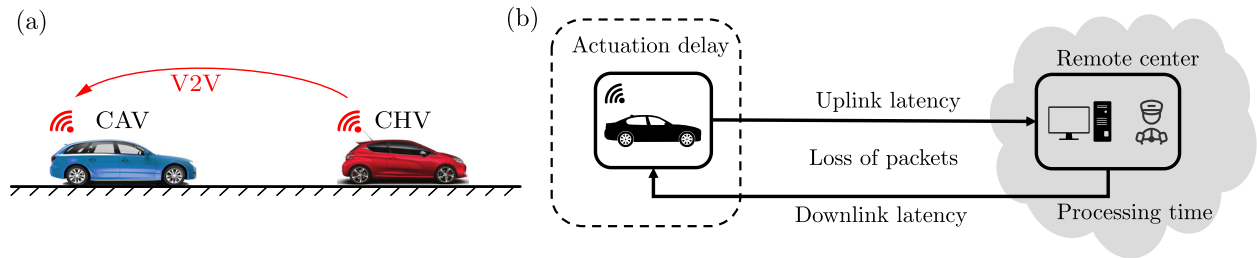


Figure 1.4: Latency components in the control loop of teleoperated driving.

In Figure 1.4(a), we illustrate that in the case of connected automated vehicle (CAV) following a connected human-driven vehicle (CHV), additional time delay is introduced by the communication network, which is usually not negligible (significant packet loss may happen due to surrounding environment). In the control loop of a teleoperated vehicle as shown in Figure 1.4(b), the overall latency originates from multiple sources:

- (i) Time needed to send and receive data packets [126, 79];
- (ii) Processing time of the remote operator [91, 97, 26];
- (iii) Actuation delay of the vehicle [125, 88].

In addition, the increase of the latency can result in packet loss that strongly impacts the teleoperated driving (ToD) service performance [79, 101]. While researchers and engineers have been working on designing low-latency communication networks and on reducing latency in video streaming [127, 23], the overall latency cannot be eliminated due to the latency

sources (ii)-(iii). For example, an automated vehicle typically has non-negligible delay in the longitudinal dynamics of around 0.6 seconds [57].

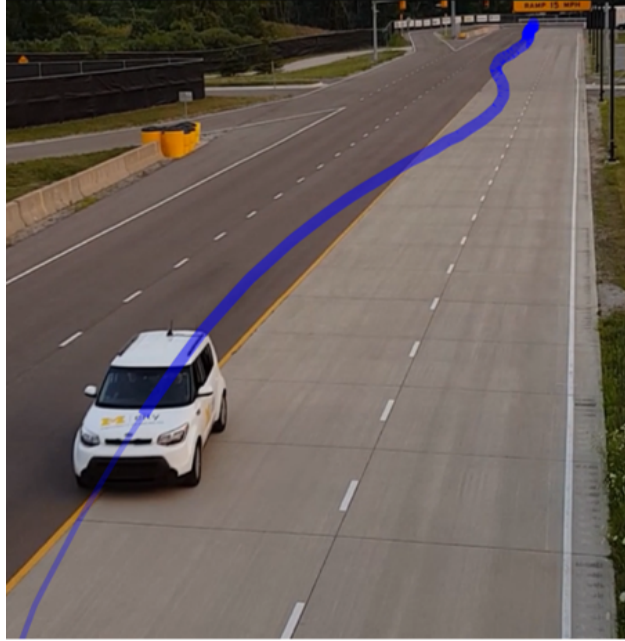


Figure 1.5: The delay in drive-by-wire steering command leads to unstable behaviors in the experiments at MCity.

Latency in the longitudinal dynamics may lead to rear-end collisions and degradation of energy and time efficiency [32], while latency in the lateral dynamics can lead to lane departure and loss of control. Figure 1.5 illustrates when a vehicle is unable to follow the desired trajectory due to the time delay in the steering actuation [3].

The analytical study of constant latency in vehicle dynamics has been carried out in [3]. Periodic and stochastic latency is considered in [86, 101] for longitudinal dynamics of a connected automated vehicle, due to the packet loss in the communication network and the zero-order-hold implementation of the control commands [79, 101]. The stochasticity in latency can lead to unstable behaviors even when the system is stable under its average latency [112].

In this dissertation, we extend the analysis to lateral dynamics under periodic and stochastic latency and present simulations for teleoperated driving. We show that it is crucial to account for the overall latency when evaluating the controller’s performance [34], because many control methods require accurate model and delay value to enhance control performance. We will use the NDDE with trainable delays to learn the delayed dynamics of intelligent vehicles. Moreover, we present a way of using the act-and-wait algorithm to mitigate the effect of delays on steering control in ToD, once the latency is known.

1.4 Dissertation Contributions and Outline

1.4.1 Contributions

The first main contribution of the dissertation is a data-driven framework for learning time delay systems in continuous time, by introducing trainable delays into the neural networks. This simplifies the structure of the conventional time delay neural networks (TDNNs) and brings advantages such as generalizability and interpretability.

By formulating the TDNNs as neural delay differential equations (NDDEs), we can evaluate the dynamical behaviors of the trained networks. For example, we utilize bifurcation diagrams and maximal Lyapunov exponents to characterize the network dynamics. This serves as a second contribution of this dissertation. With these methods, we can quantify the quality of the training dataset in terms of the “richness” of dynamics and see how the data quality affects the generalizability of the trained time delay neural network (TDNN)s. Moreover, we also compare the proposed method with other state-of-the-art methods such as sparse identification of nonlinear dynamics (SINDy).

As a third contribution, the last part of this dissertation focuses on the control of intelligent vehicles with time delays. The effects of time delays on stability are studied in details. Then our data-driven method is applied to learning the vehicle dynamics and the delay. With the knowledge of the delay, we demonstrate that the control performance can be improved by implementing advanced control algorithms.

1.4.2 Outline

The dissertation is organized as follow. In Chapter 2, we introduce the formulation of time delay systems and time delay neural networks in continuous time. We propose to use trainable delays for learning the dynamics of time delay systems from trajectory data. We compare the different loss functions and discuss both advantages and limitations of the proposed learning framework. We also compare the performance of our method with another state-of-the-art method developed for learning delayed dynamics.

In Chapter 3, we implement the proposed framework on learning the dynamics of the delayed logistic equation, Mackey-Glass system, Ikeda system, a climate system, a predator-prey system and a multiple-delay Rössler equation. We provide evaluations on the network generalizability and the training algorithm under noise. The results for the multiple-delay Rössler equation are compared with SINDy method.

In Chapter 4 and Chapter 5, we focus on the effect of time delays in longitudinal dynamics and lateral dynamics of the vehicles, respectively. Detailed analysis on the lateral dynamics

are presented for various types of latency profiles. We apply the proposed learning framework on the delayed dynamics of a connected automated vehicle and the behavior of teleoperated vehicle.

Lastly, we demonstrate an example of using act-and-wait control strategy to mitigate the effect of latency with known latency value in Chapter 6. The dissertation is concluded with discussion of future directions in Chapter 7.

CHAPTER 2

Neural Delay Differential Equations with Trainable Delays

In this chapter, we start with a brief introduction of time delay systems with discrete delays and establish a formulation of the problem in the context of neural networks. We emphasize the use of time delay neural networks in continuous time which leads to neural delay differential equations in Section 2.1. We incorporate the trainable time delays in time delay neural networks formulation and the training algorithms for learning the dynamics with trainable delays. Two main types of loss functions are introduced in Section 2.2 based on whether simulation is performed during training or not, and a training algorithm is provided in Section 2.3. Learning results are presented in Section 2.4 followed by a discussion. We introduce the evaluation of trained neural delay differential equations in Section 2.5 and another state-of-the-art method for learning delayed dynamics in Section 2.6. Our proposed method is compared to SINDy in a multiple-state example in Chapter 3.

We consider time delay systems in continuous time which can be cast in the general form

$$\dot{x}(t) = \mathcal{G}(x_t, u_t), \quad (2.1)$$

where $x \in \mathbb{R}^n$, and the latter $x_t \in C([-\tau_{\max}, 0], \mathbb{R}^n)$ represents the space of continuous functions. Namely, we define $x_t(\vartheta) := x(t + \vartheta)$, $\vartheta \in [-\tau_{\max}, 0]$ where $\tau_{\max} > 0$ is the maximum value of the delay. Similarly, for the control input we have $u \in \mathbb{R}^p$, $u_t \in C([-\tau_{\max}, 0], \mathbb{R}^p)$, and $u_t(\vartheta) := u(t + \vartheta)$, $\vartheta \in [-\tau_{\max}, 0]$. That is, the right hand side is given by the functional $\mathcal{G} : C([-\tau_{\max}, 0], \mathbb{R}^n) \times C([-\tau_{\max}, 0], \mathbb{R}^p) \rightarrow \mathbb{R}^n$. Specifically, we restrict ourselves to point delays, that is, to cases when (2.1) can be written as

$$\dot{x}(t) = g(x(t - \tau_0), x(t - \tau_1), \dots, x(t - \tau_d), u(t - \sigma_0), u(t - \sigma_1), \dots, u(t - \sigma_q)), \quad (2.2)$$

where $g : \mathbb{R}^{n(d+1)+p(q+1)} \rightarrow \mathbb{R}^n$ and $\tau_0 = \sigma_0 = 0$, while d and q represent the number of

(nonzero) delays in the state and the input, respectively. Our goal here is to learn the nonlinear functional $\mathcal{G}$, or equivalently, learn the nonlinear function g and the delays $\tau_0, \tau_1, \dots, \tau_d$ and $\sigma_0, \sigma_1, \dots, \sigma_q$, simultaneously.

2.1 Time Delay Neural Networks with Trainable Delays

Learning the functional $\mathcal{G}$ in (2.1) means finding a functional $\hat{\mathcal{G}}$ such that the norm $\|\mathcal{G}(x_t, u_t) - \hat{\mathcal{G}}(x_t, u_t)\|$ remains small while considering the set of states $\mathcal{X} \subset C([- \tau_{\max}, 0], \mathbb{R}^n)$ and set of inputs $\mathcal{U} \subset C([- \tau_{\max}, 0], \mathbb{R}^p)$. These sets will be specified with the available data. For the specific form (2.2) we shall find a function $\hat{g}$ such that the norm $\|g(\cdot) - \hat{g}(\cdot)\|$ remains small for the set of arguments specified by the data, and also find time delays $\hat{\tau}_0, \hat{\tau}_1, \dots, \hat{\tau}_d$ and $\hat{\sigma}_0, \hat{\sigma}_1, \dots, \hat{\sigma}_q$ which are close to the true delay values. In [18], a method was proposed to identify the time delay and the plant's parameters for a linear system using transfer function estimation. Here we will obtain the nonlinear function $\hat{g}$ and the delays $\hat{\tau}_0, \hat{\tau}_1, \dots, \hat{\tau}_d$ and $\hat{\sigma}_0, \hat{\sigma}_1, \dots, \hat{\sigma}_q$ directly from states and inputs.

In order to achieve this, we construct time delay neural networks (TDNNs) in continuous time. The inputs of such a neural network are the current and past values of the state and the control input, and the output of the network is the state derivative. First, we start with the conventional TDNN in the continuous time and then propose the time delay neural networks with trainable delays (TrTDNNs).

2.1.1 Conventional TDNN

Time delays have been introduced into neural networks for speech recognition in [119]. The corresponding TDNNs are essentially feed-forward neural networks with delayed information flow. These can be expressed as

$$\begin{aligned} z_1^t &= f_1 \left(\sum_{k=0}^r w_1^k z_0^{t-k} + b_1 \right), \\ z_l^t &= f_l (W_l z_{l-1}^t + b_l), \quad l = 2, \dots, L, \end{aligned} \tag{2.3}$$

where z_l^t represents the (potentially vector valued) output of layer l at time t , $f_l(\cdot)$ is the activation function which is applied element-wise and has well-defined derivative $f'_l(s) = \frac{\partial f_l(s)}{\partial s}$, w_1^k is the weight matrix associated with delay index k , while W_l and b_l are the weight and

bias used in layer l . Defining the augmented vector

$$\mathbf{z}_0^t = \begin{bmatrix} z_0^t \\ z_0^{t-1} \\ \vdots \\ z_0^{t-r} \end{bmatrix}, \quad (2.4)$$

equation (2.3) can be re-written as

$$\begin{aligned} s_1^t &= W_1 \mathbf{z}_0^t + b_1 \\ z_1^t &= f_1(s_1^t), \\ s_l^t &= W_l z_{l-1}^t + b_l \\ z_l^t &= f_l(s_l^t), \quad l = 2, \dots, L, \end{aligned} \quad (2.5)$$

with $W_1 = [w_1^0, w_1^1, \dots, w_1^r]$. Here we introduce the compact notation

$$z_L^t = \text{net}(\mathbf{z}_0^t), \quad (2.6)$$

which captures the functional relationship between the input $\mathbf{z}_0^t$ and the output z_L^t of the network.

The time history of the state x and the control input u is often referred to as memory. These are usually available in discrete time and the output can be very general such as letters or words. Since our goal is to represent the dynamics of the dynamical system (2.2), we can discretize time and assume that the delays coincide with the discrete time steps $j\Delta t$, $j \in \mathbb{Z}$ where data is available. Then one may construct

$$\mathbf{z}_0^t = \begin{bmatrix} x(t) \\ u(t) \\ x(t - \Delta t) \\ u(t - \Delta t) \\ \vdots \\ x(t - r\Delta t) \\ u(t - r\Delta t) \end{bmatrix}, \quad (2.7)$$

where $\tau_{\max} = r\Delta t$. Different from the typical use of TDNN many applications in discrete time, we view the TDNN as a delay differential equation, whose output is the predicted state

derivative at time t , i.e.,

$$z_L^t \equiv \hat{x}(t). \quad (2.8)$$

The conventional TDNN sometimes leads to computational issues. Namely, considering small Δt values result in a large weight matrix W_1 . This is not necessary for systems with discrete point delays, since many of these weights shall approach zero during training that requires extra regularization. At the same time, simulating the system with the large number of delays is also computational expensive. Instead, we propose a new architecture which can accommodate trainable time delays as laid out in the next subsection.

2.1.2 Constructing TrTDNNs

Different from the conventional TDNNs, the time delays in the TrTDNNs are continuous variables that evolve through the learning process. The concept of TrTDNN is shown in Figure 2.1(a). There is still a window of sequential data available to the network at each time moment t , but the trainable delay parameters $\tau_0, \tau_1, \dots, \tau_m$ “select” the relevant ones as the network input. As these time delays evolve during the training, the inputs change as indicated by the blue arrows. After training, similar to the weights and biases, the time delays are fixed when using the network to predict future states of the dynamical system.

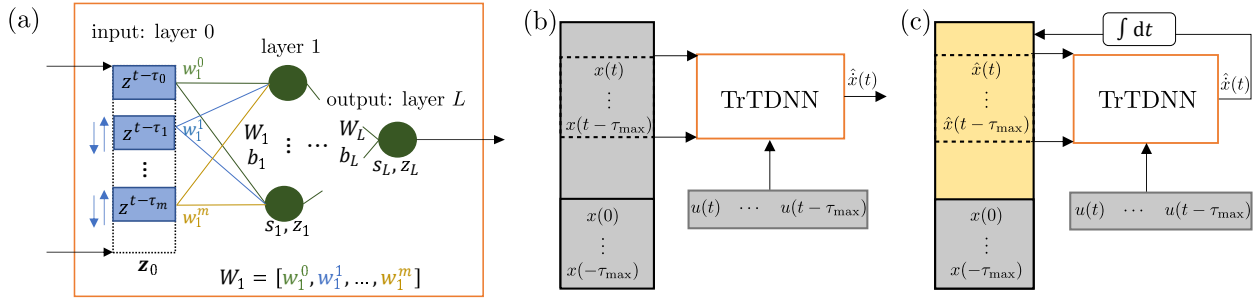


Figure 2.1: (a) Time delay neural network with trainable delay for predicting state derivative at time t (The actual inputs to the network move along the history vector, depending on the trainable delays); (b) use of TrTDNN as feed-forward mapping during training; (c) use of TrTDNN for simulation: gray color means inputs from data and the yellow color means the inputs obtained from simulation.

TrTDNNs also take the form (2.5) but how the input $\mathbf{z}_0^t$ is constructed in the two cases differ. TDNNs utilize the whole memory (2.4) while for TrTDNNs we construct the sparse

memory

$$\mathbf{z}_0^t = \begin{bmatrix} z_0^{t-\tau_0} \\ z_0^{t-\tau_1} \\ \vdots \\ z_0^{t-\tau_m} \end{bmatrix}, \quad (2.9)$$

with flexible delays which will be learned from data. By constructing such flexible delayed inputs, the number of delays in TrTDNNs can be much smaller than the number of discretization points ($m \ll r$). This allows us to greatly simplify the network structure and customize using prior knowledge, which also improves the interpretability of the results. The arrangement of the delayed states and control inputs is also flexible, i.e., the delays do not have to be in an ascending/descending order, they can be assigned as zero, the delays in u and s may be different, and their number can be overestimated. For instance, when learning the dynamics of (2.2), one may construct the input of the TrTDNN as

$$\mathbf{z}_0^t = \begin{bmatrix} x(t - \hat{\tau}_0) \\ x(t - \hat{\tau}_1) \\ \vdots \\ x(t - \hat{\tau}_d) \\ u(t - \hat{\sigma}_0) \\ u(t - \hat{\sigma}_1) \\ \vdots \\ u(t - \hat{\sigma}_q) \end{bmatrix}, \quad (2.10)$$

where the delays $\hat{\tau}_k$ and $\hat{\sigma}_k$ are continuous variables learned from the data. Since the state x and the control input u are typically available at time moments $j\Delta t$, $j \in \mathbb{Z}$, we will use linear interpolation to obtain their value.

We remark that the form (2.6) can be viewed as a recurrent neural network if the output of the network is used as input in the next time moment. Depending on whether the output is the state derivative at current time or the state itself at next time moment, the network can be used to capture the dynamics of continuous-time or discrete-time dynamical systems. In this dissertation, we only consider the former one, that is, the outputs of the TDNNs and TrTDNNs are used as the state derivatives to resemble the delay differential equations in the continuous time.

When the TrTDNN is used for simulation, the output of the network simulation will become the input of the network at the next time moment. Therefore the corresponding

neural delay differential equation (NDDE) can be described as

$$\hat{x}(t) = \text{net}(\hat{\mathbf{z}}_0^t), \quad (2.11)$$

where

$$\hat{\mathbf{z}}_0^t = \begin{bmatrix} \hat{\mathbf{x}}(t) \\ \mathbf{u}(t) \end{bmatrix}, \quad (2.12)$$

$$\hat{\mathbf{x}}(t) = \begin{bmatrix} \hat{x}(t - \hat{\tau}_0) \\ \hat{x}(t - \hat{\tau}_1) \\ \vdots \\ \hat{x}(t - \hat{\tau}_d) \end{bmatrix} \quad \text{and} \quad \mathbf{u}(t) = \begin{bmatrix} u(t - \hat{\sigma}_0) \\ u(t - \hat{\sigma}_1) \\ \vdots \\ u(t - \hat{\sigma}_q) \end{bmatrix}. \quad (2.13)$$

The simulation of the NDDE is performed by using the DDE solver

$$\hat{x}(t) = \text{DDEsolver}(\text{net}, x_0, u_t), \quad (2.14)$$

where x_0 is the initial history of the state and u_t is the sequence of control input available at time t . For autonomous systems without external inputs, we have

$$\hat{x}(t) = \text{net}(\hat{\mathbf{x}}(t)), \quad (2.15)$$

and

$$\hat{x}(t) = \text{DDEsolver}(\text{net}, x_0). \quad (2.16)$$

Learning the dynamics of the time delay system (2.2) is equivalent to sequentially adjusting the weights W_l , $l = 1, 2, \dots, L$ and biases b_l , $l = 1, 2, \dots, L$ in net as well as the delays $\hat{\tau}_k$, $k = 0, 1, \dots, d$ and $\hat{\sigma}_k$, $k = 0, 1, \dots, q$ such that the error $\|\hat{\hat{x}}(t) - \dot{x}(t)\|$ decreases. Moreover, the delays $\hat{\tau}_k$ and $\hat{\sigma}_k$ should converge to the true delays τ_k and σ_k and the TrTDNN should also give small error in $\|\hat{x}(t) - x(t)\|$.

2.2 Loss Function Design

In this section, we construct the loss functions for network training and illustrate the training algorithms based on discrete delays. We categorize the training algorithms based on whether the simulation of the NDDE is involved in the training process, which is also related to the type of loss functions, see Figure 1.3. Figure 2.2 illustrates the difference between the two loss functions.

The loss functions presented in this section are based on L_2 norm evaluation of the

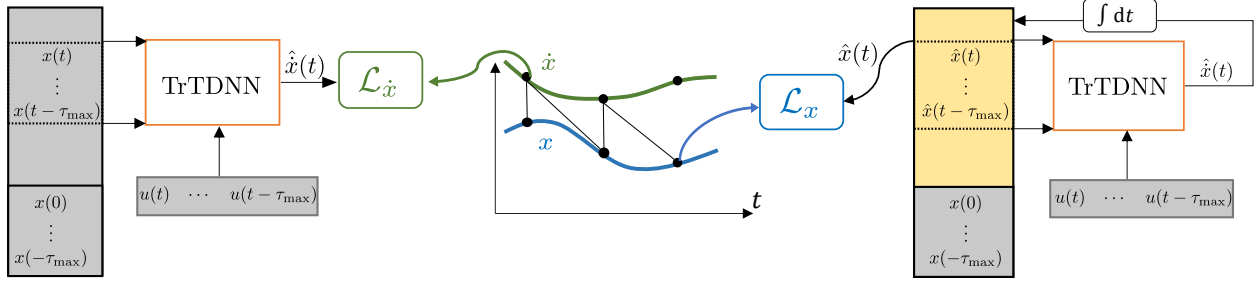


Figure 2.2: Comparison between two loss functions. The derivative loss function $\mathcal{L}_{\dot{x}}$ takes in the data directly without simulation and compares the output (state derivatives) to the ones in data (which may be calculated from numerical differentiation). The simulation loss function $\mathcal{L}_x$ compares the simulated state to the data.

difference between the prediction and data, which results in the mean squared error. In practice, one may also use other loss functions such as L_1 norm and assign weights to errors according to time, depending on the data and the application. This will be discussed later in this section and more details can be found in [59].

2.2.1 Derivative Loss

First, let us consider the derivative loss, where no simulation is performed during training and the network predicts the state derivative directly from the data. The direct map state derivative is given by

$$\hat{x}(t) = \text{net}(\mathbf{z}_0^t) = \text{net}(\mathbf{x}(t), \mathbf{u}(t)), \quad (2.17)$$

whose inputs are directly from the data. The derivative loss takes the form

$$\mathcal{L}_{\dot{x}} = \frac{1}{nN} \sum_{j=1}^N \sum_{i=1}^n \left(\hat{x}_i(t_0 + j\Delta t) - \dot{x}_i(t_0 + j\Delta t) \right)^2, \quad (2.18)$$

where x_i refers to the i -th element of vector x and N is the number of samples considered in each update (batch size). Observe that the prediction starts from $t_0 \geq \tau_{\max}$, because we need a segment of data of length $\tau_{\max}$ as the history. It is necessary to have an upper bound on the time delay $\tau_{\max}$, so that not all the data end up being used as history.

2.2.2 Simulation Loss

Alternatively we define the simulation loss

$$\mathcal{L}_x = \frac{1}{nHN} \sum_{m \in \text{batch}}^N \sum_{j=1}^H \sum_{i=1}^n \left(\hat{x}_i(t_m + j\Delta t) - x_i(t_m + j\Delta t) \right)^2, \quad (2.19)$$

where $H\Delta t$ is the simulation horizon and N is the number of segments (batch size) used for one update. To obtain $\hat{x}(t)$, one may use either the existing neural ordinary differential equation (NODE) solver by constructing augmented NODE, see Appendix D, or use the NDDE solver provided in [60] and in Appendix C.1.

2.3 Training NDDEs

Regardless of loss functions ($\mathcal{L}$ in this section can be either $\mathcal{L}_{\dot{x}}$ or $\mathcal{L}_x$), all the gradient-based training methods require the gradients of loss respect to parameters including delays. In this section we derive the gradients with respect to delays for the autonomous system (2.15) and provide the adaptive moment estimation (ADAM) method as the parameter updating rule [65]. Other parameter updating rules such as gradient descent (GD) and Levenberg-Marquardt (LM), and two more training algorithms for the derivative loss function are collected in the Appendix A. A detailed derivation for gradients is given in the Appendix B.

Consider the autonomous system (2.15), the gradient of the loss with respect to the time delay at time t is given by

$$\frac{\partial \mathcal{L}}{\partial \tau_k} = \frac{\partial \mathcal{L}}{\partial x(t - \tau_k)} \frac{\partial x(t - \tau_k)}{\partial \tau_k} = \frac{\partial \mathcal{L}}{\partial x(t - \tau_k)} (-\dot{x}(t - \tau_k)), \quad (2.20)$$

where $\frac{\partial \mathcal{L}}{\partial x(t - \tau_k)}$ is the gradient with respect to the delayed state and $\dot{x}(t - \tau_k)$ is the time derivative of x at $t - \tau_k$. The gradients of the loss with respect to the weights, biases and historical states, i.e., $\frac{\partial \mathcal{L}}{\partial W_l}$, $\frac{\partial \mathcal{L}}{\partial b_l}$ and $\frac{\partial \mathcal{L}}{\partial x(t - \tau_k)}$, can be calculated via backpropagation. After obtaining the gradients, we can update the parameters using any gradient-based training algorithms.

Here we provide the adaptive moment estimation (ADAM) algorithm devised in [65]. In the implementation, we use the `adamupdate` function from MATLAB Deep Learning Toolbox. The formula for updating each individual parameter with ADAM is given by

$$\theta_{q+1} = \theta_q - \frac{\eta}{\sqrt{\hat{v}_k} + \epsilon} \hat{m}_q, \quad (2.21)$$

with

$$\begin{aligned}\hat{m}_q &= \frac{m_q}{1 - \beta_1^q}, \\ \hat{v}_q &= \frac{v_q}{1 - \beta_2^q},\end{aligned}\tag{2.22}$$

where q is the iteration number, m_q and v_q are the first moment estimate and second moment estimate, respectively. The updates for m_q and v_q are given by

$$\begin{aligned}m_q &= \beta_1 m_{q-1} + (1 - \beta_1) \frac{\partial \mathcal{L}}{\partial \theta_q}, \\ v_q &= \beta_2 v_{q-1} + (1 - \beta_2) \left(\frac{\partial \mathcal{L}}{\partial \theta_q} \right)^2,\end{aligned}\tag{2.23}$$

using the gradient information $\frac{\partial \mathcal{L}}{\partial \theta_k}$ at iteration q . The parameters $\eta, \epsilon, \beta_1, \beta_2$ in (2.21, 2.22, 2.23) can be tuned by the user to achieve better performance; see [65].

In addition to the parameter updates with the gradient information, we restrict the delay value to be between 0 and $\tau_{\max}$ considering the positivity of the delay and the length of memory. An example of the NDDE training algorithm [60] is illustrated in Algorithm 1. One may customize the training algorithm by designing different loss functions, selecting parameter update rules and the hyper parameters, and implement parallel training techniques. We provide the [MATLAB codes](#) for some of test cases.

Note that our algorithms allow the delays to vary continuously during the learning process, therefore the value $x(t - \tau)$ may not be available directly. (The subscript of τ is dropped to simplify the notation.) Recall that the state is only observed at the time moments $t = j\Delta t$, $t = j\Delta t$, $j \in \mathbb{Z}$. Considering that $\tau \in [l\Delta t, (l+1)\Delta t)$, we may define $\alpha \in [0, 1)$ such that $\tau = (l + \alpha)\Delta t$. Then the delayed state can be approximated using linear interpolation:

$$x(t - \tau) \approx (1 - \alpha)x(t - l\Delta t) + \alpha x(t - (l + 1)\Delta t),\tag{2.24}$$

and similar procedure can be followed for the input yielding

$$u(t - \sigma) \approx (1 - \beta)u(t - l\Delta t) + \beta u(t - (l + 1)\Delta t),\tag{2.25}$$

where $\sigma = (l + \beta)\Delta t$ and $\beta \in [0, 1)$. Having a linear interpolation is equivalent to assuming a constant derivative of the state between two time moments. Thus, the derivative of the state at time $t \in [j\Delta t, (j + 1)\Delta t)$ is the same as the derivative at time $j\Delta t$, that is

$$\dot{x}(t) \approx \dot{x}(j\Delta t).\tag{2.26}$$

Goal: Learn $\theta = \{W_l, b_l, \hat{\tau}_k\}, l = 1, \dots, L, k = 1, \dots, d$ from data.

- Choose the hyper parameters according to the loss function. Choose the learning rates η_τ and η for delays and other parameters.
- Set the maximum iteration number $q_{\max}$ and the maximum allowed delay $\tau_{\max}$.
- Initialize θ_0 , e.g., initialize $\hat{\tau}_d$ uniformly between $[0, \tau_{\max}]$, b_l as zeros and W_l using Glorot initialization [36]; see Appendix C.2 for an example of initialization code.

for $q = 1, \dots, q_{\max}$ **do**

- Randomly take N segments or samples from the training data.
- Calculate the loss $\mathcal{L}$.
- Back-propagate to get the gradients with respect to parameters using `dlgradient`.
- Update θ_q from θ_{q-1} using `adamupdate` and impose positivity constraint to delays $\hat{\tau}_d$.

if $q > 1$ **and** $\mathcal{L}_q < \mathcal{L}_{q-1}$ **then**
 | $\theta_{\text{best}} = \theta_q$
end

end

$\theta = \theta_{\text{best}}$, evaluate the NDDE on testing datasets

Algorithm 1: Training algorithm for NDDE with trainable delays

In general, we assume that we only observe the state and the input at time moments $t = j\Delta t$. In this case, the state derivative can be approximated from data using finite differences. For example, using Euler method's, the state derivative can be obtained as

$$\dot{x}(j\Delta t) \approx \frac{x((j+1)\Delta t) - x(j\Delta t)}{\Delta t}, \quad (2.27)$$

but one may also use more sophisticated methods to obtain the derivative. In case the state derivative is available directly at $t = j\Delta t$, one can use those to achieve better approximation of the derivative at other time moment.

2.4 Comparisons between NDDEs of Different Architectures

Consider an autonomous delayed dynamical system which has quadratic and cubic nonlinearities in two delayed terms:

$$\dot{x}(t) = a_2 x^2(t - \tau_2) + a_3 x^3(t - \tau_3). \quad (2.28)$$

We choose the parameters $a_2 = a_3 = -1$, $\tau_2 = 1$ and $\tau_3 = 0.5$ for which the system has two equilibria at $x = 0$ and $x = -1$. We are interested in how well shallow networks with input delays can learn the nonlinearity from a limited number of trajectories and whether such networks can learn the delays in the system.

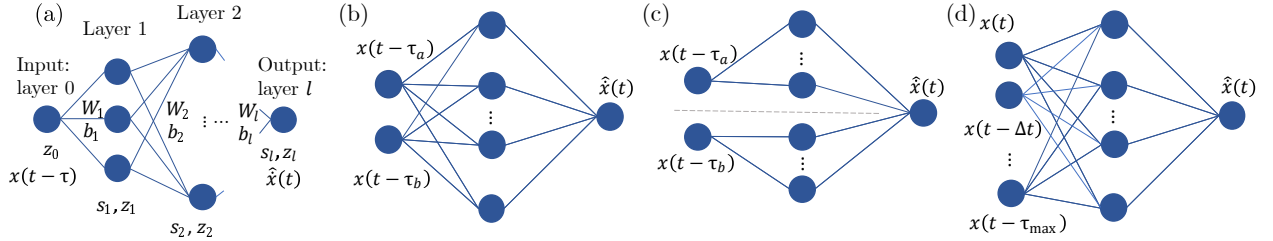
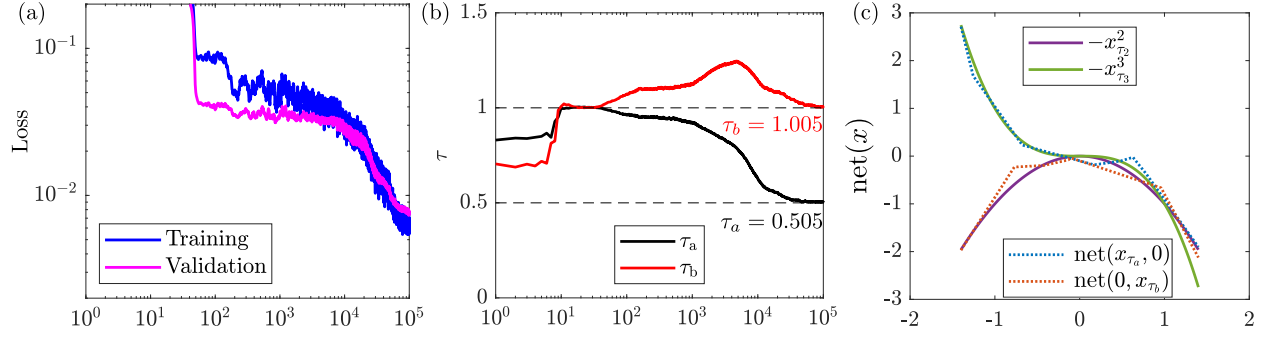


Figure 2.3: Schematic diagrams of network architectures: (a) deep neural network with one delayed input; (b) fully-connected shallow network with two trainable delayed inputs; (c) decoupled shallow network with two trainable delayed inputs; (d) conventional TDNN with a window of fixed delayed inputs.

We simulate (2.28) to obtain four batches of data for training by using constant values $c_{tr} = \{0.8, 1.0, 1.2, 1.4\}$ as histories, i.e, $x(\vartheta) \equiv c_{tr}$ for $\vartheta \in [-\tau_{max}, 0]$. Simulation data with constant history $c_{va} = 1.1$ is used for validation and we test the trained networks using the constant history $c_{ts} = 1.3$. Each batch of data contains the state information $x(t)$ over $0 \leq t \leq 5$ with a time step $\Delta t = 0.01$. We obtain the time derivative from state through the forward Euler method.

The derivative is used both as output and for calculating the gradient with respect to delay in (2.20). We normalize the input and output between $[0, 1]$ and choose the learning rate $\eta_\theta(n) \equiv 0.1$ for all parameters including delays. During training, we also include the initial constant history in the data so that the shift between inputs and outputs does not affect the length of prediction. Another reason for including the history in the training data is the fact that for autonomous delayed dynamical systems the solutions get smoother as time t increases [83, 51]. Thus, the initial segments of the solution contain the “richest” dynamics for a stable trajectory without excitation. This indicates that having longer trajectories do

Fully-connected network:



Decoupled network:

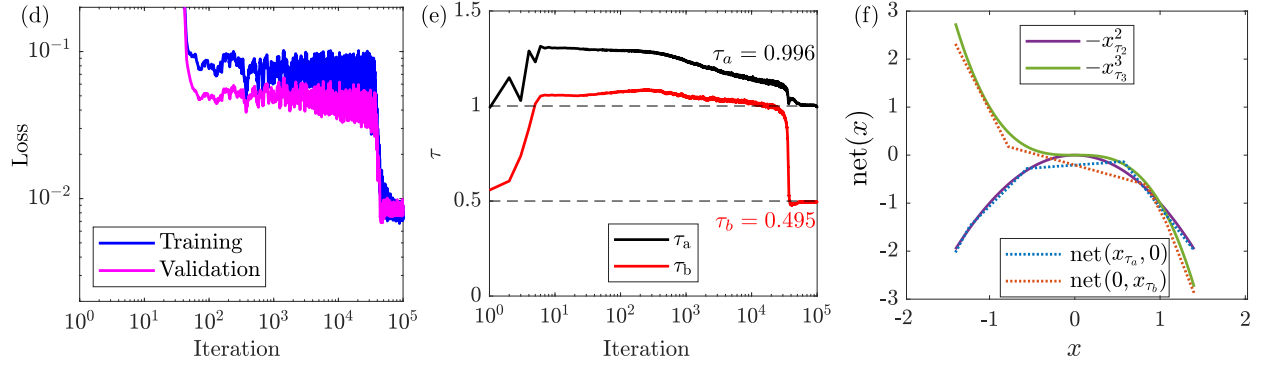


Figure 2.4: Evolution of the loss (left) and delays (middle) over the iterations during training, and the equivalent approximation F of the nonlinearities given by trained networks (right): (a)-(c) fully-connected network; (d)-(f) decoupled network.

not necessarily mean better data, as the initial segments have less weight. It also suggests that one may assign time-dependent weights in the derivative loss function.

We implement the delay training algorithm 2 with a fully-connected network and a decoupled network shown in Figure 2.3(b) and (c). Both are shallow neural networks and the difference is the structure of the hidden layer. The fully-connected network does not contain any prior knowledge about the form of (2.1), while the decoupled architecture implies that there is no cross term between delayed states. The number of neurons in the hidden layer is $l_1 = 30$ for fully-connected network and $l_1 = [20, 20]$ for the two parts of the decoupled network. Both networks use ReLU as activation function in the hidden layer, i.e., $f(z) = \max\{z, 0\}$, and we set $\tau_{\max} = 2$.

The evolution of the loss $\mathcal{L}$ and the learned delays τ_a and τ_b along the iterations of one training process is shown in Figure 2.4(a,b) for the fully-connected architecture and in Figure 2.4(d,e) for the decoupled architecture. We also show how the true nonlinearities are approximated by these two networks in Figure 2.4(c) and (f). Note that the neural networks do not distinguish the order of the two delays, that is, τ_a and τ_b can approach τ_2 and τ_3

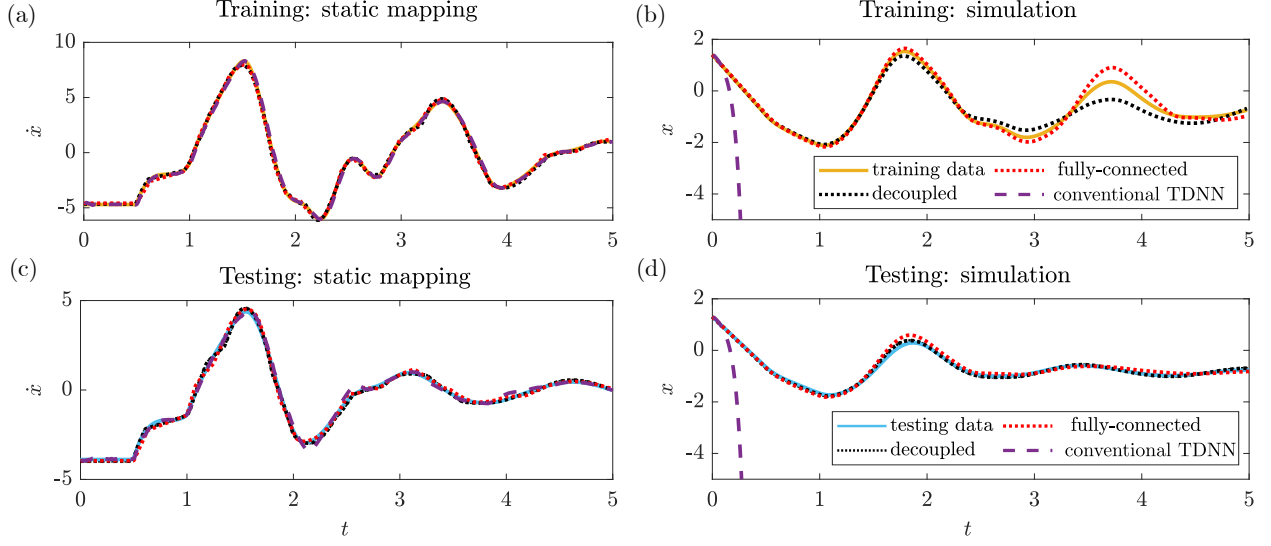


Figure 2.5: Static mapping and simulation results for the trained networks: (a), (b) performance for a trajectory with $c_{\text{tr}} = 1.4$ in the training dataset; (c), (d) performance on testing dataset with $c_{\text{ts}} = 1.3$.

or, alternatively, τ_3 and τ_2 . Nevertheless, the two delays converge to the true values 1 and 0.5 with both architectures. The delays in the input layer can be learned and they can be interpreted clearly as the trained networks represent the delayed dynamical system. The nonlinearities are also recovered from data with the meaningful learned delays. One may observe the piece-wise property of the network approximation given by the ReLU activation function in the hidden layer.

To investigate the performance of the trained networks, we analyze the system

$$\dot{x}(t) = \text{net}(x(t - \tau_a), x(t - \tau_b)), \quad (2.29)$$

with histories $c_{\text{tr}} = 1.4$ and $c_{\text{ts}} = 1.3$, where $\text{net}(\cdot)$ is the nonlinear function of the neural network given by trained weights and biases, τ_a and τ_b are the learned delays. First, we inspect the input-output relationship between $x(t)$ and $\dot{x}(t)$, which we refer to as static mapping, or direct prediction. Then, we simulate system (2.29) using the learned parameter values which we refer to as the neural network simulation results. Both of these results are compared to the those obtained from simulating the original system (2.28).

For further comparison, we also consider a conventional TDNN which uses $x(t), x(t - \Delta t), \dots, x(t - \tau_{\text{max}})$ as inputs and $\dot{x}(t)$ as output. Since $\Delta t = 0.01$ and $\tau_{\text{max}} = 2$, this TDNN has 201 inputs where 200 of them are delayed with different quantized delays. Here we use a shallow TDNN with 12 hidden neurons and ReLU activation function to learn from the same data.

		Standard TDNN	Fully-connected TrTDNN	Decoupled TrTDNN
Training	direct mapping ($\dot{x}$)	0.0229	0.0102	0.0106
	$\sqrt{\mathcal{L}_{\text{tr}}}$ Simulation (x)	∞	0.0514	0.0676
Testing	direct mapping ($\dot{x}$)	0.1453	0.0096	0.0085
	$\sqrt{\mathcal{L}_{\text{ts}}}$ Simulation (x)	∞	0.0305	0.0143
τ_a/τ_b		-	0.505/1.005	0.996/0.495

Table 2.1: Performances of different networks on training and testing trajectories. The normalized root-mean-square deviations are calculated based on $\dot{x}$ for the performance of static mapping and on x for the performance of neural network simulation.

The three trained networks are compared to the data in Figure 2.5(a) and (b) in terms of static mapping and simulation results of the trajectory with $c_{\text{tr}} = 1.4$ from training dataset. Figure 2.5(c) and (d) show the performance of the trained networks on testing data. All three networks have good static mapping performance on both training and testing dataset. In neural network simulation, the networks with two trainable delays give reasonable simulations close to the data, while the conventional TDNN with 200 delays fails (the simulation blows up). This shows that with trainable delays, the networks have the capability of approximating the true dynamics given by (2.28). Also, the simulation of the conventional TDNN with 200 delayed inputs is more time-consuming compared to the other two networks. The root-mean-square deviations on training data $\sqrt{\mathcal{L}_{\text{tr}}}$ and on testing data $\sqrt{\mathcal{L}_{\text{ts}}}$ of all trained networks are collected in Table 2.1. Since we normalize all data before training, $\sqrt{\mathcal{L}_{\text{tr}}}$ and $\sqrt{\mathcal{L}_{\text{ts}}}$ are also normalized values according to the maximum and minimum values of $\dot{x}$ and x in the training set. From the table we notice that two networks with trainable delays outperform the conventional TDNN on testing data in both static mapping and neural network simulation.

2.5 Evaluation of the Trained NDDEs

After the training is complete, one may test the accuracy of the learned nonlinearity and delays by either comparing the derivatives $\hat{\dot{x}}$ and $\dot{x}$ or comparing the solutions $\hat{x}(t)$ and $x(t)$, obtained via numerical simulation, along some time horizon. To further investigate how well the learned dynamics matches the ground truth, one may also study the stability properties and the nonlinear dynamics of the trained NDDEs.

2.5.1 Spectrum Examination

One may compare the spectrum of the trained NDDE to the one given by the true dynamics to determine how well the NDDE approximates the true dynamics locally around the equilibrium. Linearizing the dynamics (2.2) around an equilibrium we obtain

$$\dot{x}(t) = \sum_{k=0}^d A_k x(t - \tau_k) + \sum_{k=0}^q B_k u(t - \sigma_k), \quad (2.30)$$

where $A_k \in \mathbb{R}^{n \times n}$ and $B_k \in \mathbb{R}^{n \times p}$ and we used $g(0, \dots, 0) = 0$ to simplify the notation. This results in the characteristic equation

$$\det \left[\lambda I - \sum_{k=0}^d A_k e^{-\lambda \tau_k} \right] = 0, \quad (2.31)$$

see [82], which has infinitely many solution for the characteristic roots λ , see [51, 10]. Computing the rightmost characteristic roots (rightmost spectrum) allow us to compare the asymptotic behavior of the learned system to that of the ground truth.

For example, consider the system

$$\dot{x}(t) = -x(t - 1.5) + x^2(t - 1) - x^3(t - 0.5), \quad (2.32)$$

with the equilibrium at $x(t) \equiv 0$. The approximated spectrum of this system is shown in Figure 2.6 as the black crosses. For neural networks in the form of delay differential equations, similar spectrum can be obtained by linearizing the dynamics around the equilibrium and computing the characteristic roots.

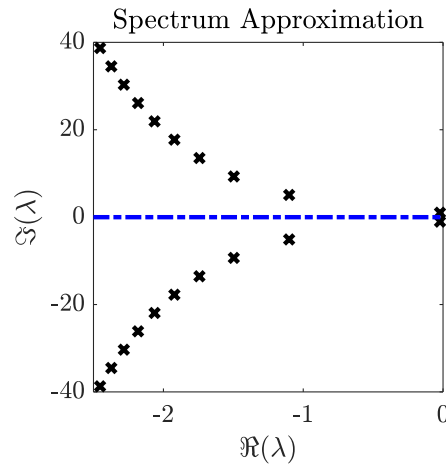


Figure 2.6: Spectrum approximation of (2.32) around its equilibrium $x(t) \equiv 0$.

2.5.2 Bifurcation Analysis

Bifurcation theory in the dynamical systems is the mathematical study of changes in the qualitative behaviors of the systems as parameters are varied. This can lead to the appearance of new equilibria and periodic orbits, the change of stability of equilibria or periodic orbits, and the birth of chaos. In many time delay systems, bifurcations occur when the delay is varied.

One may summarize the change of steady state solutions versus the change of time delays in bifurcation diagrams, which can be obtained by well-developed numerical toolboxes such as DDE-BIFTOOL [24]. For instance, in Mackey-Glass system [81],

$$\dot{x}(t) = \frac{\beta x(t - \tau)}{1 + x^\delta(t - \tau)} - \gamma x(t), \quad (2.33)$$

with $\beta = 4, \gamma = 2, \delta = 9.65$, the system's behavior changes drastically as the time delay τ is varied. When the time delay is small, for instance $\tau = 0.2$, the system has a stable equilibrium at $x(t) \equiv 1$, as shown by the numerical simulations in Figure 2.7(a). As the delay increases, the equilibrium loses the stability and a stable limit cycle rises, whose amplitude grows with the delay. Such a stable limit cycle is shown in Figure 2.7(b) for $\tau = 0.5$. Increasing the delay further, the system becomes chaotic as illustrated in panel (c) for $\tau = 1$. The appearance and stability change of equilibria and limit cycles are summarized in the bifurcation diagrams in Figure 2.7(d). The amplitude of oscillations is shown as a function of the time delay τ . The green/red color indicates the stable/unstable limit cycle.

2.5.3 Maximal Lyapunov Exponent

Lyapunov exponents represent the rate of separation of infinitesimally close trajectories in dynamical systems. Positive maximal Lyapunov exponent (MLE) $\lambda_{\max} > 0$ is used to characterize the sensitivity to initial conditions in chaotic dynamical systems [113]. Thus, we use the MLE to characterize the richness of the dynamics. In this section we introduce how to calculate the MLE for time delay systems. For more details about the numerical algorithms; see [8].

Consider a nonlinear autonomous time delay system whose time evolution is given by

$$\dot{x}(t) = \mathcal{G}(x_t), \quad (2.34)$$

where $x \in \mathbb{R}^n$ and $x_t(\vartheta) = x(t + \vartheta)$, $\vartheta \in [-\tau_{\max}, 0]$ with maximum delay $\tau_{\max} > 0$. That is, $x_t \in \mathcal{X}$ where $\mathcal{X}$ denotes the space of continuous functions $\mathcal{X} = C([- \tau_{\max}, 0], \mathbb{R}^n)$ and

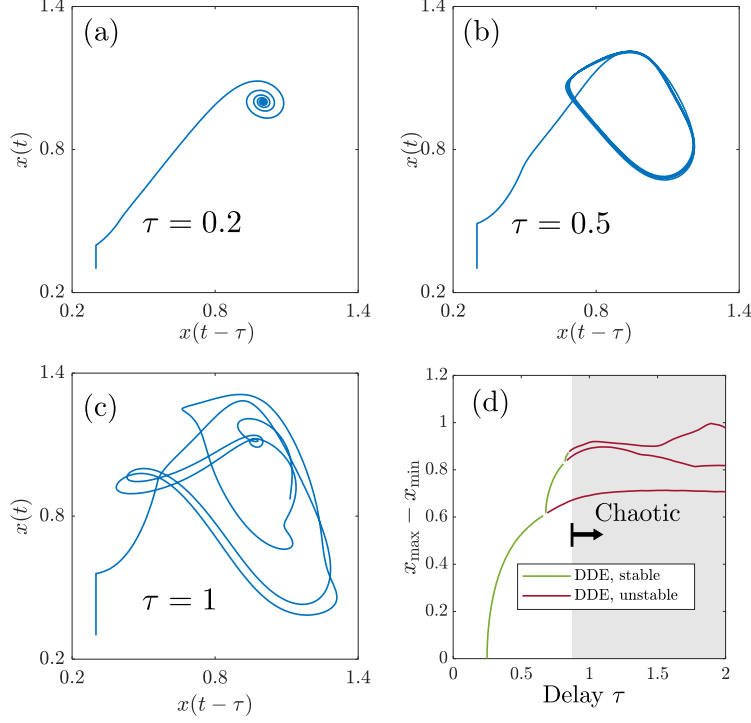


Figure 2.7: Qualitative changes in behavior when varying the time delay. (a)-(c) trajectories generated by the Mackey Glass equation for delay values $\tau = 0.2, 0.5, 1$, respectively. (d) Bifurcation diagram summarizing the change of steady state solution and periodic orbits as delay increases.

$\mathcal{G} : \mathcal{X} \rightarrow \mathbb{R}^n$ is a functional. For initial history x_0 , let us denote the solution as $\varphi(x_0, t)$, that is, $x_t(\vartheta) = \varphi(x_0(\vartheta), t)$. Then for initial history $x_0 + \delta x_0$, which is in the close vicinity of x_0 for small $\|\delta x_0\|$, we have the solution $\varphi(x_0 + \delta x_0, t)$. The MLE for the time delay system (2.34) is defined as:

$$\lambda_{\max} = \lim_{t \rightarrow \infty} \lim_{\|\delta x_0\| \rightarrow 0} \frac{1}{t} \ln \frac{\|\varphi(x_0 + \delta x_0, t) - \varphi(x_0, t)\|}{\|\delta x_0\|}, \quad (2.35)$$

where $\|\delta x_0\| = \sup_{\vartheta \in [-\tau_{\max}, 0]} \|\delta x_0(\vartheta)\|_2$ and the condition for the existence of the limit is given in [120].

By linearizing the autonomous nonlinear time delay system (2.34) about a time-dependent solution, one may obtain the linear nonautonomous system

$$\dot{x}_t = \mathcal{A}(t)x_t, \quad (2.36)$$

with the time-dependent linear operator $\mathcal{A} : \mathbb{R} \times \mathcal{X} \rightarrow \mathcal{X}$. Then, by discretizing time, this infinite-dimensional system can be reduced to a system of $M + 1$ nonautonomous linear

ordinary differential equations (ODEs):

$$\begin{bmatrix} \dot{x}_t(\vartheta_0) \\ \dot{x}_t(\vartheta_1) \\ \vdots \\ \dot{x}_t(\vartheta_M) \end{bmatrix} = \mathcal{A}_M(t) \begin{bmatrix} x_t(\vartheta_0) \\ x_t(\vartheta_1) \\ \vdots \\ x_t(\vartheta_M) \end{bmatrix}, \quad (2.37)$$

with mesh $-\tau_{\max} = \vartheta_M < \dots < \vartheta_1 < \vartheta_0 = 0$. The simplest, uniform-mesh discretization uses $\vartheta_i = ih$ with distance $h = \tau_{\max}/M$ between the mesh points; see [58]. In this dissertation, we use the Chebyshev mesh

$$\vartheta_i = \frac{\tau_{\max}}{2} \left(\cos \left(\frac{i\pi}{M} \right) - 1 \right), \quad (2.38)$$

which provides higher accuracy [10].

The Lyapunov exponents can then be calculated using discrete QR method [8]. One can factorize the fundamental matrix $X(t)$ of equation (2.37), at a strictly increasing sequence of time instances $\{t_k\}$ with $t_0 = 0$:

$$X(t_k) = Q_k R_k, \quad (2.39)$$

where Q_k is an orthogonal matrix and R_k is an upper triangular matrix. The matrix R_k can be expressed by the product of a series upper triangular matrices $R_{j,j-1}$:

$$R_k = \left(\prod_{j=1}^k R_{j,j-1} \right) R_0. \quad (2.40)$$

The matrix $R_{j,j-1}$ is computed from factorizing the solution $\psi_{t_{j-1}}(t)$ of the system

$$\dot{\psi}_{t_{j-1}}(t) = \mathcal{A}_M(t) \psi_{t_{j-1}}(t), \quad t \in [t_{j-1}, t_j], \quad (2.41)$$

with initial condition $\psi_{t_{j-1}}(t_{j-1}) = Q_{j-1}$, yielding

$$\psi_{t_{j-1}}(t_j) = Q_j R_{j,j-1}. \quad (2.42)$$

at time $t = t_j$.

The Lyapunov exponents are then recovered as

$$\lambda_i = \limsup_{k \rightarrow \infty} \frac{1}{t_k} \sum_{j=1}^k \ln [R_{j,j-1}]_{i,i}, \quad (2.43)$$

where $[R_{j,j-1}]_{i,i}$, $i = 1, \dots, M+1$ is the i -th diagonal entry of $R_{j,j-1}$ [21]. When computing (2.43), one should truncate the time at a large enough value $t_k = T$ and choose a sufficiently large mesh size M in (2.37) to obtain good approximation of λ_i for the nonlinear time delay system. With larger T and larger M , the MLE $\lambda_{\max} = \lambda_1$ can be represented with higher accuracy; see the evaluation in [8]. In later examples, the MLE $\lambda_{\max}$ is obtained with $T = 10^4$ and $M = 10$. These two parameters are chosen to balance approximation accuracy and computation efficiency. Such computation of Lyapunov exponents is also applicable for time delay systems with multiple discrete delays and even distributed delays, as long as the system can be linearized around a given solution.

2.6 Alternative Method for Learning DDEs

Sparse identification of nonlinear dynamics (SINDy) [13, 12, 17] is a data-driven framework for discovering governing equations of dynamical systems. By assuming that the dynamical system has an underlying sparse structure in terms of its right-hand-side (RHS) description, SINDy constructs interpretable models from time-series data. Assuming that the system state $x(t)$ is measured at discrete time points $t_1, \dots, t_m$ and its evolution is governed by the ODE

$$\dot{x} = f(x), \quad (2.44)$$

with an unknown RHS $f : \mathbb{R} \rightarrow \mathbb{R}$, SINDy expresses each component $f_i(x)$, $i = 1, \dots, n$, as a sparse linear combination of p candidate basis functions $\theta_j(x)$:

$$f_i(x) = \sum_{j=1}^p \theta_j(x) \xi_{j,i}. \quad (2.45)$$

Here, the functions $\theta_1, \dots, \theta_p$ serve as building blocks for approximating the underlying system behavior and they are chosen in accordance with the expected characteristics of the system. They typically include polynomials, with possible additional terms such as trigonometric functions, exponentials, rational or other nonlinear functions.

The measurement data are arranged in a matrix

$$X := \begin{bmatrix} x_1(t_1) & \cdots & x_n(t_1) \\ \vdots & \ddots & \vdots \\ x_1(t_m) & \cdots & x_n(t_m) \end{bmatrix} \in \mathbb{R}^{m \times n},$$

where rows correspond to time points and columns to state variables. Similarly, a matrix

$$\dot{X} := \begin{bmatrix} \dot{x}_1(t_1) & \cdots & \dot{x}_n(t_1) \\ \vdots & \ddots & \vdots \\ \dot{x}_1(t_m) & \cdots & \dot{x}_n(t_m) \end{bmatrix} \in \mathbb{R}^{m \times n},$$

is defined to collect the derivative values, which may be directly available from measurements or estimated using numerical differentiation techniques. A library matrix $\Theta(X) \in \mathbb{R}^{m \times p}$ is introduced, containing the p candidate basis functions evaluated at all m time points. For instance, a polynomial library with additional trigonometric terms is defined as

$$\Theta(X) := \begin{bmatrix} 1 & X & X^2 & \cdots & X^D & \cdots & \sin(X) & \cos(X) & \cdots \end{bmatrix}.$$

Above, X^d denotes a matrix containing all possible polynomial combinations of degree D in the components of x . For example, for $D = 2$,

$$X^2 := \begin{bmatrix} x_1^2(t_1) & x_1(t_1)x_2(t_1) & \cdots & x_n^2(t_1) \\ x_1^2(t_2) & x_1(t_2)x_2(t_2) & \cdots & x_n^2(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1^2(t_m) & x_1(t_m)x_2(t_m) & \cdots & x_n^2(t_m) \end{bmatrix}.$$

Similarly, for trigonometric functions we have

$$\sin(X) := \begin{bmatrix} \sin(x_1(t_1)) & \cdots & \sin(x_n(t_1)) \\ \vdots & \ddots & \vdots \\ \sin(x_1(t_m)) & \cdots & \sin(x_n(t_m)) \end{bmatrix}.$$

The total number of basis functions of a polynomial library of degree D for a system of physical dimension n is $p = \binom{n+D}{D}$.

To determine the coefficients $\xi_{j,i}$ in (2.45) we assemble them into vectors $\xi_i \in \mathbb{R}^p$ for each

$i = 1, \dots, n$ and the overall problem is formulated as

$$\dot{X} = \Theta(X)\Xi, \quad (2.46)$$

where $\Xi \in \mathbb{R}^{p \times n}$ is the matrix $\Xi = [\xi_1 \ \xi_2 \ \dots \ \xi_n]$ with columns ξ_i , $i = 1, \dots, n$. The coefficients for each column ξ_i are identified by solving independently the following optimization problem:

$$\xi_i = \operatorname{argmin}_{\xi \in \mathbb{R}^p} \left(\left\| \dot{X}_i - \Theta(X)\xi \right\|_2 + \lambda \|\xi\|_1 \right), \quad (2.47)$$

where $\dot{X}_i$ is the i -th column of $\dot{X}$.

In the case of DDEs with a finite number of constant discrete delays, the SINDy framework can be extended by incorporating delayed copies of the state variables into the library functions [68, 95, 107]. Explicitly, the problem can be reformulated as:

$$\dot{X} = \Theta(X, X_\tau)\Xi(\tau), \quad (2.48)$$

where X_τ is a shorthand for $X_{\tau_1}, \dots, X_{\tau_d}$, with

$$X_{\tau_i} := \begin{bmatrix} x_1(t_1 - \tau_i) & \dots & x_n(t_1 - \tau_i) \\ \vdots & \ddots & \vdots \\ x_1(t_m - \tau_i) & \dots & x_n(t_m - \tau_i) \end{bmatrix},$$

for $i = 1, \dots, d$, containing the delayed samples $x(t - \tau_i)$. The augmented library matrix $\Theta(X, X_\tau) \in \mathbb{R}^{m \times q}$ is constructed by combining basis functions of both the current data and the delayed data, e.g.,

$$\Theta(X, X_\tau) := \begin{bmatrix} 1 & X & X_\tau & X^2 & XX_\tau & X_\tau^2 & \dots & \sin(X) & \sin(X_\tau) & \dots \end{bmatrix}.$$

The sparse regression problem (2.48) now depends on the known delay values $\tau_1, \dots, \tau_d$ and is solved column-wise:

$$\xi_i(\tau) = \operatorname{argmin}_{\xi \in \mathbb{R}^q} \left(\left\| \dot{X}_i - \Theta(X, X_\tau)\xi \right\|_2 + \lambda \|\xi\|_1 \right). \quad (2.49)$$

The notation $\Xi(\tau)$ highlights the dependence of the sparse solution Ξ on the given vector of delay values $\tau := [\tau_1, \dots, \tau_d]^\top$. Note that including delays increases the dimensionality of a polynomial library, with the number of terms growing from $\binom{n+D}{D}$ to $\binom{(d+1)n+D}{D}$ when k delays are considered with polynomials of degree D . The corresponding reconstruction error

is denoted as

$$\epsilon(\tau) := \|\dot{X} - \Theta(X, X_\tau)\Xi(\tau)\|_2. \quad (2.50)$$

This error can be used to optimize the delay value and hyper parameters of the basis functions in the library. One may call external optimizer such as particle swarm optimization (PS) or Bayesian optimization (BO) to find the optimal delay values and hyper parameters by minimizing $\epsilon(\tau)$. This method assumes the number of delays is known and directly recover the DDE, therefore it is called expert SINDy (E-SINDy).

Pragmatic SINDy (P-SINDy) method is introduced to learn the ODE discretization of the DDE, when the number of delays is unknown [7]. This method reduces the problem to a finite-dimensional systems of ODE, through pseudospectral collocation in [9]. Unlike E-SINDy, which relies on prior knowledge of delay parameters to reconstruct an interpretable RHS, P-SINDy adopts a “black-box” approach, aiming only at matching the observed trajectory and then simulating beyond the measurements time span. While P-SINDy sacrifices the interpretability of the reconstructed RHS, it requires no prior information on the number or values of delays and thus achieves enhanced computational efficiency by possibly optimizing only the maximum delay $\bar{\tau}$. One may choose the dimension of the ODE ($M + 1$), and the size of a corresponding polynomial library of degree D becomes $\binom{(M+1)n+D}{D}$.

2.7 Summary

In this chapter, we established the link between time delay systems and time delay neural networks. Using a time delay neural network (TDNN) to approximate the RHS of a DDE results in the NDDE. We proposed to use time delay neural networks with trainable delays (TrTDNN) instead of the conventional TDNN with fixed delays. Most of the real-world time delay systems have meaningful but limited number of time delays, which can be captured by TrTDNNs in continuous time with much fewer delayed inputs at the right values. We introduced derivative and simulation loss functions and provided an example of learning two delays with nonlinear terms to demonstrate the advantage of using TrTDNN over conventional TDNN. With our proposed approach, the architecture of the corresponding neural networks can be greatly simplified and the chance of converging to undesired local minima is reduced. We introduced three evaluation methods for NDDEs, which will be implemented on the examples in Chapter 3. We also introduced alternative data-driven method called SINDy for learning DDEs. We will compare our proposed method to the SINDy on the example of learning double delayed Rössler system in Chapter 3.

CHAPTER 3

Examples of Learning Time Delay Systems from Trajectory Data

In this chapter, we show the performance of the proposed learning methods on various types of nonlinear time delay systems, including scalar autonomous systems with multiple delays, systems with chaotic behavior induced by a single delay, a system with external input and multiple-state autonomous systems. Several of these systems are widely employed in the literature to model biological processes (e.g., the hematopoiesis), climate changes (e.g., the El Niño phenomenon), and ecological systems (e.g., predator-prey dynamics). Others hold significant importance in mathematics and physics, particularly in the study of chaotic behavior.

3.1 Simple Nonlinear Autonomous Systems

First, we consider the following nonlinear autonomous systems with multiple delays:

$$\dot{x}(t) = -x(t) + x(t-1)x(t-0.5) - x^3(t-0.5), \quad (3.1)$$

and

$$\dot{x}(t) = -x(t-1.5) + x^2(t-1) - x^3(t-0.5). \quad (3.2)$$

In system (3.1), there are two delays (not including the delay-free term) and a “cross-term” containing both delays. In system (3.2), there are three delays and no delay-free term. These systems are constructed for testing the proposed learning algorithm with multiple time delays. In both cases, we use the same network architecture (one-hidden-layer TrTDNN with three input delays and only three neurons in the hidden layer) which can be represented

as

$$\hat{x}(t) = W_2 \operatorname{sig}\left(W_1 \begin{bmatrix} \hat{x}(t - \hat{\tau}_1) \\ \hat{x}(t - \hat{\tau}_2) \\ \hat{x}(t - \hat{\tau}_3) \end{bmatrix} + b_1\right), \quad (3.3)$$

to learn the dynamics from simulation data, where $W_1 \in \mathbb{R}^{3 \times 3}$, $W_2 \in \mathbb{R}^{1 \times 3}$, $b_1 \in \mathbb{R}^{3 \times 1}$ and $\operatorname{sig}(z) = 1/(1 + e^{-z})$.

The training dataset contains seven trajectories created by numerical simulation using the constant histories $x(\vartheta) \equiv c$, $\vartheta \in [-\tau_{\max}, 0]$ where $c = \{1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6\}$. The simulated trajectory with constant history $c = 1.25$ is used for validation and we test the performance on the trajectory with constant history $c = 1.35$. The time step in all datasets is $\Delta t = 0.01$. The initial time delays are chosen between 0 and $\tau_{\max} = 2$. Both the GD and LM methods were tested with Algorithm 3 and the LM method converges much faster. Thus, the results are presented here for LM method with $\mu = 0.01$. During training, we start from ten different initializations and choose the best result from them as our final trained results. We show below that more attempts in training provide better convergence in the time delay. As a trade-off, having more attempts makes the training process slower.

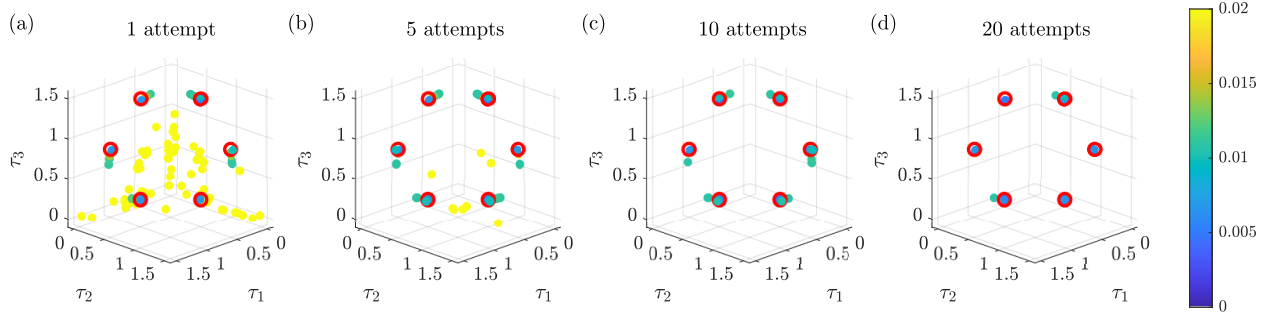


Figure 3.1: Convergence of learned delays in 100 runs: (a) 1 attempt; (b) 5 attempts; (c) 10 attempts; (d) 20 attempts. The color indicates the training loss.

To illustrate the necessity of using multiple attempts in the gradient-based training algorithms, we show the final learned time delays with different number of attempts in each training in Figure 3.1 for system (3.2). 100 runs of training are performed and the final delays are shown in the 3D plot while the color bar shows the final training error. Panel (a) shows that while training the networks multiple undesired local minima can be discovered. As we increase the number of attempts, we explore more places in the parameter space and some of these local minima can be filtered out. We can see that with 10 attempts in the training process, the delays converge to the true values. That is, multiple initializations in the training help to search globally. At the same time, the gradient information obtained from approximate state derivatives helps in local exploitation.

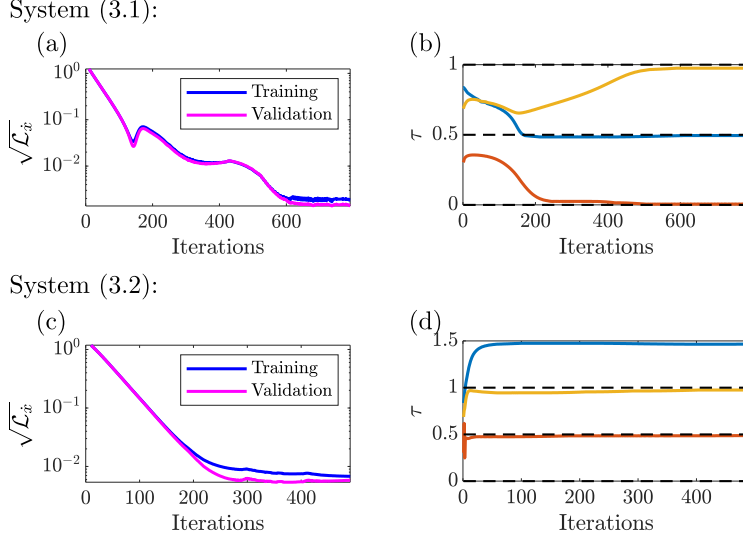


Figure 3.2: Normalized RMSE and learned delays along training iterations for system (3.1) (a)–(b), and system (3.2) (c)–(d). While only first 700 iterations are shown in the figure, the actual training stops later based on the validation loss criterion.

The training iterations of the selected attempt in one run are shown in Figure 3.2 for systems (3.1) and (3.2). Panels (a) and (c) show that the loss decreases rapidly within a few hundred iterations for both systems. Panels (b) and (d) depict the learning paths of the time delay parameters. Observe that the delays converge to the true values, i.e., to 0, 0.5 and 1 in case of (3.1) and to 0.5, 1 and 1.5 in case of (3.2). To highlight the performance of the trained TrTDNN, we display the simulation results of the trained networks in Figure 3.3 and in Figure 3.4 for training data and testing data, respectively. Static mapping refers to the input-output mapping at each time step, where the inputs are always from the data; cf. (2.17) and Figure 2.1(b). Simulation refers the case when the neural delay differential equation is simulated with given initial history; cf. (2.16) and Figure 2.1(c).

A conventional one-hidden-layer TDNN is also trained using MATLAB Deep Learning Toolbox with LM method for comparison. The conventional TDNN also has three neurons in the hidden layer and the same sigmoid activation function $\text{sig}(\cdot)$, but with 201 inputs, i.e., $\mathbf{x}(t) = [x(t), x(t - 0.01), \dots, x(t - 2)]^\top$ and $W_1 \in \mathbb{R}^{201 \times 3}$, compared to the 3 inputs with 3 trainable delays in the TrTDNN. The quantitative analysis of the training and testing performances are collected in Table 3.1. Since the conventional TDNN has a large number of inputs, the complexity of the network is much larger, even though the number of neurons in the hidden layer is the same as for the TrTDNN. Thus, the conventional TDNN can achieve a better fitting in terms of static mapping, i.e., capture $\dot{x}$ better. However, when the networks are used for simulation, the conventional TDNN fails to capture x due to

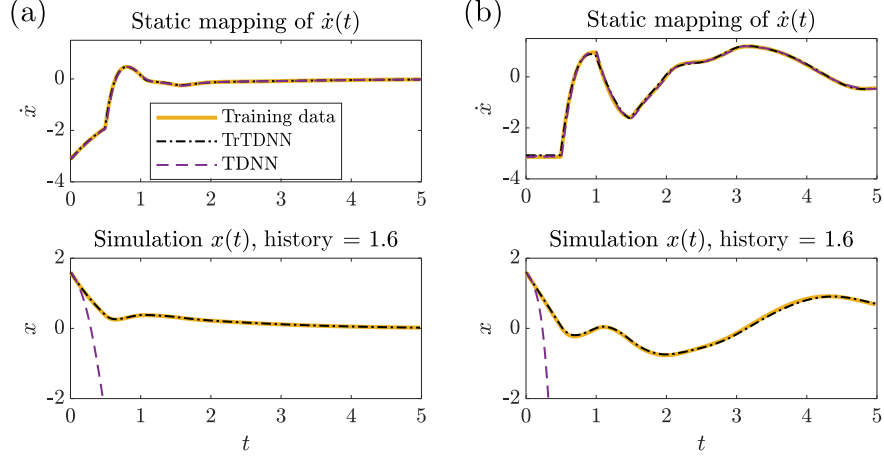


Figure 3.3: Performance of TrTDNN and conventional TDNN on the training data for system (3.1) (a), and for system (3.2) (b).

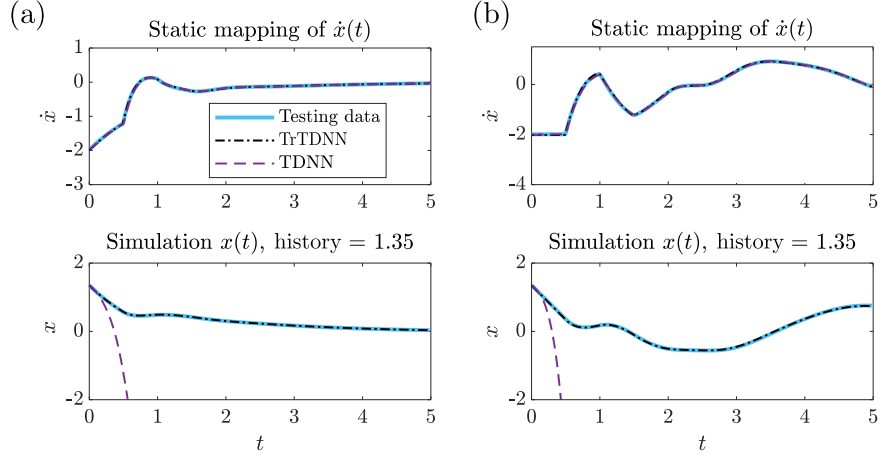


Figure 3.4: Performance of TrTDNN and conventional TDNN on the test data for system (3.1) (a), and for system (3.2) (b).

overfitting, while the TrTDNN is able to reproduce it with high accuracy. This highlights advantages of the sparsity and trainability of the proposed TrTDNN.

To emphasize that the trained TrTDNN can capture the true dynamics we also demonstrate that it reproduces the nonlinearities in the system as well as the spectrum when linearizing around equilibria. Since the delayed components in system (3.2) are decoupled, we plot them separately as shown by the solid curves in Figure 3.5(a). Also, by activating each delayed channel in the network inputs, we can plot the nonlinear approximation given by the trained network, as shown by the dashed curves. Thus, with sigmoid function $\text{sig}(\cdot)$ and only three neurons in the hidden layer, the TrTDNN can learn the nonlinearities and the delays simultaneously. In order to check the transient dynamics around the equilibrium

RMSE		System (3.1)		System (3.2)	
		TDNN	TrTDNN	TDNN	TrTDNN
Training ($c = 1.6$)	$\dot{x}$	0.0081	0.0177	0.0176	0.0381
	x	26.2978	0.0069	31.8429	0.0266
Testing ($c = 1.35$)	$\dot{x}$	0.0046	0.0071	0.0115	0.0183
	x	25.6948	0.0029	28.8596	0.0095

Table 3.1: Training and testing results for the two systems

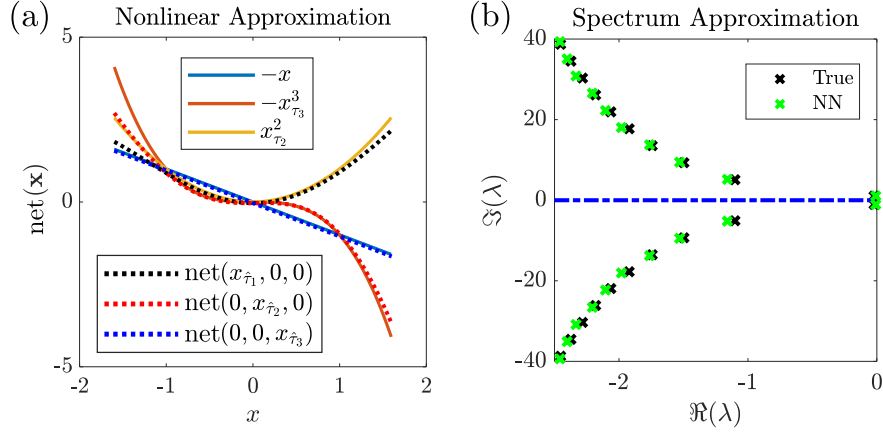


Figure 3.5: (a) Approximation the nonlinearities in (3.2). Solid curves represent the ground truth while the dashed curves are extracted from the TrTDNN. (b) Approximation of the spectrum in (3.2). Black crosses show the ground truth while the green crosses are from the trained network.

$x(t) \equiv 0$ in system (3.2), we compare the spectrum, i.e., the eigenvalues of the trained network and that of the original system in Figure 3.5(b). The spectrum given by the trained network is very close to the ground truth, i.e., to the solutions of $\lambda + e^{-1.5\lambda} = 0$. That is, the TrTDNNs approximates the underlying transient dynamics well.

3.2 Delayed Logistic Equation

The delayed logistic equation, also known as the Hutchinson equation [48], is one of the most well-known DDEs. It is given by

$$\dot{x}(t) = rx(t) \left(1 - \frac{x(t-\tau)}{K} \right), \quad (3.4)$$

where $r > 0$ is the intrinsic growth rate of the population, $K > 0$ is the carrying capacity and $\tau > 0$ represents the time delay in the feedback mechanism corresponding to internal competition. The qualitative behavior of the solution of the corresponding IVP depends on r and τ : monotonic when $r\tau < 1/e$ and oscillatory when $1/e < r\tau < \pi/2$. In both scenarios, the solution converges to the unique nontrivial equilibrium point $\bar{x} = K$ as $t \rightarrow +\infty$ and K also determines how fast the system converges. At the critical parameter value $r\tau = \pi/2$ a Hopf bifurcation [70] occurs and for $r\tau > \pi/2$ the solution tends to a limit cycle. The limit cycle can be represented using the Lambert W function and, as r increases, the structure of the cycle varies, resulting in sections with steep derivatives that are interspersed with plateaus [39].

By choosing $K = 1, \tau = 1$ and initial history $x(\vartheta) = \cos(\vartheta)$, we generate simulation data with a sampling time $\Delta t = 0.5$. The first $T_{\text{tr}} = 30$ data including the initial history are used as training dataset and the next $T_{\text{st}} = 5$ is used as testing data to examine the performance of trained model. The data is shown in Figure 3.6 (a) as a function of time and the training data is also demonstrated in panel (b) in state space.

For this problem, we use NDDE with one trainable delay in the form of

$$\hat{x}(t) = W_3 \tanh \left(W_2 \tanh \left(W_1 \begin{bmatrix} \hat{x}(t) \\ \hat{x}(t - \hat{\tau}) \end{bmatrix} + b_1 \right) + b_2 \right), \quad (3.5)$$

where $W_1 \in \mathbb{R}^{l_1 \times 2}$, $W_2 \in \mathbb{R}^{l_2 \times l_1}$, $W_3 \in \mathbb{R}^{1 \times l_2}$, $b_1 \in \mathbb{R}^{l_1 \times 1}$, $b_2 \in \mathbb{R}^{l_2 \times 1}$ and l_1, l_2 are the number of neurons in each hidden layer. Algorithm 1 with derivative loss $\mathcal{L}_{\dot{x}}$ is implemented to train the NDDE from one trajectory. The training results for different sizes of the network are compared in Section 3.8 and here we present the results for $l_1 = l_2 = 10$.

The initial delay of the NDDE is set to be 0.5 and it converges to the true value $\tau = 1$ during training, see panel (c). The maximum training iteration is $q_{\text{max}} = 8000$ for this example and the dotted vertical lane indicates the iteration with minimum training loss (yielding the parameters θ_{best}). Panel (d) shows the simulations of the trained NDDE with parameters $\theta = \theta_{\text{best}}$ on testing dataset, i.e, only the data before the dashed vertical line is provided as history in the simulation. Moreover, the direct prediction from the testing data to the state derivatives by the trained NDDE is shown in panel (e), where $\hat{x}(t) = \text{net}(x(t), x(t - \hat{\tau}))$. Note that the network direct prediction does not solve the differential equation, therefore the NDDE prediction in the panel (e) is not the numerical differentiation of the simulation result in panel (d).

When the dynamics of the data is not rich enough, the NDDE can possibly learn a “wrong” delay while still recover the data in both simulation and direct derivative prediction. To demonstrate this, we generate another trajectory using $K = 10, \tau = 1$, where the training

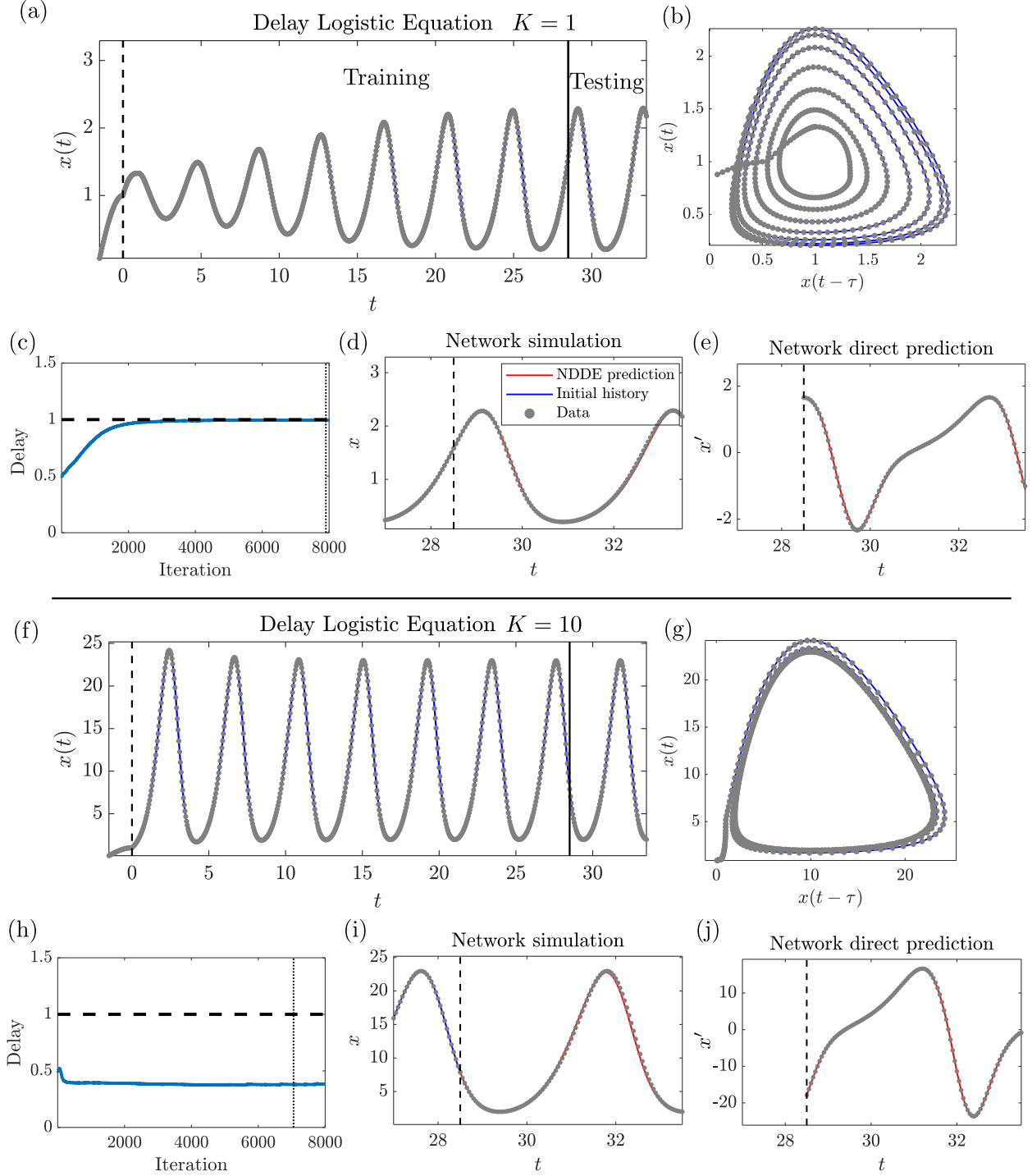


Figure 3.6: Training the NDDE (3.5) (2 hidden layers and 10 neurons in each layer, i.e., $l_1 = l_2 = 10$) with two different datasets of delayed logistic equation.

data is concentrated around the limit cycle, although the length of the data is the same. The training data is presented in Figure 3.6(f) and (g) and the convergence of the learned delay is shown in panel (h). Without any prior knowledge or restrictions on the network design, the delay $\tau = 1$ is not learned correctly. However, the network simulation and direct prediction are both close to the testing data, as shown in panels (i) and (j). This means that both simulation and derivative loss surface has local minima at this delay. From the point of view of approximating the data, the algorithm delivers good results while the truth can still remain hidden.

3.3 Mackey-Glass Dynamics

Now we examine the NDDE approach when learning the dynamics of the Mackey-Glass system [81], a scalar autonomous time delay system modeled by the equation

$$\dot{x}(t) = \frac{\beta x(t - \tau)}{1 + x^\delta(t - \tau)} - \gamma x(t), \quad (3.6)$$

with $\beta = 4, \gamma = 2, \delta = 9.65$. This mathematical model is used to describe the complex behavior in physiological systems with delayed effects, for example, the control of arterial CO_2 concentration and the regulation of hematopoiesis. This is a classical time delay system, known for the simplicity of its structure and for the complexity of its behavior. It exhibits different qualitative behaviors as the time delay τ is varied.

We use a time delay neural network with 2 hidden layers, 5 neurons in each layer, tanh nonlinearity, and 1 trainable delay to learn from four different datasets, shown in Figure 3.7. The data generated by (3.6) is captured by the NDDE

$$\hat{x}(t) = W_3 \tanh \left(W_2 \tanh \left(W_1 \begin{bmatrix} \hat{x}(t) \\ \hat{x}(t - \hat{\tau}) \end{bmatrix} + b_1 \right) + b_2 \right), \quad (3.7)$$

where $W_1 \in \mathbb{R}^{5 \times 2}$, $W_2 \in \mathbb{R}^{5 \times 5}$, $W_3 \in \mathbb{R}^{1 \times 5}$, $b_1, b_2 \in \mathbb{R}^{5 \times 1}$.

Note that the datasets shown in Figure 3.7 are not combined during training, i.e., only a single trajectory is used in each case. By doing so, we examine the effect of datasets in the training procedure as well as the generalizability. The results presented in this section are obtained by implementing Algorithm 1 with simulation loss $\mathcal{L}_x$, while the comparison between the simulation loss and the derivative loss and study on different size of networks are presented later in Section 3.8. The training horizon is chosen to be $H = 10, H\Delta t = 1$ and the simulation time step in the DDEsolver is $\delta t = 0.01$. The learning rate for weights

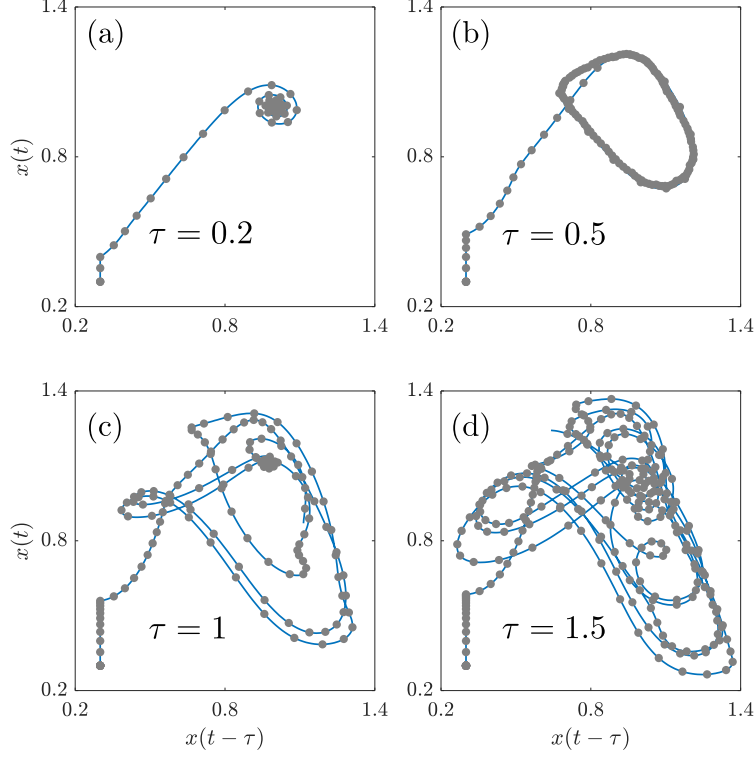


Figure 3.7: Training datasets generated by the Mackey-Glass system: each dataset contains one trajectory generated for a different delay value using the initial history $x(\vartheta) \equiv 0.3, \vartheta \in [-\tau_{\max}, 0]$ sampled with time step $\Delta t = 0.1$. (a)-(c) has the same length $T_{\text{tr}} = 15$, while (d) has the length $T_{\text{tr}} = 30$ since the richer dynamics takes more time to present.

and biases is chosen to be 0.01. The learning rate for delay is 0.01 for the limit cycle dataset (generated at $\tau = 0.5$) and for the chaotic datasets (generated at $\tau = 1, 1.5$), but it is 0.001 for the stable equilibrium dataset (generated at $\tau = 0.2$), because the target delay value is smaller. The batch size $N = 10$ is used for training, while the maximum number of iterations is set to be 2000 for the datasets generated at $\tau = 0.5, 1$ and 5000 for the datasets generated at $\tau = 0.2, 1.5$. The networks with smallest training loss along the iterations is used as final trained network for prediction and evaluation.

Algorithm 1 with simulation loss $\mathcal{L}_x$ performs well on the all four datasets. The trained neural network captures the trajectory for the training data and accurately predicts the future trajectory for a short time period. With a larger simulation horizon $H\Delta t$, the network can be used to predict the chaotic oscillations more accurately for a longer period. However, using larger horizon significantly increases the training time, and the prediction error still increases in the longer term due to the sensitivity to initial conditions in chaotic systems. To examine the performance of the trained networks, we test them for different initial histories

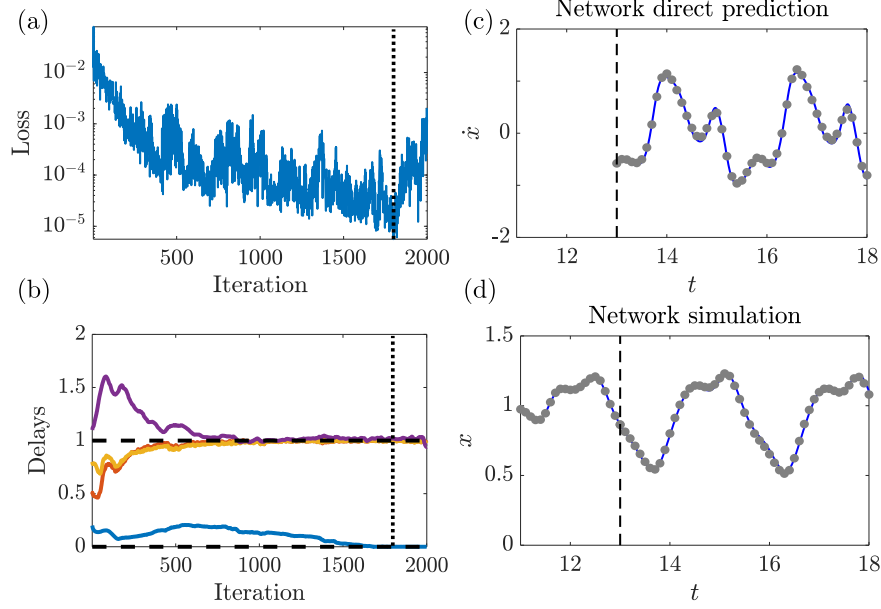


Figure 3.8: Learning the Mackey-Glass dynamics. (a) Training loss along the iterations. (b) Learned delays along the iterations. (c) Derivative prediction on test data. (d) Simulation prediction on test data.

and compare the MLE of the data as with the MLE of the trajectories given by the NDDE. We use bifurcation diagrams to demonstrate that rich data (with positive MLE) yields good generalizability. When the bifurcation diagram of the approximated system matches the bifurcation diagram of the original system, one may use the trained NDDE outside the domain of training data.

There are cases when the number of delays in the system is unknown. A demonstration of the developed algorithm under such a scenario is shown in Figure 3.8. Here we initially overestimate the number of delays to be four while there is only one delay in the system. The corresponding NDDE is now given by

$$\hat{x}(t) = W_3 \tanh \left(W_2 \tanh \left(W_1 \begin{bmatrix} \hat{x}(t - \hat{\tau}_1) \\ \hat{x}(t - \hat{\tau}_2) \\ \hat{x}(t - \hat{\tau}_3) \\ \hat{x}(t - \hat{\tau}_4) \end{bmatrix} + b_1 \right) + b_2 \right), \quad (3.8)$$

where $W_1 \in \mathbb{R}^{5 \times 4}$ and the dimensions of the other weights and biases are the same as in (3.7). As a result, some of the delays converge to the true value $\tau = 1$, while others converge to zero (representing the nondelayed terms). The training loss and learned delays are shown in panels (a) and (b), respectively. The dotted line indicates the minimum training error

where the parameters used in the NDDE (3.7) are taken. The direct network prediction of the derivative $\hat{x}$ and the simulated state $\hat{x}$ are shown as a function of time in panels (c) and (d) as blue curves, in comparison with the test data plotted as gray dots.

We show the learning path of the delay and the performance of the trained networks (3.7) on test data in Figure 3.9. The performance of the network trained on limit cycle data (generated at $\tau = 0.5$) is presented in orange on the left, and the performance of the network trained on chaotic data (generated at $\tau = 1$) is presented in yellow on the right. Panels (a)-(b) show how the delay value is learned along the training iterations. In both cases the delay starts from initial value $\hat{\tau} = 0.5\tau$ and approaches the correct delay value τ through the iterations. Both networks are tested on initial history $x(\vartheta) \equiv 0.9$ that differs from the initial history $x(\vartheta) \equiv 0.3$ used for generating the training data; see panels (c)-(f). One may observe from the phase portraits in panels (c) and (f) that both networks give good predictions for the delay value they are trained at.

The trained NDDEs are also tested for another delay value, without retraining, i.e., by keeping the same trained weights and biases; see the phase portraits in panels (d) and (e). The network trained on chaotic data at $\tau = 1$ can predict the limit cycle observed for $\tau = 0.5$, but the network trained on limit cycle data fails to predict the chaotic behavior observed for $\tau = 1$. This indicates that the networks trained on datasets with richer dynamics generalize better. Moreover, we present the difference of the state derivative between the original system and the NDDE in panels (g) and (f). Observe that the network trained on chaotic data $\tau = 1$ has a better approximation of the nonlinearity, as indicated by the blue color.

τ (DATA)	0.2	0.5	1	1.5
$\hat{\tau}$ (NDDE)	0.18	0.5	1	1.5
$\lambda_{\max}$ (DATA)	-0.66	-7.3×10^{-4}	0.143	0.155
$\lambda_{\max}$ (NDDE)	-0.77	-4.7×10^{-5}	0.140	0.153

Table 3.2: Delays and maximal Lyapunov exponents of the original Mackey-Glass system and the trained NDDEs for the four data sets in Figure 3.7 generated for initial history $x_0 \equiv 0.3$, while using mesh size $M = 10$ and truncation time $T = 10^4$.

The richness of the training data affects the learning performance. To evaluate the richness, we calculate the maximal Lyapunov exponent MLE of the original data and compare it to the MLE of the trajectories generated by the trained networks. The calculation follows the method described in Section 2.5.3. For the four networks trained on the datasets shown in Figure 3.7, the delays and MLEs are collected in Table 3.2. All networks fit their corresponding training data, thus, the delays and the MLEs are recovered by the NDDEs.

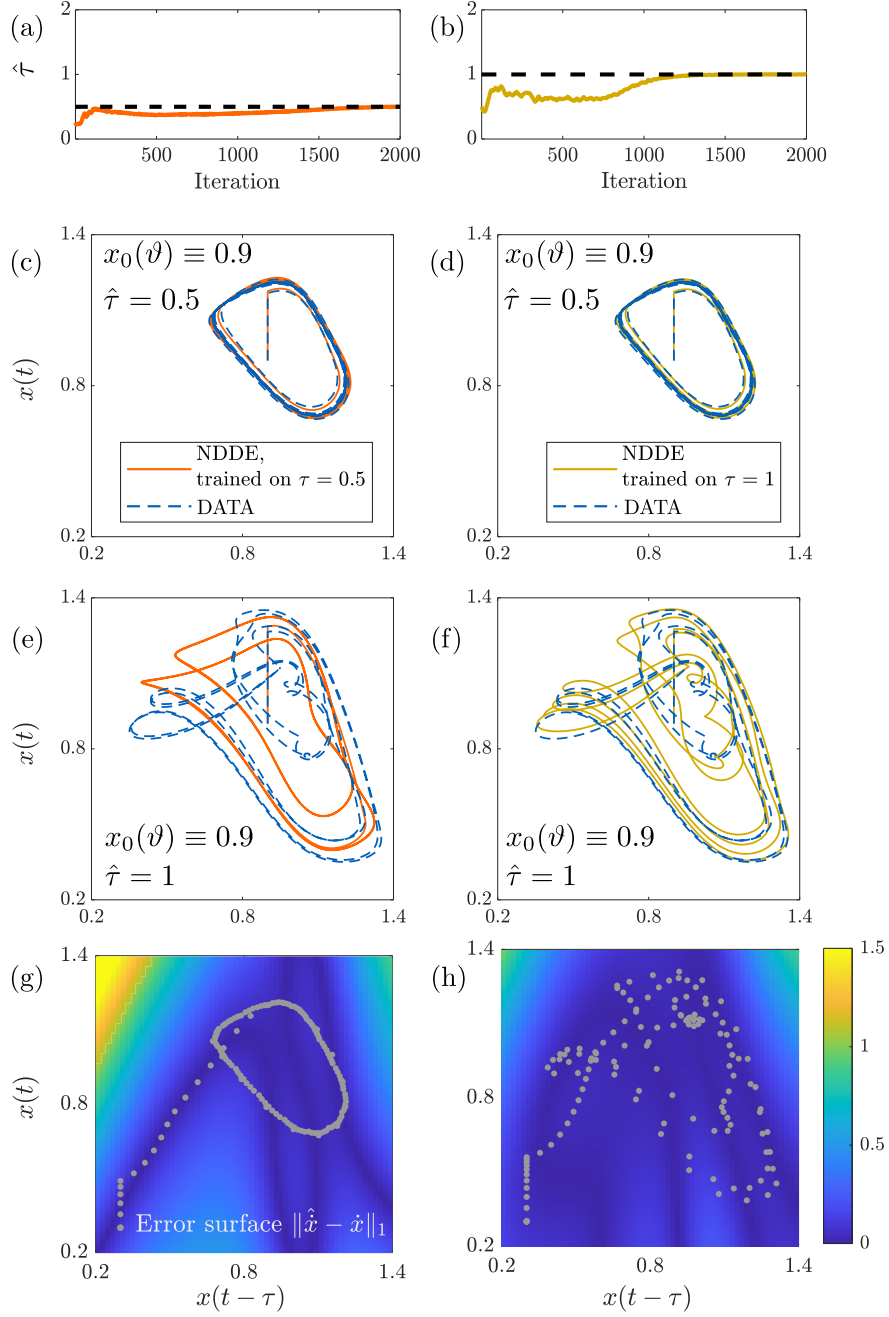


Figure 3.9: Results when the networks are trained on the limit cycle dataset $\tau = 0.5$ (left) and the chaotic dataset $\tau = 1$ (right). (a)-(b) Learning the delays from the training data. (c)-(f) Comparison of network performances on test data. (g)-(h) Absolute error between $\dot{x}$ and $\hat{x}$ in state space where the grey dots indicate training data.

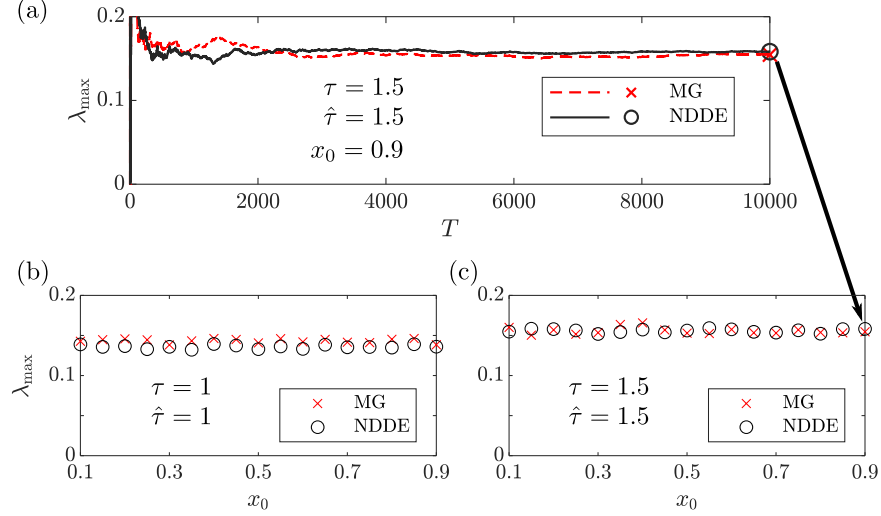


Figure 3.10: Maximal Lyapunov exponents computed for the Mackey-Glass system and the trained NDDE. Panel (a) shows the MLEs as a function of the truncation time T for delay $\tau = 1$ and history $x_0 \equiv 0.9$. Panels (b) and (c) show the exponents at $T = 10^4$ for delays $\tau = 1$ and $\tau = 1.5$, respectively, while considering the initial histories $x_0 \equiv c, c \in [0.1, 0.9]$ in the testing data.

The MLE and the learned delay for the network trained on stable equilibrium data are less accurate as most of the stable trajectory is around the equilibrium, which makes the learning difficult.

The computation of the MLE requires the simulation of the system with a given initial history [8]. The computed MLEs are shown in Figure 3.10 for different time delays and for different initial histories. The approximate MLEs of the Mackey-Glass system and the learned NDDE are shown in the panel (a), as a function of the truncation time T for $\tau = 1.5$ and history $x_0 = 0.9$. Both systems require a large truncation time to obtain good estimates (convergence) on MLEs. The MLEs are compared at $T = 10^4$ for different initial histories and different delays in panels (b) and (c). Observe that the NDDEs trained on chaotic data estimate well the MLE of the true system regardless of the initial history of the testing trajectory. One may also observe slightly larger MLE in the case $\tau = 1.5$ compared to the case $\tau = 1$.

To illustrate the generalizability of NDDEs, we generate the bifurcation diagrams using DDE-BIFTOOL [24, 110] and compare those to the bifurcation diagram of the original Mackey-Glass system in Figure 3.11, also see in Section 2.5.2. We plot the peak-to-peak amplitude of the limit cycle ($x_{\max} - x_{\min}$) as a function of delay and indicate the stability/instability of limit cycles by green/red color. The equilibrium $x(t) \equiv 1$ loses its stability via Hopf bifurcation and a stable limit cycle appears at $\tau = 0.24$. The limit cycle undergoes

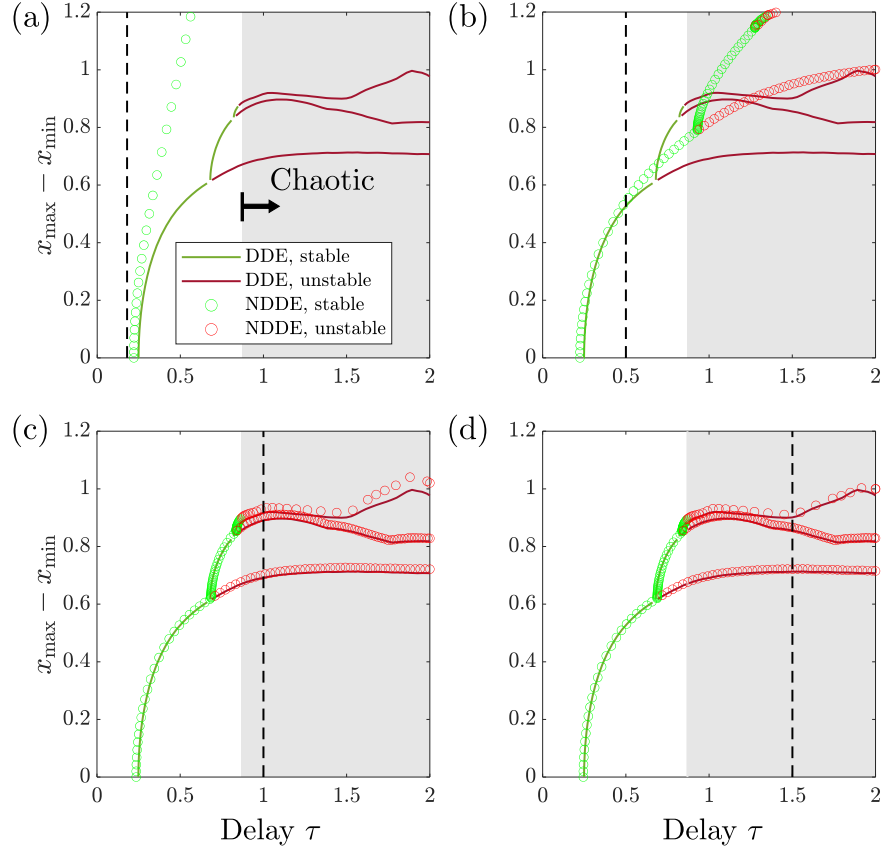


Figure 3.11: Bifurcation diagrams of the NDDE learned from data at $\tau = 0.2, 0.5, 1, 1.5$ compared to the ground truth. The dashed line indicates the underlying delay in the training data, and the shaded area indicates where the original Mackey-Glass system exhibits chaotic behavior.

a period-doubling cascade with the first two bifurcations located at $\tau = 0.61$ and $\tau = 0.84$. Beyond $\tau = 0.87$, the system has chaotic attractor, which is indicated by grey shading in Figure 3.11. From panels (a) and (b), we can see that the NDDEs with $\lambda_{\max} \leq 0$ have poor generalization for delay values greater than the delay in the training data. Contrarily, panels (c) and (d) demonstrate that the NDDEs with $\lambda_{\max} > 0$ predict the appearance of Hopf and period doubling bifurcations and capture the amplitude and stability of the limit cycles for other delay values without retraining. The network trained on data with larger MLE ($\tau = 1.5$ case) has better generalization, i.e., better approximation of the bifurcation diagram. Thus, the MLE can be used as a metric to evaluate the richness of the training data as well as the generalizability of the trained NDDE. Introducing time delays into the systems may help to improve the richness of the data.

We also present the learning performance under different levels of measurement noise in Fig. 3.12 for this system. As shown in the top row, the delay is still learned when small noise is added to the data, however, the noise does affect the learning of the nonlinearity as shown by the decreased accuracy of prediction in panels (b) and (c). The bottom row demonstrates that for stronger noise, the state derivatives fail to guide the learning of the delay. The gradients of the loss function with respect to the delay heavily rely on the state derivatives, which are sensitive to the noise.

3.4 Ikeda Dynamics

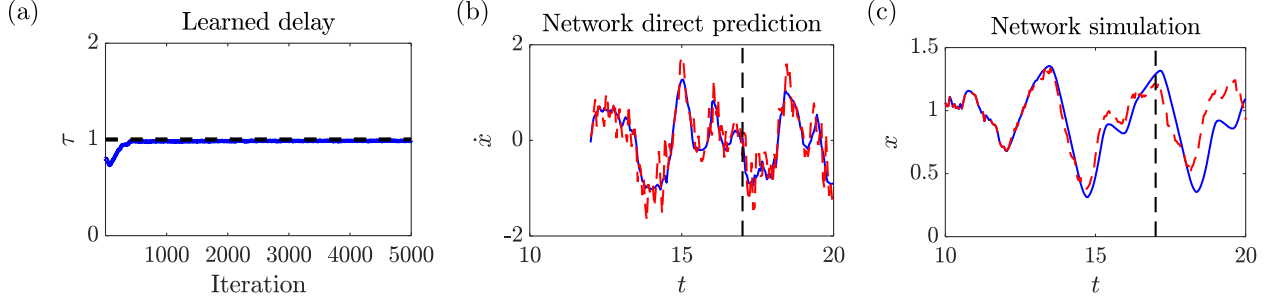
We train the same neural network (3.7) on a simulation trajectory (with $T = 30, \Delta t = 0.1$) for the Ikeda system [49, 37], which describes the dynamics of passive optical resonators

$$\dot{x}(t) = \beta \sin^2(x(t - \tau)) - \gamma x(t), \quad (3.9)$$

with parameters $\beta = 8, \gamma = 2, \tau = 1$ and initial history $x(t) \equiv 1$. This is another classical time delay system whose dynamics change qualitatively when the time delay is varied. Again, Algorithm 1 is used for training the NDDE with simulation loss $\mathcal{L}_x$. In the training process, the simulation time step for the DDE solver is set to $\delta t = 0.01$, while the horizon is $H = 10, H\Delta t = 1$. The learning rate for delay is 0.01 and for weights and biases is 0.1. Maximum number of iterations is set to 5000.

The network successfully learns the delay $\hat{\tau} = 1$ from the data and the predictions obtained by the trained network are given in Figure 3.13(a) and (b). The bifurcation diagrams of the original and the learned systems are compared in panel (c). Note that the network is only trained on data generated at $\tau = 1$. The bifurcation diagrams match well for smaller

With white noise in x ($\sigma_x = 0.02, \sigma_{\dot{x}} = 0.28$)



With white noise in x ($\sigma_x = 0.05, \sigma_{\dot{x}} = 0.71$)

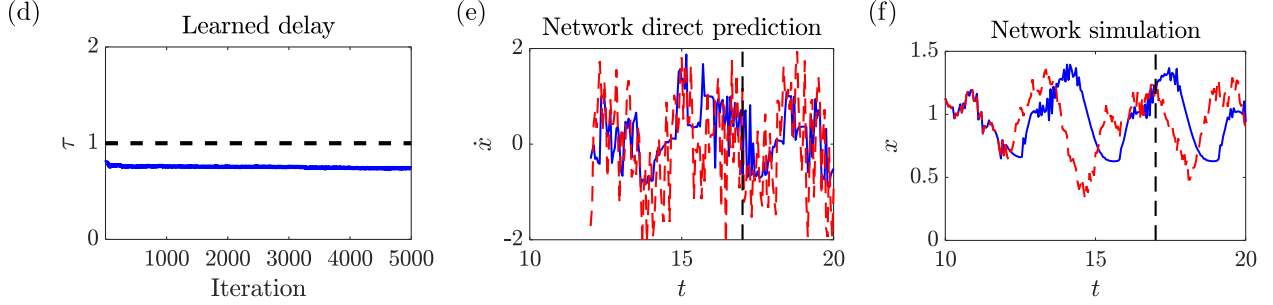


Figure 3.12: Training performance under two different levels of white noise. Both networks have the same initial delay 0.8 and are trained on chaotic data using weighted derivative loss.

delay values, while there are some differences for larger values of the delay (where even the bifurcation analysis of the original Ikeda system becomes numerically challenging). The transition between limit cycles and chaotic attractors in the Ikeda system is fast compared to the Mackey-Glass system, which makes the numerical bifurcation analysis more challenging.

3.5 Climate Model with Seasonal Excitation

Besides nonlinear autonomous systems with single delay, such as Mackey-Glass system and the Ikeda system, we can extend the training algorithms to systems with multiple delays and external forcing. In this section, we use a time delay neural network to learn the dynamics of the climate model

$$\dot{x} = aA(\kappa, x(t - \tau_1)) - bA(\kappa, x(t - \tau_2)) + cu(t), \quad (3.10)$$

$$A(\kappa, x) = \begin{cases} d_u \tanh(\frac{\kappa}{d_u} x), & \text{if } x \geq 0, \\ d_l \tanh(\frac{\kappa}{d_l} x), & \text{if } x < 0, \end{cases} \quad (3.11)$$

from a single trajectory where the input $u(t) = \cos(2\pi t)$ represents the seasonal forcing.

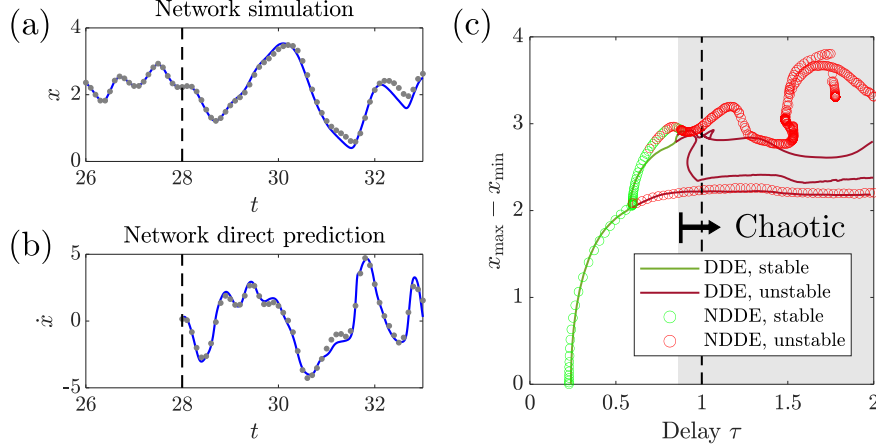


Figure 3.13: Training and testing performance of the Ikeda system. (a) Derivative prediction on test data. (b) Simulation prediction on test data. (c) Bifurcation diagram.

This model describes the deviation of thermocline depth from its long-term mean at the eastern boundary of the equatorial Pacific Ocean. The chaotic behavior of this simple delay differential equation is used to demonstrate the irregular phenomenon of the El Niño Southern Oscillation (ENSO). More detailed explanations of this model can be found in [64, 108]. In our example, the positive and negative feedbacks are associated with two different fixed delays $\tau_1 = 0.0958$ and $\tau_2 = 0.4792$, while other parameters are $a = 2.02$, $b = 3.03$, $c = 2.6377$, $d_u = 2.0$, $d_l = -0.4$. With these parameters we generate a trajectory using the constant history $x(t) \equiv 1$ as illustrated in Figure 3.14(a). The data are sampled with $\Delta t = 0.05$, indicated by grey dots. We again use Algorithm 1 and the simulation loss $\mathcal{L}_x$ to train the NDDE on a single trajectory in this section. The training under derivative loss $\mathcal{L}_{\dot{x}}$ is shown in Section 3.8 for comparison. The simulation time step is $\delta t = 0.01$ and simulation horizon is chosen to be $H = 10$.

For this learning task, we use a time delay neural network which takes the forcing $u(t)$ and state history $x(\vartheta)$ as input and gives state derivative $\dot{x}(t)$ as output. The network consists of 2 hidden layers, 5 neurons in each layer and 3 delays in the state. Note that the true number of delays in the system (3.10) is two (and there is no non-delayed state). That is, we moderately overestimate the number of delays. With three trainable delays, the dynamics of the NDDE takes the form

$$\hat{\dot{x}}(t) = W_3 \tanh \left(W_2 \tanh \left(W_1 \begin{bmatrix} \hat{x}(t - \hat{\tau}_1) \\ \hat{x}(t - \hat{\tau}_2) \\ \hat{x}(t - \hat{\tau}_3) \\ u(t) \end{bmatrix} + b_1 \right) + b_2 \right), \quad (3.12)$$

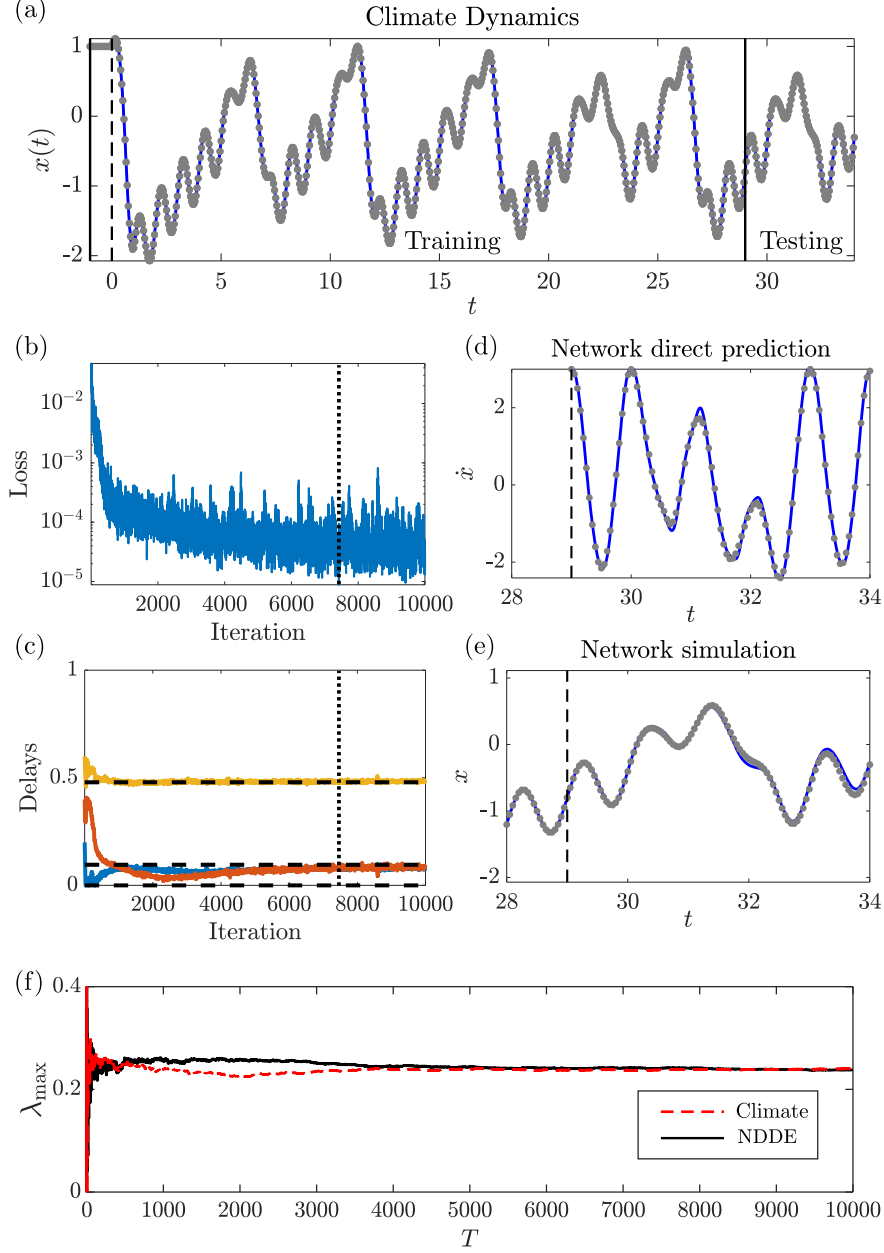


Figure 3.14: Training and testing performance of the climate model. (a) Training and testing datasets obtained from a single chaotic trajectory. (b) Training loss along the iterations. (c) Learned delays along the iterations. (d) Derivative prediction on test data. (e) Simulation prediction on test data. (f) Maximal Lyapunov exponents of the climate model and the learned NDDE while using $M = 10$.

where $W_1 \in \mathbb{R}^{5 \times 4}$, $W_2 \in \mathbb{R}^{5 \times 5}$, $W_3 \in \mathbb{R}^{1 \times 5}$, $b_1, b_2 \in \mathbb{R}^{5 \times 1}$. Here $u(t)$ is obtained from interpolating the recorded external forcing data. Thus, with a given initial history $x(\vartheta)$ and forcing $u(t)$, one can simulate the NDDE as (2.14).

Here we present the training results with simulation loss (2.19) and Algorithm 1 on single

trajectory in Figure 3.14. The learning rates for all parameters are set to 0.01, the batch size is chosen to be $N = 10$, and the maximum number of iterations is set to 10000. From panel (c) we can see that two of the three delays converge to the same value τ_1 and the third one converges to τ_2 . The final parameters of the NDDE are determined by the minimum training loss which is indicated by the dotted lines in panels (b) and (c). The derivative prediction and closed-loop network simulation on the test data set are shown in panels (d) and (e), respectively. This example shows that the developed algorithm can be extended to systems with external forcing where using a chaotic trajectory still provides good performance due to its rich dynamics. The MLE of the trained network is compared with that of the true system in panel (f). The network trained on one chaotic trajectory can reproduce the MLE of the original system.

3.6 Predator-Prey Model

Here we learn the dynamics of a two-state autonomous nonlinear delay system, which is used to model the time evolution of the species in a predator-prey system

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} rx_1(t) - sx_1^2(t) - px_1(t)x_2(t) \\ bpx_1(t - \tau)x_2(t - \tau) - dx_2(t) - mx_2^2(t) \end{bmatrix}, \quad (3.13)$$

where the time delay τ indicates the maturation time of the predators. Simulation trajectories are generated with parameters $r = 1, a = 1, p = 4, b = 1.2, m = 0.1, d = 1, \tau = 2$ and constant histories $x(\vartheta) \equiv 0.1, 0.2, 0.3, 0.4$. The training data set contains trajectories of length 34 ($t \in [-\tau_{\max}, 30]$) with time step $\Delta t = 0.1$. A two-hidden-layer network with tanh type nonlinearity, one fixed zero delay and one trainable delay is used considering $\tau_{\max} = 4$. The corresponding NDDE is given by

$$\begin{bmatrix} \hat{x}_1(t) \\ \hat{x}_2(t) \end{bmatrix} = W_3 \tanh \left(W_2 \tanh \left(W_1 \begin{bmatrix} \hat{x}_1(t) \\ \hat{x}_1(t) \\ \hat{x}_2(t - \hat{\tau}) \\ \hat{x}_2(t - \hat{\tau}) \end{bmatrix} + b_1 \right) + b_2 \right), \quad (3.14)$$

where $W_1 \in \mathbb{R}^{20 \times 4}$, $W_2 \in \mathbb{R}^{20 \times 20}$, $W_3 \in \mathbb{R}^{2 \times 20}$, $b_1, b_2 \in \mathbb{R}^{20 \times 1}$. Training is performed with Algorithm 1 and weighted derivative loss based on L_1 norm, maximum iteration 5000, batch size $N = 50$, and learning rates $\eta_\theta = 0.01, \eta_\tau = 0.001$.

Figure 3.15 shows the training result with no noise added to the data. The initial delay is set to 3 and it converges to the correct value of $\tau = 2$ during learning. The trained network

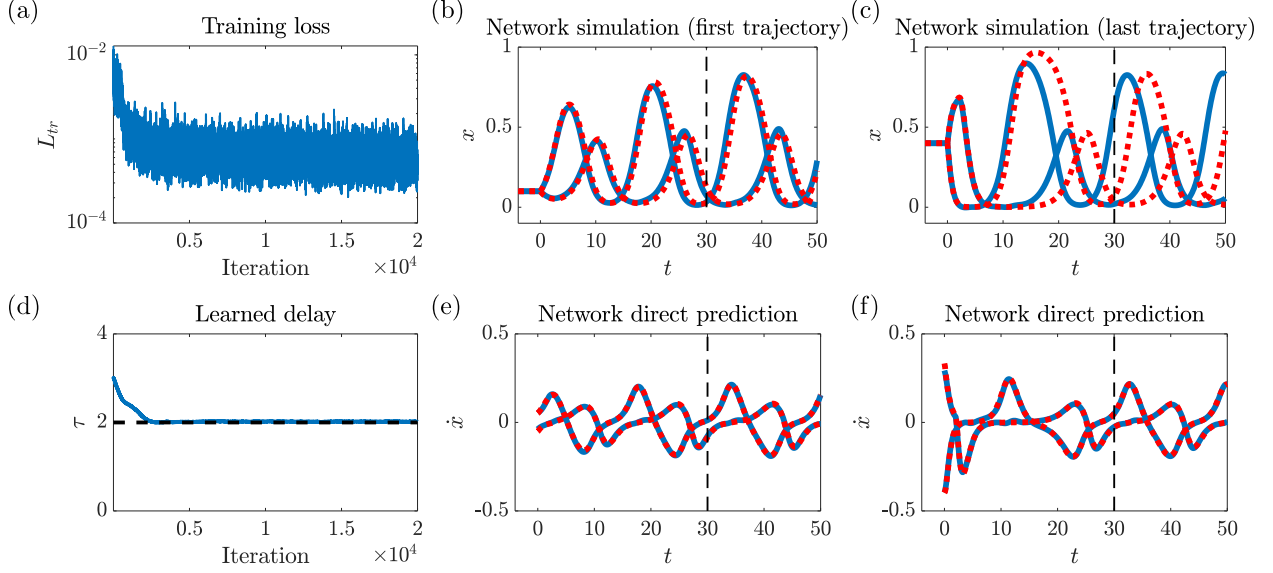


Figure 3.15: Training performance of the predator-prey model.

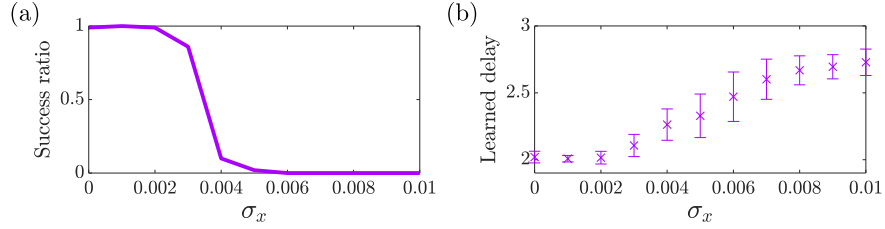


Figure 3.16: Evaluation of learning the delay from noisy data. (a) success ratio of learning the delay for different levels of noise; (b) distribution of 100 learned delays. All the training runs are initialized with different weights and biases but with the same initial delay 3.

gives good predictions on the state derivative, however, the simulation of the network on the last trajectory is not so accurate. We remark that part of the last trajectory is very close to the boundary of training data set and the portion of training data at that part of the state space is small.

To show how the delay learning algorithm performs under noise, we conduct another 100 training runs from different initial weights and biases but same initial delay 3 when different levels of noise are added. We define the success ratio of learning the delay as the number of learned delay falling in the interval 2 ± 0.15 divided by the number of conducted runs. One may observe from the Figure 3.16 that the success ratio of learned delay drops sharply as the noise level exceeds some limit.

3.7 Double-Delayed Rössler System

In this section, we compare the NDDE approach with the SINDy approach on the 3-state delayed Rössler system. The reconstructions of the time delays and the trajectories, as well as the computation time, are compared between different methods. In addition, the advantages and shortcomings of the different approaches are also discussed in the section.

The double-delayed Rössler system [123, 35, 124] is given by:

$$\begin{aligned}\dot{x}_1(t) &= -x_2(t) - x_3(t) + \alpha_1 x_1(t - \tau_1) + \alpha_2 x_1(t - \tau_2), \\ \dot{x}_2(t) &= x_1(t) + \beta_1 x_2(t), \\ \dot{x}_3(t) &= \beta_2 + x_3(t)x_1(t) - \gamma x_3(t).\end{aligned}\tag{3.15}$$

This is a variation of the Rössler system, which simplifies the Lorenz model of turbulence (as it contains only one second-order nonlinearity in one variable). When $\alpha_1 = 0.2$, $\alpha_2 = 1$, $\beta_1 = \beta_2 = 0.2$, $\gamma = 1.2$ and $\tau_1 = 1$, $\tau_2 = 2$, the system exhibits chaotic behavior. We collect data generated when starting from the constant initial function $\varphi(s) \equiv [1.5, 0.4, 0.9]^\top$. The resulting dataset is a single trajectory with a section of length $T_{\text{tr}} = 30$ used for training and section of length $T_{\text{ts}} = 5$ used for testing. The data are sampled with $\Delta t = 0.05$ and central difference is used to calculate the state derivatives numerically.

In this example, we compare the performance of the NDDE with both expert SINDy (E-SINDy) and pragmatic SINDy (P-SINDy) approaches when the delays are optimized using the particle swarm optimizer (PS).

NDDE with trainable delays We consider the network structure

$$\hat{x}(t) = W_3 \tanh \left(W_2 \tanh \left(W_1 \hat{\mathbf{x}}(t) + b_1 \right) + b_2 \right),\tag{3.16}$$

with different inputs and different number of neurons in the hidden layers. Depending on whether we know that the delays only enter the state x_1 , or consider that they may enter

x_1 , x_2 and x_3 , we can construct different network inputs:

$$\mathbf{x}_t = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_1(t - \hat{\tau}_1) \\ x_1(t - \hat{\tau}_2) \end{bmatrix}, \quad \mathbf{x}_t = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_1(t - \hat{\tau}_1) \\ x_2(t - \hat{\tau}_1) \\ x_3(t - \hat{\tau}_1) \\ x_1(t - \hat{\tau}_2) \\ x_2(t - \hat{\tau}_2) \\ x_3(t - \hat{\tau}_2) \end{bmatrix}. \quad (3.17)$$

We refer the first input as simplified input and the second as full input and we will use both networks to learn from training data considering the derivative loss. While training with derivative loss, the input vector is from data ($\mathbf{x}_t$ instead of $\hat{\mathbf{x}}_t$ in (3.16)) and the output vector is the predicted state derivative $\hat{\dot{x}}(t)$.

SINDy for DDE approach We optimize both delays τ_1 and τ_2 and compare the results when using different degrees of polynomial libraries with $D = 2, 3, 4$. All of these libraries include the true second-degree nonlinearity that appears in the RHS of (3.15), but for $D = 3$ or $D = 4$ the library also contains many other terms that are missing from (3.15). When using E-SINDy, all delays are simultaneously optimized, and when using P-SINDy, only the maximum delay is optimized. For E-SINDy, candidate delays for τ_1 and τ_2 are uniformly sampled from the intervals $[0.1, 1.5]$ and $[1, 3]$. For P-SINDy, the candidate for the maximum delay $\bar{\tau}$ is uniformly sampled from the interval $[0.1, 2.5]$. Particle swarm optimization is utilized to search for the unknown delays. For P-SINDy, the degree of pseudospectral collocation for the discretization from DDE to ODE is set to $M = 5$.

Comparisons of SINDy and NDDE approaches The trajectory recovery is shown in Figure 3.17 for the E-SINDy approach (with PS and polynomial libraries of orders 2 and 4) and for the NDDE approach (with simplified input, 5 neurons per layer and derivative loss). When the degree of the SINDy library is 2, it outperforms the NDDE in terms of prediction accuracy, while for degree 4 the SINDy predictions are less accurate than the NDDE predictions.

The E-SINDy approach with low order library can recover the data very well since the components at the RHS of (3.15) exist in the library and no higher-order nonlinearities are attempted to fit the data. However, when these higher-order terms are also included, the

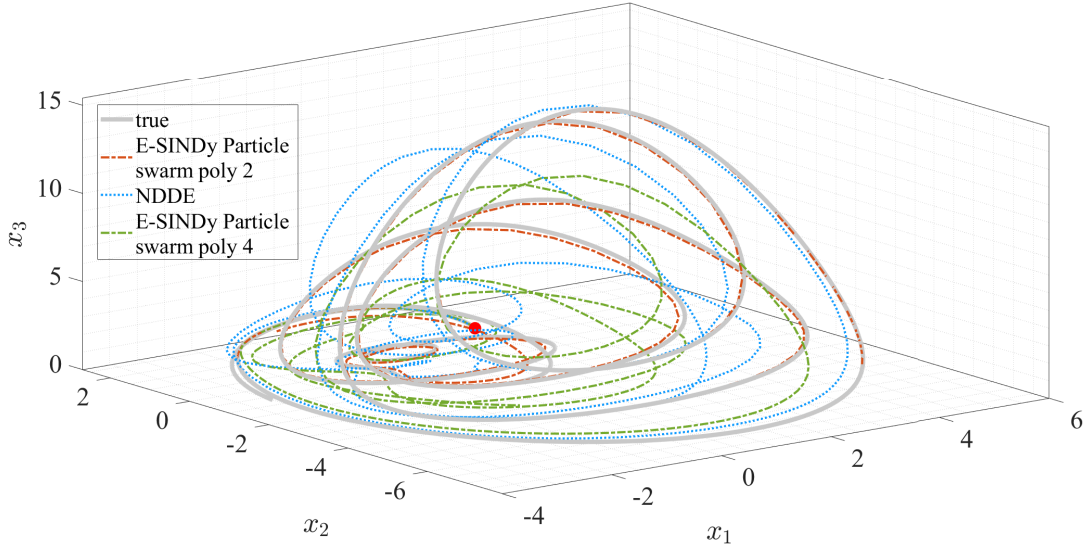


Figure 3.17: Trajectory reconstruction for the Rössler model (3.15) showing the true trajectory (gray), E-SINDy trajectory when using library with polynomial degree 2 (red), E-SINDy trajectory when using library with polynomial degree 4 (green), and NDDE trajectory (blue). The true delay values are $(\tau_1, \tau_2) = (1, 2)$. The optimized delay values obtained through PS optimization for E-SINDy are $(\hat{\tau}_1, \hat{\tau}_2) = (1.0001, 2.0000)$ when using polynomial library of order 2 and $(\hat{\tau}_1, \hat{\tau}_2) = (1.0004, 2.0000)$ when using polynomial library of order 4. The NDDE approach yields $(\hat{\tau}_1, \hat{\tau}_2) = (0.2309, 1.9064)$ when using simplified input and 5 neurons in each hidden layer.

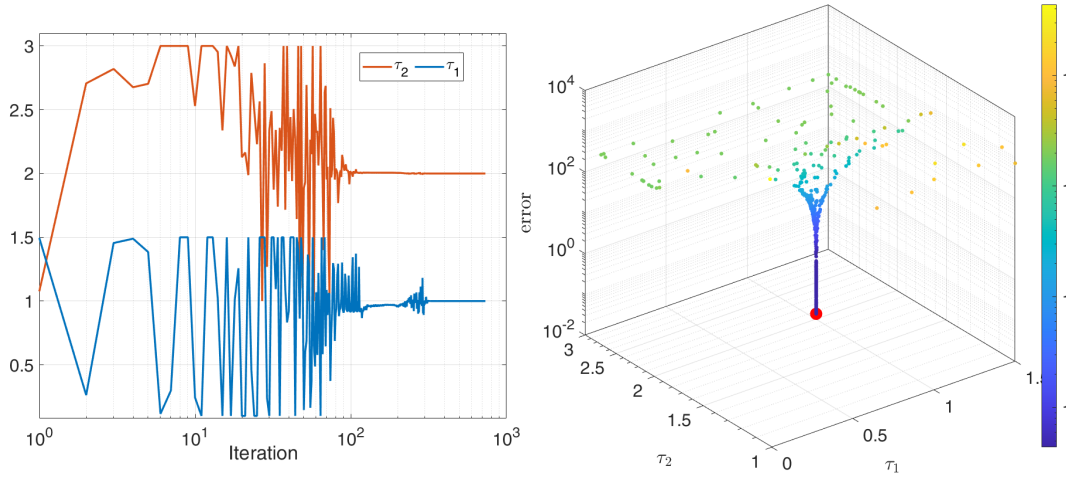


Figure 3.18: Learned delays and particle filter errors along the iterations when applying E-SINDy with polynomial library of order 2 to the Rössler model (3.15).

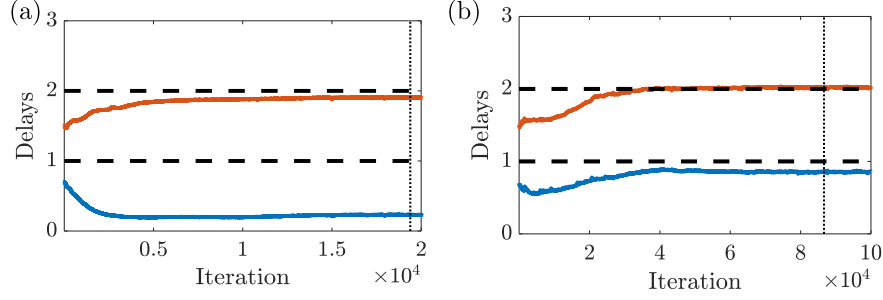


Figure 3.19: The learned delays along the iterations for the two-delay case when applying the NDDE approach to the Rössler model (3.15). (a) Simplified input and 5 neurons in each hidden layer, trained on one trajectory. (b) Simplified input and 20 neurons in each hidden layer, trained on ten trajectories.

prediction accuracy decreases, as E-SINDy tries to capture the behavior by blending polynomials of different degrees. In the meantime, the PS optimization still accurately captures the time delays in both cases. For the limited amount of data utilized the accuracy of NDDEs predictions fall between the two SINDy cases while the learned delays remain somewhat far from the true values. However, by including more training data, the performance of the NDDE can be significantly improved as will be demonstrated below.

The training procedure for E-SINDy with particle swarm optimization is shown in Figure 3.18. One may observe a clear minimum around the true delay values, which can be approached within a few hundred iterations with the help of the PS optimizer. For the NDDE, we depict the iterations in the trainable delays for different training datasets in Figure 3.19. While for the limited data described above only one of the two delays is captured accurately, see panel (a), both delays can be recovered when the amount of training data is increased 10 times (by using 10 different initial conditions to generate 10 distinct trajectories), as shown in panel (b).

Table 3.3 summarizes the E-SINDy results and the P-SINDy results (with collocation mesh size 5) while using the particle swarm (PS) optimizer. It includes the absolute errors on both delays τ_1 and τ_2 , as well as the reconstruction error $\text{RMSE}_{x'}$ for training and the errors $\text{RMSE}_{x'}$ and RMSE_x for testing. Different libraries of polynomial orders are compared in the table. Both methods achieve excellent reconstruction accuracy for polynomial order 2. E-SINDy shows superior computational efficiency but P-SINDy achieves slightly better delay estimation accuracy in most cases. When the order of polynomials increases in the library, the delay estimation remains accurate, while the reconstruction accuracy drops. This illustrates that having the right set of nonlinear functions in the library is important for SINDy to accurately identify the underlying dynamics of time delay systems.

The NDDE results are collected in Table 3.4. For each network input choice, we test

SINDy			Training	Testing	
	$ \tau_1 - \hat{\tau}_1 $	$ \tau_2 - \hat{\tau}_2 $	RMSE $_{\dot{x}}$	RMSE $_{\dot{x}}$	RMSE $_x$
Order 2					
E-SINDy	6.73×10^{-3}	3.16×10^{-5}	4.31×10^{-3}	2.82×10^{-3}	8.17×10^{-2}
P-SINDy	-	3.63×10^{-7}	5.33×10^{-5}	6.49×10^{-5}	5.84×10^{-4}
Order 3					
E-SINDy	5.89×10^{-4}	9.36×10^{-5}	1.55×10^{-2}	4.88×10^{-2}	
P-SINDy	-	2.75×10^{-7}	4.84×10^{-3}	6.38×10^{-3}	3.20×10^{-2}
Order 4					
E-SINDy	7.32×10^{-4}	1.04×10^{-5}	3.02×10^{-1}	2.64	5.36
P-SINDy	-	6.42×10^{-7}	7.66×10^{-1}	9.63	3.28

Table 3.3: Performance of E-SINDy and P-SINDy (with collocation mesh size 5) and different polynomial orders for the Rössler system (3.15), spelling out the absolute error between the true and optimized delays, the RMSEs for training and testing phases, the number of SINDy calls and the CPU time while employing particle swarm (PS).

NDDE			Training	Testing	
	$ \tau_1 - \hat{\tau}_1 $	$ \tau_2 - \hat{\tau}_2 $	$\mathcal{L}_{\dot{x}}$	RMSE $_{\dot{x}}$	RMSE $_x$
Full (5)	8.70×10^{-1}	1.10	4.53×10^{-5}	4.67×10^{-2}	1.86×10^{-1}
Full (20)	4.61×10^{-1}	6.32×10^{-1}	5.53×10^{-6}	7.09×10^{-1}	2.79
Simplified (5)	7.69×10^{-1}	9.36×10^{-2}	1.91×10^{-4}	6.84×10^{-2}	8.96×10^{-2}
Simplified (20)	3.24×10^{-1}	3.29×10^{-1}	4.15×10^{-6}	1.69×10^{-1}	1.87×10^{-1}
Simplified* (20)	1.39×10^{-1}	2.30×10^{-2}	1.73×10^{-5}	2.21×10^{-1}	1.24×10^{-1}

Table 3.4: Performance of NDDE for the Rössler system (3.15), spelling out the absolute error between the true and optimized delays and the errors in training and testing phases. The last row (Simplified*) is trained on a larger dataset, where 10 trajectories are generated from 10 different constant histories.

two different network sizes, i.e., the number of neurons in each hidden layer is either 5 or 20, as indicated in the first column. The networks with simple input structure lead to smaller errors in the learned delays, that is, having restriction on the network input helps regulating the network. In addition, increasing the number of neurons in each hidden layer from 5 to 20 yields better predictions on the testing data. Training the NDDE on a dataset consisting of 10 different trajectories results in more accurate learned delays and simulation results compared to the NDDE trained on only one trajectory. That is, while learning from a single trajectory is challenging for neural networks (without any knowledge about the nonlinearities), utilizing data with multiple trajectories stemming from different initial

histories improves the learning performance. In summary, having richer training data may compensate the missing apriori knowledge about the form of the RHS.

3.8 Discussion

In this section, we first discuss the size of the NDDE on the training performance for Mackey-Glass dynamics, logistic equation and climate model and compare the performance of the NDDE under derivative and simulation loss functions for the Mackey-Glass dynamics. We also summarize potential challenges in learning the time delay systems with NDDE and provide insights for the future work.

The training results of the NDDE in different sizes under derivative loss are compared on all three examples when applying Algorithm 1 with derivative loss $\mathcal{L}_{\dot{x}}$ on different sizes of fully-connected 2-layer networks. The learned delay and the RMSE on the testing dataset are collected in Tables 3.5, 3.6 and 3.7. As we increase the number of neurons in the hidden layers, the performance of trained networks improves. It improves significantly as we increase the number of neurons from 2 to 5, however, the improvement is not significant after 5.

Loss	Size	RMSE $_{\dot{x}}$	RMSE $_x$	$\hat{\tau}$ ($\tau = 1$)
$\mathcal{L}_{\dot{x}}$	2	0.3421	0.1825	0.8298
	5	0.0619	0.0338	0.9824
	10	0.0243	0.0245	0.9948
	20	0.0224	0.0214	0.9931

Table 3.5: RMSE of trained NDDEs with different size on testing data from delayed logistic equation (3.4). All the NDDEs have two hidden layers while the number of hidden neurons ($l_1 = l_2$) in each hidden layer is indicated in the first column.

Additionally, in Table 3.6, we compare the results given by different loss functions for the Mackey-Glass example. Once the delay is correctly learned, the performance of trained networks are similar, regardless of the loss function. The computation time is also examined in this case. Since the size of the networks is still relatively small even if we have $l_1 = l_2 = 20$, it is not a main factor that affects the computation time. The computation time mostly depends on the batchsize and the number of iterations. For the simulation loss function, the computation time also depends on the simulation horizon H . Using derivative loss leads to a much faster training process compared to the one using simulation loss.

From all the examples provided in this chapter, we summarize the following challenges in learning time delay systems with NDDE.

Loss	Size	RMSE $_{\dot{x}}$	RMSE $_x$	$\hat{\tau}$ ($\tau = 1$)	Iterations	Time
$\mathcal{L}_{\dot{x}}$	2	0.1626	0.0813	0.9886	2000	≈ 5 s
	5	0.0413	0.0114	1.0017		
	10	0.0340	0.0079	1.0024		
	20	0.0307	0.0077	1.0016		
$\mathcal{L}_x$	2	0.1357	0.0524	1.0053	2000	≈ 100 s
	5	0.0353	0.0117	0.9949		
	10	0.0337	0.0178	0.9977		
	20	0.0231	0.0037	0.9972		

Table 3.6: RMSE of trained NDDEs with different size on testing data from Mackey-Glass equation (3.6). All the NDDEs have two hidden layers while the number of hidden neurons ($l_1 = l_2$) in each hidden layer is indicated in the first column.

Loss	Size	RMSE $_{\dot{x}}$	RMSE $_x$	$\hat{\tau}_1$ ($\tau_1 = 0.0958$)	$\hat{\tau}_2$ ($\tau_2 = 0.4792$)
$\mathcal{L}_{\dot{x}}$	2	0.4937	0.4807	0	0.3086
	5	0.0577	0.1927	0.1175	0.4717
	10	0.0293	0.0389	0.1116	0.4752
	20	0.0177	0.0155	0.0540	0.4851

Table 3.7: RMSE of trained NDDEs with different size on testing data from climate model. All the NDDEs have two hidden layers while the number of hidden neurons ($l_1 = l_2$) in each hidden layer is indicated in the first column.

Data Quality and System Complexity We remark that autonomous systems with high behavior complexity typically generate “high quality” data for learning. For instance, models learned from chaotic attractors usually generalize well [74, 58, 37]. Contrarily, the models trained by data concentrated around the equilibrium or the limit cycle may generalize poorly. In such cases, we consider the data is of “low quality” (even if it does not contain noise). Incorporating more transient data can be achieved by setting the initial conditions further away from the attractors. While higher behavior complexity facilitates better learning, increasing the system complexity typically makes learning more difficult. For example, a scalar system is easier to learn compared to a system with vector valued states, as in the latter more local minima were introduced when the complexity of data-driven model increases. To address this, the amount of high-quality training data should also be increased.

Learning the Numerical Differentiation It is possible to learn numerical differentiation schemes when training NDDEs because those schemes indeed constitute local minima of the

derivative loss function, see the study in [56]. For example, the NDDE can approximate

$$\hat{x} = \frac{1}{\hat{\tau}}x(t) - \frac{1}{\hat{\tau}}x(t - \hat{\tau}), \quad (3.18)$$

instead of the right hand side of the differential equation. This can happen when data is sampled densely (the undesired solution may be the global minimum) and when the initial delay is small (starts too close to the undesired solution). Therefore, obtaining a small derivative loss does not guarantee that the network has learned the right hand side of the differential equation. The network may approximate any linear combination of the right hand side and the numerical differentiation scheme, when there are more delays (typically small delays) allowed in the network. Such behavior was also observed in [37], where a one-step-ahead prediction loss, which is equivalent to the derivative loss, was considered, and the loss function had a local minima at a very small delay value. To avoid this problem, one can consider the simulation loss, limit the number of delays and use multiple initial delays.

Robustness to Noise One may observe from the gradient calculation (2.20) that the gradient-based algorithms rely heavily on the state derivatives. If the states are measured with added noise, the time derivatives obtained via numerical differentiation and the state estimates obtained via linear interpolation are largely affected by the noise and the time step used. Smaller time step Δt leads to larger noise in the derivative, while larger time step leads to error in the differentiation. For example, consider that white noise with zero mean and standard deviation $\sigma_x = 0.02$ is added to x through the measurements, then the time derivatives obtained from Euler’s methods with $\Delta t = 0.05$ results in additive noise with zero mean and standard deviation $\sigma_{\dot{x}} = \frac{\sqrt{2}\sigma_x}{\Delta t} = 0.57$ for $\dot{x}$. If the central difference method is used to obtain the derivative, the standard deviation of the noise added to $\dot{x}$ reduces to 0.28. Thus, choosing a robust numerical differentiation method and applying appropriate filtering are critical for calculating the time derivatives accurately, which is an important research topic on its own; see [28, 115].

3.9 Summary

In this chapter, we have shown that the NDDE can be used to learn the dynamics of time delay systems from data. The trained NDDE can reproduce the dynamics of the original system and the learned delays are close to the true values. The good generalizability was demonstrated by showing that NDDEs can be used to predict the dynamics of the system outside the training data. We also compared our method to the SINDy methods on a

multiple-state multiple-delay example. We remark the keys to the success of many data-driven methods are the richness of the training data and the amount of embedded knowledge. We pointed out potential challenges in the NDDE implementation and provided solutions to overcome these.

The choice of data-driven methods is usually problem-dependent and it should be customized based on the objective of the tasks. There are still many open questions in this topic. For example, how to quantify the amount and the quality of the training data which is sufficient to learn the time delay systems, and how that is related the amount of knowledge embedded in the data-driven model. The proposed NDDE methods can be improved, for instance, restricting the NDDE for specific training tasks (to employ domain knowledge) and equip it with gradient-free delay learning. The studies on stochastic and distributed delays can be further explored, as well as the methods for neutral delay differential equations. It would be interesting to see applications on real-world data, even though the ground truth is normally unknown.

CHAPTER 4

Learning the Longitudinal Dynamics of a Connected Automated Vehicle

In this chapter, we first introduce the longitudinal vehicle model with time delays for a connected automated vehicle (CAV). We perform stability analysis for the delayed longitudinal dynamics, and then we use a NDDE with a trainable delay to learn the dynamics of the CAV from experimental data.

4.1 Delayed Longitudinal Dynamics

Considering a simple case where the air drag, rolling resistance and road grade are all neglected, one can model the longitudinal dynamics of a connected automated vehicle (CAV) as

$$\begin{aligned}\dot{s} &= v, \\ \dot{v} &= u(t - \tau),\end{aligned}\tag{4.1}$$

where s is the position of the rear bumper of the vehicle, v is the velocity, u is the acceleration command, and τ is the delay in the feedback loop. Define the distance headway h as $h = s_1 - s - l$, where s_1 is the position of the rear bumper of the leading vehicle that is a connected human-driven vehicle (CHV), and l is the length of CAV. Then, one can rewrite the model (4.1) as

$$\begin{aligned}\dot{h} &= v_1 - v, \\ \dot{v} &= u(t - \tau),\end{aligned}\tag{4.2}$$

where v_1 is the velocity of the leading vehicle. The illustration of the system is shown in Figure 4.1. Assume the longitudinal controller is designed for car-following scenario and it

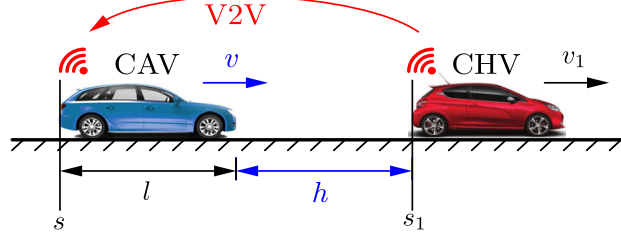


Figure 4.1: Car-following scenario of a connected automated vehicle following a connected human-driven vehicle.

is in the form of a proportional-derivative controller, i.e., the acceleration command is given by

$$u = \alpha(\kappa(h - h_{\text{st}}) - v) + \beta(v_1 - v), \quad (4.3)$$

where h_{st} is the desired minimal distance between two vehicles (the vehicle intends to stop below this distance), κ represents how the desired distance changes linearly as the velocity increases and α and β are the control gains. Note that $\alpha\kappa$ is equivalent to a proportional gain and $\alpha + \beta$ is the equivalent to a differential gain.

This model is a simplified version of the longitudinal dynamics of a connected automated vehicle (CAV) in a car-following scenario and serves as a good approximation of the vehicle dynamics in the presence of time delays. Saturation and nonlinearities can be added to the model, while these are usually omitted when performing stability analysis. We study the plant stability of the system (4.1)–(4.3) while varying the time delay τ and the control gains α and β . The plant stability is defined as the asymptotic stability of the equilibrium without external disturbance, which characterizes the transient response of the system. To quantify plant stability, we consider small perturbations around the constant speed equilibrium. The system can be expressed as

$$\begin{bmatrix} \dot{\tilde{h}} \\ \dot{\tilde{v}} \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \tilde{h} \\ \tilde{v} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \alpha\kappa & -(\alpha + \beta) \end{bmatrix} \begin{bmatrix} \tilde{h}(t - \tau) \\ \tilde{v}(t - \tau) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tilde{v}_1 + \begin{bmatrix} 0 \\ \beta \end{bmatrix} \tilde{v}_1(t - \tau), \quad (4.4)$$

where $\tilde{v}_1 = v_1 - v^*$, $\tilde{v} = v - v^*$, $\tilde{h} = h - h^*$ and $h^* = v^*/\kappa + h_{\text{st}}$. The stability of the system (4.4) can be analyzed based on the characteristic equation

$$\lambda^2 + \alpha\kappa e^{-\lambda\tau} + \beta e^{-\lambda\tau} = 0. \quad (4.5)$$

The equilibrium is plant stable if and only if all the characteristic roots of Eq. 4.5 have negative real parts, i.e., $\text{Re}(\lambda_i) < 0, i = 1, 2, \dots$. A stronger stability condition can be obtained by requiring that all the roots have real parts smaller than a certain value, i.e.,

$\text{Re}(\lambda_i) < \rho, i = 1, 2, \dots$, where ρ is a negative constant. The real parts of the leading roots indicate how fast the system converges to the equilibrium.

Substituting $\lambda = 0$ and $\lambda = j\omega$ into the characteristic equation (4.5), we obtain

$$\alpha = 0, \quad (4.6)$$

and

$$\begin{aligned} \alpha &= \frac{\omega^2}{\kappa} \cos(\omega\tau), \\ \beta &= \omega \sin(\omega\tau) - \alpha, \end{aligned} \quad (4.7)$$

that describe the stability boundaries in the (α, β) plane. As the delay τ increases, the stable region shrinks and the system may become unstable if the control gains land outside the stable region.

Substituting $\lambda = \rho$ and $\lambda = \rho + j\omega$ into the characteristic equation (4.5), we obtain

$$\alpha = -\frac{\rho}{\rho + \kappa}(\beta + \rho e^{\rho\tau}), \quad (4.8)$$

and

$$\begin{aligned} \alpha &= \frac{\rho^2 + \omega^2}{\kappa\omega} e^{\rho\tau} \left(\rho \sin(\omega\tau) + \omega \cos(\omega\tau) \right), \\ \beta &= -e^{\rho\tau} \left(\frac{\rho^2 - \omega^2}{\omega} \sin(\omega\tau) + 2\rho \cos(\omega\tau) \right) - \alpha, \end{aligned} \quad (4.9)$$

that bound the region in the (α, β) plane where real parts of the leading roots are smaller than ρ . Indeed, $\rho = 0$ gives the stability boundaries (4.6)-(4.7).

Figure 4.2(a) and (b) shows the curves (4.8)-(4.9) for $\tau = 0.5$ and $\tau = 0.6$ seconds, respectively, while using different ρ values. In each panel, as ρ decreases, the region within the curves shrinks and eventually disappears at

$$\rho_{\text{fast}} = \frac{\sqrt{2} - 2}{\tau}, \quad (4.10)$$

which is indicated by the star. The corresponding fastest-convergence control gains are given

by

$$\begin{aligned}\alpha_{\text{fast}} &= \frac{10\sqrt{2} - 14}{\kappa\tau^2} e^{\sqrt{2}-2} \approx \frac{0.0791}{\kappa\tau^2}, \\ \beta_{\text{fast}} &= \frac{2\sqrt{2} - 2}{\tau} e^{\sqrt{2}-2} - \alpha_{\text{fast}} \approx \frac{0.4611}{\tau} - \alpha_{\text{fast}}.\end{aligned}\tag{4.11}$$

Figure 4.2(c) illustrates that the control gains move along the black curve (from top to down) as the delay increases. Note that this curve intersects with the stability boundaries. This means that, if the fastest-convergence control gains are chosen for a certain delay, the system may still lose stability at a larger delay τ_{cr} . We can analytical derive the critical delay τ_{cr} by equating (4.11) at τ and the stability boundary (4.7) at τ_{cr}

$$\begin{aligned}\alpha_{\text{fast}}(\tau) &= \alpha(\omega_{\text{cr}}, \tau_{\text{cr}}), \\ \beta_{\text{fast}}(\tau) &= \beta(\omega_{\text{cr}}, \tau_{\text{cr}}).\end{aligned}\tag{4.12}$$

Let $p = e^{\sqrt{2}-2} (2\sqrt{2} - 2)$ and $q = e^{\sqrt{2}-2} (10\sqrt{2} - 14)$ in (4.11), then we have

$$\begin{aligned}\frac{q}{\tau^2} &= \omega_{\text{cr}}^2 \cos(\omega_{\text{cr}}\tau_{\text{cr}}), \\ \frac{p}{\tau} &= \omega_{\text{cr}} \sin(\omega_{\text{cr}}\tau_{\text{cr}}),\end{aligned}\tag{4.13}$$

which yields

$$\begin{aligned}\omega_{\text{cr}} &= \frac{g}{\tau}, \\ \tau_{\text{cr}} &= \frac{1}{g} \sin^{-1} \left(\frac{p}{g} \right) \tau \approx 2.5232 \tau,\end{aligned}\tag{4.14}$$

where $g = \sqrt{\frac{p^2 + \sqrt{p^4 + 4q^2}}{2}}$.

One may also study the string stability of the systems, which characterizes the attenuation of the speed fluctuations of the leading vehicle [27, 99]. As string instability corresponds to the growth of the oscillations along the chain of vehicles, it degrades the energy efficiency and results in traffic jams. The detailed analysis on the string stability is given in [3]. As the time delay increases, the string stability region also shrinks and may even disappear above a certain value.

The stability analysis demonstrates that the time delay has a significant impact on the performance of the vehicles and if one can learn the time delays from the data, the information can be used to improve the controller design.

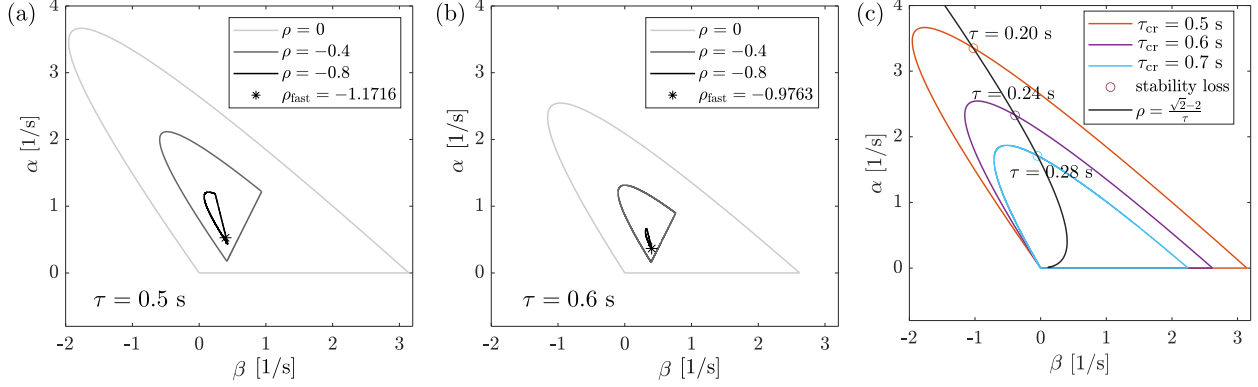


Figure 4.2: Stability charts under different delays. (a) $\tau = 0.5$ s. (b) $\tau = 0.6$ s. (c) The curve of the fastest-convergence gains intersects the oscillatory stability boundaries around $\tau_{\text{cr}} \approx 2.52 \tau$.

4.2 Learning the Delayed Longitudinal Dynamics

In this section, we use a NDDE with a trainable delay to learn the longitudinal dynamics of a CAV following a chain of CHVs from experimental data. The scenario is shown in Figure 4.3(a) where a connected automated vehicle (CAV) is inserted to a chain of CHVs. Via vehicle-to-vehicle (V2V) communication, the CAV is made aware of the velocities and positions of the vehicles ahead and it also measures its own velocity and position.

The details of the controller design and experiments are given in [32]. Here we provide a brief overview of the design procedure, while our main goal is to learn the dynamics from data. The longitudinal dynamics of the CAV can be modeled by

$$\begin{aligned} \dot{h}(t) &= v_1(t) - v(t), \\ \dot{v}(t) &= -p(v(t)) + \text{sat}(u(t - \tau)), \end{aligned} \quad (4.15)$$

where h is the distance headway to the CHV1 immediately ahead, v is the speed of the CAV, and v_1 is the speed of the CHV1. Moreover, $p(v(t))$ represents the loss term which depends on the CAV's current velocity while u represents the commanded acceleration that is limited by the saturation function

$$\text{sat}(u) = \begin{cases} a_{\min} & \text{if } u < a_{\min}, \\ u & \text{if } a_{\min} \leq u \leq a_{\max}, \\ a_{\max} & \text{if } u > a_{\max}, \end{cases} \quad (4.16)$$

to keep it within the limits given by the road friction and driver comfort. Finally, τ represents

the time delay in the control loop resulting from V2V communication, on-board computation, and actuator delays. Neither the resistance term $p(v)$ nor the time delay τ was known while designing the controller. Here we will identify these from the data collected through the experiments.

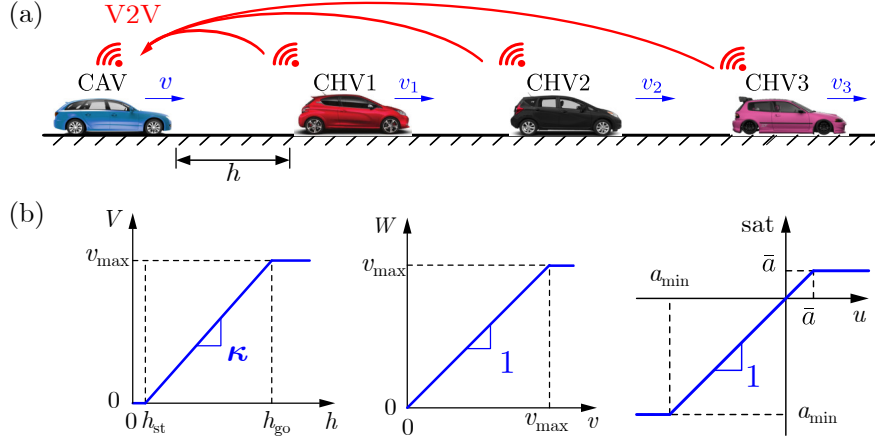


Figure 4.3: (a) Experimental setup where a CAV follows CHVs based on the data transmitted through V2V connectivity. (b) Nonlinearities used in the model.

The control law used in the experiments [129, 32, 43, 1] is given by:

$$u = \alpha(V(h) - v) + \sum_{k=1}^3 \beta_k(W(v_k) - v), \quad (4.17)$$

(c.f. (4.3)), where α is the control gain used to regulate the headway h while $\beta_1, \beta_2, \beta_3$ are the control gains used to respond to the velocities v_1, v_2, v_3 of the CHVs ahead. The control law includes the range policy

$$V(h) = \begin{cases} 0 & \text{if } h < h_{\text{st}}, \\ \kappa(h - h_{\text{st}}) & \text{if } h_{\text{st}} \leq h \leq h_{\text{go}}, \\ v_{\text{max}} & \text{if } h > h_{\text{go}}, \end{cases} \quad (4.18)$$

which gives the desired speed as an affine function of the headway. It also contains the speed policy

$$W(v_k) = \begin{cases} v_k & \text{if } v_k \leq v_{\text{max}}, \\ v_{\text{max}} & \text{if } v_k > v_{\text{max}}, \end{cases} \quad (4.19)$$

which is introduced to avoid following leaders whose speed exceeds the maximum desired speed v_{max} . The functions V , W and sat are depicted in Figure 4.3(b).

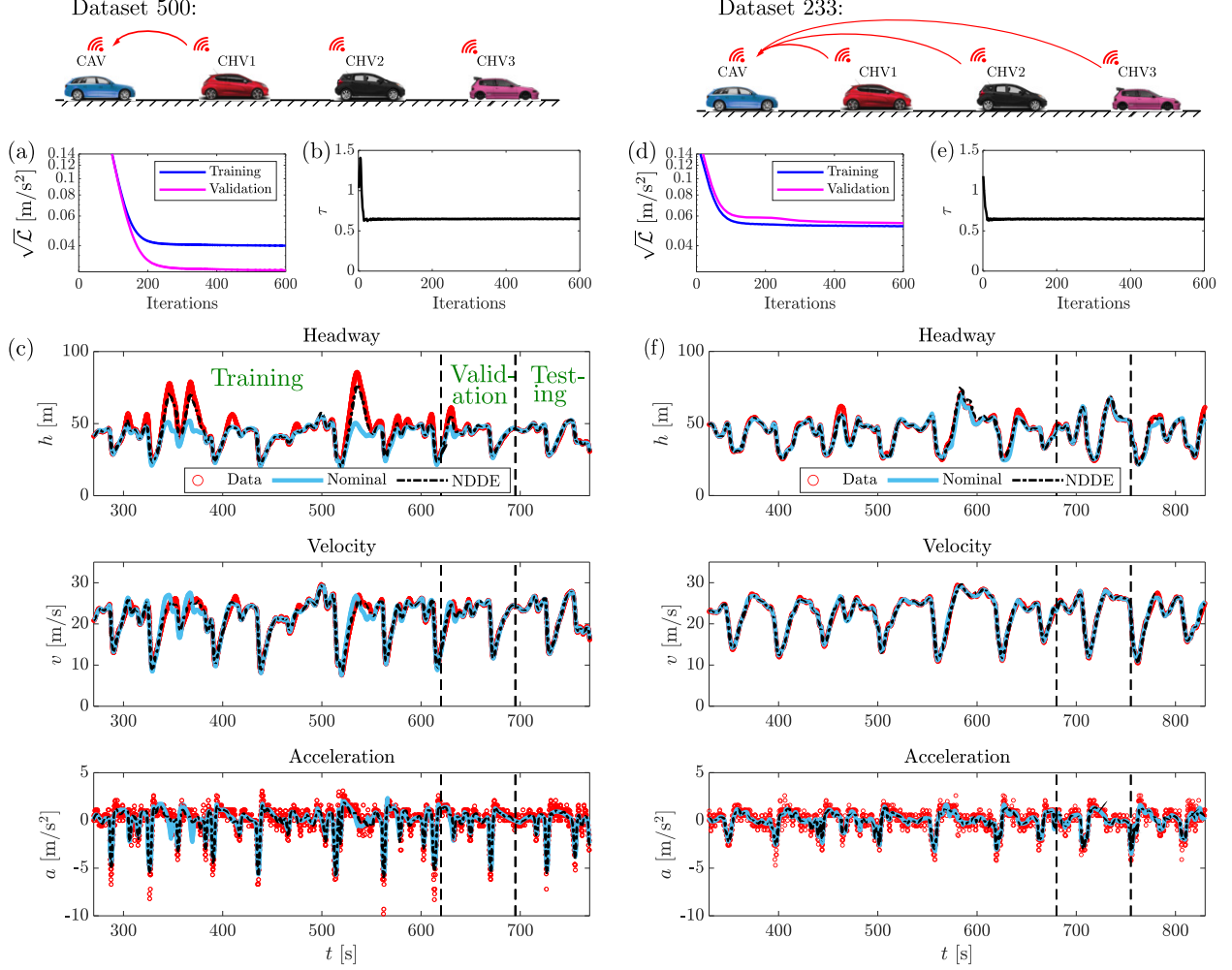


Figure 4.4: NDDE training results on dataset 500 and dataset 233. Panels (a) and (d) show the training and validation loss; panels (b) and (e) display the evolution of the time delay; (c) and (f) depict the simulation of learned NDDE (black) together with the simulation given by the nominal model (light blue) and the experimental data (red). Note that panels (a), (b), (d) and (e) only show the first 600 iterations of training, but the actual training stops later based on the validation loss criterion. The RMSE of the quantities plotted in panels (c) and (f) are collected in Table 4.1.

As shown in Figure 4.4, we utilize two experimental datasets with different controller settings. In ‘dataset 500’, the CAV only responded to the CHV immediately ahead using $(\beta_1, \beta_2, \beta_3) = (0.5, 0.0, 0.0) \text{ s}^{-1}$, while in ‘dataset 233’, the CAV responded to all three CHVs ahead with $(\beta_1, \beta_2, \beta_3) = (0.2, 0.3, 0.3) \text{ s}^{-1}$. Both settings shared the same $\alpha = 0.4 \text{ s}^{-1}$. When simulating the nominal model we use $\tau = 0.6 \text{ s}$ and $p(v) = a + cv^2$ with $a = 0.0981 \text{ m/s}^2$ and $c = 0.0003 \text{ m}^{-1}$; see [56]. The delay and the loss term were unknown in the experiments and some unmodeled dynamics (e.g., powertrain) is not represented in the nominal model (4.15)–

(4.17). This can be observed when comparing the experimental data (red) and the simulation of the nominal model (light blue) in Figure 4.4(c), (f). Our goal here is to learn the overall dynamics of the CAV and the time delay τ simultaneously from the experimental data.

A NDDE with three neurons in the single hidden layer is used when learning the dynamics. The number of the inputs depends on the availability of the data and the number of the time delays in the systems. We presume that one dominant time delay appears in the feedback loop through the inputs $h(t - \tau), v(t - \tau), v_1(t - \tau), v_2(t - \tau), v_3(t - \tau)$ and we also set a time delay to zero corresponding to the loss term containing $v(t)$. There are altogether 6 inputs to the NDDE, 5 of which share the same trainable delay. The output of the NDDE is the acceleration $\dot{v}(t)$, since the kinematic part of equation (4.15) does not require further learning. The NDDEs are trained using Algorithm 3 with LM method, $\mu = 0.01$ and 10 attempts. The initial delays for those attempts are chosen uniformly between 0 and 2 seconds and the maximum allowed delay is set to 3 s.

Approximately five hundred seconds data (with a time step 0.1 second) are utilized from each experiment. The first 70% was used for training, 15% was used for validation, and 15% was used for testing. The training processes and delay learning path are shown in Figure 4.4(a)–(b) for dataset 500 and in Figure 4.4(d)–(e) for dataset 233. In both cases the delay value converges to a number around 0.65 s. In panels (c) and (f), the simulations from the trained NDDE are shown as black dot-dashed lines capturing the experimental data (red circles) well. The simulation results of the nominal model are plotted as light blue solid curves. From panel (c), we can see that the trained NDDE can capture the headway and velocity well in the training segment and gives good predictions in validation and testing segments. The difference between the model based simulations and the learned dynamics is more pronounced for the dataset 500 than for the dataset 233 since in the former case the CAV performed more aggressive maneuvers. The simulation-based root-mean-square errors between prediction and data are collected in Table 4.1 for headway, velocity and acceleration. To show the convergence in the time delay parameters, we run the training algorithm 100 times and plot the distribution of the final learned delay in Figure 4.5. For both datasets, the delay converges to a distribution concentrated around 0.65 s.

4.3 Summary

In this chapter, we showed the importance of time delays in the longitudinal dynamics of connected automated vehicles by performing stability analysis on a car-following model. We demonstrated that the time delay can affect the stability and the convergence rate of the longitudinal dynamics. With the knowledge of the time delay, the fastest-convergence

RMSE	Dataset 500		Dataset 233	
	NDDE	Nominal	NDDE	Nominal
h [m]	2.7682	7.9383	1.6971	1.7633
v [m/s]	0.5865	1.1765	0.3680	0.3884
$\dot{v}$ [m/s ²]	0.6057	0.6933	0.4383	0.4423

Table 4.1: Testing results for the two datasets.

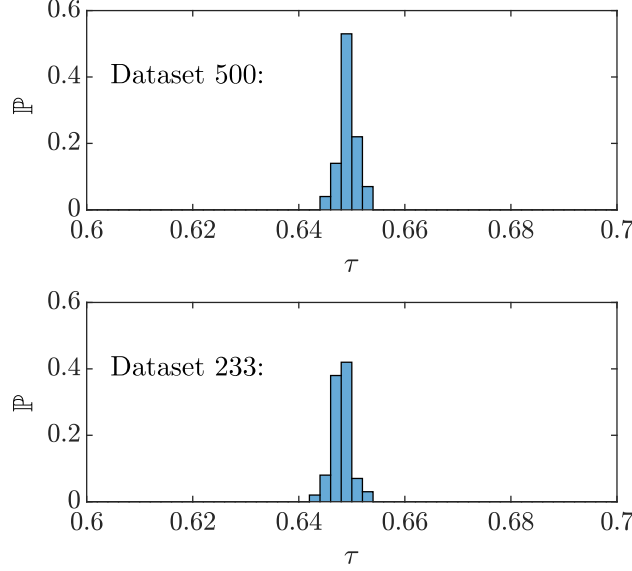


Figure 4.5: Distributions of learned delay in 100 runs for the two datasets.

control gains can be derived analytically. We also studied the robustness of such gains against changes in time delays and unveiled that the critical delay where the system loses stability is around 2.5 times of the time delay used in the design (this constant linear relationship does not depend on other parameters).

We also showed that the time delay can be learned from experimental data using a NDDE with a trainable delay. In the example, the delays converge to a value around 0.65 second consistently for two datasets and the networks give better results compared to the nominal model with the consideration of the resistance terms. This aligns well with the study in [57] where the time delay was treated as a random variable. With the knowledge of the time delay, one can design better controllers for connected automated vehicles.

CHAPTER 5

Learning the Lateral Dynamics of a Teleoperated Vehicle

As the deployment of artificial intelligence in autonomous driving has been accelerated in the past decades [25, 78, 105], the risk and safety concerns of driverless vehicles also grew, especially when these vehicles experience onboard system failures. Teleoperated driving (ToD) may be utilized to help automated vehicles in such emergency situations. In addition, ToD may be used in other applications such as fleet delivery and valet parking [47].

In this chapter, we analyze the effects of delays in the lateral dynamics of vehicles, specifically on the stability and the steering control performance in ToD. Analytical studies on the effects of constant and saw-tooth periodic latency are presented, offering insights into numerical investigations under stochastic, time-varying latency. Moreover, we construct a NDDE with the help of known physics-based nonlinearity and apply our learning algorithm to identify the overall latency in the ToD system from human-in-the-loop data. Evaluation of the ToD system is performed after the latency and the equivalent control gains are learned.

5.1 Latency in Teleoperated Driving System

Latency causes significant challenges in ToD design and operation. As illustrated in the top of Figure 5.1, the overall latency in ToD is composed of the communication latency, the actuation latency, and the processing time of the remote operator. Communication latency refers to the time required to transmit the information through the mobile networks, including the time needed for encoding/decoding as well as the travel time of packets [62, 38, 5, 6]. The actuation latency [33] arises from the time required for the actuator to execute the control command. For example, delays may arise in steering controllers and in mechanical actuators [77, 3, 117, 116, 118]. The processing time is the time required by the remote operator to generate control commands based on the input information. This

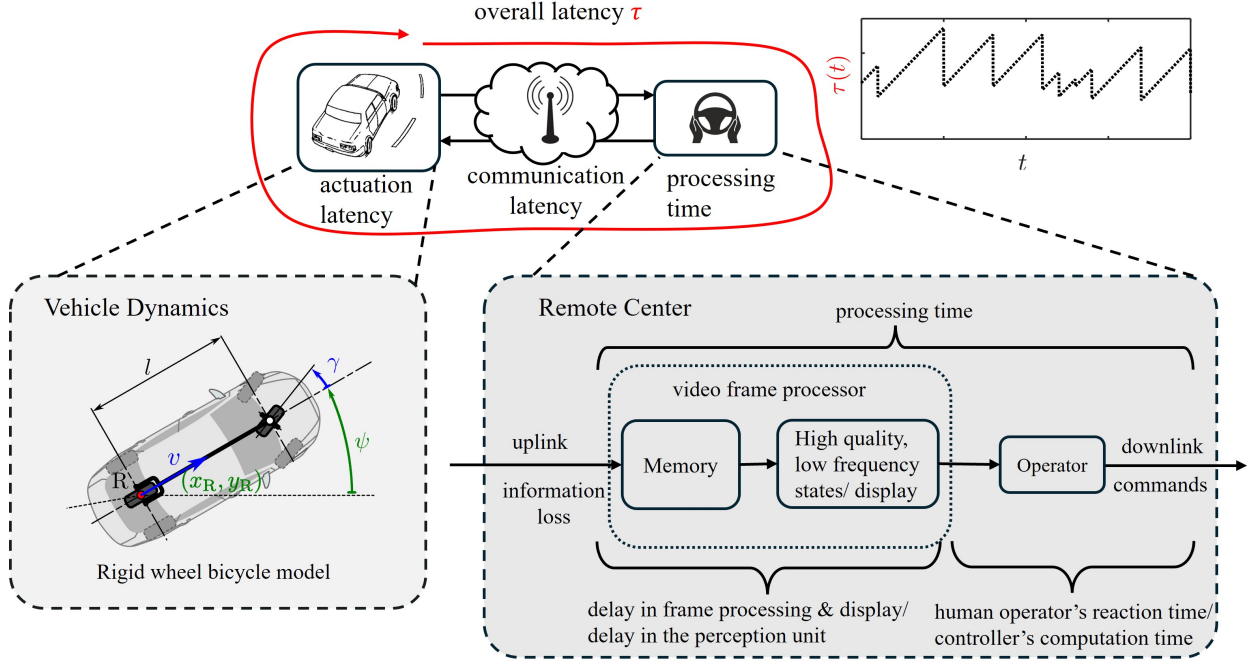


Figure 5.1: Components in a ToD control loop.

includes the image processing time [26], operator sampling time [67] and/or human reaction time [100, 98]. In this chapter, we consider the overall latency in the ToD system as the sum of the communication latency, actuation latency and processing time, and this overall latency may vary in time as explained below.

For instance, packet loss may occur in the communication network when the transmission data demand is high and the bandwidth is limited. This reduces the video quality, making teleoperation challenging for a human operator. If ToD is performed by an automated system, the latencies arising due to packet loss may lead to large state estimation errors and, in turn, to performance degradation. In response to this, the video frame processor may choose to reduce the update frequency in order to improve the video quality or state estimation [19]. However, such infrequent updates reduce the rate at which new control commands are generated. Assuming that the vehicle actuator continues to execute the same command until a new one is received, the latency increases linearly in time between command updates [112, 102, 52]. Once a new command is put in place, the latency drops to its base value, creating a sawtooth-shape profile as illustrated in the top right of Figure 5.1. The effects of such sawtooth-shape latency on the longitudinal vehicle dynamics has been studied in the context of controlling connected automated vehicles [92, 86].

In this chapter, we first analytically study the effect of constant latency on the lateral dynamics. Then, to bridge the gap between the constant delay and actual latency in the

teleoperated driving loop, we consider periodic latency approximations of the time-varying latency. We provide detailed stability and performance analysis for periodic latency profiles. This is followed by the study of stochastic time-varying latency via numerical simulations and compared to the average constant latency. In the end, we present how to use a NDDE and a trainable delay to learn the ToD behavior from human-in-the-loop data.

5.2 Stability of Lateral Vehicle Dynamics under Latency

5.2.1 Effect of constant latency

We start the analysis with a simple maneuver, where the teleoperated vehicle follows a straight path, and extend the study to the circular path following. Simulations are provided to validate the analytical results.

Following a straight path

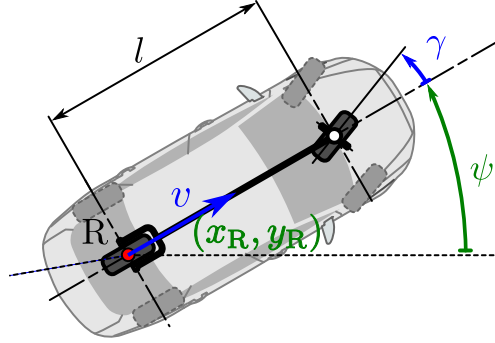


Figure 5.2: Kinematic vehicle model with respect to point R.

Consider the kinematic vehicle model [73]

$$\begin{aligned}\dot{x}_R(t) &= v \cos(\psi(t)), \\ \dot{y}_R(t) &= v \sin(\psi(t)), \\ \dot{\psi}(t) &= \frac{v}{l} \tan(\gamma(t)),\end{aligned}\tag{5.1}$$

shown in Figure 5.2 with controller

$$\gamma(t) = \arctan \left(-k_y y_R(t - \tau) - k_\psi \psi(t - \tau) \right),\tag{5.2}$$

aiming to stabilize the rectilinear motion $y_R \equiv 0$, $\psi \equiv 0$. Here (x_R, y_R) are the coordinates of the center of the rear axle, ψ is the yaw angle, and γ is the steering control input. The parameters are the longitudinal velocity v , the wheelbase, l , the time delay τ , and the control gains k_y, k_ψ .

We first nondimensionalize the system using the dimensionless quantities $\bar{t} = tv/l$, $\bar{\tau} = \tau v/l$, $\bar{y} = y_R/l$, $\bar{x} = x_R/l$, $\bar{\psi} = \tilde{\psi}$ which yields

$$\begin{aligned}\bar{x}'(\bar{t}) &= \cos(\bar{\psi}(\bar{t})), \\ \bar{y}'(\bar{t}) &= \sin(\bar{\psi}(\bar{t})), \\ \bar{\psi}'(\bar{t}) &= -l k_y \bar{y}(\bar{t} - \bar{\tau}) - k_\psi \bar{\psi}(\bar{t} - \bar{\tau}),\end{aligned}\tag{5.3}$$

where the prime represents derivative with respect to the dimensionless time $\bar{t}$. Then we linearize the system around the steady state $\bar{x}^* \equiv \bar{t} + \bar{x}(0)$, $\bar{y}^* \equiv 0$, $\bar{\psi}^* \equiv 0$. By defining the perturbations $\tilde{x} = \bar{x} - x^*$, $\tilde{y} = \bar{y} - y^*$, $\tilde{\psi} = \bar{\psi} - \psi^*$, we obtain the linear and dimensionless system

$$\begin{aligned}\tilde{x}'(\bar{t}) &= 0, \\ \tilde{y}'(\bar{t}) &= \tilde{\psi}(\bar{t}), \\ \tilde{\psi}'(\bar{t}) &= -l k_y \tilde{y}(\bar{t} - \bar{\tau}) - k_\psi \tilde{\psi}(\bar{t} - \bar{\tau}).\end{aligned}\tag{5.4}$$

The nondimensionalization does not alter the stability properties of the original system (5.1), but demonstrates that the velocity and the latency has the same effect on stability. In fact, varying v and τ such that $\bar{\tau} = \tau v/l$ is kept the same means that the vehicle is running along the same trajectory in space, see the example in [53].

The stability of lateral dynamics is given by the characteristic equation

$$\lambda^2 + k_\psi e^{-\lambda \bar{\tau}} \lambda + k_y l e^{-\lambda \bar{\tau}} = 0,\tag{5.5}$$

that is obtained from the second and third rows of (5.4). It has infinitely many characteristic roots λ in the complex plane due to the delay. The rightmost roots give the dominant behavior of the system. The state $\bar{y}^* \equiv 0$, $\bar{\psi}^* \equiv 0$ is asymptotically stable if and only if all characteristic roots of (5.5) have negative real parts, i.e., $\text{Re}(\lambda_i) < 0, i = 1, 2, \dots$

Substituting $\lambda = 0$ yields the stability boundary

$$l k_y = 0,\tag{5.6}$$

and using $\lambda = j\omega$, $\omega > 0$ results in the stability boundary

$$\begin{aligned} k_\psi &= \omega \sin \omega \bar{\tau}, \\ l k_y &= \omega^2 \cos \omega \bar{\tau}, \end{aligned} \quad (5.7)$$

which is parameterized by the angular frequency ω . The stability boundaries related to different levels of the dimensionless delay are shown in Figure 5.3(a). The stable region shrinks as the dimensionless delay $\bar{\tau}$ increases. Since $\bar{\tau} = v\tau/l$, increasing the velocity or increasing the actual delay in the system has the same effect on stability.

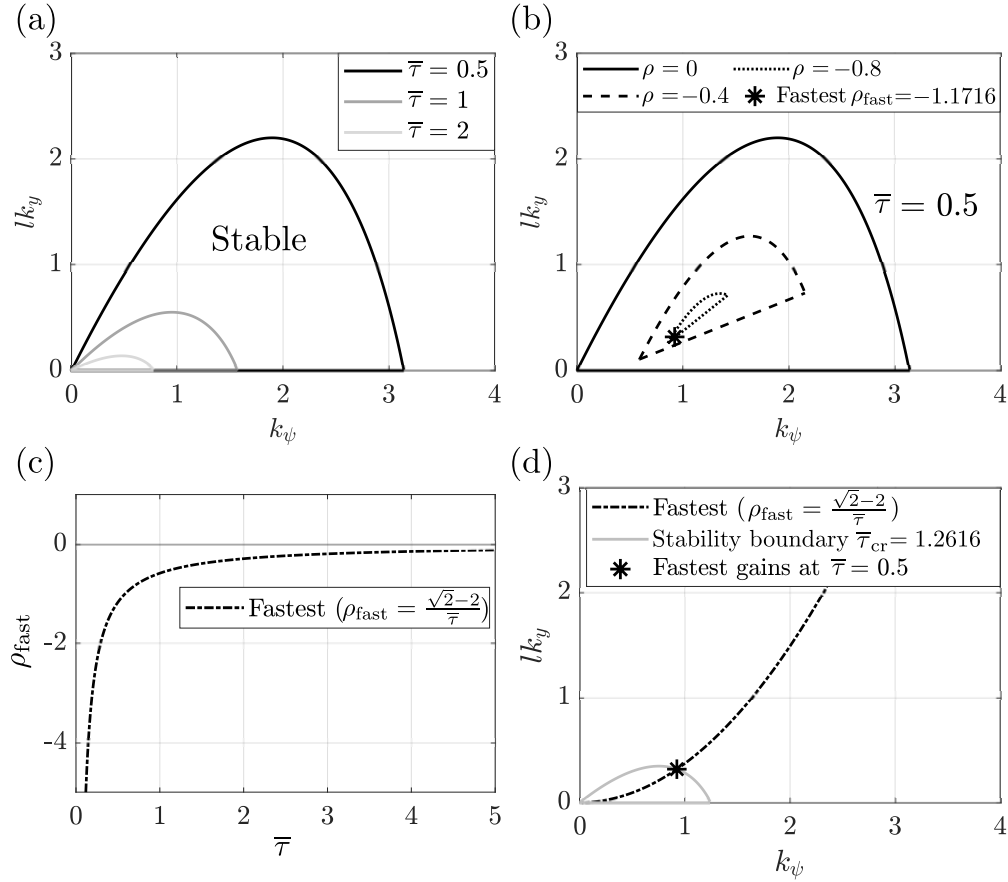


Figure 5.3: (a) Stability boundaries for different dimensionless delays $\bar{\tau}$. (b) Performance curves for different convergence rates ρ values when $\bar{\tau} = 0.5$. (c) The fastest convergence rate ρ_{fast} as a function of $\bar{\tau}$. (d) The control gains which give ρ_{fast} are plotted as the black dashed-dotted curve. This curve intersects the stability boundary of $\bar{\tau}_{\text{cr}} = 1.2616$ (grey solid curve) when $\bar{\tau} = 0.5$.

The real parts of the rightmost roots indicate how fast the state of the dynamical system converges to the steady state given an initial error. By substituting $\lambda = \rho + j\omega$, $\rho < 0$, $\omega \geq 0$

into the characteristic equation (5.5), one can solve for the curves with respect to each ρ . For $\omega = 0$, we have

$$l k_y = -\rho e^{\rho\bar{\tau}}(\rho + k_\psi e^{-\rho\bar{\tau}}), \quad (5.8)$$

and for $\omega > 0$, we obtain

$$\begin{aligned} k_\psi &= -e^{\rho\bar{\tau}}((\rho^2 - \omega^2) \sin(\omega\bar{\tau}) + 2\rho\omega \cos(\omega\bar{\tau}))/\omega, \\ l k_y &= -e^{\rho\bar{\tau}}((\rho^2 - \omega^2) \cos(\omega\bar{\tau}) - 2\rho\omega \sin(\omega\bar{\tau})) - k_\psi\rho. \end{aligned} \quad (5.9)$$

These so-called performance curves are plotted for different ρ values in Figure 5.3(b) when $\bar{\tau} = 0.5$. Note that the curves (5.9) are parameterized by the angular frequency ω such that $\omega \rightarrow 0$ locates at the left corner of the performance domain.

Indeed, the performance curve for $\rho = 0$ is the stability boundary. As the ρ decreases, the region enclosed by performance curves shrinks and eventually disappears at ρ_{fast} , which gives the fastest convergence rate. This happens when the slope of (5.8) and the slope of (5.9) for $\omega \rightarrow 0$ become equal. This yields

$$\rho_{\text{fast}} = \frac{\sqrt{2} - 2}{\bar{\tau}}, \quad (5.10)$$

which only depends on the dimensionless delay $\bar{\tau}$, as depicted in Figure 5.3(c). Substituting this into (5.9) and using $\omega \rightarrow 0$, we obtain the gains corresponding to ρ_{fast} as functions of $\bar{\tau}$:

$$\begin{aligned} k_{\psi,\text{fast}} &= \frac{p}{\bar{\tau}}, \\ l k_{y,\text{fast}} &= \frac{q}{\bar{\tau}^2}, \end{aligned} \quad (5.11)$$

where $p = e^{\sqrt{2}-2} (2\sqrt{2} - 2)$ and $q = e^{\sqrt{2}-2} (10\sqrt{2} - 14)$. This is plotted as a dotted-dashed curve in Figure 5.3(d).

This curve intersects all stability boundaries, which means that the fastest-convergence gains designed for a given delay $\bar{\tau}$ are located on the stability boundary of another (larger) delay $\bar{\tau}_{\text{cr}}$. We can derive the analytical relationship between $\bar{\tau}$ and $\bar{\tau}_{\text{cr}}$ by equating (5.7) and (5.11):

$$\begin{aligned} k_{\psi,\text{fast}}(\bar{\tau}) &= k_\psi(\omega_{\text{cr}}, \bar{\tau}_{\text{cr}}), \\ l k_{y,\text{fast}}(\bar{\tau}) &= l k_y(\omega_{\text{cr}}, \bar{\tau}_{\text{cr}}), \end{aligned} \quad (5.12)$$

which yields

$$\begin{aligned}\omega_{\text{cr}} &= \frac{g}{\bar{\tau}}, \\ \bar{\tau}_{\text{cr}} &= \frac{1}{g} \sin^{-1} \left(\frac{p}{g} \right) \bar{\tau} \approx 2.5232 \bar{\tau},\end{aligned}\tag{5.13}$$

where $g = \sqrt{\frac{p^2 + \sqrt{p^4 + 4q^2}}{2}}$. Observe that the coefficient giving the linear relationship between $\bar{\tau}$ and $\bar{\tau}_{\text{cr}}$ does not depend on any parameter. In Figure 5.3(d), we mark the gains that give the fastest convergence for $\bar{\tau} = 0.5$ (black star), which is indeed located on the stability boundary for $\bar{\tau}_{\text{cr}} = 2.5232\bar{\tau} \approx 1.2616$ (gray curve). One may notice that the results presented here are analogous to those in Section 4.1.

These results quantify the robustness of the fastest-convergence gains against the change of the dimensionless delay $\bar{\tau} = v\tau/l$, i.e., against the change of the velocity v and actual delay τ in the remote-driving control loop. If the fastest-convergence gains are chosen, the system can maintain stability even when either the velocity or the delay or their product increases up to 2.5 times. Note that, in case of human operator, the fastest-convergence gains may not be achieved and stability may not be guaranteed for larger delays.

Following a curved path

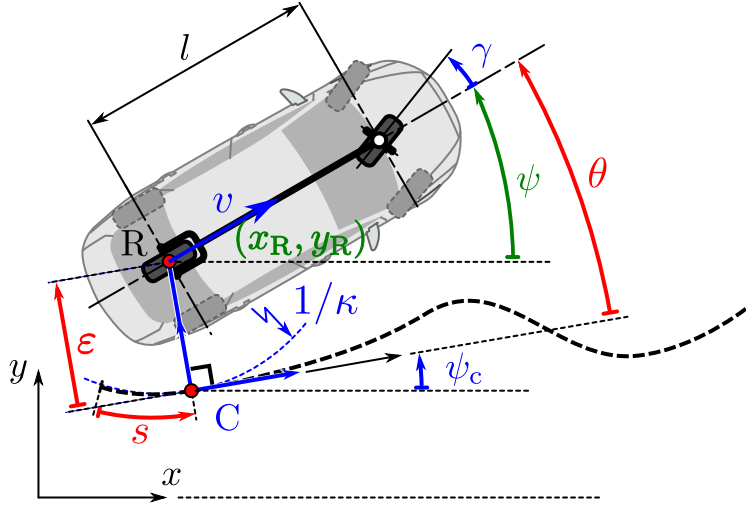


Figure 5.4: Rewrite the kinematic model with respect to the path.

If the vehicle is following a pre-defined path, we can formulate the model (5.1) in the path coordinate system, see Figure 5.4. The closest point to R along the path is point C, which

is located at (x_C, y_C) and has the tangential angle ψ_C and the curvature κ . Thus, the lateral deviation from the path is defined as

$$\varepsilon = -(x_R - x_C) \sin \psi_C + (y_R - y_C) \cos \psi_C, \quad (5.14)$$

and the relative yaw angle is defined as

$$\theta = \psi - \psi_C. \quad (5.15)$$

With these coordinates, the model (5.1) can be written as

$$\begin{aligned} \dot{s} &= \frac{v \cos \theta}{1 - \kappa \varepsilon}, \\ \dot{\varepsilon} &= v \sin \theta, \\ \dot{\theta} &= \frac{v}{l} \tan \gamma - \frac{v \kappa \cos \theta}{1 - \kappa \varepsilon}, \end{aligned} \quad (5.16)$$

where s is the arclength at point C.

Inspired by the controller in [103], we use the controller

$$\gamma_d = \arctan \left(l \kappa - k_1 (\theta + \arctan(k_2 \varepsilon)) \right), \quad (5.17)$$

to assign the desired steering angle, where k_1, k_2 are the control gains. In real-world implementations, the velocity and steering angle commands may not be achieved immediately due to the latency. Here we assume that the vehicle can realize the desired velocity v_d and steering command γ_d with some delay, that is, $v(t) = v_d(t - \tau)$, $\gamma(t) = \gamma_d(t - \tau)$. Then the closed-loop system can be represented by the delay differential equations

$$\begin{aligned} \dot{s}(t) &= \frac{v_d(t - \tau) \cos \theta(t)}{1 - \kappa(t) \varepsilon(t)}, \\ \dot{\varepsilon}(t) &= v_d(t - \tau) \sin \theta(t), \\ \dot{\theta}(t) &= - \frac{v_d(t - \tau) \kappa(t) \cos \theta(t)}{1 - \kappa(t) \varepsilon(t)} + \frac{v_d(t - \tau)}{l} \left(l \kappa(t - \tau) - k_1 \left(\theta(t - \tau) + \arctan(k_2 \varepsilon(t - \tau)) \right) \right). \end{aligned} \quad (5.18)$$

For the stability analysis, we assume that the vehicle is running at a constant velocity $v(t) \equiv v_*$ while trying to approach a circular reference path of constant curvature $\kappa(t) \equiv \kappa_*$. The system (5.18) has a desired steady-state solution

$$s_*(t) = v_* t, \quad \varepsilon_*(t) \equiv 0, \quad \theta_*(t) \equiv 0. \quad (5.19)$$

By defining the perturbations $\tilde{s} = s - s_*$, $\tilde{\varepsilon} = \varepsilon - \varepsilon_*$, $\tilde{\theta} = \theta - \theta_*$, we can obtain the linearized system

$$\begin{aligned}\dot{\tilde{s}}(t) &= v_* \kappa_* \tilde{\varepsilon}(t), \\ \dot{\tilde{\varepsilon}}(t) &= v_* \tilde{\theta}(t), \\ \dot{\tilde{\theta}}(t) &= -v_* \kappa_*^2 \tilde{\varepsilon}(t) - \frac{v_* k_1 k_2}{l} \tilde{\varepsilon}(t - \tau) - \frac{v_* k_1}{l} \tilde{\theta}(t - \tau).\end{aligned}\tag{5.20}$$

Again, this can be nondimensionalized by using the scaled quantities $\bar{t} = tv_*/l$, $\bar{\tau} = \tau v_*/l$, $\hat{s} = \tilde{s}/l$, $\hat{\varepsilon} = \tilde{\varepsilon}/l$, $\hat{\theta} = \tilde{\theta}$:

$$\begin{aligned}\hat{s}'(\bar{t}) &= l \kappa_* \hat{\varepsilon}(\bar{t}), \\ \hat{\varepsilon}'(\bar{t}) &= \hat{\theta}(\bar{t}), \\ \hat{\theta}'(\bar{t}) &= -l^2 \kappa_*^2 \hat{\varepsilon}(\bar{t}) - k_1 k_2 l \hat{\varepsilon}(\bar{t} - \bar{\tau}) - k_1 \hat{\theta}(\bar{t} - \bar{\tau}),\end{aligned}\tag{5.21}$$

where the latency is now the dimensionless delay $\bar{\tau}$.

To simplify the notation, we utilize the dimensionless gains

$$\begin{aligned}k_\theta &= k_1, \\ k_\varepsilon &= k_1 k_2 l,\end{aligned}\tag{5.22}$$

corresponding to the yaw angle error and lateral error, respectively. As the lateral stability is determined by the second and third rows of (5.21), we arrive at the system

$$\begin{aligned}\mathbf{x}'(\bar{t}) &= \mathbf{a}\mathbf{x}(\bar{t}) + \mathbf{b}\mathbf{x}(\bar{t} - \bar{\tau}), \\ \mathbf{x} &= \begin{bmatrix} \hat{\varepsilon} \\ \hat{\theta} \end{bmatrix}, \quad \mathbf{a} = \begin{bmatrix} 0 & 1 \\ -l^2 \kappa_*^2 & 0 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 & 0 \\ -k_\varepsilon & -k_\theta \end{bmatrix}.\end{aligned}\tag{5.23}$$

The constant overall latency results in a constant dimensionless delay $\bar{\tau} = \tau v_*/l$. The characteristic equation of (5.23) is given by

$$\lambda^2 + k_\theta \lambda e^{-\lambda \bar{\tau}} + k_\varepsilon e^{-\lambda \bar{\tau}} + l^2 \kappa_*^2 = 0.\tag{5.24}$$

Again, we quantify the control performance using the real part of the rightmost roots. When the leading roots are on the negative real axis, the system converges to the steady state without oscillations. When the imaginary parts of the rightmost roots are nonzero, oscillatory behavior is observed (see Figure 5.8(b) and (c) for demonstration).

By substituting $\lambda = \rho + j\omega$, $\omega \geq 0$ into the characteristic equation (5.24), one can solve

for the gains k_θ and k_ε as a function of ρ and ω . We compute performance boundaries on the plane of the gain parameters as curves that belong to a given convergence rate ρ . For $\omega = 0$, we have the performance boundaries

$$\begin{aligned} k_\theta &= -e^{\rho\bar{\tau}}((\rho^2 + l^2\kappa_*^2)\bar{\tau} + 2\rho), \\ k_\varepsilon &= -e^{\rho\bar{\tau}}(\rho^2 + l^2\kappa_*^2) - \rho k_\theta, \end{aligned} \quad (5.25)$$

that are straight lines in the $(k_\theta, k_\varepsilon)$ plane. For $\omega > 0$, we obtain

$$\begin{aligned} k_\theta &= e^{\rho\bar{\tau}} \frac{(\omega^2 - l^2\kappa_*^2 - \rho^2) \sin(\omega\bar{\tau}) - 2\rho\omega \cos(\omega\bar{\tau})}{\omega}, \\ k_\varepsilon &= e^{\rho\bar{\tau}} \left((\omega^2 - l^2\kappa_*^2 - \rho^2) \cos(\omega\bar{\tau}) + 2\rho\omega \sin(\omega\bar{\tau}) \right) - \rho k_\theta, \end{aligned} \quad (5.26)$$

which are curves parameterized by the angular frequency ω such that the angular frequency increases from $\omega = 0$ as moving along the performance boundary.

Substituting $\rho = 0$ yields the stability boundaries

$$k_\varepsilon = -l^2\kappa_*^2, \quad (5.27)$$

and

$$\begin{aligned} k_\theta &= (\omega^2 - l^2\kappa_*^2) \frac{\sin(\omega\bar{\tau})}{\omega}, \\ k_\varepsilon &= (\omega^2 - l^2\kappa_*^2) \cos(\omega\bar{\tau}). \end{aligned} \quad (5.28)$$

The stability boundaries of different scaled delays are plotted in Figure 5.5(a) and (b) for curvatures $\kappa_* = 0 \text{ m}^{-1}$ and $\kappa_* = 0.2 \text{ m}^{-1}$, respectively. The stable region shrinks as the scaled delay increases and increasing the curvature shifts the stable region downwards. In panels (c)-(f), we show the numerical simulations of the original nonlinear system (5.18) with gains $k_\theta = 1, k_\varepsilon = 0.45$, latency $\tau = 0.5 \text{ s}$, and wheelbase $l = 2.73 \text{ m}$ while following a straight path ($\kappa_* = 0 \text{ m}^{-1}$) and a 5-meter-radius circle ($\kappa_* = 0.2 \text{ m}^{-1}$). The initial position and the orientation of the vehicle are visualized using black arrows. By choosing different velocities $v_* = 2.73 \text{ m/s}$ and $v_* = 5.46 \text{ m/s}$, the scaled delays are $\bar{\tau} = 0.5$ and $\bar{\tau} = 1$, respectively. For $\bar{\tau} = 0.5$ the control gains are located within the stability boundary in panels (a)-(b) and the trajectories converge to the desired paths in panels (c) and (e). For zero curvature, as the scaled delay increases to $\bar{\tau} = 1$, the stability boundary gets close to the selected gains while still enclosing them in panel (a). This results in the oscillatory (but still stable) motion in panel (d). That is, increasing the scaled delay leads to performance degradation. For nonzero curvature, the same gains fall outside the stable region in panel (b) and simulations

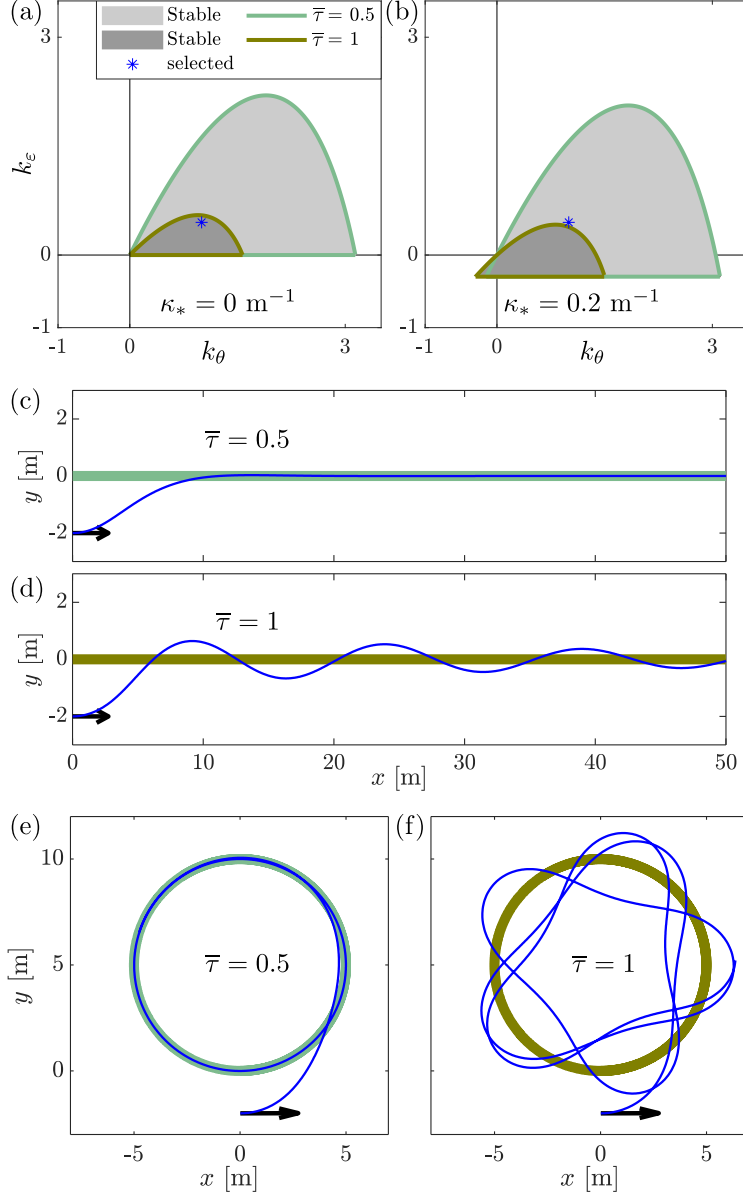


Figure 5.5: (a)-(b) Stability charts for curvatures $\kappa_* = 0 \text{ m}^{-1}$ and $\kappa_* = 0.2 \text{ m}^{-1}$. (c)-(d) Simulations when following a straight path ($\kappa_* = 0 \text{ m}^{-1}$) under different scaled delays. (e)-(f) Simulations when following a circular path ($\kappa_* = 0.2 \text{ m}^{-1}$) under different scaled delays.

in panel (f) show that the system no longer converges to the circular path. The performance of the controller is largely affected by the scaled delay, i.e., the longitudinal velocity and the overall latency in the loop. The path curvature can also limit the performance in the presence of the latency. The stability analysis provides insights about the behavior of the original nonlinear system and informs us about the performance of a given controller for different maneuvers.

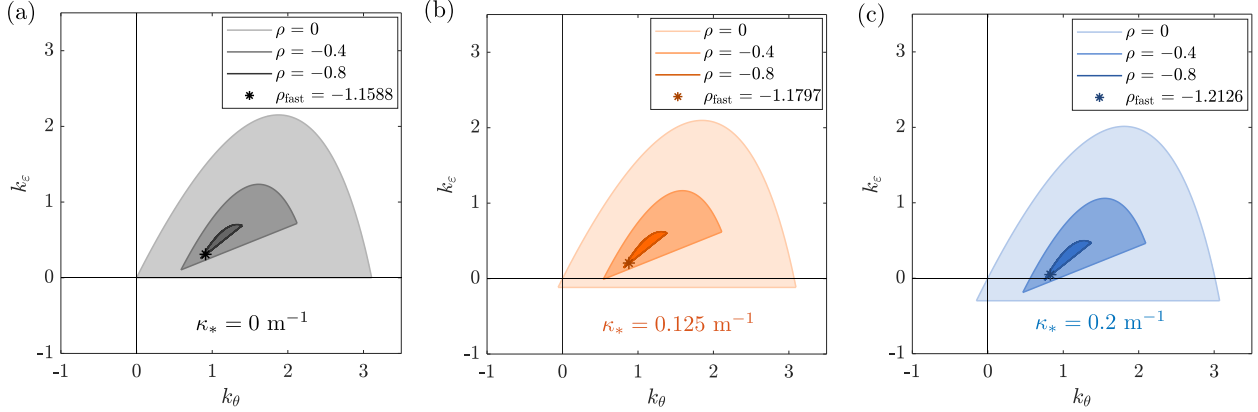


Figure 5.6: Performance boundaries and stability boundaries for $\bar{\tau} = 0.5$ ($l = 2.73$ m, $v_* = 3$ m/s and $\tau = 0.46$ s). The stars indicate gains of fastest convergence. Different panels correspond to different curvatures: (a) $\kappa_* = 0$, (b) $\kappa_* = 0.125$ m $^{-1}$, and (c) $\kappa_* = 0.2$ m $^{-1}$.

The performance boundaries of different convergence rates (including the stability boundaries) are shown in Figure 5.6(a), (b) and (c) for curvatures $\kappa_* = 0$ m $^{-1}$, $\kappa_* = 0.125$ m $^{-1}$ and $\kappa_* = 0.2$ m $^{-1}$, respectively. The other parameter are chosen as $l = 2.73$ m, $v_* = 3$ m/s and $\tau = 0.46$ s, yielding $\bar{\tau} = 0.5$. The outmost boundaries ($\rho = 0$) are the stability boundaries enclosing the stable (lightly shaded) area, for which $\rho < 0$. As ρ decreases, the region enclosed by performance curves shrinks as illustrated by darker shadings.

Eventually the area disappears at $\rho = \rho_{\text{fast}}$ (when the slope of (5.25) and the slope of (5.26) for $\omega \rightarrow 0$ become equal), which gives the fastest convergence rate in space. This yields

$$\rho_{\text{fast}} = -\frac{2}{\bar{\tau}} + \sqrt{\frac{2}{\bar{\tau}^2} - l^2 \kappa_*^2}, \quad (5.29)$$

when $\bar{\tau} \leq \sqrt{2}/(l\kappa_*)$. For $\bar{\tau} > \sqrt{2}/(l\kappa_*)$ the fastest convergence rate can be determined numerically but in this case the longitudinal velocity is so large that the kinematic bicycle model loses its validity. According to (5.25) the control gains corresponding to the fastest convergence are

$$\begin{aligned} k_{\theta}^{\text{fast}} &= -e^{\rho_{\text{fast}} \bar{\tau}} \left((\rho_{\text{fast}}^2 + l^2 \kappa_*^2) \bar{\tau} + 2\rho_{\text{fast}} \right), \\ k_{\epsilon}^{\text{fast}} &= -e^{\rho_{\text{fast}} \bar{\tau}} (\rho_{\text{fast}}^2 + l^2 \kappa_*^2) - \rho_{\text{fast}} k_{\theta}^{\text{fast}}. \end{aligned} \quad (5.30)$$

When following a straight path ($\kappa_* = 0$), the fastest convergence rate (5.29) and gains (5.30) are simplified to (5.10) and (5.11) with $k_{\theta} = k_{\psi}$, $k_y = k_{\epsilon}$.

The fastest convergence rate (5.29) is plotted as a function of $\bar{\tau}$ in Figure 5.7(a) for different curvature values. For small dimensionless delay values $\bar{\tau} < 0.5$, the fastest convergence rate

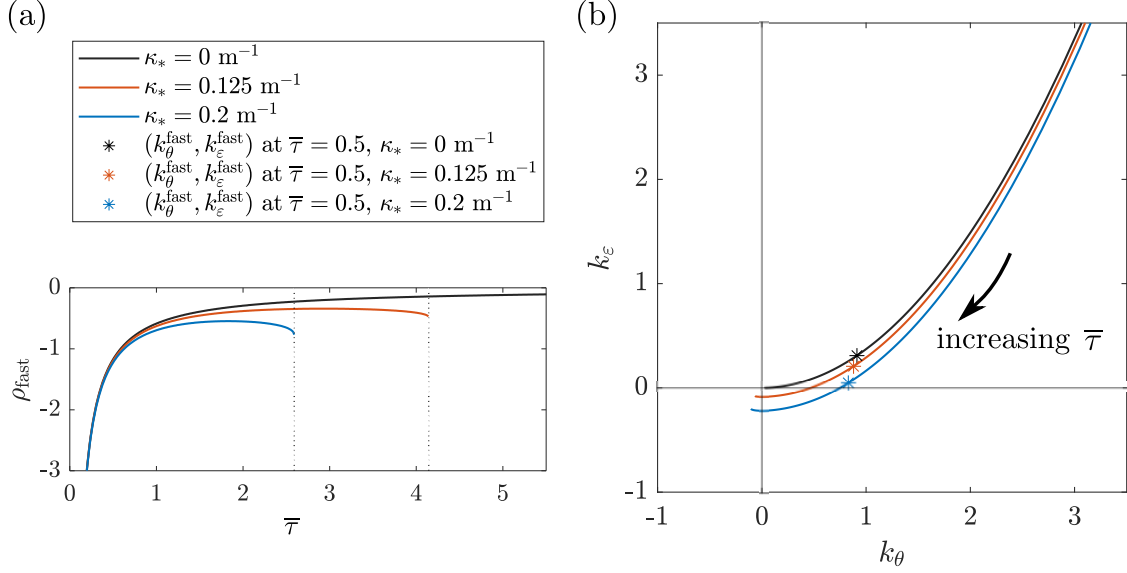


Figure 5.7: (a) Fastest convergence rates for different curvatures as a function of the dimensionless delay. (b) Fastest convergence gains for different curvatures.

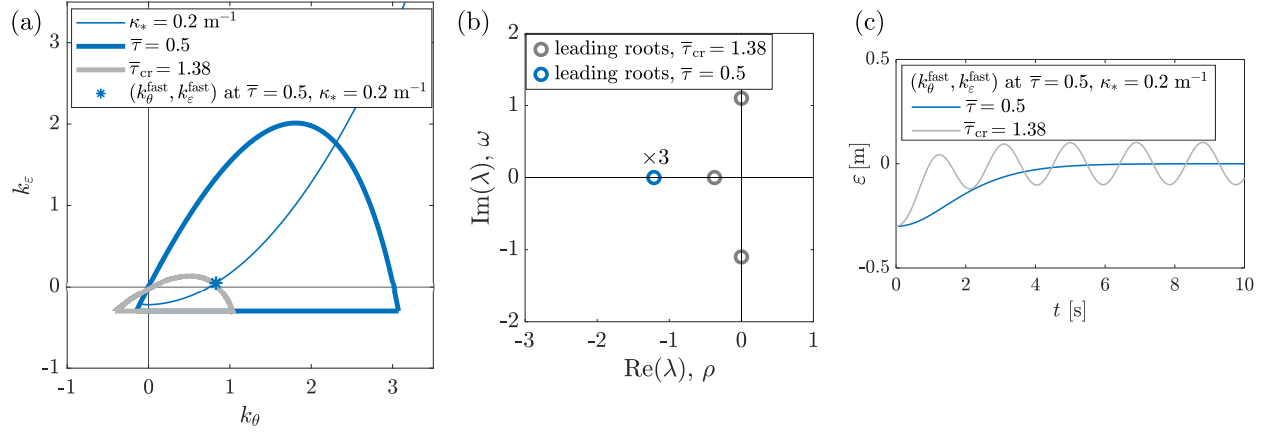


Figure 5.8: Stability chart, characteristic roots and simulations of following a circular path (radius 5 m) under $\bar{\tau}$ and $\bar{\tau}_{\text{cr}}$ while using the same control gains. (a) The fastest-convergence gains obtained for $\bar{\tau}$ are located on the stability boundary for $\bar{\tau}_{\text{cr}}$. (b) The leading characteristic roots for the same gains under different dimensionless delays. (c) Simulation of lateral error while using the two different delays.

changes rapidly and the influence of the curvature is negligible. Note that for $\kappa_* = 0$, ρ_{fast} increases monotonically with $\bar{\tau}$, while for $\kappa_* > 0$, the curve has a maximum. The location of fastest convergence gains (5.30) are shown in Figure 5.7(b) for different curvature values. Along each curve, the dimensionless delay increases from right to left and stars mark the fastest convergence gains when $\bar{\tau} = 0.5$ (also shown in Figure 5.6 and Figure 5.8).

Since these curves intersect the stability boundary (5.28) for other delay values, the fastest

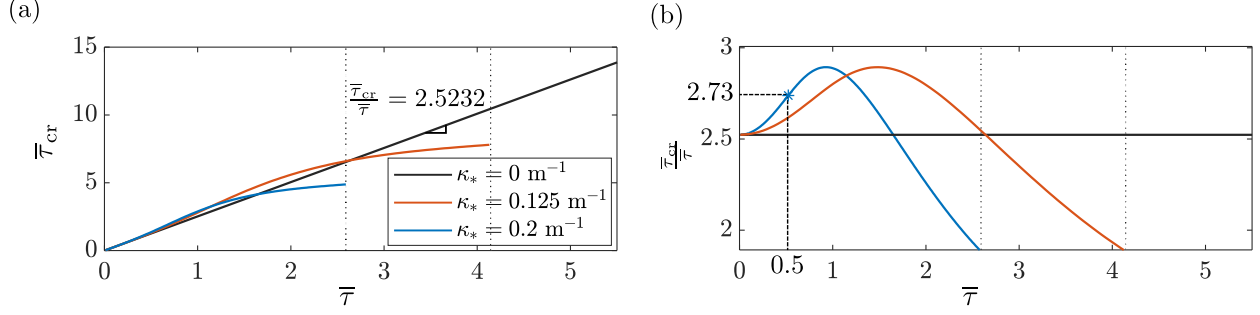


Figure 5.9: (a) Critical dimensionless delay as a function of the dimensionless delay. For the straight path ($\kappa_* = 0 \text{ m}^{-1}$), the slope is constant. (b) Robustness coefficients as a function of the dimensionless delay.

convergence gains for a certain $\bar{\tau}$ may still lead to unstable behavior if the dimensionless delay is increased to $\bar{\tau}_{cr}$. For example, in Figure 5.8(a), the blue star marks the fastest convergence gains for $\bar{\tau} = 0.5$, and this is located on the stability boundary of the larger delay $\bar{\tau}_{cr} = 1.38$. The same control gains correspond to different leading characteristic roots in panel (b), due to the change in the dimensionless delays. For $\bar{\tau} = 0.5$ the leading roots are given by a triple root with negative real part, while for $\bar{\tau}_{cr} = 1.38$ two of these roots form a purely imaginary complex conjugate pair. Simulations of the lateral error under different dimensionless delays are shown in panel (c). For $\bar{\tau} = 0.5$ the error decays, while for $\bar{\tau}_{cr} = 1.38$ periodic oscillations develop, whose angular frequency is close to that of the leading roots.

One can derive the critical dimensionless delay $\bar{\tau}_{cr}$ for which the stability is lost when the fastest convergence control gains are chosen for a certain dimensionless delay $\bar{\tau}$. By setting the equation (5.28) equal to (5.30) yields

$$\begin{aligned} k_{\theta}^{\text{fast}}(\bar{\tau}) &= k_{\theta}(\bar{\tau}_{cr}, \omega_{cr}), \\ k_{\varepsilon}^{\text{fast}}(\bar{\tau}) &= k_{\varepsilon}(\bar{\tau}_{cr}, \omega_{cr}), \end{aligned} \quad (5.31)$$

which can be solved for $\bar{\tau}_{cr}$, ω_{cr} given $\bar{\tau}$, as shown in Figure 5.9(a) for different curvatures κ_* . We define a robustness coefficient $\bar{\tau}_{cr}/\bar{\tau}$ to represent how much increase in the dimensionless delay the system can withstand before it loses stability. The robustness coefficients shown in panel (b) correspond to the slope of the curves in panel (a). For circular paths ($\kappa_* > 0$), the coefficient first increases and then decreases with $\bar{\tau}$. These changes occur faster for larger values of curvature. For $\kappa_* = 0.2 \text{ m}^{-1}$ and $\bar{\tau} = 0.5$, the coefficient is around 2.73, yielding $\bar{\tau}_{cr} = 1.38$. For straight paths ($\kappa_* = 0$), equation (5.31) can be solved analytically, giving the universal constant in (5.13). This is reflected as the straight black lines in Figure 5.9.

5.2.2 Effect of time-varying latency

In this section, we study the stability and convergence of system (5.23) under periodic sawtooth latency profiles analytically. We also perform numerical simulations to demonstrate the effect of latency on the ToD performance and extend the numerical study to stochastic time-varying latency. For the periodic latency, a corresponding discrete-time system is derived by discretizing time and the stability is studied via the state transition matrix of the augmented state over one period.

First, we consider the latency profile where the delay increases from $\tau_{\min}$ to $\tau_{\max}$ with gradient 1 before dropping back to $\tau_{\min}$, see Figure 5.10(a),(b). The period of these oscillations is $T = \tau_{\max} - \tau_{\min}$ while the average delay is $\tau_{\text{ave}} = (\tau_{\max} + \tau_{\min})/2$. Rescaling the time by v/l , the dimensionless delay increases with slope 1 from $\bar{\tau}_{\min}$ to $\bar{\tau}_{\max}$ within each period of $\bar{T} = \bar{\tau}_{\max} - \bar{\tau}_{\min}$. This variation is a result of the zero-order hold applied on the input, which is equivalent of making $\mathbf{x}(\bar{t} - \bar{\tau})$ constant over $\bar{\tau} \in [\bar{\tau}_{\min}, \bar{\tau}_{\max})$.

We discretize time while choosing the time step $\Delta\bar{t}$ such that $\bar{\tau}_{\min} = n\Delta\bar{t}$, $\bar{\tau}_{\max} = m\Delta\bar{t}$ where $n, m \in \mathbb{Z}^+$. Due to the zero-order hold, for $\bar{t} \in [k\Delta\bar{t}, (k+1)\Delta\bar{t})$, we have $\mathbf{x}(\bar{t} - \bar{\tau}) \equiv \mathbf{x}((k-i)\Delta\bar{t}) = \mathbf{x}_{k-i}$, where $\bar{\tau}(\bar{t}) = \bar{t} + (i-k)\Delta\bar{t}$ and $i = n + (k \bmod (m-n))$. Using semi-discretization [51], we can obtain the discrete-time system

$$\mathbf{x}_{k+1} = \mathbf{P}\mathbf{x}_k + \mathbf{R}\mathbf{x}_{k-i}, \quad (5.32)$$

where

$$\mathbf{P} = e^{\mathbf{a}\Delta\bar{t}}, \quad \mathbf{R} = \int_0^{\Delta\bar{t}} e^{\mathbf{a}(\Delta\bar{t}-s)} ds \mathbf{b} = (\mathbf{P} - \mathbf{I})\mathbf{a}^{-1}\mathbf{b}, \quad (5.33)$$

with $\mathbf{a}$ and $\mathbf{b}$ from (5.23). The last equality only holds when $\mathbf{a}$ is invertible, which is the case when $\kappa_* > 0$. For $\kappa_* = 0$ (straight path) $\mathbf{a}$ is not invertible and we have

$$\mathbf{P} = \begin{bmatrix} 1 & \Delta\bar{t} \\ 0 & 1 \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} -k_\theta\Delta\bar{t}/2 & -k_\varepsilon\Delta\bar{t}/2 \\ -k_\theta & -k_\varepsilon \end{bmatrix} \Delta\bar{t}. \quad (5.34)$$

From the system (5.32), we construct an augmented system

$$\mathbf{X}_{k+1} = \mathbf{A}_i\mathbf{X}_k, \quad (5.35)$$

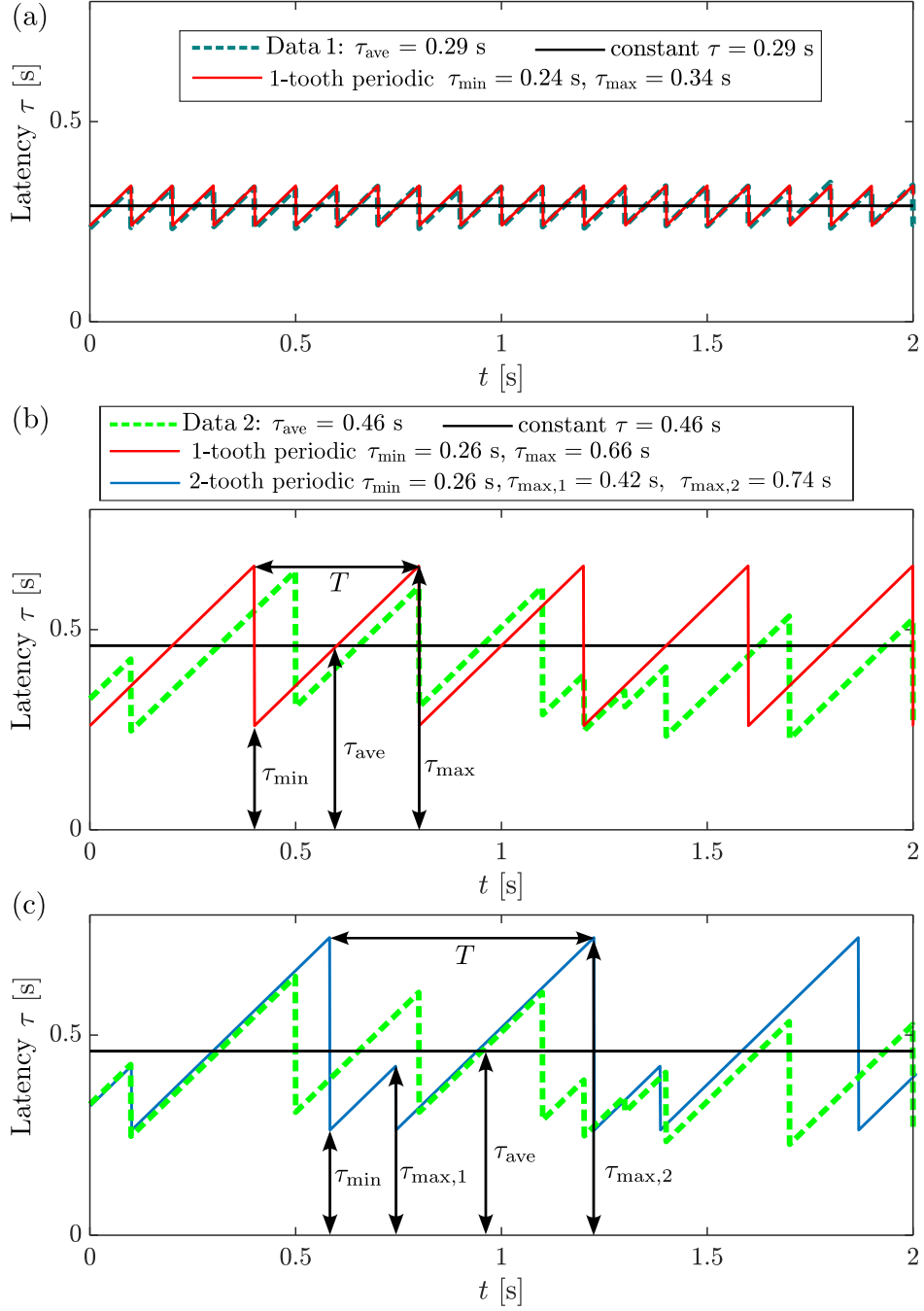


Figure 5.10: Two examples of stochastic time-varying latency under different communication network conditions. Data 1 and Data 2 are obtained from communication network simulations with different packet loss rates in [54]. These are compared to their constant latency approximations (black lines) and periodic latency approximations (red and blue curves).

where

$$\mathbf{X}_k = \begin{bmatrix} \mathbf{x}_k \\ \mathbf{x}_{k-1} \\ \vdots \\ \mathbf{x}_{k-i} \\ \vdots \\ \mathbf{x}_{k-m+1} \end{bmatrix} \in \mathbb{R}^{2m}, \quad \mathbf{A}_i = \begin{bmatrix} \mathbf{P} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{R} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{I} & \mathbf{0} & & & \cdots & & & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & & & \cdots & & & \mathbf{0} \\ \vdots & \vdots & & & \ddots & & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & & & \ddots & & \mathbf{I} & \mathbf{0} \end{bmatrix}, \quad (5.36)$$

and the matrix $\mathbf{R}$ is located in the $(i+1)$ -st block column of $\mathbf{A}_i$. This system describes the discrete time dynamics between $\bar{t} = k\Delta\bar{t}$ and $\bar{t} = (k+1)\Delta\bar{t}$, while the index i changes periodically with k . The computation cost increases with the decrease of discretization time step $\Delta\bar{t}$. We choose $\Delta\bar{t}$ such that the greatest common divisor of m and n is 1 to achieve the minimum computation cost while maintaining the accuracy of the stability boundary calculation.

For stability we look at the discrete-time system over one period $\bar{T} = (m-n)\Delta\bar{t}$:

$$\mathbf{X}_{k+m-n} = \mathbf{A}\mathbf{X}_k, \quad \mathbf{A} = \prod_{i=n}^{m-1} \mathbf{A}_i, \quad (5.37)$$

where k is the starting index of each period. This system is stable if and only if all the characteristic multipliers (i.e., the solutions of $\det(\mu\mathbf{I} - \mathbf{A}) = 0$) are within the unit circle, that is, $|\mu| < 1$. To determine the fastest convergence with periodic sawtooth latency profile, we minimize the absolute value of the outermost multiplier.

The same methodology can be applied for other periodic sawtooth type delay profiles. For instance, we consider the case where two teeth of different sizes are contained with a period, as shown in Figure 5.10(c) and Figure 5.13(a). Assume that the delay first increases from a minimum value $\tau_{\min}$ to $\tau_{\max,1}$ and then again increases from $\tau_{\min}$ to another value $\tau_{\max,2}$. The period of this profile is $T = \tau_{\max,1} + \tau_{\max,2} - 2\tau_{\min}$. When $\tau_{\max,1} = \tau_{\max,2} = \tau_{\max}$ one obtains two periods of the one-tooth periodic delay profile. The average delay can be calculated as

$$\tau_{\text{ave}} = \frac{(\tau_{\max,1} + \tau_{\min})(\tau_{\max,1} - \tau_{\min}) + (\tau_{\max,2} + \tau_{\min})(\tau_{\max,2} - \tau_{\min})}{2T}. \quad (5.38)$$

After rescaling time with v/l , we discretize it with time step $\Delta\bar{t}$ such that $\bar{\tau}_{\min} = n\Delta\bar{t}$, $\bar{\tau}_{\max,1} = m_1\Delta\bar{t}$ and $\bar{\tau}_{\max,2} = m_2\Delta\bar{t}$ where $n, m_1, m_2 \in \mathbb{Z}^+$. Then we have $\bar{T} = (m_1 + m_2 - 2n)\Delta\bar{t}$. Again, we can construct an augmented system similar

to (5.35), (5.36), where $m = \max\{m_1, m_2\}$. Then the discrete-time system over one period $\bar{T}$ gives

$$\mathbf{X}_{k+m_1+m_2-2n} = \mathbf{A}\mathbf{X}_k, \quad \mathbf{A} = \prod_{i=n}^{m_1-1} \mathbf{A}_i \prod_{j=n}^{m_2-1} \mathbf{A}_j. \quad (5.39)$$

Stability again can be determined by examining whether the outermost characteristic multipliers (i.e., the largest solutions of $\det(\mu\mathbf{I} - \mathbf{A}) = 0$) are within the unit circle.

Comparison of performance under constant and periodic latencies

To compare the stability under constant latency (which was analyzed in continuous time) to the stability under periodic latency (which was analyzed in discrete time), we need to convert the characteristic roots of the continuous-time system into the characteristic multipliers used in the discrete-time system. Namely, $\mu = e^{\lambda\bar{T}}$ which yields $|\mu| = e^{\rho\bar{T}}$ where $\bar{T}$ is the period. That is, the fastest-convergence rate ρ_{fast} in the continuous time corresponds to

$$|\mu_{\text{c,fast}}| = \left| e^{\rho_{\text{fast}}\bar{T}} \right|. \quad (5.40)$$

Below we show that if the fastest convergence control gains are chosen based on the constant latency analysis, the performance drops under the periodic latency.

The sawtooth latency profile in Figure 5.11(a) has a minimum value $\tau_{\min} = 0.25$ s and the maximum value $\tau_{\max} = 0.54$ s. The corresponding stability boundaries are shown in panel (b) together with the stability boundaries (5.27), (5.28) of the continuous time system with constant average latency $\tau_{\text{ave}} = 0.395$ s. The fastest convergence gains k_{θ}^{fast} , $k_{\varepsilon}^{\text{fast}}$ under the constant latency are marked by the black star. The characteristic multipliers of these gains are plotted in panel (c) under constant latency (black) and periodic latency (red). The periodic latency results in $(2m - 3)$ characteristic multipliers which are either zero or very close to zero where $2m$ is the dimension of the augmented state in (5.36). These $(2m - 3)$ multipliers correspond to characteristic roots with large negative real parts, which are indicated as black stars around the origin. One of the three leading multipliers is located at the real axis and the other two form a complex conjugate pair. The real multiplier has the largest magnitude and this is denoted by μ_{p} . There are also three leading multipliers for the constant latency in the discrete-time representation corresponding to the three leading characteristic roots in Figure 5.8(b). These coincide at the real axis and are denoted by $\mu_{\text{c,fast}}$. As shown by the zoom in panel (c), $\mu_{\text{c,fast}}$ is slightly closer to the origin than μ_{p} , i.e., $|\mu_{\text{p}}| - |\mu_{\text{c,fast}}| = 0.0013$.

The sawtooth latency profile in Figure 5.11(d) also has minimum value $\tau_{\min} = 0.25$ s but now the maximum value is $\tau_{\max} = 1.4$ s, yielding the average latency $\tau_{\text{ave}} = 0.825$ s. The

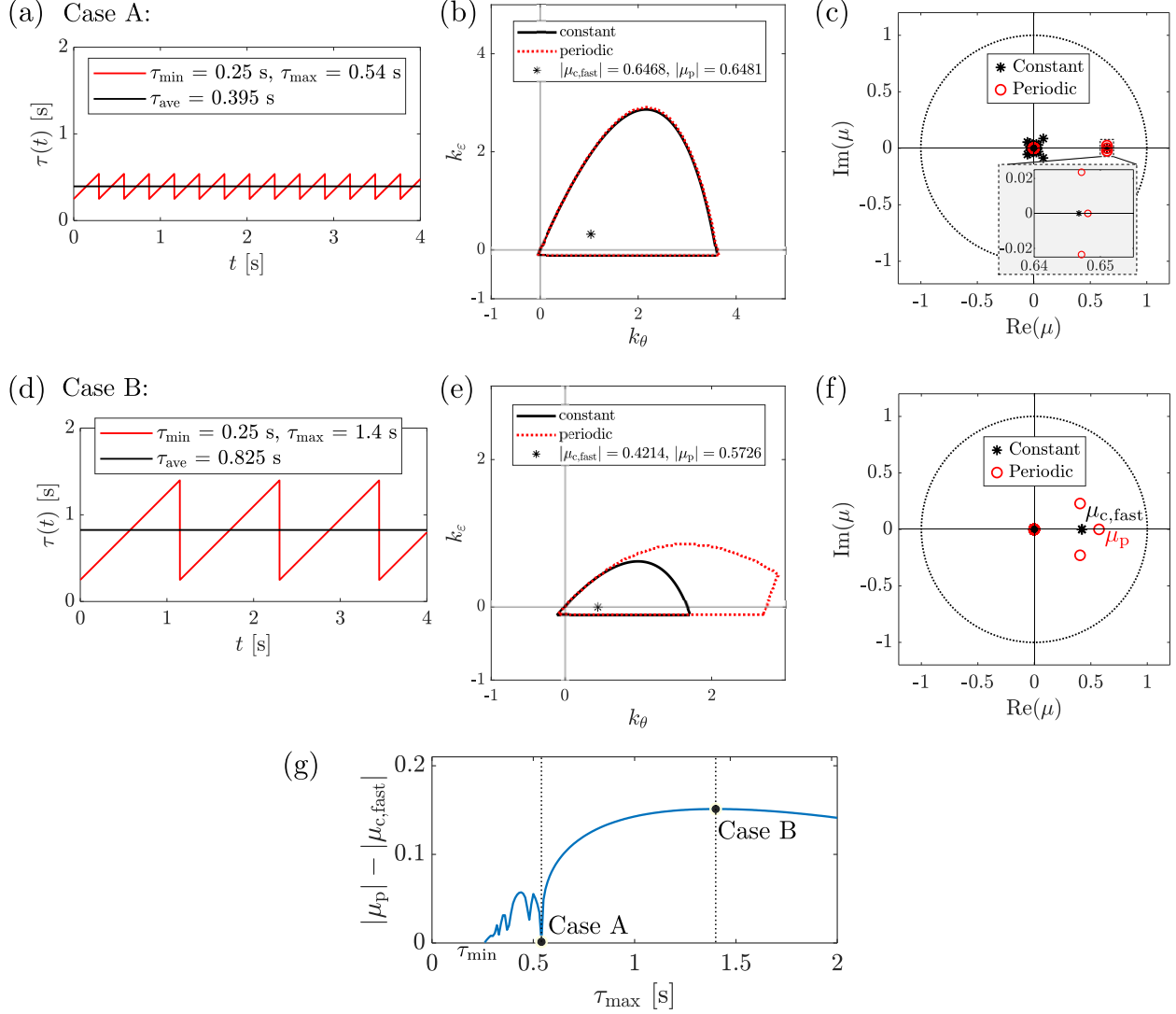


Figure 5.11: Comparing the stability boundaries and the characteristic multipliers under constant and periodic sawtooth latency profiles, when the gains are chosen based on the fastest convergence under the constant average of the periodic sawtooth latency for $v = 3$ m/s, $\kappa_* = 0.125\text{m}^{-1}$. Case A (panels (a)-(c)) and Case B (panels (d)-(f)) differ in the value $\tau_{\max}$. (g) Difference in the magnitude of the largest characteristic multiplier when considering constant vs. periodic latency.

corresponding stability boundaries are shown in panel (e) and these indeed show significant deviations between the constant delay and the periodic delay cases. The multipliers corresponding to the fastest convergence gains (calculated for constant delay) are plotted in panel (f), where the difference between the leading multipliers becomes $|\mu_p| - |\mu_{c,\text{fast}}| = 0.1512$. The positive value indicates that the periodic latency model converges to the steady state slower than the constant latency model. To further illustrate this performance degradation,

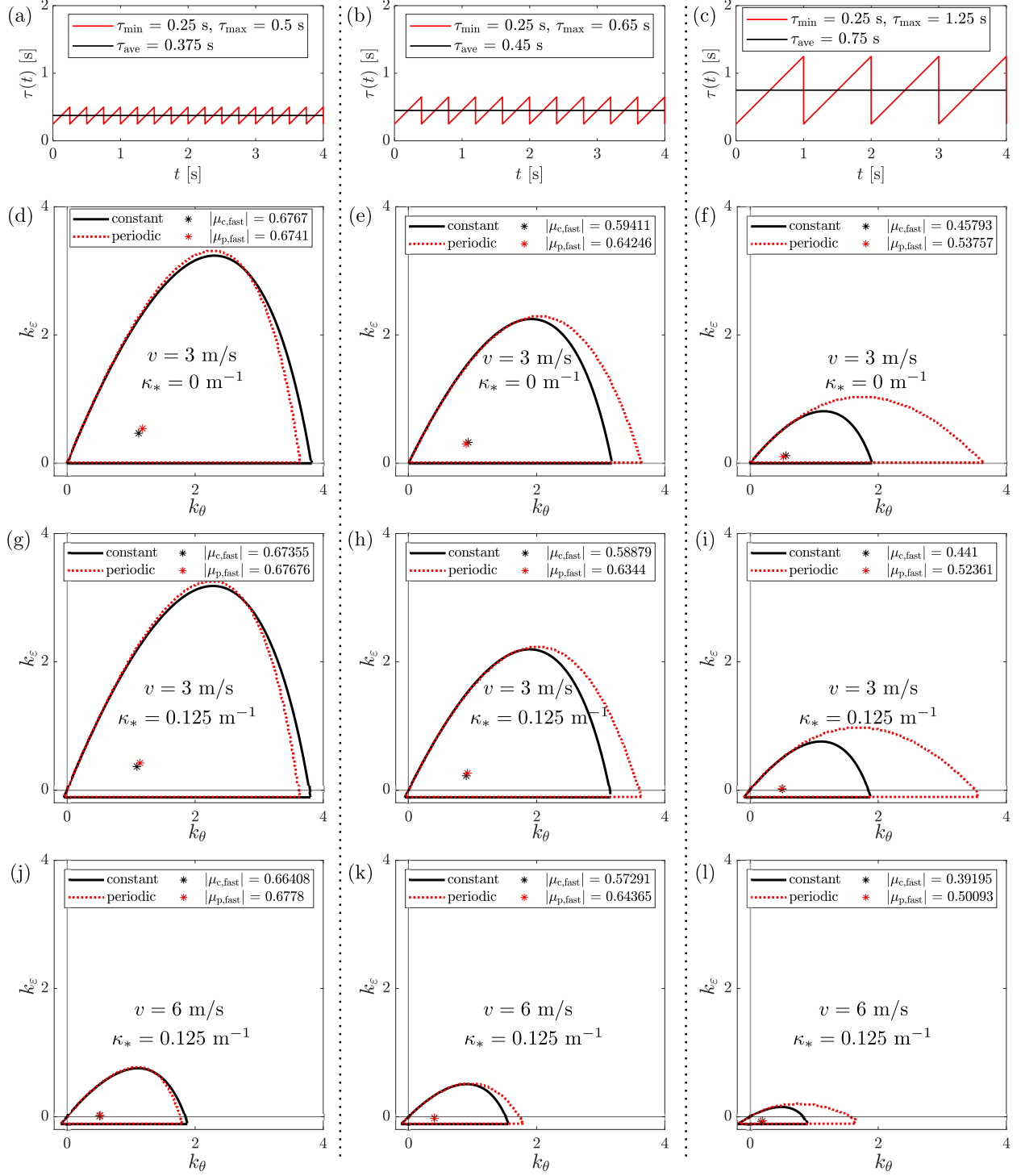


Figure 5.12: (a)-(c) Three different sawtooth latency profiles with same $\tau_{\min}$ but different $\tau_{\max}$. (d)-(l) Comparing stability charts for periodic sawtooth latency profiles and the corresponding average constant latency for different curvatures and longitudinal velocities when $l = 2.73$ m.

in Figure 5.11(g) we plot the difference $|\mu_p| - |\mu_{c,fast}|$ as a function of the maximum latency τ_{max} between 0.26 to 2 s, while keeping the minimum latency at $\tau_{min} = 0.25$ s. This becomes more pronounced when the difference between τ_{min} and τ_{max} is large. The periodic latency profiles of Case A and Case B are marked on the curve.

To further illustrate the behavioral differences between the constant and periodic latencies, we plot the stability boundaries for different values of τ_{max} in Figure 5.12. As τ_{min} is kept the same, τ_{ave} increases with τ_{max} . The charts in each column correspond to the latency profiles in the first row. As the average latency increases (from left to right), the stable regions shrink. The stability boundaries of the constant latency and the periodic latency remain close unless τ_{max} is significantly larger than τ_{min} . Moreover, we demonstrate the impacts of curvature and longitudinal velocity while increasing the curvature from 0 to 0.125 m^{-1} (from the second to the third row), and increasing the velocity from 3 m/s to 6 m/s (from the third to the fourth row). The increase of curvature shifts the stable region down while the increase of velocity reduces the size of the stable region. We can observe that the black stability boundaries in the panels (i) and (j) are identical as these correspond to the same dimensionless delay $\bar{\tau}_{ave}$.

For the case of periodic latency in discrete time, its fastest convergence gains correspond to the smallest leading multipliers denoted by $|\mu_{p,fast}|$. These can be obtained numerically and are marked by red stars in Figure 5.12(d)-(l). These are close to the fastest convergence gains obtained under the constant delay approximation, marked by the black stars. Still, the convergence at the black star (under constant latency) is always faster than at the red star (under periodic latency). That is, although having periodic latency instead of constant latency may increase the size of stable region and enable the use of larger control gains, the best performance becomes worse since the maximum latency increases.

Different sawtooth latency profiles have different stability boundaries, even when their average latency is the same. The three profiles in Figure 5.13(a) have the same average $\tau_{ave} = 0.5$ s, and the two periodic profiles share the same minimum latency $\tau_{min} = 0.25$ s. The one-tooth latency profile has the maximum $\tau_{max} = 0.75$ s (yielding the period 0.5 s), while the two-tooth latency has the maxima $\tau_{max,1} = 0.45$ s and $\tau_{max,2} = 0.85$ s (yielding the period 0.8 s). The corresponding stability boundaries are depicted in panel (b). As in the two-tooth latency profile, the maximum latency is greater than in the one-tooth latency profile, the stability boundaries deviate more from those obtained for constant delays.

To further interpret the stability chart, we select two sets of control gains from different regions of the chart and perform numerical simulations for the nonlinear system (5.18). The vehicle starts from a position of $(0, -0.3)$ m (i.e., small lateral deviation) and having 0 yaw angle (i.e., aligning with the path). The trajectories of the simulations are shown in panels

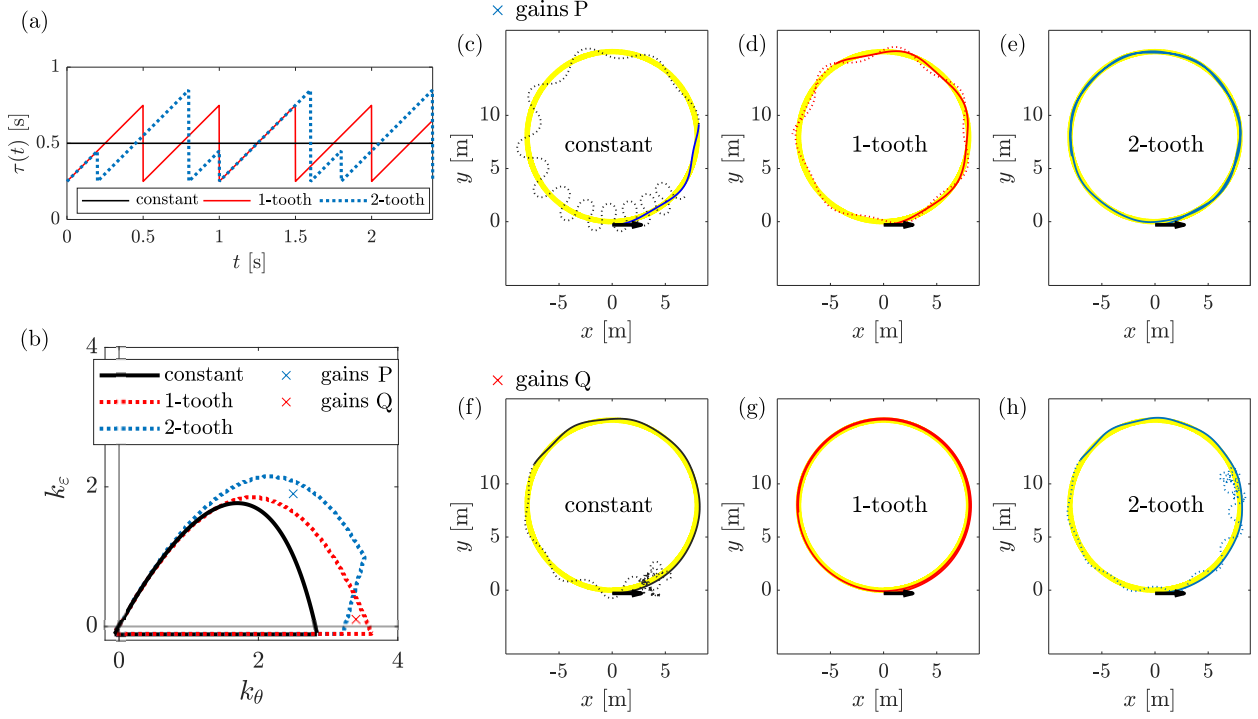


Figure 5.13: Stability and simulations under latency profiles for $v = 3$ m/s, $\kappa_* = 0.125$ m⁻¹. (a) Three delay profiles share the same average latency $\tau_{\text{ave}} = 0.5$ s. (b) Stability charts for three latency profiles. Two gain combinations marked as P and Q are selected for simulations. (c)-(e) Simulations with gains P under different latency profiles. (f)-(h) Simulations with gains Q under different latency profiles. Dotted sections of the trajectories indicate that the steering angle exceeds 45 degrees.

(c)-(e) for gain combination $k_\theta = 2.5, k_\epsilon = 1.9$ (marked by P in panel (b)) and in panels (f)-(h) for gain combination $k_\theta = 3.4, k_\epsilon = 0.1$ (marked by Q in panel (b)). Note that in the system (5.18), there is no additional saturation on the steering angle (except the arctan which saturates at ± 90 degrees). For this reason, the unstable trajectories may reach unrealistically large steering angles. We use dotted line for the part of the trajectory after the steering angle exceeds the limit of 45 degree. Since point P is within the stable region of two-tooth latency profile but outside the other two stable regions, the trajectory only converges to the path in the former case. For point Q, which is located within the stable region of one-tooth latency profile but outside the other two stable regions, the trajectory only converges under the one-tooth scenario. This demonstrates that with periodic latency, a larger stability region can be possibly achieved, and that the shape of the periodic latency plays an important role when $\tau_{\min} \ll \tau_{\max}$.

Simulations of teleoperated driving for a parking lot maneuver

We perform numerical simulations of ToD maneuvers to demonstrate how the latency and speed impact the performance. We relate these simulations to the stability analysis.

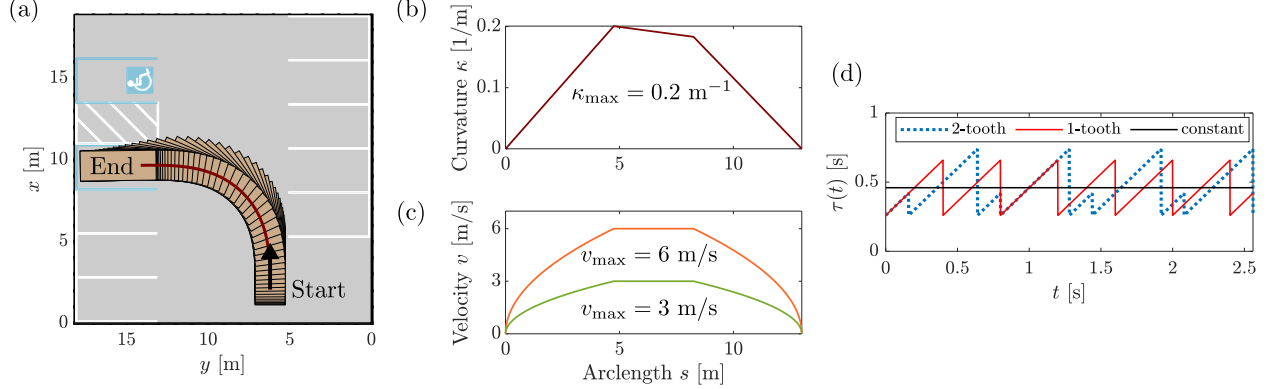


Figure 5.14: Path reference used for simulations. (a) Reference path (brown curve) of the rear axle center point with the vehicle depicted as moving along the path (b) Curvature plan for a left turn path. (c) Two velocity plans defined along the path. (d) Three types of latency profiles with $\tau_{\text{ave}} = 0.46 \text{ s}$.

We consider the task of maneuvering a vehicle in a parking lot, following the desired path indicated by the brown curve in Figure 5.14(a). The map of the parking lot is generated from measurements of a real lot. The curvature along the path is shown in panel (b) with maximum value $\kappa_{\max} = 0.2 \text{ m}^{-1}$ which corresponds to a typical minimum turning radius for a conventional automobile. Velocity plans along the path are shown in panel (c) for two different maximum velocities $v_{\max} = 3 \text{ m/s}$ and $v_{\max} = 6 \text{ m/s}$. The simulations are conducted under three different latency profiles as shown in panel (d). These all possess the same average $\tau_{\text{ave}} = 0.46 \text{ s}$. For the two periodic latency profiles we set $\tau_{\min} = 0.26 \text{ s}$. For the one-tooth profile we have $\tau_{\max} = 0.66 \text{ s}$ (yielding the period 0.4 s) while for the two-tooth profile we have $\tau_{\max,1} = 0.42 \text{ s}$ and $\tau_{\max,2} = 0.74 \text{ s}$ (yielding the period 0.64 s). They are also shown in Figure 5.10(b) and (c) as examples to approximate the stochastic delay Data 2.

The control gains are selected as $k_{\epsilon} = 0.5$, $k_{\theta} = 1.5$ and these are marked in the stability charts in Figure 5.15(a), (d), (g) as a blue diamonds. When simulating the system (5.18) we also include the same delay for realizing the desired velocity. That is, we use $v(t) = v_d(t - \tau)$, where $v_d(t) = v(s(t))$ is given by the velocity plan along the path. Without any latency the vehicle can follow the path and the velocity plans perfectly.

Considering a constant non-zero latency, the vehicle starts to deviate from the path, as shown in Figure 5.15(b) and (c). Since having larger velocity corresponds to a larger dimensionless delay and a smaller stable region in panel (a), the control gains, which are

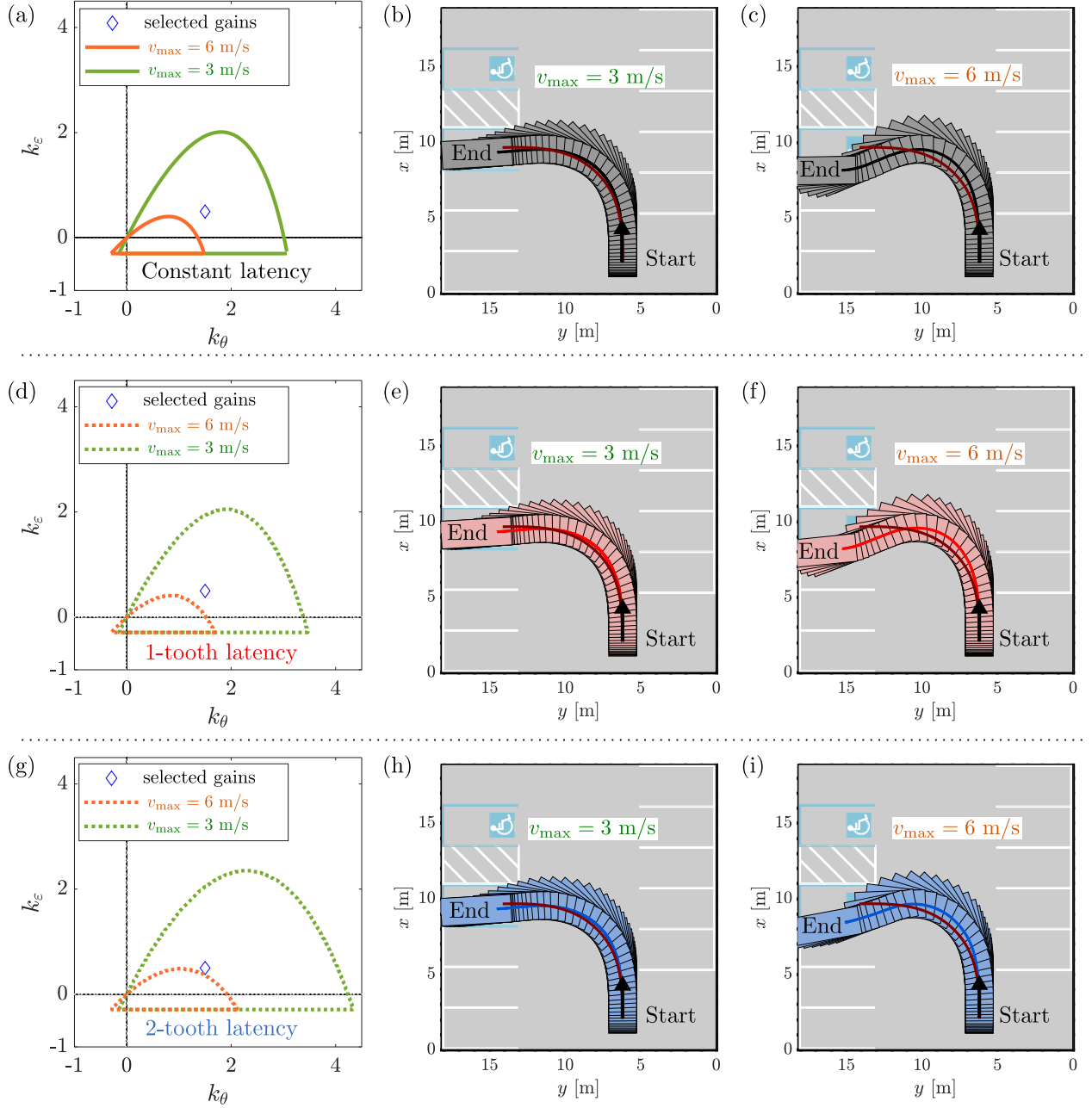


Figure 5.15: Stability charts and simulations under different latency profiles and velocity profiles. The stability charts correspond to constant curvature $\kappa_* = \kappa_{\max} = 0.2 \text{ m}^{-1}$ and a constant velocity $v_* = v_{\max} = 3 \text{ m/s}$ or $v_* = v_{\max} = 6 \text{ m/s}$ as indicated. (a)-(c) Stability charts and simulations under constant latency $\tau_{\text{ave}} = 0.46 \text{ s}$. (d)-(f) Stability charts and simulations under one-tooth latency profile. (g)-(i) Stability charts and simulations under two-tooth latency.

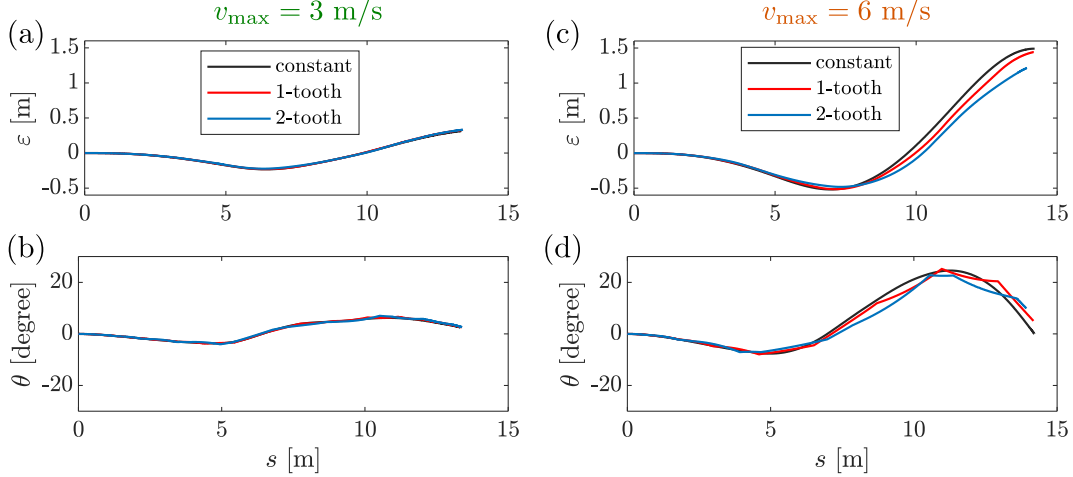


Figure 5.16: The lateral and yaw errors along the path under different latency profiles.

inside the stable region for smaller maximum velocity, are located outside the stable region for larger maximum velocity. Correspondingly, the vehicle is not able to follow the desired path in panel (c) even though the latency is the same as in panel (b). When ToD is performed under one-tooth periodic latency and two-tooth periodic latency, the corresponding stability charts and simulations are depicted in panels (d)-(f) and panels (g)-(i), respectively. The vehicle exhibits similar behaviors under the three different latency profiles when trying to follow the same path and the same velocity plan.

We show the lateral error and the yaw error between the realized trajectory and the closest point on the desired path under different delay profiles in Figure 5.16. The lateral and yaw errors along the path for the sawtooth latency simulations are close to those under constant latency. Panels (a) and (b) indicate that the effect of sawtooth variation in latency on the trajectory is negligible for smaller maximum velocity. For larger maximum velocity, the deviations between the trajectories obtained for different latency profiles become more significant as demonstrated in panels (c) and (d). As the constant latency results in the smallest stability region, see Figure 5.15(a), (d) and (g), it also has the largest lateral error along the path among three cases, see Figure 5.16(c). This shows that using the constant average latency approximation for the actual time-varying latency may expect a worse performance, depending on the choice of the control gains. Lastly, observe that the yaw error is not smooth along the path in panels (b) and (d). This is due to the discontinuities in the sawtooth latency which directly affects the yaw angle.

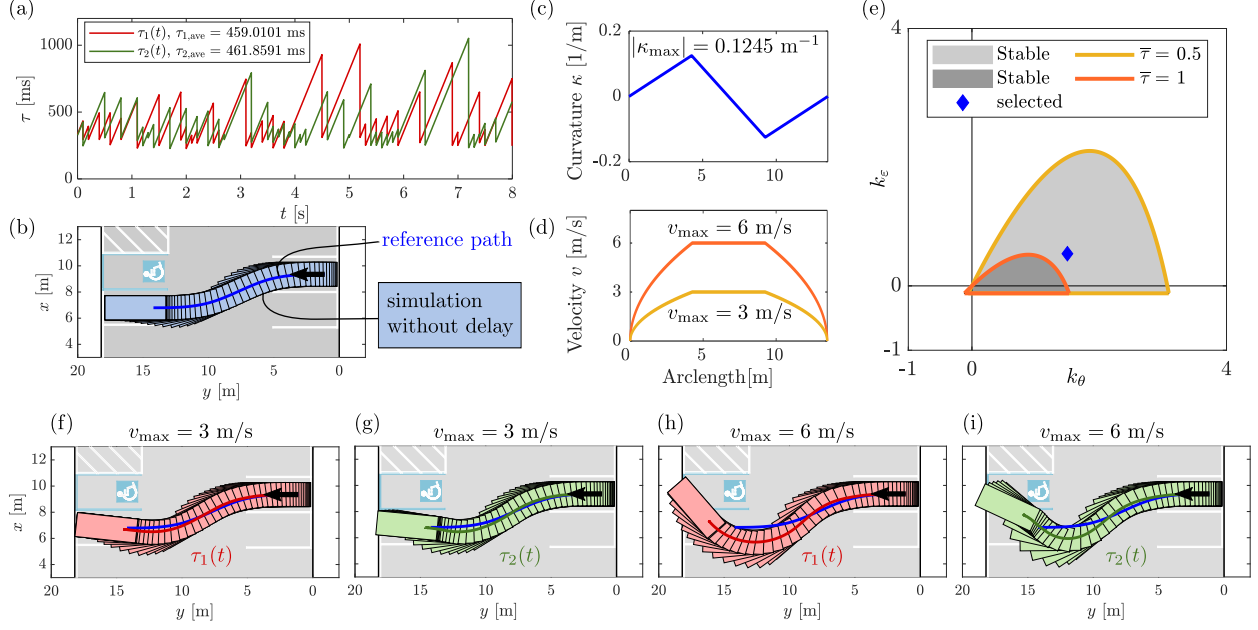


Figure 5.17: Simulation results for different latency profiles and velocity plans. (a) Two time-varying latency profiles both have $\tau_{\text{ave}} = 0.46$ s. (b) Reference path (blue solid curve) and the simulation of the ToD without any latency. (c) The curvature function along the reference path with $|\kappa_{\text{max}}| = 0.1245 \text{ m}^{-1}$. (d) Two velocity plans along the given path with $v_{\text{max}} = 3 \text{ m/s}$ and $v_{\text{max}} = 6 \text{ m/s}$. (e) Stability chart for $\kappa_* = 0.1245 \text{ m}^{-1}$, $\tau = 0.46 \text{ s}$, $v_* = 3 \text{ m/s} \Rightarrow \bar{\tau} = 0.5$, and $\tau = 0.46 \text{ s}$, $v_* = 6 \text{ m/s} \Rightarrow \bar{\tau} = 1$. (f)-(i) Simulation results for the different velocity plans, with latency profiles $\tau_1(t)$ (red) and $\tau_2(t)$ (green) respectively.

Effect of stochastic time-varying latency

Here we demonstrate the effect of stochastic time-varying latency via simulation results of ToD for a different scenario in the same parking lot. The vehicle's goal is to go from one parking spot to another following a given curved path, see Figure 5.17. This case can be generalized to many low-speed maneuvers with predefined curvatures and velocity plans along the desired paths. The two stochastic latency profiles shown in panel (a) are generated for the same network configuration with average overall latency $\tau_{\text{ave}} = 0.46$ s. This case is chosen since the effects of the stochastic latency are more pronounced due to the high packet drop ratio. The vehicle starts from standstill at a spot on the right and terminates its motion at the spot on the left following the blue path in panel (b), with two different velocity plans. The desired path is determined by a continuous curvature function shown in panel (c), using the 3-clothoid method [4, 90]. The maximum curvature of the path is $\kappa_{\text{max}} = 0.1245 \text{ m}^{-1}$ and the maximum velocities for the two velocity plans are $v_{\text{max}} = 3 \text{ m/s}$ and $v_{\text{max}} = 6 \text{ m/s}$, as depicted in panel (d). When there is no latency in the teleoperated control loop, the

vehicle can follow the path perfectly for both velocity profiles, see the light blue simulation in the panel (b).

The stability boundaries for constant curvature $\kappa_* = 0.1245 \text{ m}^{-1}$, constant scaled delays $\bar{\tau} = 0.5$ and $\bar{\tau} = 1$ are shown in panel (e). The two scaled delays correspond to the same latency $\tau = 0.46 \text{ s}$, wheelbase $l = 2.73 \text{ m}$, but different velocities $v_* = 3 \text{ m/s}$ and $v_* = 6 \text{ m/s}$. We choose the control gains $k_\theta = 1.5$, $k_\varepsilon = 0.5$ in the controller (5.17) as indicated by blue diamond, which is inside the stable region for $\bar{\tau} = 0.5$ but outside the stable region of $\bar{\tau} = 1$. We simulate the nonlinear closed-loop system (5.18) with time-varying $\tau(t)$, $v(t)$ and $\kappa(t)$. The simulation results for $v_{\max} = 3 \text{ m/s}$ are shown in panels (f) and (g), with the latency profiles $\tau_1(t)$ and $\tau_2(t)$, respectively. The difference between the vehicle trajectories is small in these two simulations. On the contrary, the simulations for the larger velocity $v_{\max} = 6 \text{ m/s}$ look very different in panels (h) and (i), which means that this maneuver is more sensitive to the stochasticity in the latency profiles. The linear stability analysis for the constant parameters (overall latency, longitudinal velocity and path curvature) still provides useful insights regarding the qualitative results of the nonlinear simulations with time-varying parameters.

5.3 Learning the Delayed Lateral Dynamics

In this section, we utilize the physics-based form (5.1)-(5.2) to construct a neural delay differential equation (NDDE) with a trainable delay. Then we train the NDDE to learn the dynamics of the vehicle and the remote operator, including the overall latency. The learned weights and biases carry physical meanings, such as the longitudinal velocity and the steering control gains.

Consider the following NDDE

$$\dot{x}(t) = \text{net}(x(t), x(t - \tau_{\text{est}})), \quad (5.41)$$

with $x = [x_R, y_R, \psi]^\top$ representing the position and yaw angle of the vehicle, and

$$\text{net} := \begin{bmatrix} w_2 \cos(x_3(t)) \\ w_2 \sin(x_3(t)) \\ \frac{w_2}{l} \tan\left(\text{sat}(W_1 x_{2:3}(t - \tau_{\text{est}}) + b_1)\right) \end{bmatrix}, \quad (5.42)$$

where the wheelbase l is assumed to be known and the rest of the trainable NDDE parameters are the weights $w_2 \in \mathbb{R}$, $W_1 = [w_{11}, w_{12}] \in \mathbb{R}^{1 \times 2}$, the bias $b_1 \in \mathbb{R}$, and the overall latency $\tau_{\text{est}} \in \mathbb{R}$. The saturation function $\text{sat}(\cdot) = \frac{1}{\pi} \arctan(\pi \cdot)$ limits the steering angle between

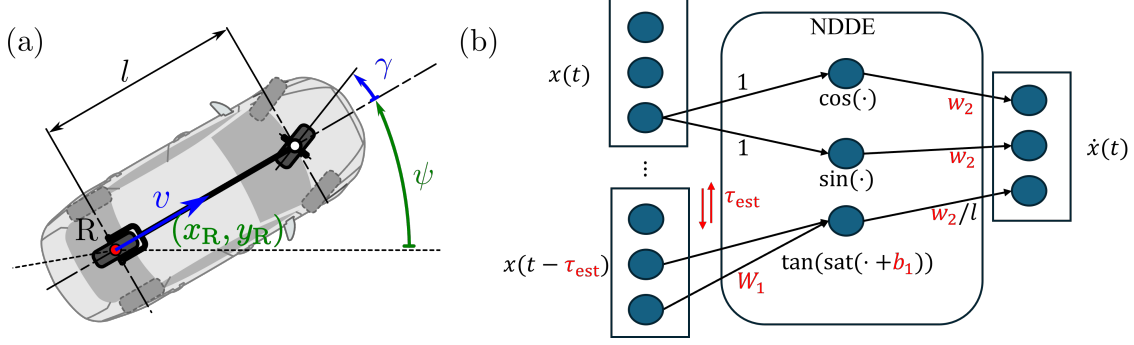


Figure 5.18: (a) Kinematic vehicle model. (b) Structure of the NDDE. The trainable parameters are indicated in red. As the trainable delay in the NDDE updates, the delayed input to the NDDE changes accordingly.

± 28.65 degrees, taken from [103]. This model is built based on the kinematic model (5.1)-(5.2), and thus, the NDDE parameters have physical meanings. For example, w_2 represents the longitudinal velocity, $-w_1$ represents the control gains in the steering controller and τ_{est} represents the overall latency in the loop. The bias b_1 represents the lateral offset of the straight path $y^* = -b_1/w_{11}$, which may not be zero when the remote operator is a human (who may follow the straight path with some constant lateral deviation). The structure of NDDE (5.42) is visualized in Figure 5.18. The input data is the state trajectory and the output is the time derivative of the state, which can be estimated by numerical differentiation.

We consider the derivative loss function

$$\mathcal{L} = \frac{1}{Nn} \sum_{j=1}^N \sum_{i=1}^n \left(\hat{x}_i(t_j) - \dot{x}_i(t_j) \right)^2, \quad (5.43)$$

where N is batchsize (the number of samples used for one update) and n is the dimension of the NDDE state $\mathbf{x}(t)$, in our case $n = 3$. In the training process, the predicted state derivative

$$\hat{\dot{x}}(t_j) = \text{net}(x(t_j), x(t_j - \tau_{\text{est}})), \quad (5.44)$$

is directly calculated from data.

We utilize experimental driving data where a human operator drives in a high-fidelity simulation environment. In particular, we establish a remote driving simulator using TELECARLA [46], in which we can introduce an additional delay as the communication latency. As shown in Figure 5.19, the experimental setup consists of three parts: the client computer, the Ethernet wire connection, and the server computer. The CARLA simulations are running on the server computer and the data is transmitted to the client computer via the Ethernet wire with a network emulation Web-GUI. The network emulation introduces

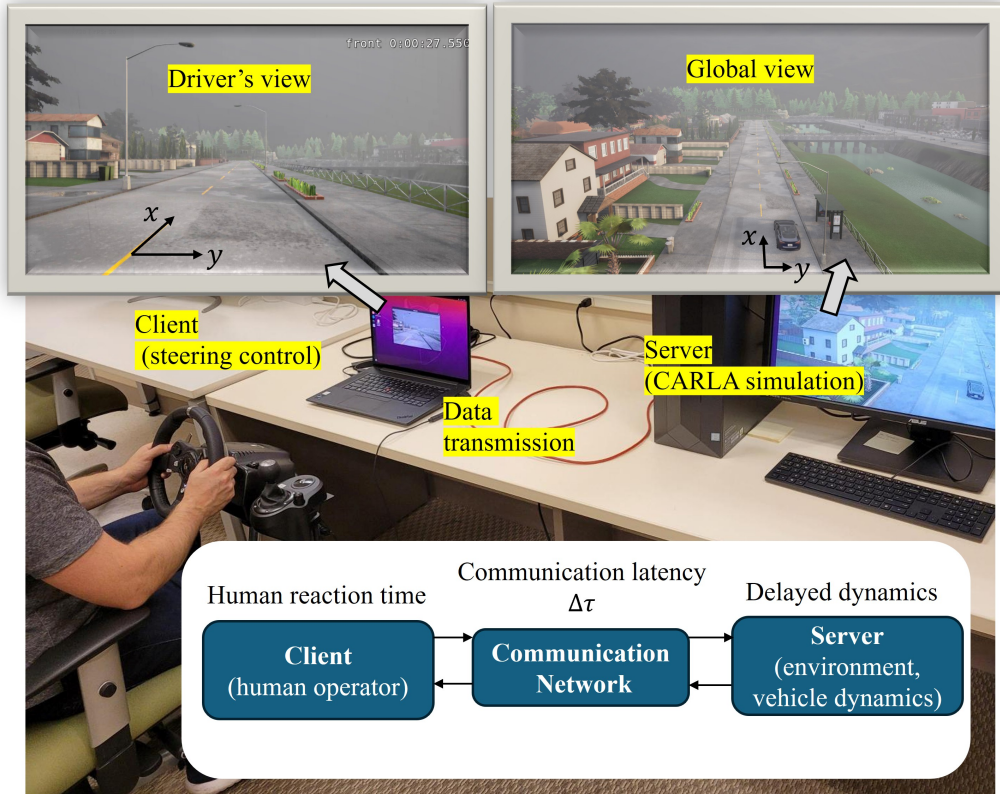


Figure 5.19: TELECARLA setup used for collecting experimental data.

a constant delay $\Delta\tau$ to represent the communication latency. The client computer shows a driver's view to the remote operator and the steering commands generated by the operator are recorded and sent back to the server for execution.

In the experiments, the human operator performs a lane change maneuver after the vehicle reaches an equilibrium speed. The speed is maintained at a constant value around 4 m/s by a controller during the lane-changing maneuver and the human operator is only controlling the steering wheel. Multiple experiments are conducted under different values of the communication latency $\Delta\tau = 0, 0.15, 0.3, 0.4$ s. Each set of experiments contains three runs. The trajectory data are presented in Figure 5.20 under different introduced latency $\Delta\tau$. It can be observed that when $\Delta\tau \geq 0.4$ s, the human operator starts losing control of the vehicle.

When training the neural networks, we do not combine the data from multiple runs since human behavior may differ even under the same driving conditions. Figure 5.21(a) and (c) depict runs under $\Delta\tau = 0$ and $\Delta\tau = 0.4$ s, respectively. The NDDE (5.42) is utilized to learn from individual trajectories, each containing 24-seconds of data with sampling time 0.05 second (480 data points), see the segment between 0 and the second vertical black line.

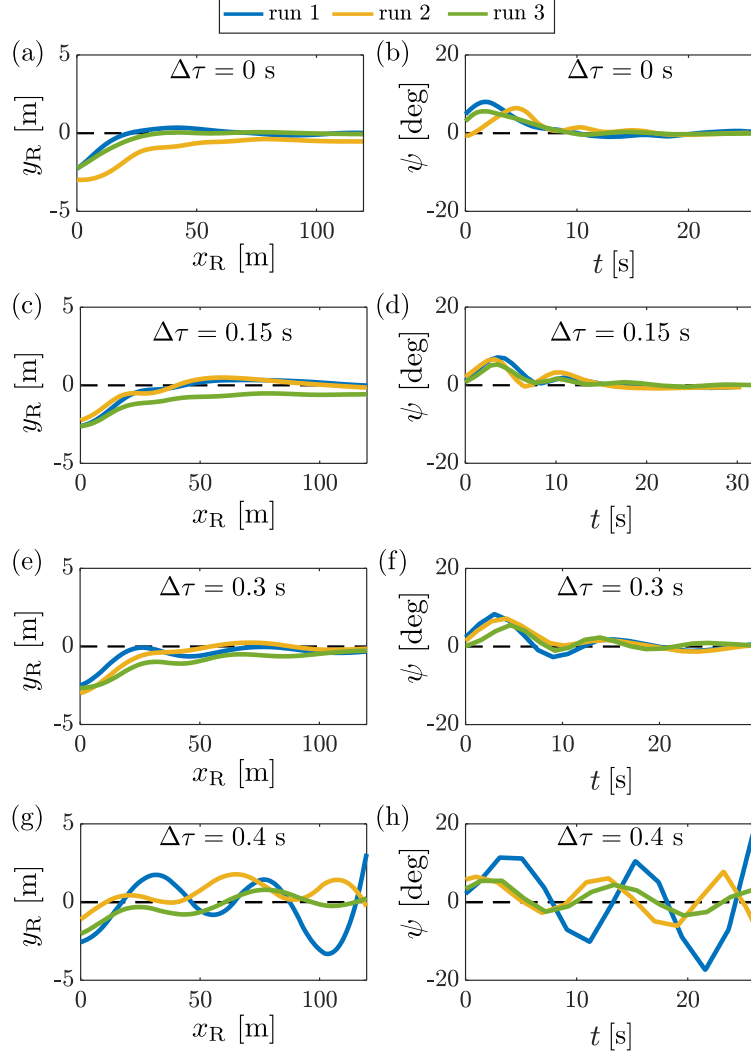


Figure 5.20: Experimental data of a lane-change maneuver with a human remote operator in the loop. Different network latencies $\Delta\tau$ were introduced in the experiments and three runs of data were collected under each condition.

The maximum allowed latency is set to be 3 seconds, so the first 3 seconds of training data (the segment between 0 and the first vertical black line) is not included in the loss (5.43).

After training, the NDDE is simulated to predict the trajectory from the given history, beyond the training data. This allows us to examine how this simplest nontrivial model captures the complex dynamics of the human-operator-in-the-loop TELECARLA experiments, under different latencies. Since the simulation error is not considered in the loss function during training, it is necessary to examine the simulations even on the training data. When the latency is small, as in Figure 5.21(a), the NDDE aligns well with the data. When the latency is larger and the trajectory is unstable, as in Figure 5.21(b), the NDDE still captures

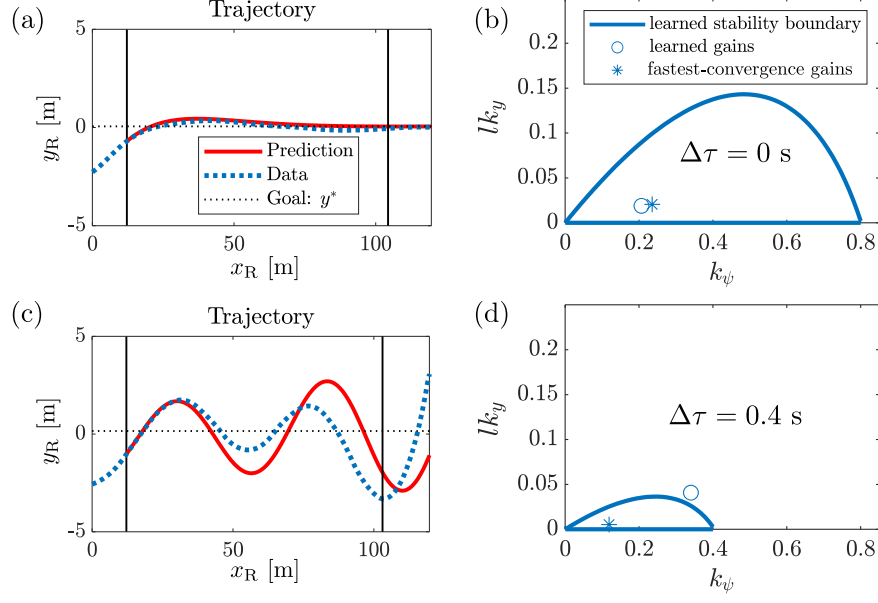


Figure 5.21: Simulations of trained NDDE are compared to the experimental data on the left and the stability charts of the learned remote human operator are shown on the right. (a)-(b) a run of data collected under $\Delta\tau = 0$ s. (c)-(d) a run of data collected under $\Delta\tau = 0.4$ s

the unstable behavior qualitatively, while the prediction accuracy decreases.

Stability and performance analysis can be done for the trained NDDE in the same way as for classical delay differential equations since the data-driven model is in the form (5.1)-(5.2). Figure 5.21(b) and (d) show the stability boundaries and the fastest-convergence gains (blue stars) under the learned scaled latency $\bar{\tau} = \tau_{\text{est}} w_2 / l$ where τ_{est} is the learned overall latency and w_2 is the learned speed. The equivalent control parameters k_y, k_ψ can also be extracted from the learned weight W_1 , which are indicated as blue circles. When the human gains are close to the fastest-convergence gains, the trajectory of the teleoperated vehicle converges to the straight line quickly without oscillations. When the learned human gains are located outside the stability boundary, the ToD trajectory is indeed oscillatory and unstable.

We plot the stability charts and the learned human gains of all runs in Figure 5.22. When the introduced latency is 0, the overall latency in the loop is small. Then the stable region is large and the human can perform close to the fastest-convergence gains. However, when the introduced latency is large, the stable area is small, and the human gains are located outside or near the stability boundaries. We remark that similar behavior is observed in balancing problems where human reaction delay also plays an important role [14].

We show the correlation between the learned latency and the communication latency for all experiments in Figure 5.23. The learned delay τ_{est} increases with the introduced communication latency $\Delta\tau$ and it varies under same $\Delta\tau$. This indicates that the human

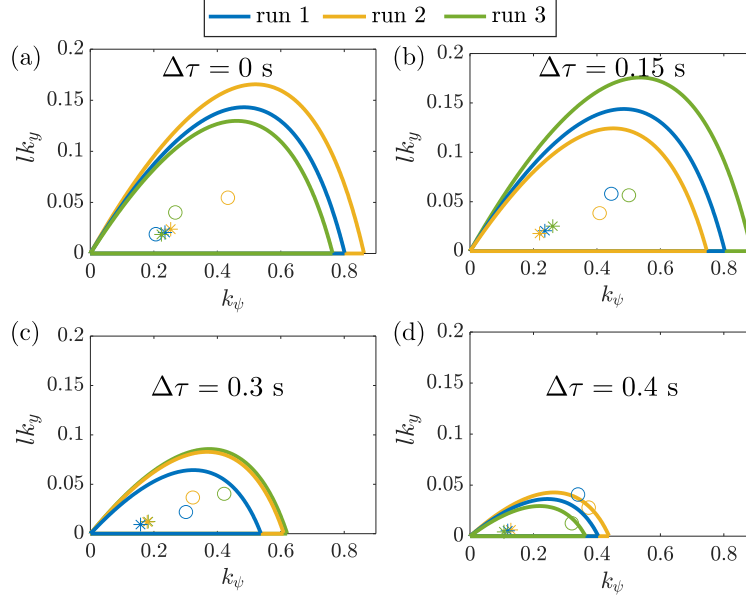


Figure 5.22: Stability charts extracted from the TELECARLA experiments.

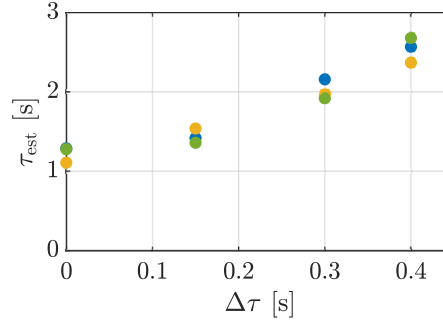


Figure 5.23: The overall latency in the loop estimated from individual trajectories for different values of the communication latency $\Delta\tau$.

reaction time may differ in individual experiments, under same network conditions.

5.4 Summary

In this chapter, we provided the detailed analysis of the stability of teleoperated driving systems under various types of latency profiles. By rescaling the system, we summarized the effect of the latency and longitudinal velocity into one variable, the dimensionless delay. We demonstrated the similarity between longitudinal and lateral dynamics and obtained the same results in the performance and robustness study, when the curvature is zero. Moreover, we showed that the curvature also plays an important role and we derived fastest convergence

rate (gains) analytically. In the case of periodic latency, we showed that the stability region may increase, but the best performance usually decreases compared to the constant latency. The difference between the constant and periodic latency is small unless the variation of the latency is large in the periodic latency profile. The stochastic latency profiles were examined via numerical simulations and it was shown that the stability analysis under their constant average values can still provide insights on the ToD performance.

We also proposed to use a neural delay differential equation (NDDE) to learn the dynamics of the vehicle and the remote operator, including the overall latency. The nonlinearity from a kinematic vehicle model was embedded in the NDDE structure. Experiments with human remote operators were conducted in a high-fidelity simulation environment with an added latency. The NDDE was able to learn an overall latency and equivalent control gains from one trajectory. It was shown that the human operator’s behavior can be captured by the NDDE and the learned parameters can be used to analyze the stability and the performance of the teleoperated system. Human operators may perform differently under the same network conditions, and as future direction, we will consider the uncertainty and time-varying characteristic in the NDDE parameters. More complex model and experimental data in the real world will be considered in the future work.

CHAPTER 6

Mitigating the Effect of Delay with Act-and-Wait

To address the negative effects of the overall delay in the ToD control loop, some researchers have been trying to reduce the delays [121, 11]. Others tried to work with the inevitable delay. For example, a predictive display system [109] was developed to compensate the communication latency in teleportation using model-based extended Kalman filter. The Smith predictor [89] can be used to achieve delay-free performance if the delay and the plant model are accurately known. Intermittent control [31, 75] reduces the infinite-dimensional time delay system to finite dimension, by utilizing predictive control intermittently. Other methods include optimal control [15] and H_∞ control [130, 80]. These methods result in standalone controllers and are not applicable on top of an existing automatic controller or on a human operator.

In this chapter, we will study the act-and-wait control strategy [50, 30, 84, 131], in the context of ToD. Compared to the mitigation methods mentioned above, act-and-wait algorithms can be directly implemented as an add-on to the existing controller. By switching the control on and off periodically, the original infinite-dimensional delay system can be reduced to a finite-dimensional system. As illustrated in Fig. 6.1(b), when the act-and-wait signal $G(t)$ is used on top of the remote operator the systems achieves superior control performance. For large enough delay, the original controller may lose stability (gray curve). With the act-and-wait algorithm applied on the same controller (same control gains), the system converges to the steady state much faster (red curve). Besides addressing the delays in the control loop, one also needs to evaluate the robustness of the controller against the changes in the delay. Methods for examining the stability and the performance of time delay systems are well established [51] and these can be generalized to study the robustness of stability against changes in system parameters, including the delay.

We first introduce act-and-wait algorithm and study the effect of the delay and velocity on ToD in terms of stability and convergence rate analytically with the act-and-wait algorithm.

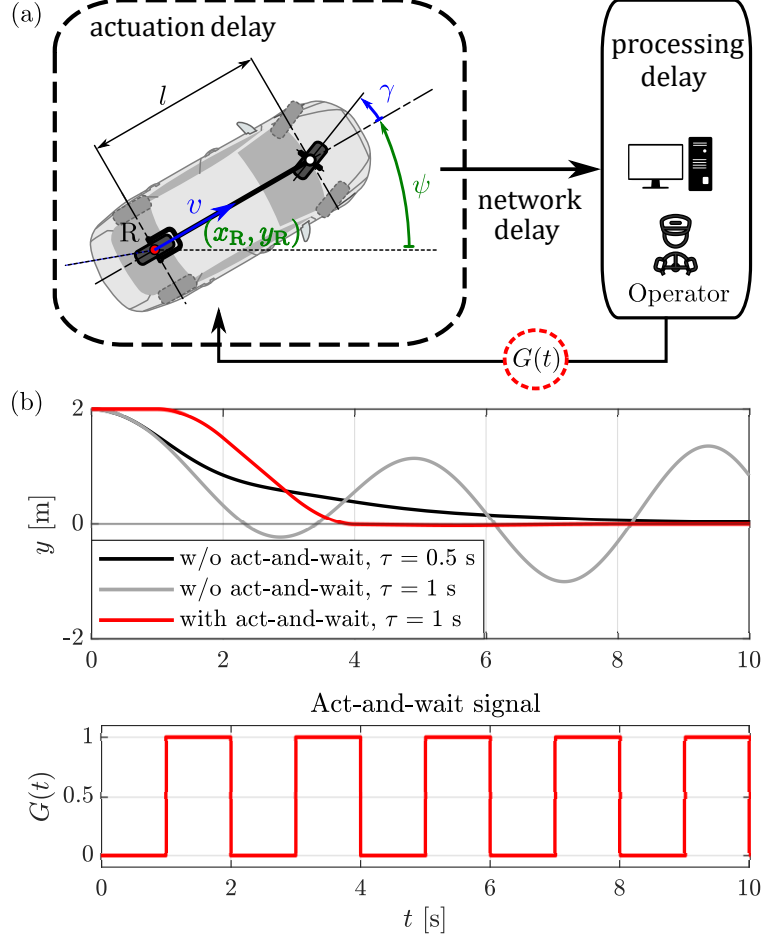


Figure 6.1: (a) The framework of ToD with delays. (b) Using act-and-wait strategy on top of existing remote operator to stabilize the system under large delay. Top: Simulations of a vehicle following a straight line $y = 0$ under different delays. The nominal control gains chosen for delay $\tau = 0.5$ s (black) lose stability when the delay increases to 1 s (grey). After applying the act-and-wait signal on top the nominal controller (without changing the control gains), the system becomes stable again. Bottom: The periodic act-and-wait signal used in the simulation.

Then we derive the robustness coefficient that represents how much delay the system may tolerate before losing stability. In the end, the advantages of using the act-and-wait algorithm on top of a nominal controller are demonstrated.

6.1 Act-and-wait Algorithm

Consider a generic linear system with feedback control based on the delayed state [50]:

$$\begin{aligned}\dot{X}(t) &= \mathbf{A}X(t) + \mathbf{B}U(t), \\ U(t) &= G(t)\mathbf{K}X(t - \tau),\end{aligned}\tag{6.1}$$

where X is the state, U is the control input, $\mathbf{K}$ contains the control gains, and $G(t)$ is the act-and-wait signal defined as

$$G(t) = \begin{cases} 0, & \text{if } 0 \leq (t \bmod T) < t_w, \\ 1, & \text{if } t_w \leq (t \bmod T) < T. \end{cases}\tag{6.2}$$

Here, $T = t_w + t_a$ is the period, t_w is the waiting time, and t_a is the acting time. The closed-loop dynamics are given by

$$\dot{X}(t) = \mathbf{A}X(t) + G(t)\mathbf{A}_\tau X(t - \tau),\tag{6.3}$$

where $\mathbf{A}_\tau = \mathbf{B}\mathbf{K}$.

With the assumptions $t_w > \tau$ and $0 < t_a < t_w$, one can convert the continuous-time system (6.3) into a discrete-time map linking the state at the beginning of the act-and-wait interval to the state at the end of that interval:

$$X(kT + T) = \Phi X(kT), \quad k = 0, 1, \dots,\tag{6.4}$$

where the monodromy matrix Φ is derived as

$$\Phi = e^{\mathbf{A}T} + \int_{t_w}^T e^{\mathbf{A}(T-s)} \mathbf{A}_\tau e^{\mathbf{A}(s-\tau)} ds.\tag{6.5}$$

The stability of the discrete-time linear system (6.4) is determined by the eigenvalues of the monodromy matrix Φ , also known as the characteristic multipliers.

6.2 Act-and-wait Steering Control

In the ToD application (5.1), (5.2), we apply the act-and-wait signal $G(t)$ to the argument of the “arctan” function in the controller. We further assume $t_w = \tau$ and use the act-wait ratio $a = t_a/t_w$, $0 < a \leq 1$, to tune the act-and-wait signal. Then the closed-loop dynamics (5.1)-

(5.2) become

$$\begin{aligned} \dot{x}_R(t) &= v \cos(\psi(t)), \\ \dot{y}_R(t) &= v \sin(\psi(t)), \\ \dot{\psi}(t) &= \frac{v}{l} G(t) (-k_y y_R(t - \tau) - k_\psi \psi(t - \tau)), \end{aligned} \quad (6.6)$$

After nondimensionalization and linearization, the stability of the lateral dynamics (given by the last two rows in (6.6)) are characterized by the monodromy matrix

$$\Phi = e^{\mathbf{A}\bar{T}} + \int_{\bar{t}_w}^{\bar{T}} e^{\mathbf{A}(\bar{T}-s)} \mathbf{A}_{\bar{\tau}} e^{\mathbf{A}(s-\bar{\tau})} ds, \quad (6.7)$$

where $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$, $\mathbf{A}_{\bar{\tau}} = \begin{bmatrix} 0 & 0 \\ -l k_y & -k_\psi \end{bmatrix}$, $\bar{T} = (1+a)\bar{\tau}$ and $\bar{t}_w = \bar{\tau}$. The analytical form of the monodromy matrix is

$$\Phi = \begin{bmatrix} 1 - \frac{l k_y a^2 \bar{\tau}^2}{2} & (a+1)\bar{\tau} - \frac{k_\psi a^2 \bar{\tau}^2}{2} - \frac{l k_y a^3 \bar{\tau}^3}{6} \\ -l k_y a \bar{\tau} & 1 - k_\psi a \bar{\tau} - \frac{l k_y a^2 \bar{\tau}^2}{2} \end{bmatrix}. \quad (6.8)$$

The eigenvalues μ_1, μ_2 of this matrix can be obtained analytically and the discrete-time system (6.4) with $X = [\tilde{y}, \tilde{\psi}]^\top$ is exponentially stable if and only if $|\mu_1| < 1$ and $|\mu_2| < 1$. As shown by [2], this is equivalent to satisfying inequalities $|\text{tr}(\Phi)| - 1 < \det(\Phi)$ and $\det(\Phi) < 1$, which result in the stability conditions

$$\begin{aligned} k_\psi &< \frac{1}{24} a^3 \bar{\tau}^3 (l k_y)^2 - \frac{1}{2} a \bar{\tau} l k_y + \frac{1}{2} \bar{\tau} l k_y + \frac{2}{a \bar{\tau}}, \\ k_\psi &> \frac{1}{12} a^3 \bar{\tau}^3 (l k_y)^2 + \bar{\tau} l k_y, \\ l k_y &> 0. \end{aligned} \quad (6.9)$$

The stable regions enclosed by these three boundaries are shown in Figure 6.2(a) and (b) for different values of the dimensionless delay $\bar{\tau}$ and different choices of the act-wait ratio a . Comparing these to the chart in Fig 5.3(a), one may observe that applying the act-and-wait strategy significantly increases the stable region. The stable region still shrinks as $\bar{\tau}$ increases, and it expands as a decreases. Therefore, adjusting the act-and-wait signal could help to stabilize the system under large delay.

Moreover, if $\mu_1 = \mu_2 = 0$, the solution converges to zero after two periods (since $\Phi^2 = \mathbf{0}$), which we refer to as dead-beat (DB) control. The corresponding control gains can be calcu-

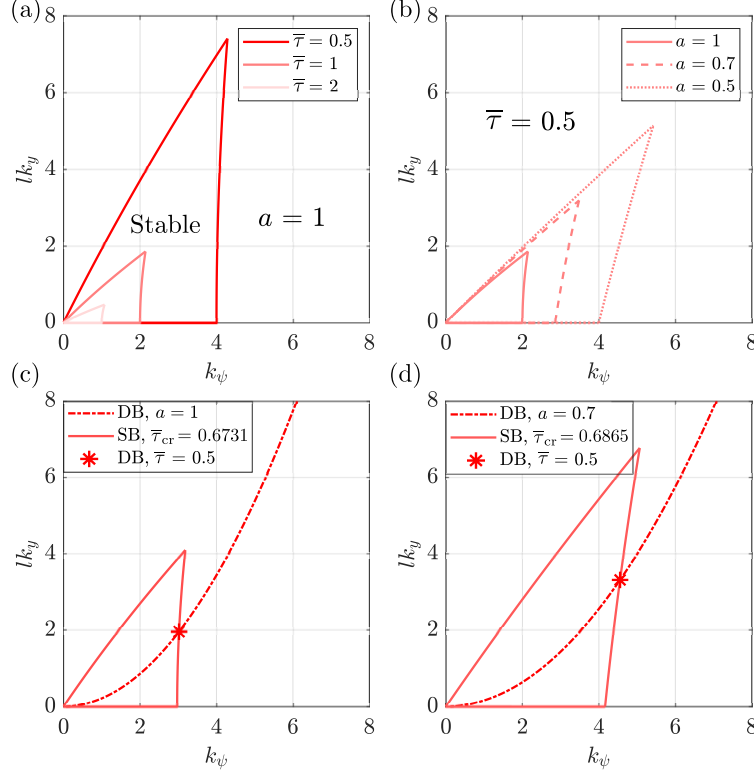


Figure 6.2: (a) Stability boundaries for different values of the dimensionless delay $\bar{\tau}$ when $a = 1$. (b) Stability boundaries for different values of the act-wait ratio a when $\bar{\tau} = 1$. (c) The control gains which give the dead-beat (DB) control are plotted as a red dashed-dotted curve for $a = 1$. The DB curve intersects the stability boundary of $\bar{\tau}_{cr} = 0.6731$ (red solid curve) at $\bar{\tau} = 0.5$. (d) The DB curve (red dashed-dotted curve) for $a = 0.7$ intersects the stability boundary of $\bar{\tau}_{cr} = 0.6865$ (red solid curve) at $\bar{\tau} = 0.5$.

lated analytically:

$$\begin{aligned}
 lk_{y,db} &= \frac{-6a - 6 + 2\sqrt{12a^2 + 18a + 9}}{a^3\bar{\tau}^2}, \\
 k_{\psi,db} &= \frac{8a + 6 - 2\sqrt{12a^2 + 18a + 9}}{a^2\bar{\tau}},
 \end{aligned} \tag{6.10}$$

and these depend on a and $\bar{\tau}$ only. The DB control gains are shown in Figure 6.2(c) and (d) as red dashed-dotted curves for $a = 1$ and $a = 0.7$, respectively. Similar to the continuous-time system, these curves intersect the stability boundaries. The DB gains for $\bar{\tau} = 0.5$ are marked by red stars and these are located at the stability boundary of the critical delay value $\bar{\tau}_{cr}$. However, the relationship between $\bar{\tau}$ and $\bar{\tau}_{cr}$ now depends on the choice of a .

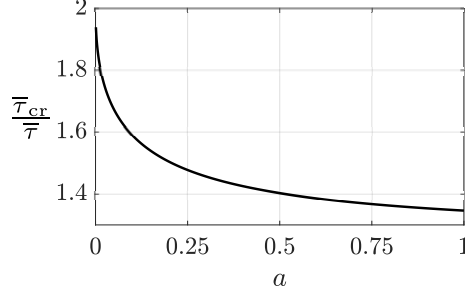


Figure 6.3: Robustness coefficient $\bar{\tau}_{cr}/\bar{\tau}$ as a function of the act-wait ratio a .

From (6.9), the right boundary is given by

$$k_\psi = \frac{1}{24}a^3\bar{\tau}_{cr}^3(lk_y)^2 - \frac{1}{2}a\bar{\tau}_{cr}lk_y + \frac{1}{2}\bar{\tau}_{cr}lk_y + \frac{2}{a\bar{\tau}_{cr}}. \quad (6.11)$$

Therefore, substituting (6.10), we obtain the fourth-order equation

$$\frac{r^2}{3a} \left(\frac{\bar{\tau}_{cr}}{\bar{\tau}} \right)^4 + \frac{2ar - 2r}{a} \left(\frac{\bar{\tau}_{cr}}{\bar{\tau}} \right)^2 - (4a + 4r) \frac{\bar{\tau}_{cr}}{\bar{\tau}} + 4a = 0, \quad (6.12)$$

where $r = 3a + 3 - \sqrt{12a^2 + 18a + 9}$. This can be solved numerically, for $\bar{\tau}_{cr}/\bar{\tau}$ and the solution is plotted in Figure 6.3 as a function of a . The coefficient $\bar{\tau}_{cr}/\bar{\tau}$ decreases with a , changing from 2 to 1.3463. Recall that this coefficient is a constant around 2.5 for the continuous-time system without act-and-wait, cf. (5.13). Although the robustness coefficient is smaller, applying act-and-wait greatly increases the absolute stability region and significantly improves the performance.

We simulate nonlinear system (5.1), (5.2) and the nonlinear system (6.6) with different choices of control gains and relate the linear stability charts with the nonlinear behaviors. In the simulation, we use the constant longitudinal velocity $v = 2.5$ m/s, the wheelbase $l = 2.5$ m, and the delay $\tau = 1$ s, which give the dimensionless delay $\bar{\tau} = 1$. For the act-and-wait algorithm, we select $a = 1$ and $t_w = \tau$. This means that the acting period is the same as the waiting period, as well as the delay in the system. Although the system is nonlinear, the effect of the nonlinearity is small as we consider small yaw angles.

First, we compare the performance of the original system without act-and-wait control to the one with act-and wait. In Figure 6.4(a), we compare the stable domain of the original system (bounded by the black curves) to that of the system with act-and-wait on the top of the same controller when using $a = 1$ (bounded by the red curves). Observe the significant expansion of the stable domain due to act-and-wait. The black star corresponds to the fastest

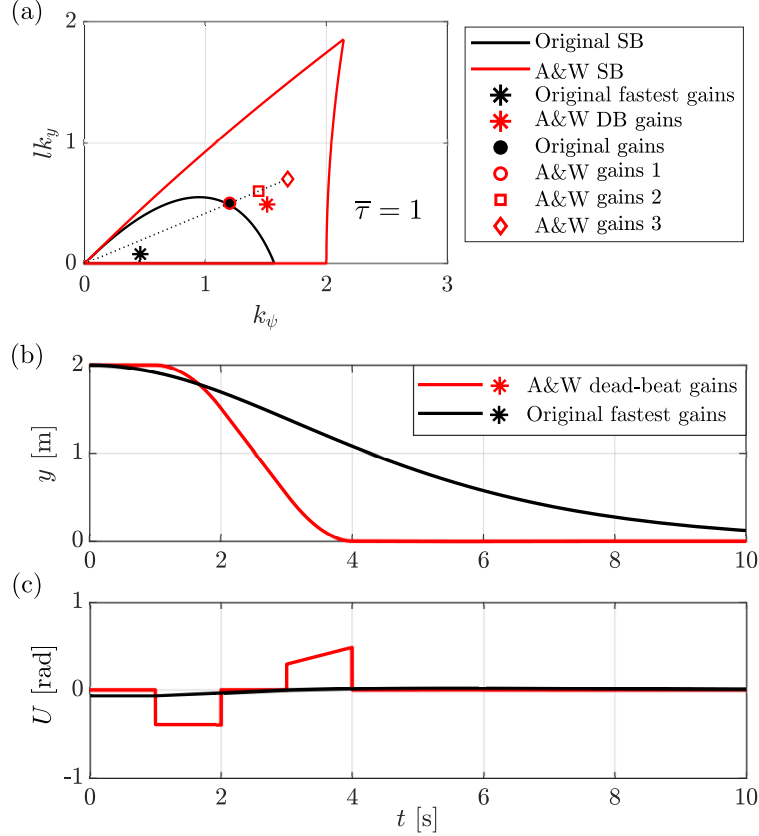


Figure 6.4: (a) Stability boundaries and selected control gains for scenarios with and without act-and-wait algorithm shown by red and black colors, respectively. (b) and (c) Simulation results for different choices of control gains with and without act-and-wait algorithms.

convergence rate $\rho_{\text{fast}} = -0.5858$ for the original controller, and the red star corresponds to the dead-beat control gains for act-and-wait enhanced controller. The lateral error and the control input are shown in Figure 6.4(b) and (c), when the “best-performance” gains are chosen in both scenarios. With act-and-wait algorithm and the DB gains, the system can converge in 4 seconds (two periods), while the fastest convergence without act-and-wait takes around 10 seconds (and is only asymptotic in nature).

The system may lose stability for some control gains, see, for instance, the gains marked by the black dot just outside the stability boundary in Figure 6.4(a). One can apply act-and-wait on top of the current controller to stabilize the system. Additionally, the final control input can be amplified by multiplying a coefficient to achieve better performance while exploiting the larger stable region under act-and-wait. This is equivalent to increasing control gains along the dotted black line. We amplify the control input by different coefficients 1, 1.2 and 1.4 as indicated by the red circle, square and diamond, respectively. In Figure 6.5, we

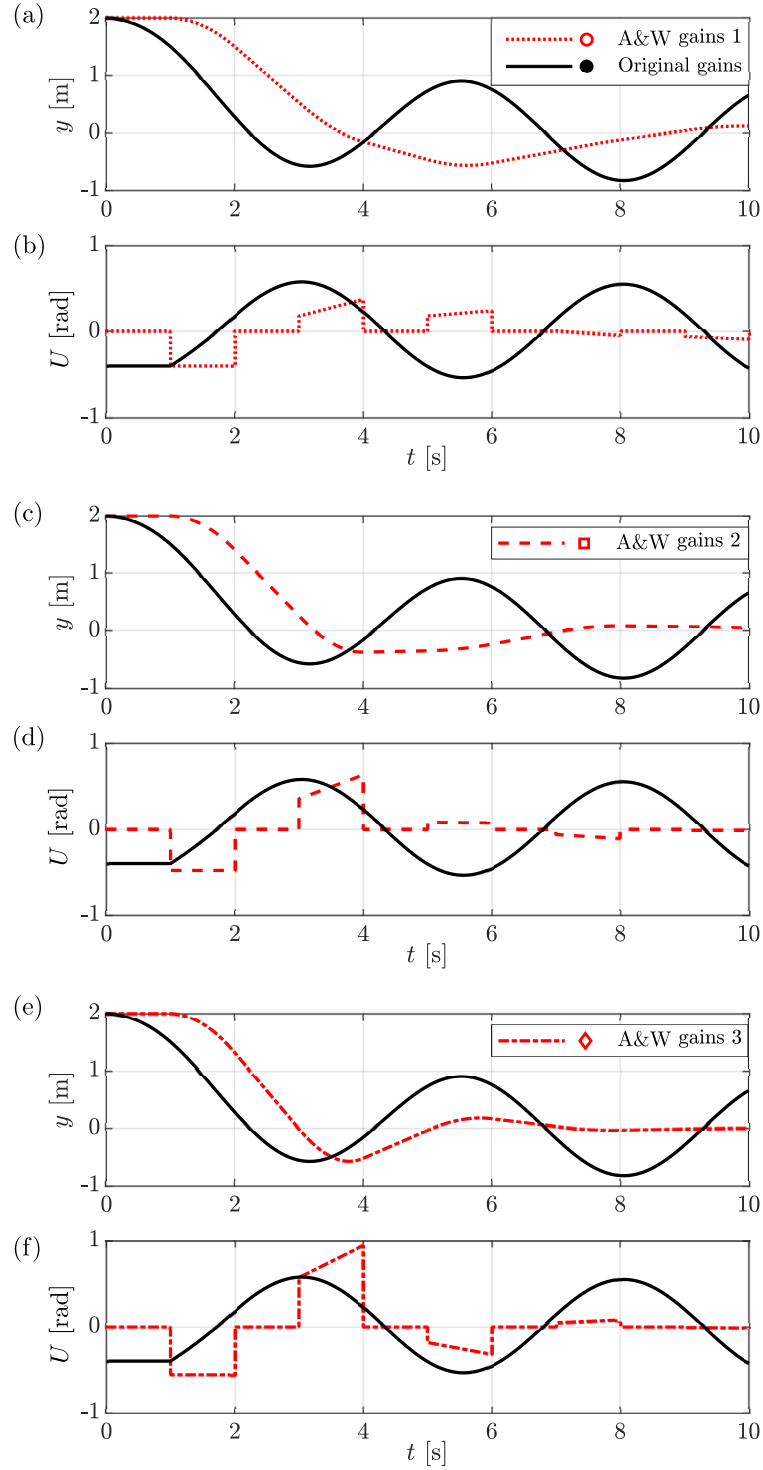


Figure 6.5: Simulations when using the act-and-wait algorithm with different amplifications of the control gains. (a)-(b) No amplification of the control gains. (c)-(d) Amplification with coefficient 1.2. (d)-(e) Amplification with coefficient 1.4.

show the simulation results for those three gains with act-and-wait (red curves) and compare the results to the original system without act-and-wait (black curves) to demonstrate the performance improvements in terms of convergence time. These highlight the benefits of modulating the control signal without changing the original controller, which can potentially be applied on top of a human remote operator to assist ToD. Note that the location of the original gains also influences the performance of the act-and-wait algorithm. Thus, to achieve the best performance, it is necessary to tune the two gains individually.

6.3 Summary

In this chapter, we provided the act-and-wait steering control strategy as a solution to mitigating the effect of delay on the lateral dynamics with the knowledge of learned delay. With act-and-wait strategy, the stable region in the parameter space increases while the robustness coefficient decreases. The system with act-and-wait strategy also converges much faster at its best performance. It means that the act-and-wait could potentially stabilize the system and achieve better control performance, but the performance is also more sensitive to the change of delay and velocity. As future directions, one may advance the act-and-wait strategy to generate smoother and more realistic input signals and compare the results to other delay mitigation methods. The implementations on other maneuvers such as following a curved path need to be studied. Moreover, it would be interesting to integrate the act-and-wait strategy with the data-driven model. For example, one can linearize the trained NDDE of the vehicle dynamics and adjust the act-and-wait parameters based on the learned delay.

CHAPTER 7

Conclusion

In this dissertation, we proposed to learn the time delay systems from trajectory data with the time delay neural networks and trainable delays (TrTDNNs). We focused on capturing the time delay systems in continuous time, which results in neural delay differential equations (NDDE) and we highlighted the trainability of the delays in the neural networks. The proposed methods are applied to learn the delayed dynamics of intelligent vehicles, including a connected automated vehicle and a teleoperated vehicle with human operator. This dissertation contributes to the field of data-driven methods for learning time delay systems, and provides insights on the effects of latency on vehicle control.

Compared to the conventional time delay neural networks (TDNNs) with fixed time delays, we demonstrated that TrTDNNs can greatly reduce the complexity of the neural networks. It was shown that the simpler structure of the TrTDNN enables fast simulations and allows more sophisticated analysis on the trained model for further improvement. We unveiled the fundamental problem of converging to the undesired local minima and showed that the sparsity of the trainable delays in the network can mitigate this situation. Although the TrTDNNs do not fully remove the chance of converging to undesired local minima, multiple algorithms with different loss functions were presented to improve the training performance.

We provided various examples of learning the time delay systems with our proposed methods, demonstrating that with the trainable delays, both delays and the nonlinearity can be learned simultaneously. The independence of delay learning improves the generalizability of the network. We evaluated the trained TrTDNNs in continuous time (which become neural delay differential equations, NDDEs) using well-developed tools for delay differential equations (DDEs), such as spectrum analysis, bifurcation diagrams and the maximum Lyapunov exponent (MLE). We showed that a NDDE trained on one single chaotic trajectory at a certain delay can reproduce the behavior at other delay values. This indicates that the quality of the training data is crucial, especially for autonomous systems without excitation. The proposed method was compared with the sparse identification of nonlinear dynamics (SINDy) method, which is a well-known data-driven method for learning the nonlinear dynamics from

data. The SINDy method requires a library of candidate functions which includes the correct nonlinearities (while not including similar functions) to restrict the search space. The NDDE with trainable delays provides interpretable results without the need of a library and the learning performance can be improved by increasing the amount of high-quality training data.

As future work, we can improve the training algorithm for the NDDE approach, by restricting the nonlinearities and adapting different parameter update algorithms. For instance, one may use external optimizers for the delays and extend the current algorithms to online training. Moreover, the extension of the NDDE to other types of delay differential equations such as distributed delays and time-varying delays is also very intriguing. Other potential future directions include applying the proposed NDDE with trainable delays on typical machine learning problems such as image classification and natural language processing. With the delays in the differential equations, the size of the model can be reduced significantly while still exhibiting rich dynamics. Additionally, one may apply the existing algorithms on reduced state space or latent space, though the advantage of having interpretable learned delays in the neural networks may diminish.

Besides putting forward the data-driven methods for learning the time delay systems, we also studied the effects of latency thoroughly on the vehicle control in this dissertation. We considered the acceleration and the steering angle as control inputs in the longitudinal and lateral control, respectively. We demonstrated through analytical studies that the choice of control gains can be determined to achieve fastest convergence with the knowledge of time delays in both longitudinal and lateral dynamics. For the lateral dynamics, one can nondimensionalize the system and summarize the effect of longitudinal velocity and the overall latency in a scaled delay. The similarity of the scaled delay in the lateral dynamics and the actual time delay in the longitudinal dynamics indicates the same generic results in both analytical studies. If the control gains are selected under a nominal (scaled) delay to achieve the fastest convergence, the vehicle can maintain stability up to 2.5 times of that nominal (scaled) delay. In addition, the road curvature can play a crucial role in the stability of the lateral dynamics.

In the context of teleoperated driving (ToD), we showed the change of stability and performance under periodic time delays and stochastic time-varying time delays arising in the communication network. The presented comprehensive stability and performance analysis can be used to select appropriate control gains, maneuvers, and wireless network resources for different ToD scenarios. This work can be extended to more complex vehicle models in the future. For instance, studying the joint effects of different delays in the longitudinal and lateral dynamics is particularly intriguing.

Equipped with the knowledge of vehicle dynamics and time delays, we were able to embed the delay and nonlinearities in the NDDEs to learn the overall latency and an equivalent controller from limited trajectory data. The stability analysis also enabled us to evaluate the learned model and the controller, especially when the controller is a remote human operator. The method was applied to the data generated from human-in-the-loop experiments, in high-fidelity driving simulator with the addition of a constant communication delay. After the delayed dynamics are learned, stability and performance analyses were performed to evaluate the ToD behavior under latency. We also proposed an act-and-wait algorithm to improve the stability and achieve deadbeat control, with the knowledge of delay value.

In the future, we can study the uncertainty in the human behavior through experiments in simulation environment, implement online training and evaluating algorithms for NDDEs. Our results showed that even under the same wireless network conditions, human may behave differently. It is necessary to learn the time-varying parameters in the real-world deployments. Apart from the act-and-wait algorithm presented in this dissertation, more advanced control methods should also be integrated to improve the performance of the teleoperated driving behaviors and ensure safety, by utilizing the knowledge of time delays. It would be also valuable to embed more complex vehicle models in the NDDE, explore other data-driven methods including “end-to-end” modeling and compare those methods on intelligent vehicle applications. The current NDDE model designed for learning ToD behaviors is restricted by the embedded physics, so the accuracy of the model largely depends on the correctness of this knowledge. In general, data-driven methods involve a tradeoff between the amount of data required and the reliance on prior knowledge. Future work is needed to strike a balance between these two aspects and to adapt data-driven approaches for various vehicle-related applications. High-quality data is especially vital and valuable in advancing the field of intelligent vehicles. It is important to collect and utilize such data wisely, with the aid of domain knowledge.

APPENDIX A

Training Algorithms Used in the Examples

We first present the mini-batch gradient descent (GD) training algorithm proposed for the derivative loss and GD updating rule in Algorithm 2. In this section, we specify $\mathcal{L} = \mathcal{L}_{\dot{x}}$ and consider the training loss $\mathcal{L}_{\text{tr}}$ over training dataset and a validation loss $\mathcal{L}_{\text{va}}$ over validation dataset, which are both based on L_2 norm evaluation on the state derivatives. In both algorithms [56], the inputs and outputs of the network is normalized between $[0,1]$ and the training stops when the validation loss $\mathcal{L}_{\text{va}}$ starts increasing consecutively. This is the algorithm used in the example in Section 2.4 and it is extend to the Algorithm 3 for addressing undesired local minima [61]. We also introduce the Levenberg-Marquardt (LM) method as an alternative update rule. The details of the gradient calculation are given in Appendix B.

In the GD method, the updating formula of parameter $\theta \in \mathbb{R}^M$ at iteration $q + 1$ uses the gradient of the loss function at iteration q :

$$\theta_{q+1} = \theta_q - \eta_{\theta} \frac{\partial \mathcal{L}}{\partial \theta_q}. \quad (\text{A.1})$$

The learning rate η_{θ} can be a constant or be adaptive during the training. In order to calculate the gradient $\frac{\partial \mathcal{L}}{\partial \theta_q}$ we reformulate the cost function(2.18) as

$$\mathcal{L}(\theta) = \frac{1}{N} E^{\top}(\theta) E(\theta), \quad (\text{A.2})$$

where

$$E(\theta) = \begin{bmatrix} e^1 \\ e^2 \\ \vdots \\ e^N \end{bmatrix} \in \mathbb{R}^{Nn}, \quad (\text{A.3})$$

and

$$e^j = \hat{x}(t_0 + j\Delta t) - \dot{x}(t_0 + j\Delta t). \quad (\text{A.4})$$

Then the updating formula (A.1) becomes

$$\theta_{q+1} = \theta_q - \eta_\theta \frac{2}{N} \frac{\partial E^\top}{\partial \theta_q} E(\theta_q), \quad (\text{A.5})$$

which contains the Jacobian matrix $\frac{\partial E^\top}{\partial \theta} \in \mathbb{R}^{M \times (Nn)}$.

For the LM method, we use the formula

$$\theta_{q+1} = \theta_q - \left(\frac{2}{N} \frac{\partial E^\top}{\partial \theta_q} \frac{\partial E}{\partial \theta_q} + \mu I \right)^{-1} \frac{2}{N} \frac{\partial E^\top}{\partial \theta_q} E(\theta_q), \quad (\text{A.6})$$

where μ is the tuning parameter and I denotes the $M \times M$ identity matrix. For large μ , the LM method approximates the GD method with a small learning rate. For small μ , it approximates Newton's method with an approximate Hessian matrix $\frac{\partial E^\top}{\partial \theta} \frac{\partial E}{\partial \theta} \in \mathbb{R}^{M \times M}$. Both algorithms (A.5) and (A.6) contain the Jacobian matrix

$$\frac{\partial E^\top}{\partial \theta} = \left[\left(\frac{\partial e^1}{\partial \theta} \right)^\top, \dots, \left(\frac{\partial e^N}{\partial \theta} \right)^\top \right], \quad (\text{A.7})$$

which requires the calculations of the gradients $\frac{\partial e^j}{\partial \theta}$, $j = 1, \dots, N$.

Data: training data and validation data

Goal: learn $\theta = \{W_l, b_l, \tau_k\}$, $l = 1, \dots, L, k = 1, \dots, d$

- Normalize inputs and outputs between $[0,1]$ and get derivatives of normalized inputs.
- Choose the learning rate η_θ and learning rate updating method.
- Set maximum iteration number $q_{\max}$, maximum violation number $v_{\max}$ and maximum allowed delay $\tau_{\max}$.
- Initialize W_l between $[-1,1]$, $b_l = 0$ and τ_k within interval $[0, \tau_{\max}]$, $v = 0$.

for $q = 1, \dots, q_{\max}$ **do**

- Randomly take one batch of data from the training set.
- Construct the input $\mathbf{z}_0$ for selected trajectories based on τ_k .
- Calculate the losses $\mathcal{L}_{\text{tr}}$ and $\mathcal{L}_{\text{va}}$ for training and validation data.
- Obtain the gradient information and update parameters $\theta_{q+1} \leftarrow \theta_q$, using (A.5).

if $\mathcal{L}_{\text{va}}$ Increases consecutively for $v_{\max}$ times **or** any $\tau_k \notin [0, \tau_{\max}]$ **then**
| break loop

end

end

$\theta = \theta_q$

Algorithm 2: Mini-batch GD training algorithm for NDDEs with trainable delays

Data: training data and validation data

Goal: learn $\theta = \{W_l, b_l, \tau_k\}$, $l = 1, \dots, L, k = 1, \dots, d$

- Normalize inputs and outputs between $[0, 1]$ and obtain the derivatives of the inputs.
- Choose the updating formulas (A.5) or (A.6) and tune parameters η_θ or μ .
- Set maximum iteration number $q_{\max}$, maximum violation number $v_{\max}$, maximum allowed delay $\tau_{\max}$.
- Choose different initializations for W_l uniformly between $[-1, 1]$, set $b_l = 0$, and initialize τ_k within interval $[0, \tau_{\max}]$.

For each initialization, set $v = 0$ and start training

for $q = 1, \dots, q_{\max}$ **do**

- Construct the input $\mathbf{z}_0$ for selected trajectories based on τ_k .
- Calculate the losses $\mathcal{L}_{\text{tr}}$ and $\mathcal{L}_{\text{va}}$ for training and validation data.
- Obtain the gradient information and update parameters $\theta_{q+1} \leftarrow \theta_q$, using (A.5), (A.6) or (2.21)-(2.23).

if $\mathcal{L}_{\text{va}}$ Increases consecutively for $v_{\max}$ times **or** any $\tau_k \notin [0, \tau_{\max}]$ **then**
| break loop

end

end

After collecting all parameters from multiple attempts at the end of training, choose the set of parameters θ which gives the smallest $\mathcal{L}_{\text{tr}}$ among them.

Algorithm 3: Multi-attempt gradient-based training algorithm for NDDEs with trainable delays

APPENDIX B

Analytical Derivation of the Gradients

The analytical derivation of the gradients of the derivative loss function $\mathcal{L}_{\dot{x}}$ is given by the follow. To simplify the notation we drop the superscript j . Also, we vectorize the weight matrices $W_l \in \mathbb{R}^{n_l \times n_{l-1}}$ into the vectors

$$\overline{W}_l = \left[w_{1,1}, \dots, w_{1,n_{l-1}}, w_{2,1}, \dots, w_{2,n_{l-1}}, \dots \right]^\top \in \mathbb{R}^{n_l n_{l-1}}, \quad (\text{B.1})$$

where n_l denotes the dimension of z_l . Then the gradients with respect to weights $\overline{W}_l$ and biases b_l can be expressed as

$$\frac{\partial e}{\partial \overline{W}_l} = \frac{\partial e}{\partial z_L} h_l \frac{\partial s_l}{\partial \overline{W}_l} = h_l (I_{n_l \times n_l} \otimes z_{l-1}^\top), \quad (\text{B.2})$$

and

$$\frac{\partial e}{\partial b_l} = \frac{\partial e}{\partial z_L} h_l \frac{\partial s_l}{\partial b_l} = h_l, \quad (\text{B.3})$$

respectively, where

$$h_l = h_{l+1} \frac{\partial s_{l+1}}{\partial z_l} \frac{\partial z_l}{\partial s_l} = h_{l+1} W_{l+1} f'_l(s_l), \quad (\text{B.4})$$

for $l = 1, \dots, L-1$ and

$$h_L = f'_L(s_L) = I_{n_L \times n_L}, \quad (\text{B.5})$$

cf. (2.5). Note that $\otimes$ denotes the Kronecker product so that $(I_{n_l \times n_l} \otimes z_{l-1}^\top) \in \mathbb{R}^{n_l \times n_l n_{l-1}}$. $h_l \in \mathbb{R}^{n_L \times n_l}$, therefore $\frac{\partial e}{\partial \overline{W}_l} \in \mathbb{R}^{n_L \times n_l n_{l-1}}$ and $\frac{\partial e}{\partial b_l} \in \mathbb{R}^{n_L \times n_l}$. In our case, the dimension of the last layer is equal to the dimension of the state derivative $\dot{x}$, i.e., $n_L = n$.

In case of the time delay parameters we define the vector

$$T = [\hat{\tau}_0, \dots, \hat{\tau}_d, \hat{\sigma}_0, \dots, \hat{\sigma}_q]^\top \in \mathbb{R}^D, \quad (\text{B.6})$$

with $D = d + q + 2$, which is part of the parameter θ . The gradient with respect to the delay

vector T can be calculated as

$$\frac{\partial e}{\partial T} = \frac{\partial e}{\partial z_L} h_1 \frac{\partial s_1}{\partial \mathbf{z}_0} \frac{\partial \mathbf{z}_0}{\partial T} = h_1 W_1 \frac{\partial \mathbf{z}_0}{\partial T}. \quad (\text{B.7})$$

Here the network input $\mathbf{z}_0 = [z_{0,1}, \dots, z_{0,n_0}]^\top \in \mathbb{R}^{n_0}$ is an augmented vector of states and control inputs corresponding to each delay value, where $n_0 = D(n+p)$ and n and p are the dimension of the state x and the input u . The matrix $\frac{\partial \mathbf{z}_0}{\partial T} \in \mathbb{R}^{n_0 \times D}$ can be written as

$$\frac{\partial \mathbf{z}_0}{\partial T} = \left[\frac{\partial \mathbf{z}_0}{\partial \tau_1}, \dots, \frac{\partial \mathbf{z}_0}{\partial \tau_D} \right]. \quad (\text{B.8})$$

Each element $z_{0,i}, i = 1, \dots, n_0$, is a scalar associated with one specific delay $\tau_b \in T, b = 1, \dots, D$. Thus, we can write

$$\frac{\partial z_{0,i}}{\partial \tau_b} = \frac{\partial z_{0,i}(t - \tau_b)}{\partial \tau_b} = -\frac{\partial z_{0,i}(t - \tau_b)}{\partial t} = -\dot{z}_{0,i}(t - \tau_b), \quad (\text{B.9})$$

where $\dot{z}_{0,i}(t - \tau_b)$ is the time derivative of the network input at time $t - \tau_b$, which can be approximated from the data. We can set $\frac{\partial z_{0,i}}{\partial \tau_b} = 0$ if we regulate the network to have a delay-free input $x(t)$ or $u(t)$. We also remark that to ensure the non-negativeness of time delays, after using the updates (A.5) or (A.6), we apply $\max\{0, T_{k+1}\}$ element wise; cf. (B.6).

Recall that in the formulas above, the parameter vector θ collects vectorized weights and biases and the time delays. Therefore, we obtain

$$\frac{\partial e}{\partial \theta} = \left[\frac{\partial e}{\partial \bar{W}_1}, \frac{\partial e}{\partial b_1}, \dots, \frac{\partial e}{\partial \bar{W}_L}, \frac{\partial e}{\partial b_L}, \frac{\partial e}{\partial T} \right] \in \mathbb{R}^{n_L \times M}, \quad (\text{B.10})$$

and concatenate the e^j -s of different time steps as in (A.7) to construct the Jacobian $\frac{\partial E^\top}{\partial \theta} \in \mathbb{R}^{M \times (N n_L)}$ with $n_L = n$.

APPENDIX C

MATLAB Codes of NDDE Solver for Simulation Loss Training Algorithms

C.1 Example code for NDDE solver

```
%% 4 step Adams-Bashforth scheme to solve DDEs
%% with multiple delays
function xsolu=dl_ddeab4(fun,par,delay,hist,time)
dim=length(hist(time(1)));
h=time(2)-time(1);
Maxdelay = max(ceil(max(delay)/h)*h,h);
timehist=fliplr(time(1)-Maxdelay:h:time(1));
z0=dlarray(zeros(dim,length(timehist)));
for i=1:length(timehist)
    z0(:,i)=hist(timehist(i));
end
z0=z0(:);
xsolu=dlarray(zeros(dim,length(time)));
for kk=1:length(time)
    timestep
    tnew=time(kk);
    if kk==1
        znew=z0;
    else
        xold=zold(1:dim);
        Z = reshape(zold,[dim length(timehist)]);
        xdelay = interp1(fliplr(timehist),...
```

```

        flipplr(Z)', time(1)-delay)';
        xdelay = reshape(xdelay,[],1);
        rhs1=fun(told,xold,xdelay,par);
        % calculation of the solution
        if kk==2
            xnew = xold+h*rhs1;
        elseif kk==3
            xnew = xold+h*(3*rhs1-rhs2)/2;
        elseif kk==4
            xnew = xold+h*...
                (23*rhs1-16*rhs2+5*rhs3)/12;
        elseif kk>4
            xnew=xold+h*...
                (55*rhs1-59*rhs2+37*rhs3-9*rhs4)/24;
        end
        znew=[xnew;zold(1:end-dim)];
    end
    if kk>3
        rhs4=rhs3;
    end
    if kk>2
        rhs3=rhs2;
    end
    if kk>1
        rhs2=rhs1;
    end
    told=tnew;
    zold=znew;
    xsolu(:,kk)=znew(1:dim);
end

```

C.2 Example code for the NDDE initialization

```

%% an example of function
%% as the input of DDE solver above

```

```

function dx=fun(t,x,xdelay,par)
dx = par.fc3.Weights*tanh(par.fc2.Weights...
    *tanh(par.fc1.Weights*...
    [x;xdelay]+par.fc1.Bias)...
    +par.fc2.Bias);
end

function bias = initializeZeros(sz)
bias = zeros(sz,'single');
bias = dldarray(bias);
end

function weights = initializeGlorot(sz)
Z = 2*rand(sz,'single') - 1;
bound = sqrt(6 / (sz(2)+ sz(1)));
weights = bound * Z;
weights = dldarray(weights);
end

%% initialization of the parameters
% guess the number of delay 'nd',
% initialize randomly as dldarray
% NDDE will be the par in the fun
tau = dldarray(tau_max*rand(1,nd));
% an example of network
NDDE = struct;
NDDE.fc1 = struct;
sz = [hiddenSize nx*(1+nd)];
NDDE.fc1.Weights = initializeGlorot(sz);
NDDE.fc1.Bias     = initializeZeros([sz(1) 1]);
NDDE.fc2 = struct;
sz = [hiddenSize hiddenSize];
NDDE.fc2.Weights = initializeGlorot(sz);
NDDE.fc2.Bias     = initializeZeros([sz(1) 1]);
NDDE.fc3 = struct;
sz = [nx hiddenSize];

```

```
NDDE.fc3.Weights = initializeGlorot(sz);
```

C.3 Code snippets for training

```
% xsolu is from dde solver 'dl_ddeab4'  
% x_target is the observed data  
loss = l2loss(xsolu,x_tartget)  
grad_par = dlgradient(loss,par);  
grad_tau = dlgradient(loss,par);  
% adamupdate as an example  
[par,averageGrad_p,averageSqGrad_p] = ...  
adamupdate(par,grad_par,averageGrad_p,...  
averageSqGrad_p,iteration);  
[tau,averageGrad_t,averageSqGrad_t] = ...  
adamupdate(tau,grad_tau,averageGrad_t,...  
averageSqGrad_t,iteration);  
tau = min(max(1e-5,tau),tau_max-1e-5);
```

APPENDIX D

Creating NODE from TrTDNN

D.1 ODE Approximation of DDE

Consider an autonomous time delay system

$$\dot{x}(t) = g(x_t), \quad (\text{D.1})$$

where $x \in \mathbb{R}^n$ and $x_t \in \mathcal{X}$. Here $\mathcal{X} = C([- \tau_{\max}, 0], \mathbb{R}^n)$ is the space of continuous functions, i.e., $x_t(\vartheta) := x(t + \vartheta)$, $\vartheta \in [- \tau_{\max}, 0]$, and $\tau_{\max} > 0$ is the maximum delay. The system (D.1) contains the functional $g: \mathcal{X} \rightarrow \mathbb{R}^n$ and it can be re-written as an operator differential equation:

$$\dot{x}_t = \mathcal{G}(x_t), \quad (\text{D.2})$$

where the operator $\mathcal{G}: \mathcal{X} \rightarrow \mathcal{X}$ is defined as

$$\mathcal{G}(\varphi) = \begin{cases} g(\varphi), & \vartheta = 0, \\ \frac{d\varphi}{d\vartheta}, & \vartheta \in [- \tau_{\max}, 0), \end{cases} \quad (\text{D.3})$$

see [111, 10].

We discretize in the variable ϑ using a mesh of size $M + 1$ and define the vector

$$X(t) = \begin{bmatrix} X_1(t) \\ X_2(t) \\ \vdots \\ X_{M+1}(t) \end{bmatrix} = \begin{bmatrix} x(t) \\ x(t - h) \\ \vdots \\ x(t - Mh) \end{bmatrix} \quad (\text{D.4})$$

where $h = \tau_{\max}/M$. This way we can approximate (D.2) with the ordinary differential equation

$$\dot{X}(t) = G(X(t)), \quad (\text{D.5})$$

where the function $G: \mathbb{R}^{n(M+1)} \rightarrow \mathbb{R}^{n(M+1)}$ is given by

$$G(X) = \begin{bmatrix} \tilde{g}(X) \\ D_M X \end{bmatrix}. \quad (\text{D.6})$$

Here $\tilde{g}: \mathbb{R}^{n(M+1)} \rightarrow \mathbb{R}^n$ is the approximation of the functional $g: \mathcal{X} \rightarrow \mathbb{R}^n$. For example, considering

$$g(x_t) = f(x(t), x(t - (j + \alpha)h), x(t - \tau_{\max})) \quad (\text{D.7})$$

with $j \in \{0, 1, \dots, M-1\}$, $\alpha \in (0, 1)$, and $f: \mathbb{R}^{3n} \rightarrow \mathbb{R}^n$, one may use linear interpolation to obtain

$$x(t - (j + \alpha)h) = (1 - \alpha)x(t - jh) + \alpha x(t - (j + 1)h), \quad (\text{D.8})$$

yielding

$$\tilde{g}(X) = f(X_1, (1 - \alpha)X_{j+1} + \alpha X_{j+2}, X_{M+1}). \quad (\text{D.9})$$

The matrix $D_M \in \mathbb{R}^{(n-1)(M+1) \times n(M+1)}$ represents the numerical differentiation scheme used to approximate the time derivative of past states. For instance, choosing the forward Euler method to approximate the time derivative

$$\dot{x}(t - jh) = \frac{1}{h}(x(t - (j - 1)h) - x(t - jh)), \quad (\text{D.10})$$

gives

$$\dot{X}_{j+1}(t) = \frac{1}{h}(X_j(t) - X_{j+1}(t)), \quad (\text{D.11})$$

yielding

$$D_M = \frac{1}{h} \begin{bmatrix} I & -I & & & \\ & I & -I & & \\ & & \ddots & \ddots & \\ & & & I & -I \end{bmatrix}, \quad (\text{D.12})$$

where $I \in \mathbb{R}^{n \times n}$ is the n -dimensional unit matrix and only the nonzero blocks of D_M are spelled out.

In summary, (D.5,D.6) discretizes (D.2,D.3) in the history variable ϑ . As illustrated in Figure D.1, we still maintain the continuous dynamics in time t . This enables us to propagate the state forward with high accuracy, which will be beneficial for learning. Note that, however, data samples will only be available with sampling time h .

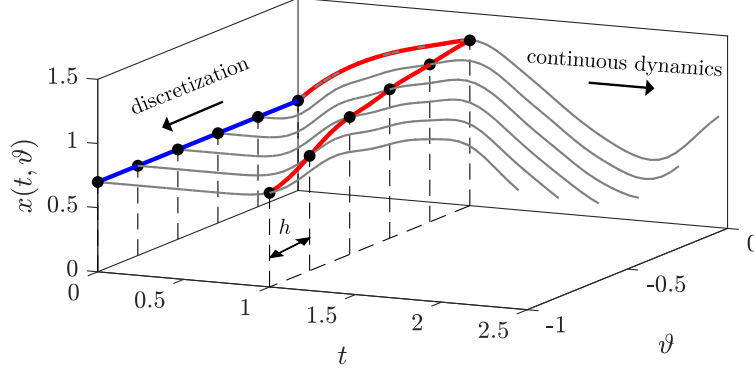


Figure D.1: Illustration of discretization of a DDE in history.

D.2 NODE approximation of TrTDNN

We define a matrix which allows us to approximate the delayed states $z_0^{t-\tau_i}$ in (2.5) with the components X_j in (D.4), namely

$$\mathbf{z}_0^t = PX(t), \quad (\text{D.13})$$

where $P \in \mathbb{R}^{d \times n(M+1)}$ and since typically $d \ll M$, P is a wide matrix. For the case of the linear interpolation given in example (D.7,D.8,D.9) we have

$$P = \begin{bmatrix} I & & \\ & (1-\alpha)I & \alpha I \\ & & I \end{bmatrix} \quad (\text{D.14})$$

where the block $(1-\alpha)I$ is the $j+1$ -st column and the block αI is the $j+2$ -nd column. Indeed, one may use higher-order interpolation methods which result in more nonzero blocks in P .

The output of the neural network (2.3) is set to be the derivative

$$\dot{x}(t) = z_L^t = \text{net}(PX(t)), \quad (\text{D.15})$$

and thus the neural ordinary differential equation becomes

$$\dot{X}(t) = \widehat{G}(X(t)), \quad (\text{D.16})$$

where the function $\widehat{G}: \mathbb{R}^{n(M+1)} \rightarrow \mathbb{R}^{n(M+1)}$ is given by

$$\widehat{G}(X) = \begin{bmatrix} \text{net}(PX) \\ D_M X \end{bmatrix}, \quad (\text{D.17})$$

cf. (D.5,D.6). We remark that to obtain D_M , instead of the forward Euler method we will use the central difference method which yields

$$D_M = \frac{1}{2h} \begin{bmatrix} I & O & -I & & \\ & \ddots & \ddots & \ddots & \\ & & I & O & -I \\ & & & 2I & -2I \end{bmatrix}, \quad (\text{D.18})$$

where $O \in \mathbb{R}^{n \times n}$ is the n -dimensional zero matrix while the other nonzero blocks are not spelled out; cf. (D.12). Note that the last row in D_M is still derived from forward Euler method.

When training the NODE (D.16,D.17), the weights W_l and biases b_l in (2.3) as well as the delays τ_i are updated at every iteration. The update in τ_i iteratively changes P defined in (D.13).

BIBLIOGRAPHY

- [1] S. S. Avedisov, G. Bansal, and G. Orosz. Impacts of connected automated vehicles on freeway traffic patterns at different penetration levels. *IEEE Transactions on Intelligent Transportation Systems*, 23:4305–4318, 2022.
- [2] A. O. Belyakov and A. P. Seyranian. Stability boundary approximation of periodic dynamics. In *1st International Nonlinear Dynamics Conference (NODYCON 2019), Volume I*, pages 13–23. Springer, 2020.
- [3] S. Beregi, S. S. Avedisov, C. R. He, D. Takács, and G. Orosz. Connectivity-based delay-tolerant control of automated vehicles: theory and experiments. *IEEE Transactions on Intelligent Vehicles*, 8(1):275–289, 2023.
- [4] E. Bertolazzi and M. Frego. On the g2 hermite interpolation problem with clothoids. *Journal of Computational and Applied Mathematics*, 341:99–116, 2018.
- [5] D. G. Black, D. Andjelic, and S. E. Salcudean. Evaluation of communication and human response latency for (human) teleoperation. *IEEE Transactions on Medical Robotics and Bionics*, 6(1):43–63, 2024.
- [6] C. Blackett, A. Fernandes, E. Teigen, and T. Thoresen. Effects of signal latency on human performance in teleoperations. In *5th International Virtual Conference on Human Interaction and Emerging Technologies (IHIET) and the 6th IHIET: Future Systems (IHIET-FS)*, pages 386–393. Springer, 2022.
- [7] E. Bozzo, D. Breda, and M. Tanveer. Sparse identification of time delay systems via pseudospectral collocation. *IFAC-PapersOnLine*, 58(27):108–113, 2024.
- [8] D. Breda and S. Della Schiava. Pseudospectral reduction to compute Lyapunov exponents of delay differential equations. *Discrete and Continuous Dynamical Systems-B*, 23(7):2727–2741, 2018.
- [9] D. Breda, O. Diekmann, M. Gyllenberg, F. Scarabel, and R. Vermiglio. Pseudospectral discretization of nonlinear delay equations: new prospects for numerical bifurcation analysis. *SIAM Journal on applied dynamical systems*, 15(1):1–23, 2016.
- [10] D. Breda, S. Maset, and R. Vermiglio. *Stability of linear delay differential equations: A numerical approach with MATLAB*. Springer, 2014.

- [11] D. Brunello, I. Johansson, M. Ozger, and C. Cavdar. Low latency low loss scalable throughput in 5g networks. In *93rd IEEE Vehicular Technology Conference (VTC2021-Spring)*, pages 1–7. IEEE, 2021.
- [12] S. L. Brunton and J. N. Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2022.
- [13] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.
- [14] G. Buza, J. Milton, L. Bencsik, and T. Insperger. Establishing metrics and control laws for the learning process: ball and beam balancing. *Biological Cybernetics*, 114:83–93, 2020.
- [15] F. Cacace, F. Conte, A. Germani, and G. Palombo. Optimal control of linear systems with large and variable input delays. *Systems & Control Letters*, 89:1–7, 2016.
- [16] A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.
- [17] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116(45):22445–22451, 2019.
- [18] F. Chen, H. Garnier, and M. Gilson. Robust identification of continuous-time models with arbitrary time-delay from irregularly sampled data. *Journal of Process Control*, 25:19–27, 2015.
- [19] Q. Chen and D. Wu. Delay-rate-distortion model for real-time video communication. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(8):1376–1394, 2015.
- [20] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018.
- [21] L. Dieci, M. S. Jolly, and E. S. Van Vleck. Numerical techniques for approximating Lyapunov exponents and their implementation. *Journal of Computational and Nonlinear Dynamics*, 6(1), 2011.
- [22] O. Diekmann, S. A. van Gils, S. M. Verduyn Lunel, and H.-O. Walther. *Delay Equations*. Springer, 1995.
- [23] O. El Marai and T. Taleb. Smooth and low latency video streaming for autonomous cars during handover. *IEEE Network*, 34(6):302–309, 2020.
- [24] K. Engelborghs, T. Luzyanina, and D. Roose. Numerical bifurcation analysis of delay differential equations using DDE-BIFTOOL. *ACM Transactions on Mathematical Software*, 28(1):1–21, 2002.

- [25] T. Ersal, I. Kolmanovsky, N. Masoud, N. Ozay, J. Scruggs, R. Vasudevan, and G. Orosz. Connected and automated road vehicles: state of the art and future challenges. *Vehicle System Dynamics*, 58(5):672–704, 2020.
- [26] J. Fayyad, M. A. Jaradat, D. Gruyer, and H. Najjaran. Deep learning sensor fusion for autonomous vehicle perception and localization: A review. *Sensors*, 20(15):4220, 2020.
- [27] S. Feng, Y. Zhang, S. E. Li, Z. Cao, H. X. Liu, and L. Li. String stability for vehicular platoon control: Definitions and analysis methods. *Annual Reviews in Control*, 47:81–97, 2019.
- [28] S. Fioretti and L. Jetto. Accurate derivative estimation from noisy data: a state-space approach. *International Journal of Systems Science*, 20(1):33–53, 1989.
- [29] E. Fridman. *Introduction to Time-Delay Systems*. Birkhäuser, 2014.
- [30] P. Gawthrop. Act-and-wait and intermittent control: Some comments. *IEEE Transactions on Control Systems Technology*, 18(5):1195–1198, 2010.
- [31] P. J. Gawthrop and L. Wang. Intermittent model predictive control. *Proceedings of the Institution of Mechanical Engineers, Part I*, 221(7):1007–1018, 2007.
- [32] J. I. Ge, S. S. Avedisov, C. R. He, W. B. Qin, M. Sadeghpour, and G. Orosz. Experimental validation of connected automated vehicle design among human-driven vehicles. *Transportation Research Part C*, 91:335–352, 2018.
- [33] J.-M. Georg, J. Feiler, S. Hoffmann, and F. Diermeyer. Sensor and actuator latency during teleoperation of automated vehicles. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 760–766. IEEE, 2020.
- [34] P. Ghorai, A. Eskandarian, and Y.-K. Kim. Study the effect of communication delay for perception and collision avoidance in cooperative autonomous driving. In *ASME International Mechanical Engineering Congress and Exposition*, volume 84553, page V07BT07A015. American Society of Mechanical Engineers, 2020.
- [35] D. Ghosh, A. R. Chowdhury, and P. Saha. Multiple delay rössler system—bifurcation and chaos control. *Chaos, Solitons & Fractals*, 35(3):472–485, 2008.
- [36] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *13th International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
- [37] M. Goldmann, C. R. Mirasso, I. Fischer, and M. C. Soriano. Learn one size to infer all: Exploiting translational symmetries in delay-dynamical and spatiotemporal systems using scalable neural networks. *Physical Review E*, 106:044211, Oct 2022.
- [38] N. Goodall. Non-technological challenges for the remote operation of automated vehicles. *Transportation research part A: policy and practice*, 142:14–26, 2020.

- [39] K. Gopalsamy. *Stability and Oscillations in Delay Differential Equations of Population Dynamics*. Mathematics and Its Applications. Springer Dordrecht, 1992.
- [40] K. Gu, V. L. Kharitonov, and J. Chen. *Stability of Time-Delay System*. Springer, 2003.
- [41] A. Gupta and P. F. Lermusiaux. Neural closure models for dynamical systems. *Proceedings of the Royal Society A*, 477(2252):20201004, 2021.
- [42] A. Gupta and P. F. Lermusiaux. Generalized neural closure models with interpretability. *Scientific Reports*, 13:10634, 2023.
- [43] D. Hajdu, J. I. Ge, T. Insperger, and G. Orosz. Robust design of connected cruise control among human-driven vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 21:749–761, 2020.
- [44] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.
- [45] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [46] M. Hofbauer, C. B. Kuhn, G. Petrovic, and E. Steinbach. Telecarla: An open source extension of the carla simulator for teleoperated driving research using off-the-shelf components. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 335–340. IEEE, 2020.
- [47] Y. Hossain, M. Metzler, J. Edelmann, and M. Plöchl. On the stability of the closed-loop teleoperated vehicle and teledriver system. In *Advanced Vehicle Control Symposium*, pages 557–563. Springer, 2024.
- [48] G. E. Hutchinson. Circular Causal Systems in Ecology. *Annals of the New York Academy of Sciences*, 50(4):221–246, 1948.
- [49] K. Ikeda and K. Matsumoto. High-dimensional chaotic behavior in systems with time-delayed feedback. *Physica D*, 29(1-2):223–235, 1987.
- [50] T. Insperger. Act-and-wait concept for continuous-time control systems with feedback delay. *IEEE Transactions on Control Systems Technology*, 14(5):974–977, 2006.
- [51] T. Insperger and G. Stépán. *Semi-Discretization for Time-Delay Systems*. Springer, 2011.
- [52] A. Javani, M. Zorgui, and Z. Wang. Age of information for multiple-source multiple-server networks. *IEEE Transactions on Networking*, 33(1):17–34, 2025.
- [53] X. A. Ji, S. S. Avedisov, M. I. Khan, O. Altintas, and G. Orosz. The effects of latency on teleoperated driving maneuvers. In *30th ITS World Congress*, 2024.

- [54] X. A. Ji, S. S. Avedisov, M. I. Khan, M. C. Lucas-Estañ, B. Coll-Perales, I. Vörös, O. Altintas, and G. Orosz. Fundamental rules of teleoperated driving with network latency on curvy roads. In *35th IEEE Intelligent Vehicles Symposium*, pages 1841–1846. IEEE, 2024.
- [55] X. A. Ji, T. G. Molnár, S. S. Avedisov, and G. Orosz. Feed-forward neural networks with trainable delay. In *2nd Conference on Learning for Dynamics and Control*, volume 120, pages 127–136. PMLR, 2020.
- [56] X. A. Ji, T. G. Molnár, S. S. Avedisov, and G. Orosz. Learning the dynamics of time delay systems with trainable delays. In *3rd Conference on Learning for Dynamics and Control*, volume 144, pages 930–942. PMLR, 2021.
- [57] X. A. Ji, T. G. Molnár, A. A. Gorodetsky, and G. Orosz. Bayesian inference for time delay systems with application to connected automated vehicles. In *IEEE Conference on Intelligent Transportation Systems Conference (ITSC)*, pages 3259–3264, 2021.
- [58] X. A. Ji and G. Orosz. Learning time delay systems with neural ordinary differential equations. *IFAC-PapersOnLine*, 55(36):79–84, 2022.
- [59] X. A. Ji and G. Orosz. Learning the dynamics of autonomous nonlinear delay systems. In *Learning for Dynamics and Control Conference*, pages 116–127. PMLR, 2023.
- [60] X. A. Ji and G. Orosz. Learn from one and predict all: single trajectory learning for time delay systems. *Nonlinear Dynamics*, pages 1–14, 2024.
- [61] X. A. Ji and G. Orosz. Trainable delays in time delay neural networks for learning delayed dynamics. *IEEE Transactions on Neural Networks and Learning Systems*, 2024.
- [62] S. B. Kamtam, Q. Lu, F. Bouali, O. C. Haas, and S. Birrell. Network latency in teleoperation of connected and autonomous vehicles: A review of trends, challenges, and mitigation strategies. *Sensors*, 24(12):3957, 2024.
- [63] A. Keane, B. Krauskopf, and H. A. Dijkstra. The effect of state dependence in a delay differential equation model for the el niño southern oscillation. *Philosophical Transactions of the Royal Society A*, 377(2153):20180121, 2019.
- [64] A. Keane, B. Krauskopf, and C. M. Postlethwaite. Climate models with delay differential equations. *Chaos*, 27(11):114309, 2017.
- [65] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations*, 2015.
- [66] J. Koch, T. Maxner, V. Amatya, A. Ranjbari, and C. Dowling. Physics-informed machine learning of parameterized fundamental diagrams. *arXiv preprint arXiv:2208.00880*, 2022.

- [67] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE intelligent vehicles symposium (IV)*, pages 1094–1099. IEEE, 2015.
- [68] A. Köpeczi-Bócz, Tamás, H. Sykora, and D. Takács. Data-driven delay identification with sindy. In *International Conference on Nonlinear Dynamics and Applications*, pages 481–491. Springer, 2023.
- [69] M. Krstic. *Delay Compensation for Nonlinear, Adaptive, and PDE Systems*. Birkhäuser, 2003.
- [70] Y. A. Kuznetsov, I. A. Kuznetsov, and Y. Kuznetsov. *Elements of applied bifurcation theory*, volume 112. Springer, 1998.
- [71] L. Larger, J.-P. Goedgebuer, and V. Udaltsov. Ikeda-based nonlinear delayed dynamics for application to secure optical transmission systems using chaos. *Comptes Rendus Physique*, 5(6):669–681, 2004.
- [72] M. Levine and A. Stuart. A framework for machine learning of model error in dynamical systems. *Communications of the American Mathematical Society*, 2(07):283–344, 2022.
- [73] D. J. Limebeer and M. Massaro. *Dynamics and optimal control of road vehicles*. Oxford University Press, 2018.
- [74] D.-T. Lin, J. E. Dayhoff, and P. A. Ligomenides. Adaptive time-delay neural network for temporal correlation and prediction. In *Intelligent Robots and Computer Vision XI: Biological, Neural Net, and 3D Methods*, volume 1826, pages 170–181. International Society for Optics and Photonics, 1992.
- [75] B. Liu, T. Liu, and P. Xiao. Dynamic event-triggered intermittent control for stabilization of delayed dynamical systems. *Automatica*, 149:110847, 2023.
- [76] H. Lu, G. Stépán, J. Lu, and D. Takács. Dynamics of vehicle stability control subjected to feedback delay. *European Journal of Mechanics-A/Solids*, 96:104678, 2022.
- [77] H. Lu, G. Stépán, J. Lu, and D. Takács. Dynamics of vehicle stability control subjected to feedback delay. *European Journal of Mechanics A*, 96:104678, 2022.
- [78] Y. Lu Murphey, I. Kolmanovsky, and P. Wattai. Advances, opportunities and challenges in ai-enabled technologies for autonomous and connected vehicles. In *AI-enabled Technologies for Autonomous and Connected Vehicles*, pages 1–16. Springer, 2023.
- [79] M. C. Lucas-Estañ, B. Coll-Perales, I. K. Khan, S. S. Avedisov, O. Altintas, J. Gozalvez, and M. Sepulcre. Support of teleoperated driving with 5G networks. In *98th IEEE Vehicular Technology Conference (VTC2023-Fall)*, pages 1–6. IEEE, 2023.
- [80] X. Ma, P. K. Wong, W. Li, J. Zhao, M. A. Ghadikolaei, and Z. Xie. Multi-objective H_2/H_∞ control of uncertain active suspension systems with interval time-varying delay. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 237(2):335–347, 2023.

- [81] M. C. Mackey and L. Glass. Oscillation and chaos in physiological control systems. *Science*, 197(4300):287–289, 1977.
- [82] W. Michiels and S.-I. Niculescu. *Stability and Stabilization of Time-delay Systems*. SIAM, 2007.
- [83] W. Michiels and S.-I. Niculescu. *Stability and stabilization of time-delay systems: An eigenvalue-based approach*. SIAM, 2007.
- [84] W. Michiels and B. Zhou. On the fixed-time stabilization of input delay systems using act-and-wait control. *Systems & Control Letters*, 146:104807, 2020.
- [85] T. G. Molnár, Z. Dombóvári, T. Insperger, and G. Stépán. On the analysis of the double Hopf bifurcation in machining processes via centre manifold reduction. *Proceedings of the Royal Society A*, 473(2207):20170502, 2017.
- [86] T. G. Molnár, W. B. Qin, T. Insperger, and G. Orosz. Application of predictor feedback to compensate time delays in connected cruise control. *IEEE Transactions on Intelligent Transportation Systems*, 19(2):545–559, 2017.
- [87] T. G. Molnár, A. W. Singletary, G. Orosz, and A. D. Ames. Safety-critical control of compartmental epidemiological models with measurement delays. *IEEE Control Systems Letters*, 5(5):1537–1542, 2020.
- [88] A. Nahidi, A. Khajepour, A. Kasaeizadeh, S.-K. Chen, and B. Litkouhi. A study on actuator delay compensation using predictive control technique with experimental verification. *Mechatronics*, 57:140–149, 2019.
- [89] M. Najafi. Stabilization techniques for large input-delay systems with uncertainty. In *Control Strategy for Time-Delay Systems*, pages 83–110. Elsevier, 2021.
- [90] S. Oh, Q. Chen, H. E. Tseng, G. Pandey, and G. Orosz. Sharable motion planning for connected automated vehicles. In *International Symposium on Advanced Vehicle Control (AVEC)*, 2022.
- [91] G. L. Oliveira, W. Burgard, and T. Brox. Efficient deep models for monocular road segmentation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4885–4891. IEEE, 2016.
- [92] G. Orosz. Connected cruise control: modelling, delay effects, and nonlinear behaviour. *Vehicle System Dynamics*, 54(8):1147–1176, 2016.
- [93] G. Orosz, R. E. Wilson, R. Szalai, and G. Stépán. Exciting traffic jams: Nonlinear phenomena behind traffic jam formation on highways. *Physical Review E*, 80(4):046205, 2009.
- [94] P. Panday, S. Samanta, N. Pal, and J. Chattopadhyay. Delay induced multiple stability switch and chaos in a predator–prey model with fear effect. *Mathematics and Computers in Simulation*, 172:134–158, 2020.

- [95] A. Pecile, N. Demo, M. Tezzele, G. Rozza, and D. Breda. Data-driven discovery of delay differential equations with discrete delays. *Journal of Computational and Applied Mathematics*, 461:116439, 2025.
- [96] J.-S. Pei, E. C. Mai, J. P. Wright, and S. F. Masri. Mapping some basic functions and operations to multilayer feedforward neural networks for modeling nonlinear dynamical systems and beyond. *Nonlinear Dynamics*, 71(1-2):371–399, 2013.
- [97] S. D. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. H. Eng, D. Rus, and M. H. Ang. Perception, planning, control, and coordination for autonomous vehicles. *Machines*, 5(1):6, 2017.
- [98] M. Plöchl and J. Edelmann. Driver models in automobile dynamics application. *Vehicle system dynamics*, 45(7-8):699–741, 2007.
- [99] J. Ploeg, N. Van De Wouw, and H. Nijmeijer. L_p string stability of cascaded systems: Application to vehicle platooning. *IEEE Transactions on Control Systems Technology*, 22(2):786–793, 2013.
- [100] M. Poliak, L. Svabova, J. Benus, and E. Demirci. Driver response time and age impact on the reaction time of drivers: a driving simulator study among professional-truck drivers. *Mathematics*, 10(9):1489, 2022.
- [101] W. B. Qin and G. Orosz. Scalable stability analysis on large connected vehicle systems subject to stochastic communication delays. *Transportation Research Part C*, 83:39–60, 2017.
- [102] W. B. Qin and G. Orosz. Experimental validation of string stability for connected vehicles subject to information delay. *IEEE Transactions on Control Systems Technology*, 28(4):1203–1217, 2019.
- [103] W. B. Qin, Y. Zhang, D. Takács, G. Stépán, and G. Orosz. Nonholonomic dynamics and control of road vehicles: moving toward automation. *Nonlinear Dynamics*, 110(3):1959–2004, 2022.
- [104] A. Rahman, J. Drgoňa, A. Tuor, and J. Strube. Neural ordinary differential equations for nonlinear system identification. In *2022 American Control Conference (ACC)*, pages 3979–3984. IEEE, 2022.
- [105] U. Rajasekaran, A. Malini, and M. Murugan. Artificial intelligence in autonomous vehicles—a survey of trends and challenges. *Artificial Intelligence for Autonomous Vehicles*, pages 1–24, 2024.
- [106] D. Rontani, A. Locquet, M. Sciamanna, D. S. Citrin, and S. Ortin. Time-delay identification in a chaotic semiconductor laser with optical feedback: A dynamical point of view. *IEEE Journal of Quantum Electronics*, 45(7):879–1891, 2009.

- [107] A. Sandoz, V. Ducret, G. A. Gottwald, G. Vilmart, and K. Perron. Sindy for delay-differential equations: application to model bacterial zinc response. *Proceedings of the Royal Society A*, 479(2269):20220556, 2023.
- [108] A. Saunders and M. Ghil. A boolean delay equation model of enso variability. *Physica D*, 160(1-2):54–78, 2001.
- [109] G. Sharma and R. Rajamani. Teleoperation enhancement for autonomous vehicles using estimation based predictive display. *IEEE Transactions on Intelligent Vehicles*, 9(3):4456–4469, 2024.
- [110] J. Sieber, K. Engelborghs, T. Luzyanina, G. Samaey, and D. Roose. DDE-BIFTOOL manual-bifurcation analysis of delay differential equations. *arXiv preprint arXiv:1406.7144*, 2014.
- [111] G. Stépán. *Retarded Dynamical Systems*. Longman, 1989.
- [112] H. T. Sykora, M. Sadeghpour, J. I. Ge, D. Bachrathy, and G. Orosz. On the moment dynamics of stochastically delayed linear control systems. *International Journal of Robust and Nonlinear Control*, 30(18):8074–8097, 2020.
- [113] T. Tél and M. Gruiz. *Chaotic Dynamics*. Cambridge University Press, 2006.
- [114] E. M. Turan and J. Jäschke. Multiple shooting for training neural differential equations on time series. *IEEE Control Systems Letters*, 6:1897–1902, 2021.
- [115] F. Van Breugel, J. N. Kutz, and B. W. Brunton. Numerical differentiation of noisy data: A unifying multi-objective optimization framework. *IEEE Access*, 8:196865–196877, 2020.
- [116] I. Vörös, G. Orosz, and D. Takács. On the global dynamics of path-following control of automated passenger vehicles. *Nonlinear Dynamics*, 111(9):8235–8252, 2023.
- [117] I. Vörös and D. Takács. Lane-keeping control of automated vehicles with feedback delay: nonlinear analysis and laboratory experiments. *European Journal of Mechanics A*, 93:104509, 2022.
- [118] I. Vörös, D. Takács, and G. Orosz. Safe lane-keeping with feedback delay: bifurcation analysis and experiments. *IEEE Control Systems Letters*, 7:1147–1152, 2023.
- [119] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3):328–339, 1989.
- [120] H. Wernecke, B. Sándor, and C. Gros. Chaos in time delay systems, an educational review. *Physics Reports*, 824:1–40, 2019.
- [121] P. Willars, E. Wittenmark, H. Ronkainen, C. Östberg, I. Johansson, J. Strand, P. Lédl, and D. Schnieders. Enabling time-critical applications over 5G with rate adaptation. *White Paper BNEW-21*, 25455, 2021.

- [122] S. Wong, L. Jiang, R. Walters, T. G. Molnár, G. Orosz, and R. Yu. Traffic forecasting using vehicle-to-vehicle communication. In *Learning for Dynamics and Control*, pages 917–929. PMLR, 2021.
- [123] Y. Wu. Reconstruction of delay differential equations via learning parameterized dictionary. *Physica D: Nonlinear Phenomena*, 446:133647, 2023.
- [124] M. Xu, Y. Wei, and J. Wei. Bifurcation analysis of rössler system with multiple delayed feedback. *Electronic Journal of Qualitative Theory of Differential Equations*, 2010(63):1–22, 2010.
- [125] D. Yanakiev, J. Eyre, and I. Kanellakopoulos. Analysis, design, and evaluation of AVCS for heavy-duty vehicles with actuator delays. *UC Berkeley: California Partners for Advanced Transportation Technology*, 1998.
- [126] Q. Yang and J. H. Yang. Hd video transmission of multi-rotor unmanned aerial vehicle based on 5G cellular communication network. *Computer Communications*, 160:688–696, 2020.
- [127] Y. Yu and S. Lee. Remote driving control with real-time video streaming over wireless networks: Design and evaluation. *IEEE Access*, 10:64920–64932, 2022.
- [128] J.-Z. Zhang, Z. Jin, Q.-X. Liu, and Z.-Y. Zhang. Analysis of a delayed sir model with nonlinear incidence rate. *Discrete Dynamics in Nature and Society*, 2008(1):636153, 2008.
- [129] L. Zhang and G. Orosz. Motif-based design for connected vehicle systems in presence of heterogeneous connectivity structures and time delays. *IEEE Transactions on Intelligent Transportation Systems*, 17(6):1638–1651, 2016.
- [130] Q.-C. Zhong. *Robust Control of Time-delay Systems*. Springer, 2006.
- [131] B. Zhou, W. Michiels, and J. Chen. Fixed-time stabilization of linear delay systems by smooth periodic delayed feedback. *IEEE Transactions on Automatic Control*, 67(2):557–573, 2021.
- [132] Q. Zhu, Y. Guo, and W. Lin. Neural delay differential equations. In *International Conference on Learning Representations*, 2021.
- [133] Q. Zhu, Y. Guo, and W. Lin. Neural delay differential equations: system reconstruction and image classification. *arXiv preprint arXiv:2304.05310*, 2023.