Hardware-in-the-Loop Plant Modeling for Autonomous Vehicle

Hannah Grady, Nicholas Nauman, and Dr. Suruz Miah

***Abstract*— This report presents the results of modeling vehicle systems and testing them using Hardware-in-the-Loop (HIL). First, data was collected for each system using a Lexus RX450H vehicle. After data collection, models were developed in order to accurately simulate the physical vehicle system. MATLAB's System Identification Toolbox was initially used to create these models. However, the transfer function models it provided weren't able to accurately represent these systems and their non-linear behaviors. After this was established, modeling was done using MATLAB's Neural Network Time Series application. Once a Neural Network model was created, it was then exported to Simulink and modified to be able to run without the use of the Deep Learning toolbox. These models gave much more accurate results when compared to the transfer function models developed using the System Identification Toolbox. (Add section about testing and results).**

***Index Terms*— Hardware-in-the-Loop, Neural Network, System Identification**

I. Introduction

AUTONOMOUS vehicles are being developed by many companies for commercial and personal use. These autonomous vehicles (see the AutonomouStuff vehicle fleet shown in Fig. 1(a)) would allow companies to continue crucial deliveries or transports of their products even if there is a shortage of drivers. In addition, autonomous vehicles have the ability to make roads safer for drivers and pedestrians alike. To develop a reliable and safe transportation system for modern world, a large body of research in the field of autonomous vehicles is being conducted in recent decades [1] [2]. Therefore, it is apparent that researchers of the autonomous vehicle community have been focusing on analysis, design, and development of different subsystems of self-driving/autonomous vehicles. Furthermore, modeling vehicle subsystems is a pre-requisite for the development of reliable controllers for these vehicles to be managed in the era of modern transportation systems in general.

Usually, there are six main subsystems of a self-driving vehicle:

1) Steering,
2) acceleration,
3) brake,
4) shift,
5) speed, and
6) speed control (cruise) subsystems.

These subsystems are to be modeled to get an accurate representation of how the vehicle should be controlled. Within the scope of this project and to expedite the modeling process, commercially available modeling tools in Mathworks' MATLAB, System Identification app in the System Identification Toolbox and Neural Network Time Series app in the Deep Learning toolbox, were used to model six subsystems for a Lexus vehicle platform, which is an experimental autonomous vehicle platform that belongs to AutonomouStuff Solutions [1]. The first subsystem we will model is the steering model, and then we will move onto the other subsystems. There are already controllers in place for the Lexus vehicle platform, but their reliability is lower than expected due to non-linear behaviors of the torque voltages required to control each subsystem. The scope of this project includes developing mathematical models for each subsystem so AutonomouStuff can implement control systems that improve the reliability of the autonomous vehicle platform. These models will be considered reliable if they can track actual vehicle subsystem behaviors with a minimum best fit percentage between 85 to 95 percent and fall within the error bounds defined for each subsystem. Once these models are developed, they will be tested using AutonomouStuff's Hardware-in-the-Loop (HIL) bench. Once there is confidence in each model, AutonomouStuff can use these models on their HIL bench to develop controllers that will remove the non-linear behaviors of each vehicle subsystem. There are two operating modes of the Lexus vehicle platform used in this project: Manual-drive and by-wire or autonomous. Fig. 1(b) depicts how the by-wire mode controls the vehicle. Each subsystem, like the steering system in this example, sends torque voltages to a motor that will control the subsystem. In Fig. 1(b), the motor would turn the pinion arms which would change the steering angle of the vehicle. This principle is applied to all other subsystems.

II. Literature Review

Authors in [3] illustrates identification of multiple-input single-output model for maximum power point tracking of

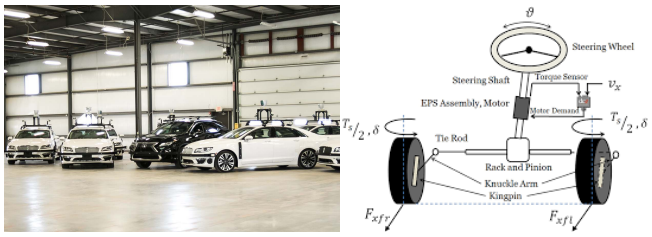
This paper was submitted for review on 5/3/22. This project was supported by Hexagon's AutonomouStuff with special thanks to Joe Buckner, Erik Guetz, Sai Chittemsetty, Matt Goben, and Travis Catton.

Hannah Grady is an electrical engineering student at Bradley University, Peoria, IL 61625 USA (e-mail: hgrady@mail.bradley.edu).

Nicholas Nauman is an electrical engineering student at Bradley University, Peoria, IL 61625 USA (e-mail: nnauman@mail.bradley.edu).

Dr. Suruz Miah is a professor for the Electrical Engineering Department, Bradley University, Peoria, IL 61625 USA, (e-mail: smiah@bradley.edu).

[1] https://autonomoustuff.com/



(a) (b)

Fig. 1: (a) Autonomous vehicle fleet in AutonomouStuff Solutions and (b) steering model setup (courtesy of AutonomouStuff).

photovoltaic system. A significant effort was conducted to model the photovoltaic system, where the two inputs were Solar irradiance and Cell temperature, and the output was DC current. To model this system, Matlab's System Identification Toolbox was used. In order to create different models, they collected and used data from an energy center in Malaysia. After generating different models, the authors ended up going with a fourth order ARX (also known as ARXQS) model because it was the most accurate, with a best fit percentage of 93.42%. The polynomial model equation for ARX is shown below.

$$y(t) + a_1 y(t-1) + ... + a_{na} y(t-n_a) = b_1 u_1(t-n_k) + ... + b_{nb} u_{nb}(t-n_k-n_b+1) + e(t),$$

$y(t)$ is the system output at time t, while $u(t)$ is the input. The noise disturbance of the system is represented by $e(t)$. The variables na, nb, and nk are the system's number of poles, amount of b parameters, and the samples before the inputs begin to affect the system's output.

III. System Identification Preliminaries

System identification is the process of developing mathematical models for a dynamic system using the measurement of input and output signals of that system. There are many components that are used to accomplish a task that they are assigned without knowing the exact behavior of the system for given input signals. Without knowing the response of this black-box, there could be unexpected consequences from a system. The goal of the system identification methodology is to get an accurate estimation of the system response to any given input. Mathworks' MATLAB has the System Identification Toolbox, where a few existing examples demonstrate the working principle of this toolbox.

A. Example 1: Dealing with Multi-Variable Systems: Identification and Analysis

The example given in [4] shows how to create an iddata object from a dataset in order to get the inputs and the outputs. The next step was to look at the impulse and step responses in order to learn more about how the inputs and outputs act. From there the state space model was estimated using the first part of the given data. This model was compared to the step responses and with the second part of the data to see if it was a good fit. The model had a best fit percentage of 83.55% for the generated voltage data, and a best fit of 39.33%

for the speed data. The frequency response of the model was estimated with spectral analysis and bode plots were also created. The tutorial explained that if the data doesn't give nice models, then it is best to try out submodels for the different channels. Two single-input single-output (SISO) models were created and compared with the existing multiple-input multiple-output (MIMO) model and the actual data. The Nyquist plots were also compared. Both SISO models performed well during these comparisons. The next step in the tutorial was to create a multiple-input single-output (MISO) model in order to get a model that more accurately reflected the generated voltage data. By creating this model and comparing it to the validation data and previous models, we saw a best fit percentage of 90.18%. The last thing the example showed was how to merge the two SISO models we created earlier.

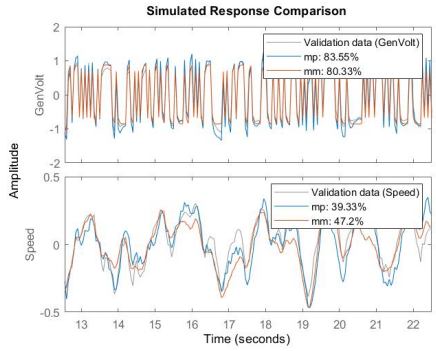
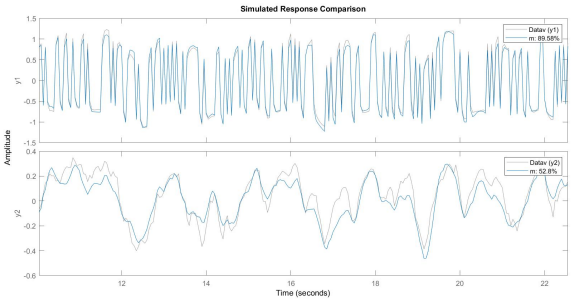


Fig. 2: Comparison of the state space model and merged SISO models with the validation data

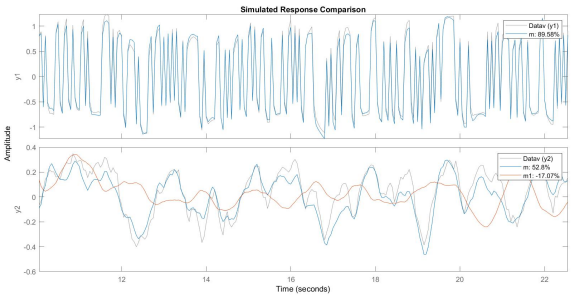
B. Example 2: Selecting Model Structures for Multivariable Systems

This example [5] discusses solutions for modeling both MISO and MIMO models using Mathworks' Matlab System Identification Toolbox. As the article discusses, MISO system models are easier to develop because all model structures used by the toolbox support models with a single output and multiple inputs. Therefore, the process for developing a model for a MISO system is importing the data as an iddata object, removing the mean from the data, and estimating the solution using any model structure available in the toolbox. The command line can be used by using the function associated with the model structure name and then using the compare function to get the best fit percentage. For MIMO systems, there are not model structures built into the System Identification Toolbox and they must be imported instead. Otherwise the process is very similar to that of a MISO system. For a MIMO system, using the compare function can be crucial. The compare function will tell which output channel is the most difficult to develop a model for, if it is possible at all. With this information, the output channel that is hardest to model should first be modeled individually because there will be less freedom in what model structures are available. The

other channels should be able to closely relate to the model for the output channel you selected.



(a)



(b)

Fig. 3: (a) State-space model of a MIMO system with a validation data comparison and (b) State-space model of a MIMO system and system-sized based state-space model with a validation data comparison

C. Example 3: Identify Linear Models Using the Command Line

This example in [6] shows how to create models for MISO systems using the command line. Before starting the model estimation process, the equilibrium values of the inputs and outputs had to be taken out. The data from each experiment also had to be separated into different iddata objects. The example showed how to estimate and compare non-parametric impulse response, transfer function, ARMA, state-space, and Box-Jenkins models with the measured experimental data. The state-space model with the five-step response prediction was the most accurate, with a best fit percentage of 85.83%.

IV. System Architecture

The overall system architecture of this project consists of six subsystems which are the steering, acceleration, brake, speed, shift, and speed control systems. Each of the six subsystems will be treated as a multiple-input single-output (MISO) system. For each subsystem, every input that is a torque voltage is actually two torque voltage signals, and thus can not be treated as a single input. Also, each subsystem will be treated as a single output system. The brake subsystem is shown to have two outputs but the behaviors of one of the outputs is already known so it will be modeled based on the other output's behavior.

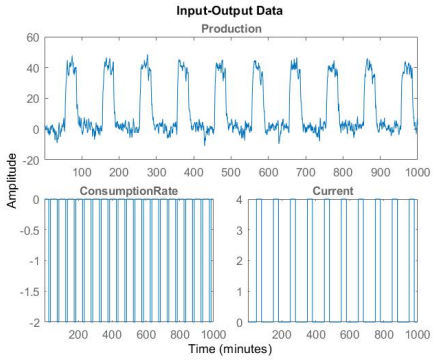


Fig. 4: Inputs and outputs of the given system

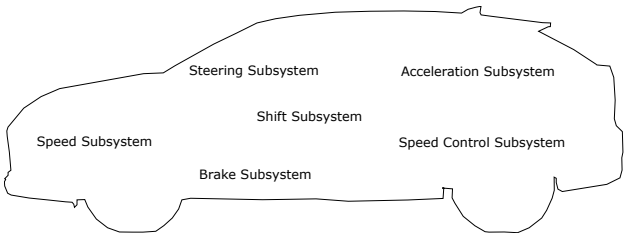


Fig. 5: Hexagon Lexus self-driving vehicle showing different subsystems

A. Steering Subsystem

The steering subsystem consists of steering, the power steering motors, ... The ultimate goal of this subsystem is to control the steering angle for the vehicle to navigate in the desired heading. Therefore, the control system is designed to produce appropriate voltages to be applied to the power steering motors for the steering orient in the target direction. The block diagram of the control system designed for this subsystem is shown in Fig. 7

B. Brake Subsystem

The brake subsystem takes the Brake Pedal Pressure Voltages, Brake Pedal Stroke Voltages, and the Brake Pedal On/Off Switch values as inputs. Using these values, it generates a new Brake Pedal Position and a boolean value called Brake Pressed. This boolean value indicates to the user whether or not the brake pedal is being pressed. The brake subsystem is another example of a multiple-input multiple-output system.

Fig. 6: AutonomouStuff Lexus RX450H vehicle.

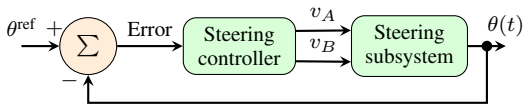


Fig. 7: Steering subsystem block diagram.

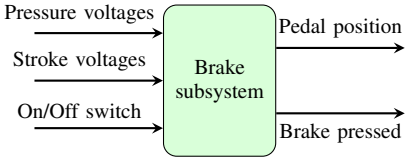


Fig. 8: Brake subsystem block diagram

C. Acceleration Subsystem

The acceleration subsystem is a multiple-input multiple-output system. Acceleration pedal voltages are sent to the subsystem. A new acceleration pedal position value is generated to better match the real-time pedal position. This is then the output of the acceleration subsystem.

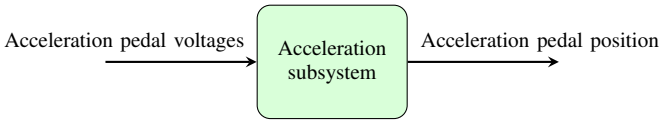


Fig. 9: Acceleration subsystem block diagram

D. Shift Subsystem

This subsystem can be classified as a single-input single-output system. The subsystem takes the desired shifter gear value from the user. Within the subsystem, the actual shifter gear changes to better reflect the desired gear. This actual shifter gear value is then the output of the shift subsystem.

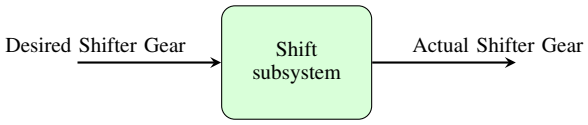


Fig. 10: Shift subsystem block diagram

E. Speed Subsystem

This speed subsystem would fall under the multiple-input single-output system. There are four inputs, the position of the acceleration pedal and the brake pedal, along with the shifter actual gear. Taking these inputs, the speed subsystem finds the vehicle speed. This vehicle speed is then the output of the system.

F. Speed Control Subsystem

The speed control subsystem is a straightforward single-input single-output system. The desired vehicle speed is set by the user and sent to the speed control subsystem. Taking this input, the subsystem calculates the new vehicle speed. This value is then sent out to the rest of the vehicle system.

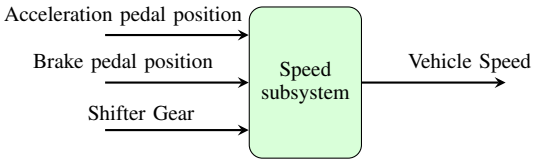


Fig. 11: Speed subsystem block diagram

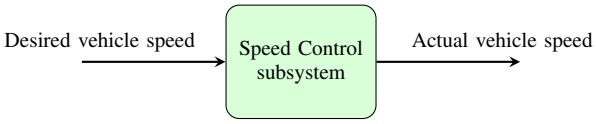


Fig. 12: Speed Control subsystem block diagram

V. Modeling Dynamic Systems using Neural Networks

In this section, we illustrate how a neural network can be used to model a dynamic system in general. Neural network is a system approximation technique that utilizes the Universal Approximation Theorem, which states that any neural network architecture can find a mathematical function $y = f(x)$ that maps inputs x to outputs y. The architecture of the neural network and the complexity of the dataset employed affect the accuracy of the function. This theorem shows that there will be a determinable function from some neural network architecture and this theorem holds for any number of inputs and outputs, like the system show in Fig. 13. The main structure of a neural network consists of neurons, input layer of neurons, one or more hidden layers, and the output layer of neurons. The neurons of the hidden layers and the output layer have processing capabilities. The actual structure of a neural network follows parallel and distributed system architecture.

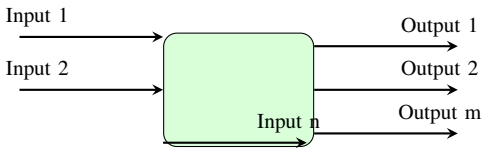


Fig. 13: Multiple Input Multiple Output Dynamic System Block Diagram

Figure 14 shows a generic actor neural network structure which represents a neural network designed for a multiple input multiple output system.

The neural network inputs are mapped to the outputs by being assigned a set of weights [2]. The approximated neural network signal can be written as:

$$\hat{\mathbf{u}}_{a,k} = \mathbf{W}_a^T \boldsymbol{\sigma}(\mathbf{e}_k), \tag{1}$$

where $\boldsymbol{\sigma} : \mathbb{R}^n \to \mathbb{R}^{\bar{n}}$ with $\bar{n} = n(n+1)/2$ and components

[2] https://towardsdatascience.com/machine-learning-for-beginners-an-introduction-to-neural-netwo

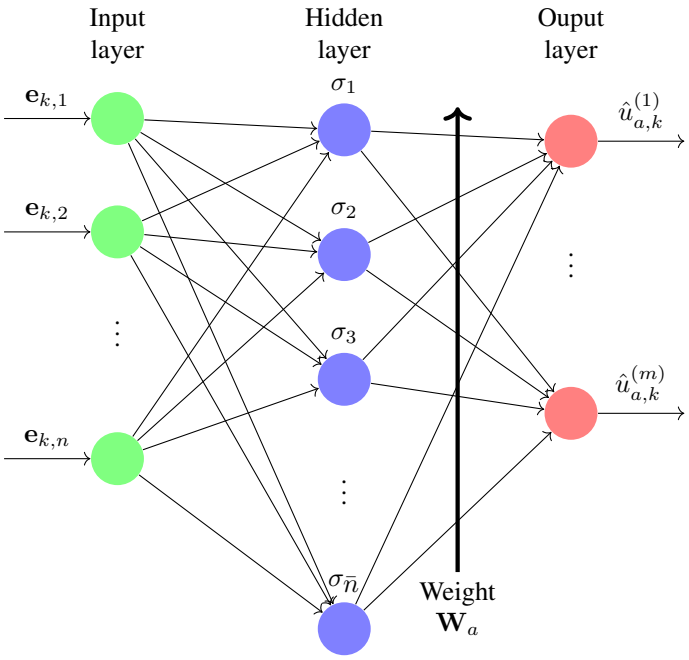


Fig. 14: Actor neural network structure for approximating control input.

of weight matrix $\mathbf{W}_a$ shown in Fig. 14 are given by

$$\mathbf{W}_a = \begin{bmatrix} w_a^{1,1} & w_a^{1,2} & \dots & w_a^{1,m} \\ w_a^{2,1} & w_a^{2,2} & \dots & w_a^{1,m} \\ \vdots & \ddots & \dots & \vdots \\ w_a^{\bar{n},1} & w_a^{\bar{n},2} & \dots & w_a^{\bar{n},m} \end{bmatrix}.$$

In compact form:

$$\mathbf{\Sigma}\mathbf{W}_a \approx \mathbf{U}^{[d]}, \qquad \text{with}$$

$\mathbb{R}^{\eta\times\bar{n}} \ni \mathbf{\Sigma} = [\mathbf{\Sigma}_0, \mathbf{\Sigma}_1, \dots, \mathbf{\Sigma}_{\eta-1}]^T$ with $\mathbf{\Sigma}_\kappa = \boldsymbol{\sigma}^T(\mathbf{e}_{k+\kappa})$ and $\mathbb{R}^{\eta\times m} \ni \mathbf{U}^{[d]} = [\mathbf{U}_0, \mathbf{U}_1, \dots, \mathbf{U}_{\eta-1}]^T$ with $\mathbf{U}_\kappa = (\mathbf{u}_{a,\kappa})^T$ for $\kappa = 0, 1, \dots, \eta-1$.

Let $\mathbf{\Sigma}\mathbf{W}_a \equiv \hat{\mathbf{U}}$, sum-of-squared error, δ_a can be written as:

$$\delta_a = \frac{1}{2}\|\mathbf{U}^{[d]} - \hat{\mathbf{U}}\|^2$$

Actor weight $\mathbf{W}_a$ can then be found minimizing the sum–of–square error δ_a defined by

$$\delta_a = \frac{1}{2}\|\mathbf{U}^{[d]} - \mathbf{\Sigma}\mathbf{W}_a\|^2 = \frac{1}{2}\mathrm{Tr}\left[(\mathbf{U}^{[d]} - \mathbf{\Sigma}\mathbf{W}_a)^T(\mathbf{U}^{[d]} - \mathbf{\Sigma}\mathbf{W}_a)\right].$$

Taking the derivative of δ_a with respect to $\mathbf{W}_a$ yields

$$\frac{\partial\delta_a}{\partial\mathbf{W}_a} = -\mathbf{\Sigma}^T\mathbf{U}^{[d]} + \mathbf{\Sigma}^T\mathbf{\Sigma}\mathbf{W}_a \tag{2}$$

Setting $\frac{\partial\delta_a}{\partial\mathbf{W}_a}$ to zero yeilds

$$\mathbf{W}_a = \left(\mathbf{\Sigma}^T\mathbf{\Sigma}\right)^T\mathbf{\Sigma}^T\mathbf{U}^{[d]}.$$

The neural network is trained by altering the weights to minimize the least square error by utilizing the gradient descent approach which is defined as:

$$\mathbf{W}_a^{(r+1)} = \mathbf{w}_a^{(r)} - \ell_a\frac{\partial\delta_a}{\partial\mathbf{W}_a} = \mathbf{W}_a^{(r)} - \ell_a\left(-\mathbf{\Sigma}^T\mathbf{U}^{[d]} + \mathbf{\Sigma}^T\mathbf{\Sigma}\mathbf{W}_a^{(r)}\right)$$
$$= \mathbf{w}_a^{(r)} - \ell_a\mathbf{\Sigma}^T\left(\mathbf{\Sigma}\mathbf{W}_a^{(r)} - \mathbf{U}^{[d]}\right) \tag{3}$$

The neural network is validated by passing another subset of data that was not used for training purposes and comparing the output results of the neural network with the expected output results from the data set. An acceptable neural network model will meet the desired error requirements [3].

VI. Modeling Vehicle Subsystems

To start the project, we researched methods of plant modeling for autonomous vehicles and more generically, nonlinear systems. From our research, we found a few methods of modeling that we employed to discover the most accurate representation of each vehicle subsystem. The first method that we tried was developing different system types using Mathworks' MATLAB System Identification Toolbox. Using data we collected from the Lexus vehcile platform at AutonomouStuff,this toolbox allowed us to add input and output data for each vehicle subsystem. With this data, we used one subset to train the system and another subset for validation. Using this toolbox, we tried developing state-space models, NARX models, and transfer function models. We also used Mathworks' MATLAB Neural Network Time Series toolbox to develop a model of each vehicle subsystem using neural network modeling making use of the Bayesian Regularization training method. The following requirements were used to evaluate the effectiveness of our models to determine which plant model should be presented to AutonomouStuff.

A. System Requirements

In order to accurately model each vehicle subsystem so that safe, reliable controllers can be developed for autonomous control, the following criteria will be met:

- Each vehicle subsystem will be able to track nonlinearities depicted in Fig. 15 associated with small changes in the output of each subsystem
- Each vehicle subsystem plant model will track any desired inputs within the specified error bounds
  - Steering Angle within 5 degrees
  - Pedal Positions within 5%
- Each vehicle subsystem will be modeled independently from other vehicle subsystems

The vehicle plant model will fulfill the requirements listed below:

- The resulting plant model will consist of accurate subsystem models, as defined above
- The subsystems can be used to create a HIL testbench
- The subsystems can handle very small changes in their output accurately

[3] https://medium.com/autonomous-agents/bayesian-regularization-for-neuralnetworks-2f2d34f03adc

All of the subsystems have nonlinear behaviors when there are small changes in the output of the subsystem. The steering subsystem, for example, should behave in a smooth, continuous manner. However, AutonomouStuff observed that when trying to implement features such as lane tracking for the Lexus vehicle platform, that when small changes in the steering angle (less than five degrees) occurred, the torque voltages would momentarily stall and then suddenly change causing a more drastic change. This behavior, depicted in Fig. 15, made it very challenging to complete features like lane tracking. These models will aid in the development of controllers that will remove the nonlinearities allowing features like lane tracking to be implemented in a safer, smoother manner.

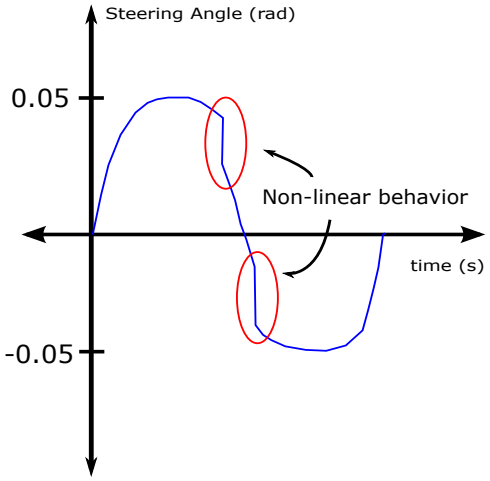


Fig. 15: Steering Non-linear Behavior

B. Steering System Modeling

To develop a model of the steering system, we employed multiple methods. The first method that we tried was generating a NARX model and a transfer function model, making use of the System Identification Toolbox in MATLAB. The NARX model we developed was not accurate and did not track the output of the steering system data. The transfer function model we developed was more reliable tracking the output of the steering system data, but was unable to meet the error requirements, as there were sudden changes that forced high error peaks. Then we employed a neural network model, making use of the Neural Network Time Series Toolbox in MATLAB. With the neural network model, the model was able to reliably and accurately track the output of the steering system data and stayed within the error requirements.

C. Acceleration System Modeling

To develop a model of the acceleration system, we employed multiple methods. The first method that we tried was generating a NARX model and a transfer function model, making use of the System Identification Toolbox in MATLAB. The NARX model we developed was not accurate and did not track the output of the acceleration system data. The transfer function model we developed was more reliable tracking the output of the acceleration system data, but was unable to meet the error requirements, as there were sudden changes that forced high error peaks. Then we employed a neural network model, making use of the Neural Network Time Series Toolbox in MATLAB. With the neural network model, the model was able to reliably and accurately track the output of the acceleration system data and stayed within the error requirements.

D. Brake System Modeling

To create a brake system model, we first tried generating transfer function models using MATLAB's System Identification Toolbox. After generating multiple transfer function models and trying many different data log combinations, we still couldn't find a model that was completely accurate across all combinations and that was below the error bound of 5% for all combinations. Once this became clear Neural Network models were created using the Neural Network Time Series Toolbox provided by MATLAB to see if they could create accurate brake models. Using this we were able to create a model that accurately tracked the data we collected from the brake system and that stayed within the 5% error bound.

VII. Validation and Testing

A. Experimental Setup

We traveled to AutonomouStuff in order to collect data from the steering, acceleration, and braking subsystems in an autonomous vehicle. The data we collected will be used to generate and then verify our models. We collected data on the Lexus RX450H vehicle platform shown in Fig. 6.
The following is the hardware components required to collect the data:

- Laptop
- PACMod ECU
- CANCase
- CAN bus

The laptop is used to pass commands or log data, such as steering angle or acceleration or brake pedal position. This data is sent or received using the CANCase and CAN bus. These are connected to the AutonomouStuff designed PACMod ECU, which sends torque voltages to the desired vehicle subsystem allowing the laptop to either control the desired vehicle system or log data. The Vector CANAlyzer software is installed on the laptop and is used to parse the collected data that is sent from the CANCase. This is how we were able to collect logs of data that we would use to develop models of the autonomous vehicle subsystems.

Each subsystem that we are modeling is set up in a similar manner. In manual mode, the torque voltages that control each subsystem are sent by the vehicle's electronic control unit (ECU). In order to control the vehicle autonomously, the vehicle subsystem switches to by-wire mode. In by-wire mode, the torque voltages from the vehicle's ECU are discarded by open-circuiting the motors that control each subsystem. Instead, the PACMod ECU built by

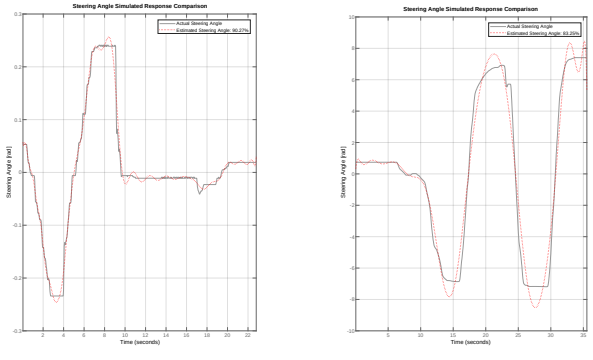
AutonomouStuff sends the torque voltages to the motor using relays. In Fig. 16, the experimental setup for data collection is shown. The laptop is used to collect the data from the desired vehicle subsystem by the use of Vector's CANAlyzer software. The CAN Case collects data from the PAC Mod and ECU and sends the data using a CAN bus to the laptop which is then parsed and displayed through the use of CANAlyzer. The ECU and PAC Mod are not shown in Fig. 16 as they are fixed behind panels of the vehicle.

Fig. 16: Autonomous Vehicle Data Collection Setup

Using the data we collected for the steering, acceleration, and braking subsystems, we initially separated the data so we could analyze the by-wire and manual modes individually. This effort was made to identify if the models we developed could be used interchangeably regardless of what mode the autonomous vehicle subsystems were operating in. After analyzing the models we developed, we determined that they are not interchangeable as there were differences in the models. When the vehicle subsystem is in by-wire mode, the controller AutonomouStuff developed is generating the torque voltages that are applied to each subsystem motor. As a result, we decided the most accurate model of each subsystem would be developed using the data when the subsystems are in manual mode. For reference, the differences in the models created for by-wire and manual mode are depicted in VII for the steering and acceleration subsystems. For all other models we develop, we will only use manual mode data and will not create models for by-wire mode.

B. Transfer Function Modeling

1) Steering Subsystem: In Fig. 17(a), the output steering angle is plotted versus time when the vehicle is in by-wire mode. The figure shows the output steering angle from some data collected to depict the steering system behavior plotted with the behavior of the estimated model. The best fit percentage for this model is 85.54%. Likewise, Fig. 17(b) depicts the output steering angle plotted versus time when the vehicle is in manual mode. The best fit percentage for this model is 90.27%.



(a) Output of Estimated By-Wire System Model (b) Output of Estimated Manual System Model

Fig. 17: Steering System Estimated Steering Angle Comparison

The model of the by-wire steering system is a twentieth order transfer function. The twentieth order transfer function is the best proposed estimation considering the system costs associated with a higher order transfer function while also obtaining an acceptable best fit percentage. Table **??** shows the coefficients of the twentieth order transfer function for the output steering angle with respect to the input torque voltage A signal.

TABLE I: By-Wire Mode Steering Transfer Function Torque Voltage A Coefficient Table

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{18}	a_{19}	a_{20}
1.949E-16	1.455E-14	8.797E-13	2.304E-11	9.041E-10	9.01E-9	2.714E-7	9.241E-7	1.946E-5	3.889E-5	0.0005721	0.0008035	0.008695	0.008922	0.07499	0.05422	0.3631	0.1693	0.9394	0.2118	1

b_0	b_1	b_2	b_3	b_4	b_5
3.44E-16	-2.203E-15	1.24E-13	5.975E-13	3.001E-12	5.23E-12

Table **??** shows the coefficients of the twentieth order transfer function for the output steering angle with respect to the input torque voltage B signal.

TABLE II: By-Wire Mode Steering Transfer Function Torque Voltage B Coefficient Table

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{18}	a_{19}	a_{20}
2.92E-15	2.758E-13	1.761E-11	4.228E-10	1.422E-8	1.561E-7	1.501E-6	8.426E-6	5.981E-5	0.0002016	0.001226	0.002649	0.01456	0.02044	0.104	0.09228	0.4417	0.2257	1.025	0.2305	1

b_0	b_1	b_2	b_3	b_4	b_5
-6.645E-15	2.161E-14	-2.605E-14	-4.744E-13	-1.813E-13	-4.858E-12

The manual steering system is modeled by a twentieth order transfer function. After considering the system costs that come with a higher order transfer function and the need to achieve a sufficient best fit percentage, it is clear that a twentieth order transfer function is the best estimation of the system. Table III shows the coefficients of the twentieth order transfer function for the output steering angle with respect to the input torque voltage A signal.
Table IV shows the coefficients of the twentieth order transfer function for the output steering angle with respect to the input torque voltage B signal.

TABLE III: Manual mode steering transfer function torque voltage A coefficient table.

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{18}	a_{19}	a_{20}
1.224E5	4.227E5	2.202E6	3.54E6	7.98E6	7.62E5	1.071E7	6.745E6	6.85E6	3.006E6	2.365E6	7.37E5	4.669E5	1.021E5	5.319E4	7752	3353	287.5	102.4	3.653	1

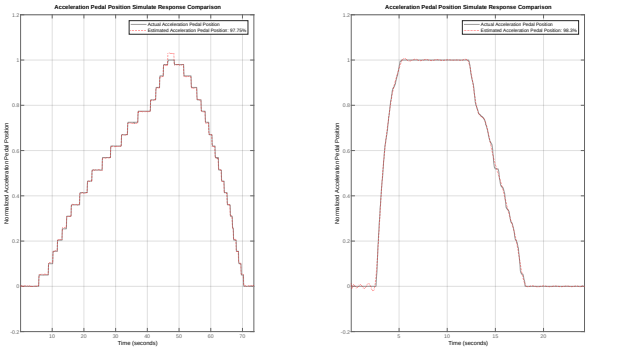
b_0	b_1	b_2	b_3	b_4	b_5
6.516E4	-2.944E5	2.328E5	-2.487E5	8.25E4	-3.038E4

TABLE IV: Manual Mode Steering Transfer Function Torque Voltage B Coefficient Table

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{18}	a_{19}	a_{20}
4.687E4	7.037E5	2.624E6	6.133E6	1.197E7	1.487E7	1.887E7	1.446E7	1.314E7	7.01E6	4.79E6	1.886E6	9.836E5	2.937E5	1.149E5	2.617E4	7205	1232	200	23.54	1

b_0	b_1	b_2	b_3	b_4	b_5
1.209E4	1.855E5	-2.801E5	4.72E5	-3.976E4	7.693E4

2) Acceleration System: In Figure 18(a), the output acceleration pedal position is plotted versus time when the vehicle is in by-wire mode. The figure shows the output acceleration pedal position from some data collected to depict the acceleration system behavior plotted with the behavior of the estimated model. The best fit percentage for this model is 97.69%. Likewise, Figure 18(b) depicts the output acceleration pedal position plotted versus time when the vehicle is in manual mode. The best fit percentage for this model is 98.3%.



(a) Output of Estimated By-Wire System Model (b) Output of Estimated Manual System Model

Fig. 18: Acceleration System Estimated Pedal Position Comparison

The model of the by-wire acceleration system is a fourth order ARX model. The fourth order ARX model defined below represents the output acceleration pedal position, A(z), based on input torque voltage A, $B_1(z)$, and input torque voltage B, $B_2(z)$.

$$A(z)y(t) = B_1(z)u_1(t) + B_2(z)u_2(t) + e(t),$$

where na = 4, nb = 4, nk = 0, and,

$$A(z) = 1 - 1.018z^{-1} - 0.002901z^{-2} + 0.4631z^{-3} - 0.2038z^{-4}$$
$$B_1(z) = -0.0207 - 0.01912z^{-1} - 0.02159z^{-2} - 0.03307z^{-3}$$
$$B_2(z) = 0.02017 + 0.01833z^{-1} + 0.09461z^{-2} + 0.05683z^{-3}$$

The model of the manual acceleration system is a twenty-fourth order transfer function. The twenty-fourth order transfer function is the best proposed estimation considering the system costs associated with a higher order transfer function while also obtaining an acceptable best fit percentage. Table V shows the coefficients of the twenty-fourth order transfer function for the output acceleration pedal position with respect to the input torque voltage A signal.

TABLE V: Manual Mode Acceleration Transfer Function Torque Voltage A Coefficient Table

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{18}	a_{19}	a_{20}	a_{21}	a_{22}	a_{23}	a_{24}
2.307E7	8.368E7	9.137E8	1.992E9	3.753E9	5.454E9	5.548E9	5.869E9	3.453E9	3.161E9	1.54E9	9.654E8	3.556E8	1.789E8	5.082E7	2.071E7	4.569E6	1.498E6	2.583E5	6.532E4	8443	1559	147.3	15.5	1

b_0	b_1	b_2	b_3	b_4	b_5	b_6
-5.741E6	-2.644E7	-2.797E6	-1.304E7	-3.203E6	-1.159E6	-5.519E5

Table VI shows the coefficients of the twenty-fourth order transfer function for the output acceleration pedal position with respect to the input torque voltage B signal.

TABLE VI: Manual Mode Acceleration Transfer Function Torque Voltage B Coefficient Table

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{18}	a_{19}	a_{20}	a_{21}	a_{22}	a_{23}	a_{24}
5.224E6	4.205E7	1.429E8	3.233E8	5.444E8	7.326E8	8.247E8	7.402E8	6.301E8	3.990E8	2.731E8	1.266E8	7.223E7	2.48E7	1.216E7	5.084E6	1.324E6	2.422E5	9.264E4	1.159E4	4003	307.4	96.87	3.451	1

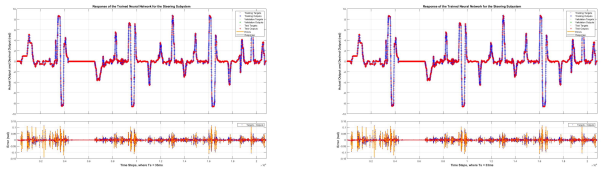
b_0	b_1	b_2	b_3	b_4	b_5	b_6
-4.159E6	1.851E6	-4.457E6	9.043E5	-7.45E5	1.865E5	-1.545E4

3) Brake System: While some by-wire models were created for the brake system, the models created using manual data were the only ones considered once we decided to use only manual data. It was found that a transfer function of 28 poles and 12 zeros gave an acceptable best fit percentage for most of the data log combinations that were tried. However, it was only within the 5% error bound for one of the data log combinations. Another problem was that the brake system required four transfer functions, one for each input. Once these transfer functions were exported, there was uncertainty on how to connect them in order to produce the output value. As a result, we ended up switching to using Neural Networks to create a model for the brake system.

C. Neural Network Modeling

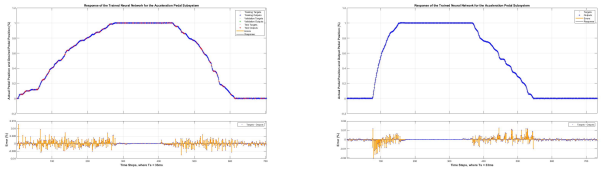
1) Steering System: To develop the model of the steering system using a neural network architecture, we used one data subset to train the model and then used an additional subset of data to verify the model worked. The neural network model used a feedforward network that had one hidden layer containing three neurons. The model was also trained with twelve delay units to achieve the required accuracy, meaning that the first twelve output samples of the model should be ignored.

2) Acceleration System: To develop the model of the acceleration system using a neural network architecture, we used one data subset to train the model and then used an additional subset of data to verify the model worked. The neural network model used a feedforward network that had one hidden layer containing three neurons. The model was also trained with three delay units to achieve the required accuracy, meaning that the first three output samples of the model should be ignored.



(a) Model Output Tracking and Error Plots for Data Subset 1 (b) Model Output Tracking and Error Plots for Data Subset 2

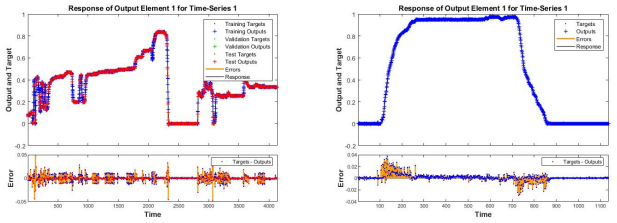
Fig. 19: Steering System Training Plots



(a) Model Output Tracking and Error Plots for Data Subset 1 (b) Model Output Tracking and Error Plots for Data Subset 2

Fig. 20: Acceleration System Training Plots

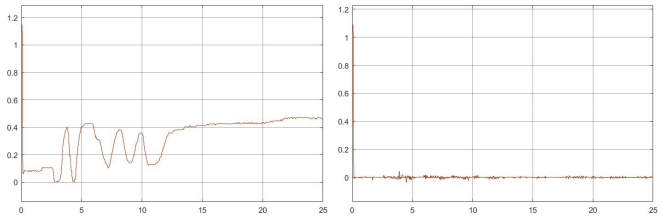
3) Brake System: The brake system was modeled using only the manual log data. It was trained using the data from the first brake log, and the result and error plots can be seen in Figure 21(a). The plots show that the model is able to predict the outputs relatively accurately, along with keeping the error below 5%. In order to further verify the accuracy of the model, it was tested using data from the second brake log. The results from that test, shown in Figure 21(b), also show that the model can accurately predict the output values and do this within the accepted error bound of 5%.



(a) Model Output Tracking and Error Plots (b) Model Output Tracking and Error Plots for Log Two Data

Fig. 21: Brake System Training Plots

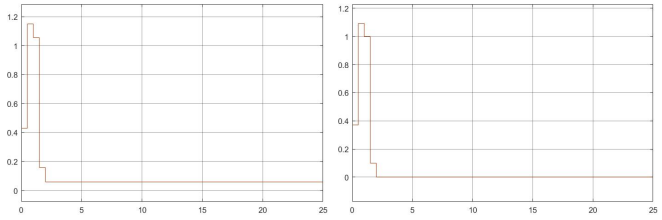
Once the model was created, it was exported to Simulink and tested again. One test involved using To Workspace blocks to access the pressure voltages, position voltages, and desired pedal position values in the first brake log. These values were used as inputs for the model, and the actual pedal position along with the error between the desired and actual pedal positions were then plotted. The actual pedal position plot can be seen in Figure 22(a). The resulting error plot between the desired pedal position and the model generated position can be found in Figure 22(b). Despite an initial spike within the first few seconds of the graph, the error plot shows that the difference between the pedal positions generated by the system and the desired pedal positions is never more than 5%.



(a) Actual Pedal Position (b) Error Between the Actual and Desired Pedal Position

Fig. 22: Actual Pedal Position and Error Plots Using the From Workspace Blocks

The second test used constant blocks to represent the voltages and desired pedal position at a specific time instant in the first brake log. The error and the actual pedal position were then plotted again. The pedal position plot is shown in Figure 23(a), and the error plot is represented by Figure 23(b). As the plots show, the model is able to produce the desired pedal position and the error falls to 0% after a spike at the start.

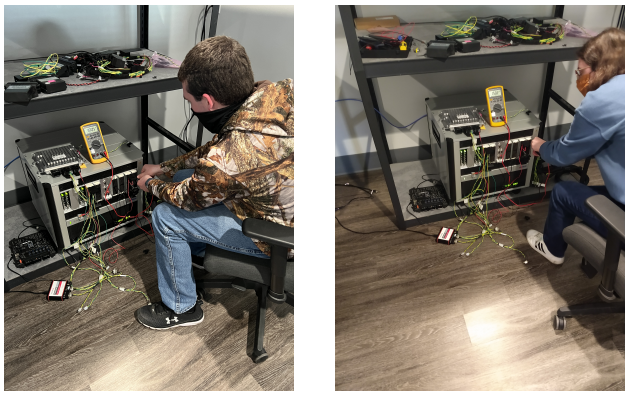


(a) Actual Pedal Position (b) Error Between the Actual and Desired Pedal Position

Fig. 23: Actual Pedal Position and Error Plots Using the Constant Blocks

D. Discussion

For each vehicle subsystem, the results show that the neural network models were the most accurate models. Other models that we developed, such as the transfer function models, were relatively accurate when tracking the data, but those models were consistently falling outside of the error bounds that were defined at the start of the project. The neural network models, however, track the data more accurately and reduced the error significantly so that all error requirements were met. We first proved that the neural network models that we developed were accurate using subsets of data as inputs to a Simulink model. Then we visited AutonomouStuff to generate the models on their HIL bench, as shown in Fig. 25, using Control Desk to interface. Due to time constraints, AutonomouStuff was not able to get the proper hardware for this project connected to the HIL bench but we were able to test our models in an open-loop configuration and measure outputs to show that the correct voltages would be fed to any hardware that would be connected, shown in Fig. 24. One downside to using neural networks as the plant model is it causes some complexities when trying to include these models into the HIL bench software for use in developing and testing controllers.



(a) (b)

Fig. 24: Verifying voltage measurements on the HIL Bench

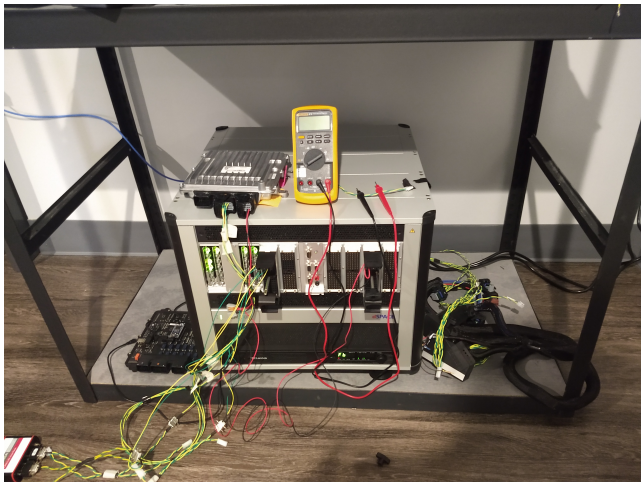


Fig. 25: AutonomouStuff HIL Bench Configuration

VIII. Conclusion and Future Work

Our initial approach was to model the subsystems using the System Identification Toolbox from MATLAB, but it produced unsatisfactory results. Switching to the Neural Network Time Series app allowed us to train models using the data we collected. These models that were developed met the predefined accuracy requirements, and so were tested using an open loop setup. In the future if AutonomouStuff would like to model other vehicle subsystems, we would recommend modeling them using Neural Networks. Creating the Neural Networks and exporting the models to Simulink was easy to do and produced accurate results.

For this project, further testing should be completed for these subsystem models. Since time and availability constraints prevented us from testing the models using Hardware-in-the-Loop, this would be a good place to start. Further testing is important to make sure the developed models are accurate and compatible with testing using hardware. Depending on what AutonomouStuff decides, future engineers working on this project could start developing controllers based off of these models. Currently, AutonomouStuff develops their own controllers for their vehicle subsystems, but they could allow students to design some as part of this senior project.

References

[1] M. Liu, G. Li, and B. Liu, "Real-time knowledge discovery from public data of internet to improve decision-making of autonomous vehicles," in *2017 Second International Conference on Reliability Systems Engineering (ICRSE)*, 2017, pp. 1–3.

[2] B. Prasad, Q. Y. Huang, and J.-J. Tang, "Development of a prototype ev autonomous vehicle for systematic research," in *2020 International Computer Symposium (ICS)*, 2020, pp. 459–461.

[3] M. Hussain, A. Omar, and A. Samat, "Identification of multiple input-single output (miso) model for mppt of photovoltaic system," in *2011 IEEE International Conference on Control System, Computing and Engineering*, 2011, pp. 49–53.

[4] "Dealing with multi-variable systems: Identification and analysis," https://www.mathworks.com/help/ident/ug/dealing-with-multi-variable-systems-identification-and-analysis.html.

[5] L. Ljung, "Selecting model structures for multivariable systems," http://www.ece.northwestern.edu/local-apps/matlabhelp/toolbox/ident/ch3tut44.html, 2021.

[6] "Identify linear models using the command line," https://www.mathworks.com/help/ident/gs/identify-linear-models-using-the-command-line.html.