Online model identification and runtime adaptation of a closed-loop control system

Ma, Chaolun

Award date:
2021

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Electronic Systems Research Group

Online model identification and runtime adaptation of a closed-loop control system

Master's Thesis

Chaolun Ma

Supervisors:
Assistant Professor Dr. Dip Goswami
PhD Candidate Sajid Mohamed

1.0 version

Eindhoven, January 2021

Abstract

In modern industry, an industrial system is composed of lots of components or subsystems. Because the manufacturers produce these subsystems according to a generic standard or design guideline without full knowledge about the target system, the control model and system dynamics will only follow a generic pattern with typical parameters, which will cause parameter changes on the real system. Also, the system dynamics may be affected by thermal effects and mechanical and electrical ageing, which cannot be directly measured and is challenging to account for in the generic control model. These factors will result in the performance deterioration of the control system under different scenarios. Therefore, it is necessary to provide a method for tuning and adapting the subsystems at runtime to improve the system performance.

In this report, we propose an online model identification and adaptation of a control system at runtime. The NARMA-L2 controller, a kind of neural network controller, is used for identification and controller design. We show that the NARMA-L2 controller can achieve high training accuracy and realize the control requirement in both linear and nonlinear systems. We validate our approach using the cruise control system designed in Webots. Based on the NARMA-L2 controller, online model identification method and an adaptive controller are designed to adapt to parameters' changes. Also, for validation of the controller, we develop a MATLAB toolbox for SiL and HiL simulation to validate the controller, which supports code generation for different parameters, integration of widgets like SDF3 dataflow analysis, and switch between different implementation platforms and controllers. Our evaluation shows that adapting the controller at runtime has better performance than the traditional PID controller.

Preface

Before the start of the thesis report, I would like to express my thankfulness to my supervisor Dr. Goswami who gave me the opportunity of this project, guided me with great patience, and gave me valuable advice and support during my graduation. Also, I would like to thank my tutor Mr. Mohamed, who helped me a lot and gave me much inspiration during the project. I also want to thank my parents who support me during my studies.

Contents

List of Figures

List of Tables

Acronyms

API Application Programming Interface. 7, 29

HiL Hardware-in-the-loop. 2

IBC Image-Based Control. 7

LQG Linear–Quadratic–Gaussian. 22

LQR Linear–Quadratic Regulator. 22

LS-SVM Least-Square Support Vector Machine. 4

MIMO Multi Input Multi Output. 12, 13

MSE Mean Square Error. 17

NARMA Nonlinear Auto-Regressive Moving-Average. 4, 12, 13

NARMA-L2 Nonlinear Auto-Regressive Moving-Average Level-2. 4, 12, 13, 17, 31, 33, 35, 40

RBFNN Radial Basis Function Neural Network. 4, 5

SGD Stochastic Gradient Descent. 17

SiL Software-in-the-loop. 2, 3

SISO Single Input Single Output. 12

SVM Support Vector Machine. 4

SVR Support Vector Regression. 4, 10

XiL X-in-the-loop, including software and hardware. 3, 4, 7, 8, 40

Chapter 1

Introduction

1.1 Motivation

With the development of industrial control, it becomes essential to precisely model the dynamic system for analysis and controller design. However, in most cases, the system is unknown to outside, time-varying, too complicated or fuzzy. It cannot be represented or modelled directly by traditional pre-defined models. Also, the parameters recorded in testing will be changed on real systems due to different environment or disturbance from other subsystems. For instance, an electric motor is affected by thermal effects [4] or mechanical and electrical ageing [5], which are both difficult to measure and affects system dynamics. For example, a car in cruise control may experience slopes, change of road surfaces like rain and snow, and loads on the vehicle. To not degrade the controller's performance when the system dynamics and parameters are varying with different situations, an online identification method and adaptive controller design are required.

For analyzing the system in detail, especially to reproduce the problematic situations on physical systems, simulators and suitable frameworks are used. Using the simulation framework, the designer can evaluate the controller's performance under the desired situations.

1.2 Background introduction

1.2.1 System identification and controller design

System identification is an effective method to identify control systems in industrial practice described above. It aims at finding a possible model that approximate the behaviours of the unknown system based on the input and output data, as is described in [6]. The model obtained from identification can then be applied in further analysis and controller design. Figure 1.1 shows the general procedure of system identification, including appropriate model selection, parameter identification and adjustment based on input and output data, and verification using error between the identified and original system.

System identification can be used in various aspects of industrial applications. As is summarized in [7], the use of identification includes simulation and prediction, which derives from the past input to the output of next state, to predict the state of a complex system or generate a simulated system like a software emulated sensor. It also includes control and optimization, which requires the control input to adjust frequently based on the reference output and the change of model parameters to give the optimal control input and optimize the control system's performance.

1.2.2 Online and offline identification

Identification methods can be divided into two categories, namely, online and offline. Offline identification means that the data must be collected before training. Online identification can

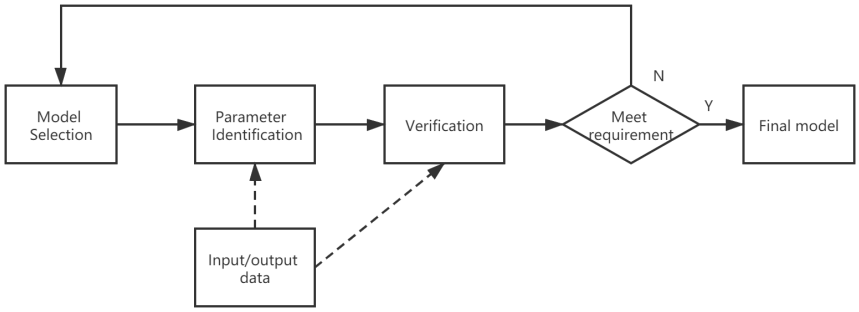


Figure 1.1: General procedure of system identification

process the data one by one and train the model whilst the model is running [8].

Online and offline models have their advantages and disadvantages. As is summarized in [9], an offline model can handle extensive data as computation time is not critical to this structure. They are robust to small variations but fail to adapt to more extensive changes in the system. The online model adapts quickly to variations in the non-linear behaviour but is less accurate because of small sets of training data given as batches.

1.2.3 Adaptation of model

The traditional controller, including the controller designed with system identification described above, can keep its functionalities to some extent when the parameters change. As the changes become extensive, the performance will be reduced until the controller diverges. Therefore, the controller requires adaptation at runtime based on the condition and parameters of the controlled system to enhance the control performance and keep converged in different types of parameter changes.

According to [10], adaptation can be divided into two categories. The direct adaptive control can directly adjust the controller's parameters based on the reference model and errors. The indirect adaptive control first identifies the model with system identification methods and adjust the parameters based on the identified model. The controller adaptation based on system identification is also called model identification adaptive controller, which will also be addressed in this report.

The process of model identification adaptive controller is shown in Figure 1.2. Figure 1.2 shows that the system identification technique identifies the model on runtime in the model identification adaptive controller. The result is used to update the controller using some kinds of update strategies.

1.2.4 XiL simulation

In a real system and its control loop, the physical system is directly connected to and controlled by a control system run on hardware platforms. However, it is difficult to use real physical systems for simulation of the controller because of the cost and the fact that the real system cannot perform extreme situations like extreme temperature. Therefore, when simulating a system, some parts of the real system need to be replaced with the simulated part to observe the behaviour of each part [11]. These kinds of simulation methods are called *XiL simulation* in general."XiL" contains two categories of simulation, Software-in-the-loop (SiL) and Hardware-in-the-loop (HiL). The "X" represents where the controller codes are executed, on controller hardware or on normal software platforms like PC.

Hardware-in-the-loop simulations mean that the evaluated controllers are run on the target hardware and interact with the simulator real-time with the same protocols as the ones on the

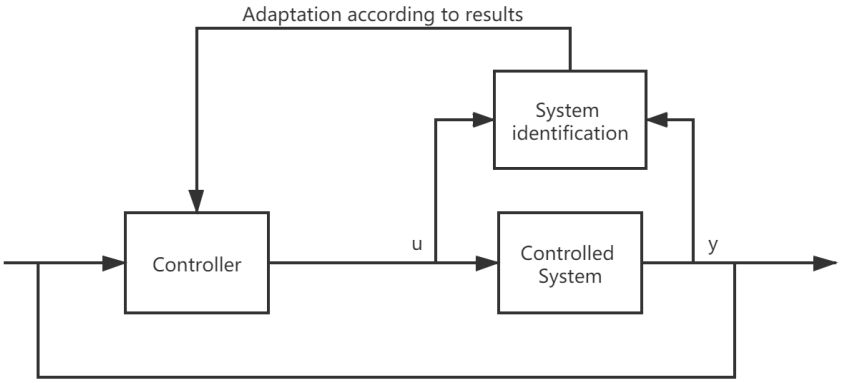


Figure 1.2: Procedure of model identification adaptive controller

actual robot sensors and actuators, as is summarized in [12]. Before writing the controller code to the hardware, the code needs to be run on PC simulators to test if its basic functions and logic meet the requirement of the control model. This is called Software-in-the-loop simulation [11].

1.2.5 Simulation environment

In order to build the XiL simulation environment introduced in Chapter 1.2.4, a robot simulator is required to model the physical system in reality for simulation and measurement. The simulators currently used in robotics and control simulation includes Gazebo[1] [13], V-REP[2] [14] and Webots[3] [15].

Most of these simulators allow users to define customized controller to control the movement of robot parts, either inside the simulator or using one of the programming language and environment provided by operating systems and third-party libraries. However, changing control parameters of the external controller requires direct modification of codes, which is vulnerable to mistakes and hard to maintain. It is not user-friendly to users who are not familiar with the language used for the controller.

1.3 Report structure

The report consists of nine chapters. Chapter 1 introduces the motivation and background introduction of the report. Chapter 2 discusses related work. Chapter 3 states the problems to be solved in this project and describes the contribution of the project. Chapter 4 describes the overall design approach. Chapter 5 introduces the controller's theoretical background and mathematical derivations, and Chapter 6 talks about the implementation and verification of the selected controller. Chapter 7 introduces the design of the XiL simulation framework. Chapter 8 applies the identification and controller design on the chosen system and discuss the adaption strategies for parameter changes. Finally, Chapter 9 concludes the whole project and describes future work after the project.

[1] http://gazebosim.org/
[2] https://www.coppeliarobotics.com/
[3] https://www.cyberbotics.com/

Chapter 2

Related Work

This chapter introduces the related work about the thesis. Chapter 2.1 discusses standard system identification approaches, including traditional methods, machine learning methods, and discussion about online and offline identification methods. Chapter 2.2 discusses common approaches to deal with parameter changes and adaptation of the system. Chapter 2.3 discusses the simulators, their comparison and related works about XiL simulation using them.

2.1 System identification methods

2.1.1 Classical identification

System identification has been a traditional topic since L.A.Zadeh raised the concept in [6]. Before the introduction of neural network or statistical learning, several mathematical methods are used to approximate the model to be identified into a chosen model.

Least square methods are the most traditional and popular statistical methods for linear regression [16]. It is widely used to identify linear models, but it does not apply for nonlinear models. Kalman filter is optimal in approximating linear models and is commonly used in linear approximation [17]. The nonlinear variations of Kalman filter, like extended Kalman filter in [18] and unscented Kalman Filter in [19] can deal with special cases of nonlinear systems by linearization. These methods mostly focus on mathematical solutions of system functions.

2.1.2 Identification with machine learning methods

In order to solve the problem of nonlinear approximation, which is proved difficult using traditional identification, Levin et al. introduced the use of the neural network in system identification in [20]. In [21] and [22], Narendra et al. gave a brief introduction about the method and raised the complete process and system architecture to identify dynamic time-varying systems. Also in [23], Narendra et al. applied Nonlinear Auto-Regressive Moving-Average (NARMA) model and its approximation NARMA-L2 to predict the output with the time-delayed past inputs and outputs, which sharply simplified the network structure and controller design.

Apart from using a traditional neural network like multi-layer perceptron and back-propagation, neural networks with other models and algorithms are raised to solve the problem. Support Vector Machine (SVM) and its variation Least-Square Support Vector Machine (LS-SVM) proposed by Vapnik et al. [24] are usually used in classification problems. SVMs can be modified to construct a hyperplane in high-dimensional space to transform nonlinear problems into linear optimization problems in higher-dimensional space. They can be applied to system identification and used in nonlinear regression, often referred to as Support Vector Regression (SVR). Lu et al. presented an algorithm for time-varying nonlinear systems using Radial Basis Function Neural Network (RBFNN) [25], which can achieve higher speed and suitable for the time-varying nonlinear system. Lin et al. in [1] proposed a fuzzy network using type-2 fuzzy sets to model uncertainties associated

with information and used for online system identification while eliminating the uncertainties in the typical fuzzy model. The process of fuzzy network identification is shown in Figure 2.1, which is cited from [1].

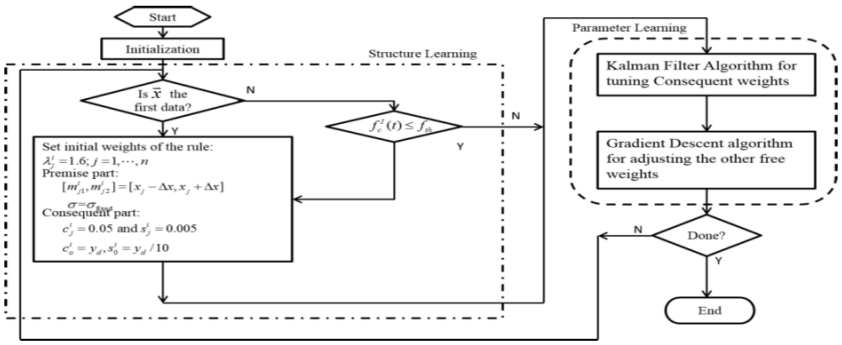


Figure 2.1: Identification using fuzzy network [1]

2.1.3 Online and offline model identification

Based on the basic identification methods in Chapter 2.1.1 and Chapter 2.1.2, the algorithms listed are adapted and optimized to various systems using either online or offline approaches.

Among online approaches, Puttige et al. introduced a real-time neural network-based online identification used to identify a UAV system in [26], using autoregressive moving average exogenous (ARMAX), which is a generalized model of ARMA and NARMA described above. Park et al. designed an online global model identification using MLP and RBF neural networks to identify a synchronous generator connected to an infinite bus [27], where the data is collected offline and trained online. In [28] Liu et al. used FCA-CMAC Neural Networks, some kinds of fuzzy network, for fault identification and fault-tolerant control for an unmanned underwater vehicle.

Among offline approaches, Erazo et al. described an offline method to use Bayesian identification for an output-only system by recursive estimation using the unscented Kalman filter in [29]. In [30], Caponio et al. introduced a fast adaptive memetic algorithm and applied it to both offline and online identification and controller design.

2.2 Adaptive control

A traditional implementation of adaptive control is adaptive PID controller used for auto-tuning of PID parameters. Ping Ge el al in [31] introduced an adaptive PID controller for tracking a piezoceramic actuator by adding feedforward to the feedback loop in PID. Training neural networks using PID parameters for the self-tuning of PID is also widely discussed, either using single-neuron network [32] or using RBFNN [33].

For adaptive control concerning neural network controllers, there have been lots of research in this direction. [2] used internal model control to control the temperature of an oven. They used RBFNN to identify the model with initial parameters offline, and updated the network online by replacing the outdated data points with newly generated data. The control process is shown in Figure 2.2, which is cited from [2]. The model has similar structures with the model identification adaptation introduced in Chapter 1.2.3, while the controller design requires training of another neural network. It will lead to larger computational pressure on hardware controllers with limited performance.

Also, [34] presents an adaptive inverse controller using neural networks that applied adaptive control for the water level of a boiler drum. The author uses neural networks to train the inverse

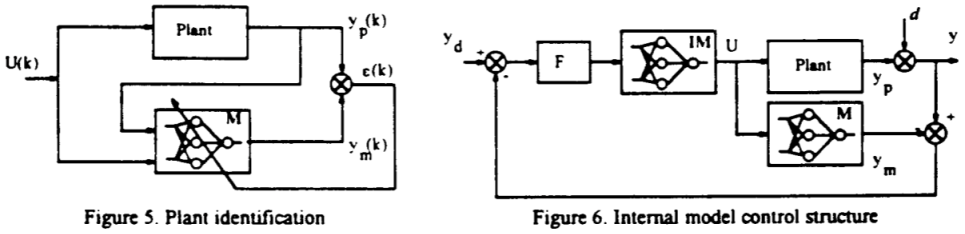


Figure 2.2: Procedure of internal control using RBFNN [2]

model of the controller, and the NN controller is combined with a robust feedback controller to provide better adaptation.

2.3 Simulators

As is introduced in Chapter 1, the simulators play an essential role in XiL simulation. The three simulators are widely used to simulate robot-related physical objects, and they are proved their abilities to use in XiL simulation in plenty of related work.

V-REP is developed by Rohmer et al. and introduced in [14]. Compared with Gazebo, it is more user-friendly on model design and integrates more features while having a similar performance of controller design and external APIs, according to the summary given by Nogueira in [35]. As a part of the simulation framework, Mohamed et al. implemented an XiL framework for IBC using V-REP in [36].

Webots is an open-source (commercial before 2018) robot simulation platform, which provides complete modelling, control application and simulation functions. Webots provides a large number of pre-designed robots and vehicle scenarios, and it also provides APIs of several popular programming languages with better readability or systems in robotics like C++, MATLAB, ROS, Python, etc [15]. The use of Webots in simulation is similar to V-REP, like [3] by Mohamed et al., which is the optimization of [36] using Webots instead of V-REP. An example of the use of Webots in the simulation is shown in Figure 2.3, which is cited from [3].

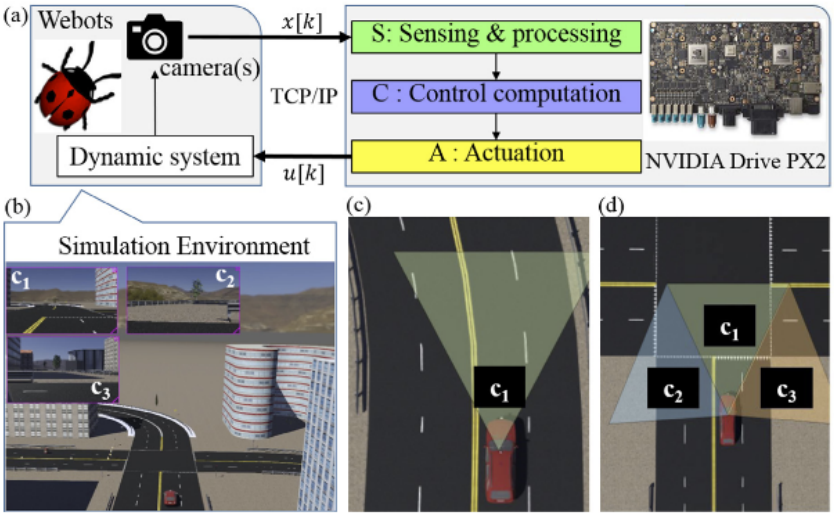


Figure 2.3: The use of Webots in simulation [3]

Chapter 3

Problem Statement

This section introduces the research question this thesis tries to solve and list the contributions of this thesis.

3.1 Problem statement

According to Chapter 1 and 2, a control system, in reality, will experience changes in parameters, which will cause the system to behave differently in different states, and it is not always possible to evaluate the physical changes, like thermal and ageing effects on an actual system.

In order to identify the controller and design the controller, system identification is necessary for complex dynamic systems described above. Furthermore, because of the need to design the controller and adapt the control input to the current controller when the parameters change, a strategy to make the controller adapt to the changes is highly demanded. However, online identification suffers from the methods for its inaccuracy and strict real-time timing requirements. Thus, a mixed-method using offline algorithms and online update strategy is preferred. Furthermore, simulation and simulation environments are proved to be useful for simulating extreme conditions and parameters for a dynamic system. Further, the performance of the controller can also be evaluated under these conditions.

Therefore, the research question is: **Can we design a controller that can identify a control system model online and adapts to parameter changes of the systems during its lifetime?**

3.2 Contribution

The contribution of this thesis includes the following items:

1. Study algorithms of system identification using neural network and justify its feasibility.
2. Choose the appropriate neural network based controller and test its performance.
3. Integrate the selected controller in the control system and do model learning and controller design for the chosen dynamic system.
4. Design an adaptation strategy to deal with parameter changes and improve the control performance.
5. Performance evaluation of the identification and adaptation method.
6. Design an XiL framework with MATLAB front-end for simulation and validation of the controller.

Chapter 4

Design approach

This chapter includes two parts. First, we give an overview of the design approach for identification and runtime adaptation and describe the functionality and structure of each step. Second, we describes the framework used for validation.

4.1 Overview of the design approach

The overall structure of the design approach is shown in Figure 4.1. In Figure 4.1, the whole process is divided into four steps, namely data collection, training for identification, controller design and runtime adaptation, which will be explained in detail below.

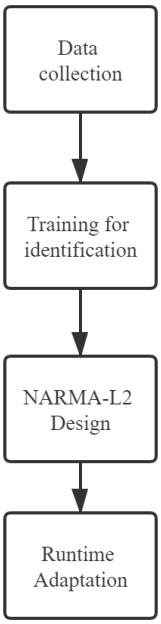


Figure 4.1: Overall design approach

4.1.1 Data collection

The first step of the process is to collect enough data for system identification and adaptation. The dynamic system required to be identified is more likely to be a black box, and the only available information for the system is the input and output data. Therefore, the input and output data of the system, including the input and output of the past moments to cover the time delays of the control system, needs to be collected and sent for training. Also, the number of data points to be collected depends on the performance of the identification model and the time permitted for running the system to collect data. In our experiments, 1500 to 2500 data points (about 15s to 20s) is enough to train and identify an adaptive cruise control system.

4.1.2 Training for system identification

After collecting enough data for identification, the data will be used for training of the identification model. Chapter 2 mentions several methods for identification, both traditional methods like least-square or Kalman filtering, and methods using machine learning methods, like neural networks or SVR. To be prepared for a physical model with non-linearity and tine-varying, a proper identification model should be chosen to present the model precisely and correctly.

Firstly, because the model must support non-linear and time-varying systems, traditional methods like least-square will then be exempted. Secondly, among machine learning methods, including neural networks and SVR, SVR cannot deal with large samples [37], which is common in control problems with high sample frequency. Therefore, the neural network approach is chosen. For a standard neural network controller, it is challenging to figure out the control input $u(k)$, which is not suitable for adaptive controller design. Therefore, as is stated in Chapter 2.2, the NARMA-L2 controller, whose control input can be calculated directly by algebraic methods, is chosen for the model to be implemented. Therefore, the final identification model should be a NARMA-L2 model.

After selecting the proper model, the selected model, NARMA-L2, will be used for identification and controller design. The model will be trained using the data provided by the data collection block.

4.1.3 NARMA-L2 Controller Design

After the NARMA-L2 model identifies the dynamic model accurately enough, the model (or the neural networks in it) will be stored and called in the NARMA-L2 controller. As is mentioned in the comparison between NARMA-L2 and other kinds of identification model in the last subsection, the controller accompanied with NARMA-L2 can be designed by simple algebraic calculation using the networks, system output and the reference output. Firstly, the output that the dynamic system is required to produce needs to be determined. Then the controller can produce the input to be imported in the system. The controller and the dynamic system will then form a closed-loop system, where the controller receives the control output from the system and produces the control input of the next state, while the dynamic system receives the input generated by the controller and gives the output corresponding to the input.

4.1.4 Runtime adaptation

The first three steps shown in Figure 4.1 do system identification and controller design assuming that the system does not change. Therefore, this step aims at defining a strategy to deal with the requirement about parameter changes in the system. In the proposed approach, the occurrence of the update of the controller needs to be determined based on the data points needed to collect. When the required data have completed collection, the collected data will be used to re-train the NARMA-L2 model and replace the current model if the current model shows higher error rate, which means that the original model cannot represent the dynamic system well.

4.2 Validation of the approach

The approach described in the previous chapter discusses how the system is identified, and how the controller is designed and adapted. To validate the approach and provide a detailed view of the whole system, the following validation process is proposed, which is shown in Figure 4.2

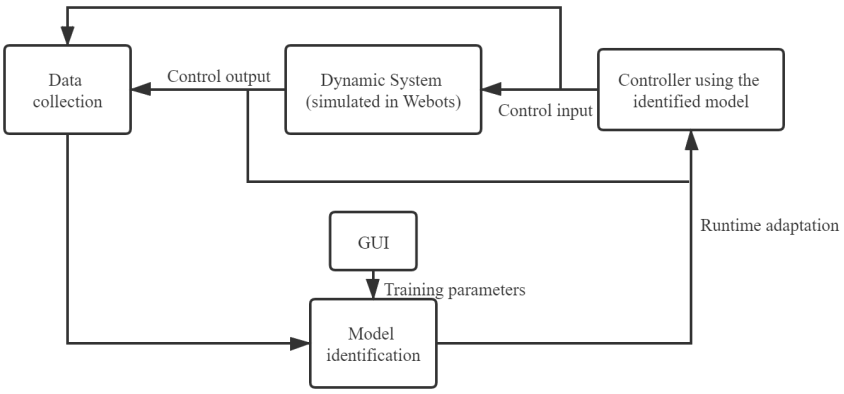


Figure 4.2: Validation process for the approach

The validation process solves two problems of the overall approach in Figure 4.1. First, Webots, the simulation platform of the dynamic system is added to replace the use of physical systems. Second, training parameters and properties, like learning rates, training epochs, amount of data points and the network structures are defined using the GUI in the XiL simulation framework. The validation can be done by selecting the parameters in the GUI, use the GUI to connect to Webots and observe the performance of the controller. The detailed description of the XiL simulation framework will be given in Chapter 7.

Chapter 5

Mathematical Background of Identification

This chapter introduces the mathematical background of identification methods and controller design.

5.1 Description of control system

A discrete-time control system can be represented in state-space equation as follows:

$$\begin{aligned} x(k+1) &= f[x(k), u(k)] \\ y(k) &= g[x(k)] \end{aligned} \tag{5.1}$$

where the state $x(k) \in \mathbb{R}^k$, the input $u(k) \in \mathbb{R}^m$, and the output $y(k) \in \mathbb{R}^n$. The functions $f : \mathbb{R}^m \times \mathbb{R}^k \to \mathbb{R}^k$ is the mappings from the current state and input to the next state, and $g : \mathbb{R}^k \to \mathbb{R}^n$ is from the current state to the output, which are both unknown in this scenario to be identified. $x(k)$ is unknown to observers too, so the unknown system is observed as an input-output black box from outside. If $m = n = 1$, the system is called Single Input Single Output (SISO) system, while otherwise it is classified as Multi Input Multi Output (MIMO) system.

A discrete-time linear system can be represented as follows, which can be regarded as linearization of Equation 5.1:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) \end{aligned} \tag{5.2}$$

Because f and g in Equation 5.1 are unknown linear or nonlinear functions, the objective of system identification is to derive the two functions from the unknown input-output plant.

5.2 NARMA and NARMA-L2 model

5.2.1 NARMA model

As is discussed in [38], a SISO system represented in the form of Equation 5.1 can be converted into the following form:

$$y(k+1) = F_n[y(k), y(k-1), ..., y(k-n+1), u(k), u(k-1), u(k-n+1)] \tag{5.3}$$

where F_n is a map that $F_n : \mathbb{R}^{2n} \to \mathbb{R}$.

Equation 5.3 can be generalized to make the following equation exist:

$$y(k+d) = \mathcal{F}_n[y(k), y(k-1), ..., y(k-n_y+1), u(k), u(k-1), ..., u(k-n_u+1)] \tag{5.4}$$

where $\mathcal{F}_n$ is a map that $F_n : \mathbb{R}^{n_y+n_b} \to \mathbb{R}$ and $d \geq 2$

Equation 5.4 shows that the time-delayed output can be represented by a nonlinear function of the past inputs and states. It is called a Nonlinear Auto-Regressive Moving-Average (NARMA) model.

5.2.2 NARMA-L2 model

In theory, NARMA model can be directly used in system identification by using a neural network to approximate $\mathcal{F}_n$. This method can achieve higher identification accuracy, but it is difficult to calculate the desired $u(k)$ in adaptive controller design, because according to Equation 5.4,the control input required to get the desired reference output $y^*(k)$ is

$$u(k) = \mathcal{H}_n[y(k), y(k-1), ..., y(k-n_y+1), y^*(k+d), u(k), u(k-1), ..., u(k-n_u+1)] \tag{5.5}$$

$u(k)$ is nonlinear and needs to be approximated by neural network rather than algebraic calculation. Therefore, an approximated model, NARMA-L2 is raised in [39] in order to simplify the controller design and accelerate the training process. Equation 5.4 can be derived by separating $u(k)$ apart and applying Taylor expansions at an equilibrium point u_0 as follows:

$$\begin{aligned} y(k+d) &= \mathcal{F}_n[y(k), y(k-1), ..., y(k-n_y+1), u(k), u(k-1), ..., u(k-n_u+1)] \\ &= \mathcal{F}_n[\phi(k), u(k)] \\ &= \mathcal{G}_n[\phi(k), u_0(k)] + \frac{\partial \mathcal{G}_n}{\partial u(k)}|_{\phi(k),u_0(k)}(u(k) - u_0(k)) + R_1(k) \\ &\Rightarrow \{\mathcal{G}_n[\phi(k), u_0(k)] - \frac{\partial \mathcal{G}_n}{\partial u(k)}|_{\phi(k),u_0(k)} u_0(k)\} + \frac{\partial \mathcal{G}_n}{\partial u(k)}|_{\phi(k),u_0(k)} u(k) \\ &= f_0[\phi(k)] + g_0[\phi(k)]u(k) \end{aligned} \tag{5.6}$$

where f_0 and g_0 are two nonlinear maps that can be approximated by two separate neural networks. The network structure described in Equation 5.6 is shown in Figure 5.1, which visualizes how the output $y(k+1)$ is calculated with the trained two networks and the input data series.

Adaptive controller design in NARMA-L2 can be much easier because it can be achieved only with algebraic calculation, which is shown in the following:

$$u(k) = \frac{y^*(k+d) - f_0[\phi(k)]}{g_0[\phi(k)]} \tag{5.7}$$

where $\phi(k) = [y(k), y(k-1), ..., y(k-n_y+1), u(k-1), ..., u(k-n_u+1)]$. Because $u(k)$ is determined by $y(k)$, the output at the same time, it is difficult to implement Equation 5.7 in reality. Therefore, Equation 5.7 is modified to the following form:

$$u(k+1) = \frac{y^*(k+d) - f_0[\phi^{'}(k)]}{g_0[\phi^{'}(k)]} \tag{5.8}$$

where $\phi^{'}(k) = [y(k), y(k-1), ..., y(k-n_y+1), u(k), ..., u(k-n_u+1)]$.

5.2.3 NARMA-L2 model in MIMO system

Chapter 5.2.1 and 5.2.2 discusses how NARMA-L2 is derived and applied in SISO system with one input and one output. This chapter will introduce how NARMA-L2 can be generalized in MIMO systems [23].

Suppose an MIMO system described as Equation 5.4, while $y(k) = [y_1(k), y_2(k), ..., y_n(k)]$, and $u(k) = [u_1(k), u_2(k), ..., u_n(k)]^T$. The NARMA-L2 model that applies to this model is

$$\begin{aligned} y_i(k+d) &= f_i[y(k), y(k-1), ..., y(k-n+1), u(k), u(k-1), u(k-n+1)] \\ &\quad + \sum_{j=1}^{n} g_{ij}[y(k), y(k-1), ..., y(k-n+1), u(k), u(k-1), u(k-n+1)]u_j(k) \end{aligned} \tag{5.9}$$

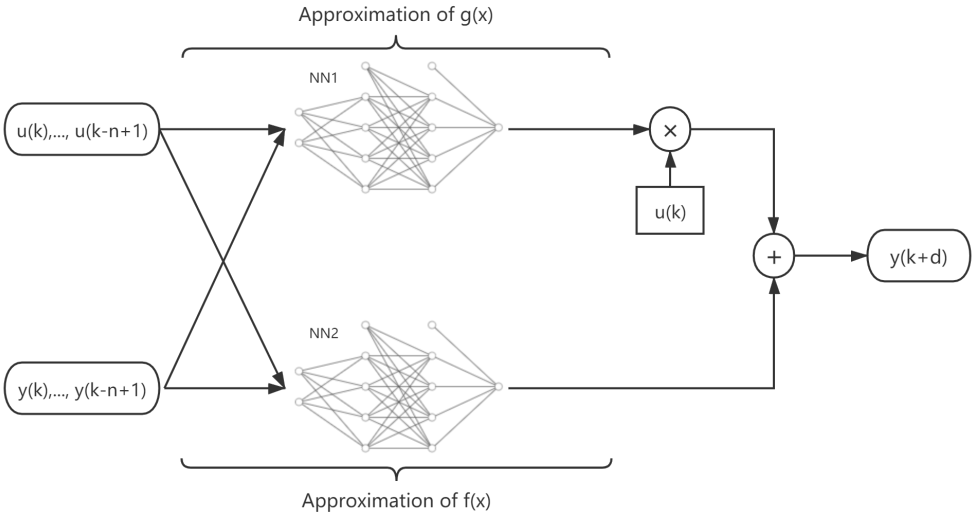


Figure 5.1: Structure of NARMA-L2 network

The controller design result $u(k)$ is given below so that the output follows the reference $y^*(k)$:

$$u(k) = \begin{bmatrix} g_{11}[\phi(k)] & ... & g_{1n}[\phi(k)] \\ ... & ... & ... \\ g_{n1}[\phi(k)] & ... & g_{nn}[\phi(k)] \end{bmatrix}^{-1} \begin{bmatrix} y_1(k) - f_1[\phi(k)] \\ ... \\ y_n(k) - f_n[\phi(k)] \end{bmatrix} \tag{5.10}$$

in which f_i and g_{ij} is given by the approximation of neural networks trained using Equation 5.9.

5.3 Smoothing of controller performance

According to Section 6.1 and 6.2, the NARMA-L2 controller is able to generate the system input that will make the control system act following the reference output. However, the controller can only guarantee the system output, while the control input is not restrained. The generated control input may have significant chatter or fluctuation. The fluctuation is caused by the sharp changes in the reference input, which will lead the input to hit the upper or lower limit and form chatters.

Therefore, [40] raised a method to smooth the controller performance by adding time-delays and linear feedback on the controller, inspired by the PID controller. In Equation 5.8, if some past value of the output y is added to the equation as linear feedback, the equation will be transformed as follows:

$$u(k+1) = \frac{c_0 y^*(k+d) - f_0[\phi^{'}(k)] - d^T y_p(k)}{g_0[\phi^{'}(k)]} \tag{5.11}$$

where $d = [d_1, d_2, ..., d_n]^T$, $y_p(k) = [y(k), y(k-1), ..., y(k-n+1)]$.

Take z-transform for Equation 5.11, the result is:

$$\begin{aligned} Y(z) &= \frac{c_0}{D(z)} Y^*(z) \\ D(z) &= 1 + d_1 z^{-1} + d_2 z^{-2} + ... + d_n z^{-n} \end{aligned} \tag{5.12}$$

According to Equation 5.12, the steady-state error can be eliminated if

$$c_0 = 1 + d_1 + d_2 + ... + d_n \tag{5.13}$$

where d_n needs to be defined by the users.

Equation 5.11, 5.12 and 5.13 apply to the scenario that the generated input requires to be smoothed. However, because there are time-delays added to the input, the response time of the controller will increase. Whether to use the method in this part requires analysis on the trade-off of response time and control performance.

Chapter 6

Implementation and verification of NARMA-L2 controller

This chapter describes the implementation of the NARMA-L2 model and verification of its correctness and effectiveness using both linear and non-linear systems.

6.1 Implementation

The implementation of the NARMA-L2 model aims at implementing the two neural networks to approximate f_0 and g_0 mentioned in Equation 5.6. The two neural networks are trained using the following algorithms:

Algorithm 1 Identification of NARMA-L2 model

Input: Time-delay of inputs and outputs $\phi(k)$=[$y(k)$, ..., $y(k-n+1)$, $u(k)$, ..., $u(k-n+1)$];
Output: Network represented as f and g

$N \leftarrow$ number of datapoints, $epoch \leftarrow$ time of iteration, $k \leftarrow 0$, $i \leftarrow 0$, $lr \leftarrow$learning rate
$X(layer)(neuron) \leftarrow$ weights of the nerual network in a single neuron in a certain layer. X represents f or g.
while $k \leq N$ **do**
    while $i \leq$ epoch **do**
        $yr(k) \leftarrow f[\phi(k)] + g[\phi(k)]u(k)$;
        $e(k) \leftarrow MSE(yr(k) - plant(u(k)))$;
        for l **in** $layers$ of f **do**
            for n **in** l **do**
                $f(l)(n) \leftarrow f(l)(n) - lr * diff(e(k), f(l)(n))$
            end for
        end for
        for l **in** $layers$ of g **do**
            for n **in** l **do**
                $g(l)(n) \leftarrow g(l)(n) - lr * diff(e(k), g(l)(n))$
            end for
        end for
        $i \leftarrow i + 1$;
    end while
    $k \leftarrow k + 1$;
end while
return f, g;

After obtaining the parameters in neural networks f and g, the control input $u(k)$ derived from the reference output $y^*(k)$ can be computed using the following algorithm, where $d1$ and $d2$ need to be summarized from experiments.

Algorithm 2 Controller Design of NARMA-L2 model

Input: Output reference $y^*(k)$, f and g
Output: control input $u(k)$, network output $y(k)$
1: $N \leftarrow$ number of datapoints, $k \leftarrow 0$, $[u(0), ..., u(n)] \leftarrow 0$, $[y(0), ..., y(n)] \leftarrow 0$.
2: $c \leftarrow 1 + d1 + d2$
3: **while** $k \leq N$ **do**
4: $yr(k+1) = plant(u(k))$
5: $\phi(k) = [yr(k), ..., yr(k-n+1), u(k), ..., u(k-n+1)]$
6: $u(k+1) = [c * y^*(k+2) - f(\phi(k)) - d1 * y(k) - d2 * y(k-1)/g(\phi(k)]$;
7: $k \leftarrow k+1$;
8: **end while**
9: **return** u, y;

The algorithm is implemented by Python and PyTorch and is plotted using matplotlib. Both networks are multi-layer perceptrons with two hidden layers and 64 neurons for each layer. The loss function selected is Mean Square Error (MSE), the optimizer is SGD or Adam, and the activation function in the hidden layers is ReLU.

6.2 Simulation and verification

In this section, two examples are proposed to verify that the NARMA-L2 model and its implementation can successfully identify the model and give satisfying results on adaptive control.

6.2.1 Non-linear system

Suppose a non-linear control system from [41] as follows:

$$y(k+1) = \frac{y(k)}{1+y^2(k)} + u^3(k) \tag{6.1}$$

The input training signal is set to:

$$u(k) = sin(\frac{\pi k}{30}) + sin(\frac{\pi k}{120}) \tag{6.2}$$

the input test signal is set to

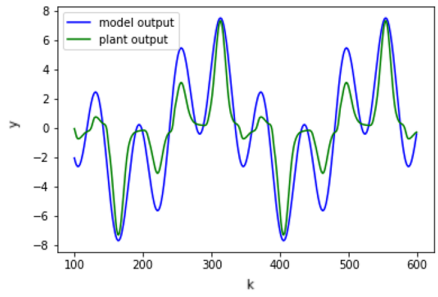
$$u^{'}(k) = sin(\frac{2\pi k}{10}) + sin(\frac{2\pi k}{25}) \tag{6.3}$$

and the reference output is set to

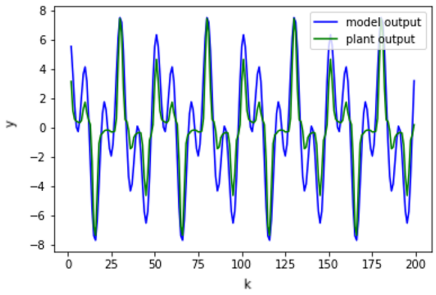
$$yr(k) = 4sin(\frac{2\pi k}{10}) + 4sin(\frac{2\pi k}{25}) \tag{6.4}$$

The simulation result is shown in Figure 6.1, and the control result is shown in Figure 6.2.

In Figure 6.1 , "model output" is the output from the trained and identified NARMA-L2 controller, "plant output" is the output from the controlled system. The input used to generate both outputs is the training or test signals u or u'. And in Figure 6.2, "predicted input" is the input value generated by the NARMA-L2 controller, "model output" is the output from the system using the input from the NARMA-L2 controller, and "reference" is the reference output defined previously, which the controller is hoped to follow.

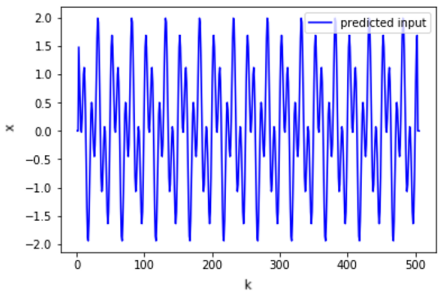


Training

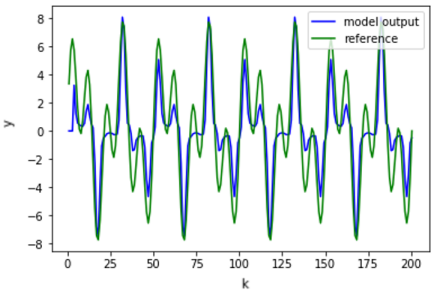


Testing

Figure 6.1: Identification result for the non-linear system



Control input



Control output

Figure 6.2: Controller Design result for the non-linear system

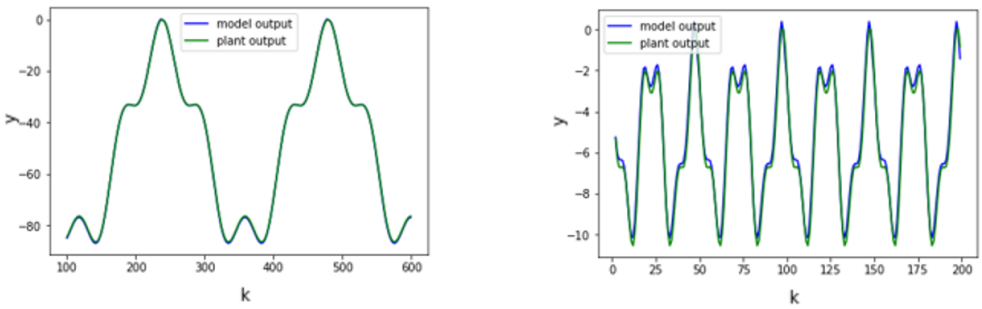
Figure 6.1 shows that the identified NARMA-L2 model is able to produce almost the same output as the real system when excited by the same input signal, which proves that the model is able to identify the given example non-linear system with satisfactory accuracy, And Figure 6.2 shows that the control input given by the controller can produce the output that can follow the reference signal. The shape and trend in changes of the control output keep almost the same as the reference model, while there are minor errors in some sharp turning points. The controller is able to accomplish the goal to control the system to generate desired outputs.

6.2.2 Linear system

Suppose a linear control system as follows:

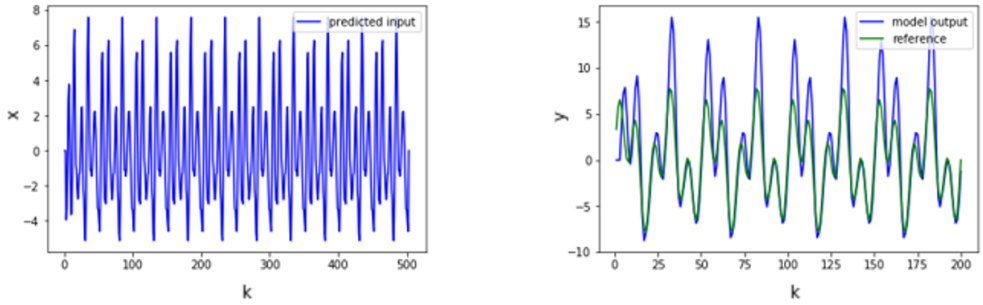
$$\begin{aligned} x_1(k+1) &= x_1(k) + u(k) \\ x_2(k+1) &= -0.016x_1(k) + x_2(k) + u(k) \\ y(k) &= -x_1(k) \end{aligned} \tag{6.5}$$

The input and reference output signals are the same as Equation 6.2, 6.3 and 6.4. The result is shown in 6.3 and 6.4. The definition of elements in these figures are the same as that in Figure 6.1 and 6.2.



Training Testing

Figure 6.3: Identification result for the linear system



Control input Control output

Figure 6.4: Controller Design result for the non-linear system

Figure 6.3 also show accurate identification of the system, which can be proved by the match between the output of the NARMA-L2 model and the real system. According to 6.4, the control output also keeps the same shapes as the reference, but there are errors on the upper limit of the curve. The error can be compensated by further tuning of the network or trying other training input.

In conclusion, the results given above prove that the implemented model can identify the example models well, and the calculated control input can make the model output follow the given reference model. And the linearity of the system does not affect the performance of the controller.

Chapter 7

XiL setup and configuration

7.1 Environment configuration

The following table lists the environment used in the development of the toolbox. The platforms and software used can be newer than listed except for Guest OS platform and Virtual machine (if the user chooses to run the environment in VM), because version difference of dependency libraries may cause unpredictable issues, especially on controller codes if the controller is written in C or C++.

Parameters	Type and version
Virtual Machine	VMWare Workstation 15
Host OS Platform	Windows 10 Patch 2004
Guest OS Platform	Ubuntu 18.04.3 LTS
Desktop Manager	Xfce (GNOME also compatible)
MATLAB version	MATLAB R2019a
Webots version	Webots R2020b

Table 7.1: Required environment of toolbox

7.2 Design of XiL setup

7.2.1 Overview

The GUI to be designed should have the following functions:

1. It should be able to call supported simulators from the GUI, start or stop the simulation.
2. It should be able to modify parameters of the external controllers of simulators and keep transparency between users and raw codes.
3. It should be able to react in real-time when a parameter changes.
4. It would be better to have some portability. Some extent of portability can be implemented by reducing hard-coded paths.
5. If needed, It would be better to be able to call or integrate required programs other than simulators to provide a reference to analysis.

Based on the requirements above, the toolbox is designed with the following features:

1. Support V-REP, Webots and MATLAB simulation engines as part of IMACS framework [36].

2. Paths of V-REP and Webots can be selected.

3. The control parameters are temporarily stored after generation and updated to the plant at the next frame. The control parameters considered are adopted from and explained in [3, 42, 43, 44, 45, 46, 47, 48].

4. Several pre-defined controllers - LQR and LQG are integrated. The LQG controller for the IMACS framework is explained in [46].

5. Integrate SDF3[1], a dataflow graph analysis tool developed by [49] to give an analysis of given dataflow graphs, and timing analysis [50] or profiling tools.

The toolbox is designed as a GUI application in MATLAB. Compared with other GUI libraries on Linux platform like Qt or GTK, MATLAB GUI can integrate appropriately with MATLAB and Simulink scripts and models, and it can achieve some extent of cross-platform portability provided that MATLAB is installed.

Within MATLAB toolboxes, The GUI is designed using MATLAB App Designer[2], a novel integrated GUI designer introduced in MATLAB 2016a to replace the traditional and deprecated GUIDE GUI designer. App Designer improves designing and coding experience, add a number of components, and fully converted to object-oriented programming in code generation.

7.2.2 Front-end design

The design of GUI is shown in Figure 7.1, 7.2 and 7.3.

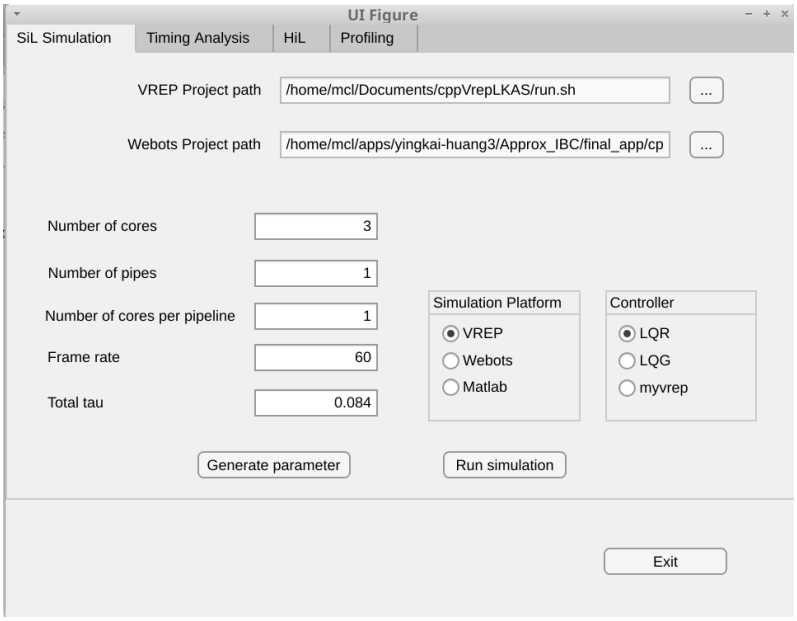


Figure 7.1: SiL Simulation page of the GUI

Figure 7.1 shows the fist page of the toolbox GUI, which focuses on the main function of software-in-the-loop simulation and parameter modification. The simulation platforms and controller types can be selected at the right side, and the button "Generate parameter" and "Run simulation" will have different clicking actions depending on the selection of platforms and. In this demo, five of the parameters in LQR and LQG controller can be modified at the left side. The paths can be changed by navigating explorer windows or typing in the full path.

[1] http://www.es.ele.tue.nl/sdf3/
[2] https://www.mathworks.com/products/matlab/app-designer.html

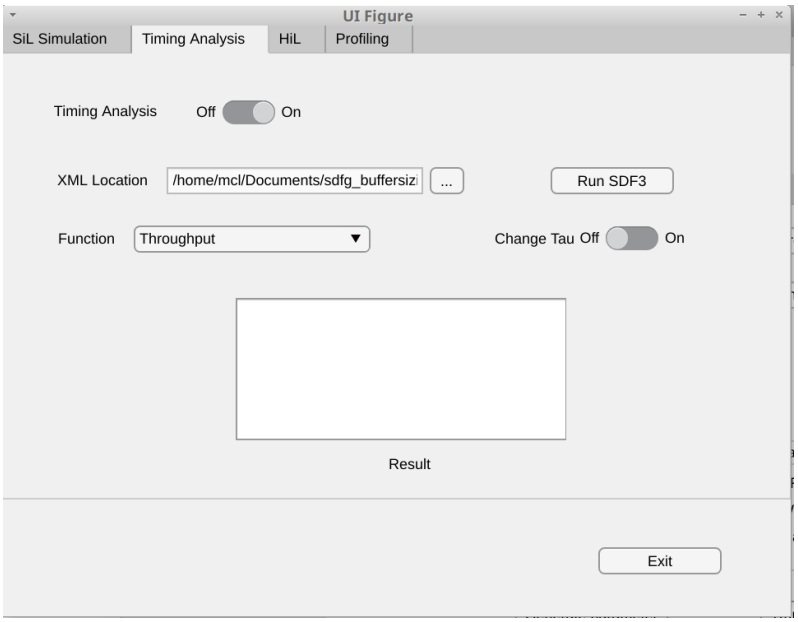


Figure 7.2: Timing analysis page of the GUI

Figure 7.2 shows the timing analysis page, where the GUI can call SDF3 and run several supported analysis, including throughput, cycle mean, etc. The result will be displayed in the window below, and there is an option that can use the given throughput to calculate tau (τ) on the first page.

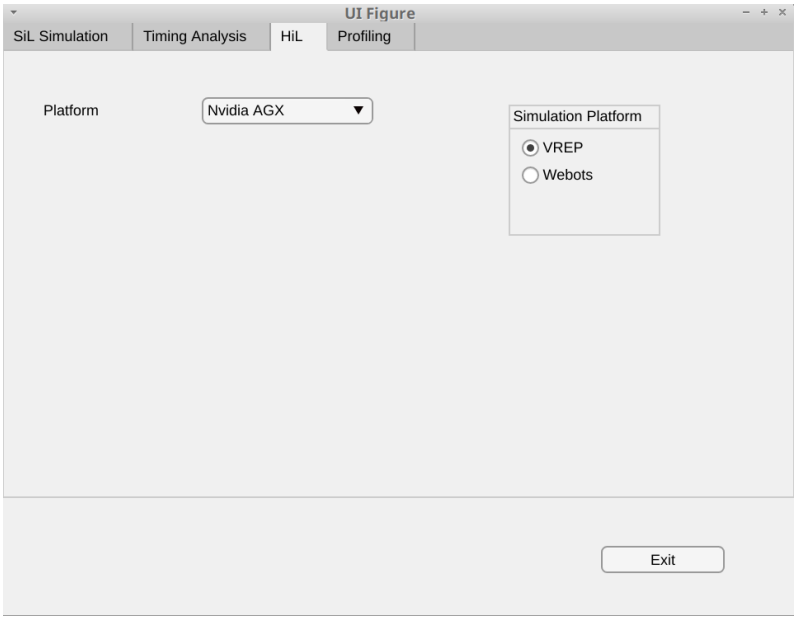


Figure 7.3: HiL simulation page of the GUI

Figure 7.3 is used for HiL analysis or profiling. Now the platform selection between Nvidia AGX and CompSOC and simulation platform selection has been implemented.

7.2.3 Back-end design

The back-end design is about how the buttons, edit files and selection boxes can interact with the code and how they can be given functionality.

According to [51], MATLAB GUI uses callback function for activities of components. A callback function can be defined for a particular interaction or state change. When the state changes or interaction occurs, the callback function will be executed.

The following code is a simple callback function that calls a MATLAB function to allow users to select an XML file for SDF3 analysis in a file navigation window.

```
% Button pushed function: ViewButton
function ViewButtonPushed(app, event)
    [file,path] = uigetfile('*.xml');
    if file~=0
        app.XMLLocationEditField.Value=join([path,file]);
    end

end
```

Figure 7.4: An example of callback function

When the button "Generate parameter" is clicked, the callback function will call a MATLAB script that calculates the controller parameters based on the inserted value. If "V-REP" or "Webots" is selected as the simulator, the script will back up the external controller code (C++ in this project) and generate the cpp and hpp files with the value derived from the given parameters using MATLAB built-in file operation functions. If "MATLAB" is selected, then the code generation will be disabled, and a plot of control output will be returned.

For the second part of timing analysis, the button "Run SDF3" will call the command of SDF3 and return the throughput or other value in the window below.

Chapter 8

Case study: Adaptive cruise control system

This chapter introduces the controlled system used for identification and controller adaptation. The control model and its properties is described first, and a Webots world which contains the required simulation scenarios is proposed.

8.1 Description of the cruise control system

The cruise control system consists of several vehicles of different types and several kinds of roads, including plain surface and slopes. The process of the cruise control is to adjust the throttle angle of the vehicle to accelerate and decelerate, in order to make the vehicle keep a certain speed. We assume that in a particular scenario, the brake is kept unchanged.

In the cruise system, the vehicle is affected by two forces, one is the torque in the same direction to the forward, and the other is the friction and other resistance to the backward direction. The resistance is assumed to be linearly related to the velocity of the vehicle. The free-body diagram of the system is shown in Figure 8.1.

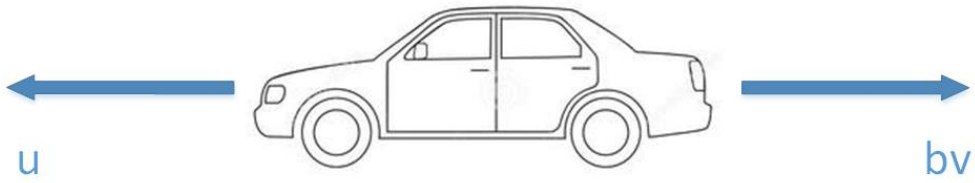


Figure 8.1: Force figure of the cruising vehicle

If we assume that the torque generated by the road and tire surface is determined by the throttle angle of the vehicle and can be directly controlled by the driver, the input value of the control system can be set as the force u. Therefore, according to Newton's second law, the system can be represented as follows [52]:

$$u - bv = m\dot{v} \tag{8.1}$$

where b represents the damping coefficient.

Assume the state $x = v$, we can convert Equation 8.1 into state-space equation as Equation 8.2.

$$\begin{aligned} \dot{x} &= -\frac{b}{m}x + \frac{1}{m}u \\ y &= x \end{aligned} \tag{8.2}$$

However, the driver can only control the torque by pushing the accelerator pedal to adjust the throttle angle instead. The relationship between the torque and throttle angle is complicated and out of the control of drivers. For instance, in Webots, the relationship between the torque and throttle on combustion engines is shown in Equation 8.3, according to [53]:

$$T = t * (a * rpm^2 + b * rpm + c) \tag{8.3}$$

T represents output torque, t represents the coefficient of the throttle, which is ranged from 0 to 1. t is the only value in Webots that can be adjusted using user-oriented APIs, like `setThrottle()`. Moreover, a, b and c are determined by the vehicle design and gears. The rpm value is related to the throttle angle, but the exact relation is based on the mechanical properties of the vehicle, which is unknown to drivers.

Therefore, the cruise system in Webots, with **throttle coefficient** (from 0 to 1) as input and the **velocity** (in kilometre per hour) as output, is more complicated and requires more insight on it.

8.2 Parameter changes and related simulation environment

According to Equation 8.2 and 8.3, the parameters that influence the control system includes the mass of the vehicle (affect m), the brake intensity, the slope grade (affect b), and the engine model (affect the relation between u and the throttle angle). Among these parameters, the mass and engine model is constant during a single driving period, while the slope grade and the brake intensity will vary as time goes.

To fully simulate the situations above, the following Webots world is built, in which a variety of road surface, slope grades and vehicle models are tested.

8.2.1 World settings

The simulation takes part in the following Webots world, whose screenshot is shown in Figure 8.2.

The road is built with `Road` nodes in Webots, including `StraightRoadSegment` node for plain road surface and `Road` node for slopes and custom roads. The length of the roads is from 450 meters to 600 meters, and the grade of slope roads is 10%.

8.2.2 Vehicle properties

Webots integrates several pre-defined vehicle models in the Webots project in PROTO node type, which include different types and values of mass, gears, engines, etc.

Figure 8.3 shows two types of Car PROTO nodes, `BmwX5` and `MercedersBenzSprinter`. The difference between these two vehicle models, according to Webots documentation, is listed in Table 8.1.

Model	BmwX5	MercedesBenzSprinter
mass (kg)	2000	4500
engine coefficient	[34.11 0.136 -0.00001461]	[600,0.2,0]
brake coefficient	1800	3500
maximum speed on Gear 1 (km/h)	73	60

Table 8.1: Properties of Car PROTO models

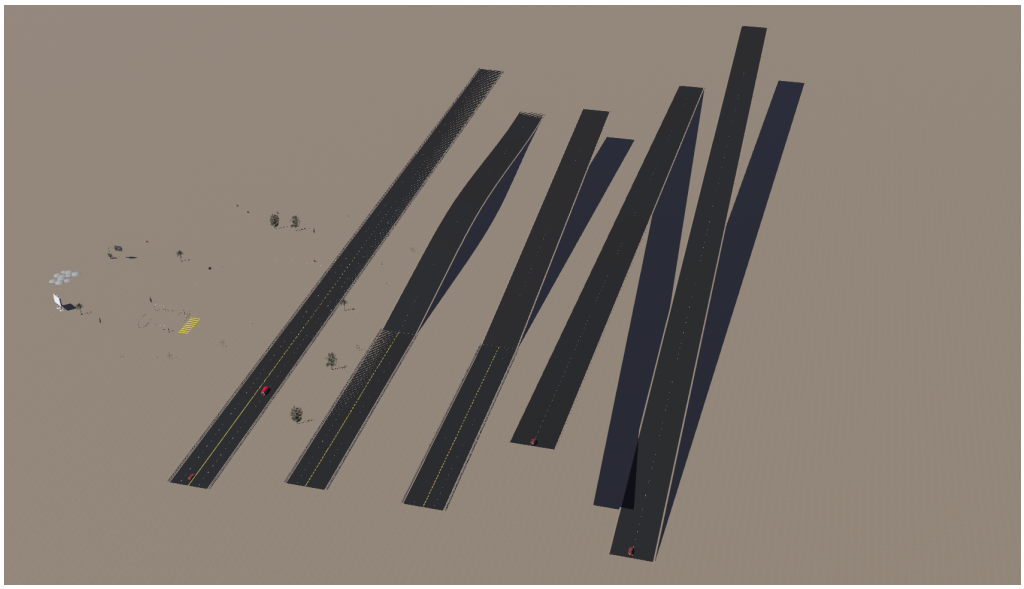


Figure 8.2: Appearance of the scenario

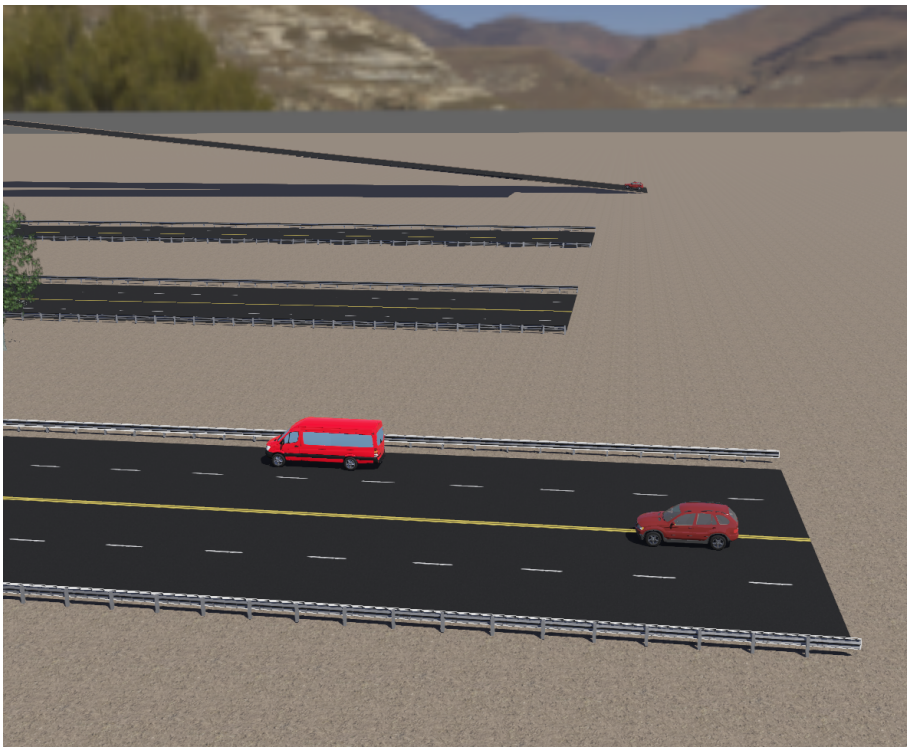


Figure 8.3: Different car models in Webots

8.2.3 Summary

From the information listed above, we can find out that the world includes all the components and possible scenarios that can influence the control system. The environment can then be used for data collection, training and controller design in the following chapter.

Chapter 9

Experimental results

9.1 Webots controller structures

Because the identification method is dependent on PyTorch and is developed in Python, and automobile APIs in Webots does not support MATLAB, the Webots controller for cruise control is written in Python.

A controller first starts with the initialization of packages and devices. In this part, required packages are imported (included), including `Robot` or `Driver` packages that contain Webots APIs. Also, a `Driver` instance and arrays that contain the collected data are created, and some parameters, like the initial throttle, gear and brake intensity is set here. Then the controller enters the main loop, which loops every 10ms (time steps in the simulation). The first several loops (about 100 iterations) must be discarded because Webots may generate irregular value at the beginning of the simulation, like `nan` or unusual large value. In every time steps, the controller calculates the input, set the throttle and store the throttle and the current speed. When the required data are collected, the loop is jumped out, and the stored input and output are dumped and saved into CSV files.

In the cruise control scenario, operations to be made by the controller are listed in Table 9.1 [53].

API Name	Input Range	Description
`setThrottle()`	[0,1]	Set the percentage of throttle angle
`setBrakeIntensity()`	[0,1]	Set the percentage of brake intensity
`getCurrentSpeed()`		Get the current speed in km/h
`setCruisingSpeed()`	per car model	Set the cruise speed in km/h.
`setGear()`	per car model	Set gearbox. Set before setting throttle.
`step()`		The main loop. One step equals 10ms in default.

Table 9.1: Webots APIs used in the controller

The structure of vehicle controller is shown in Figure 9.14.

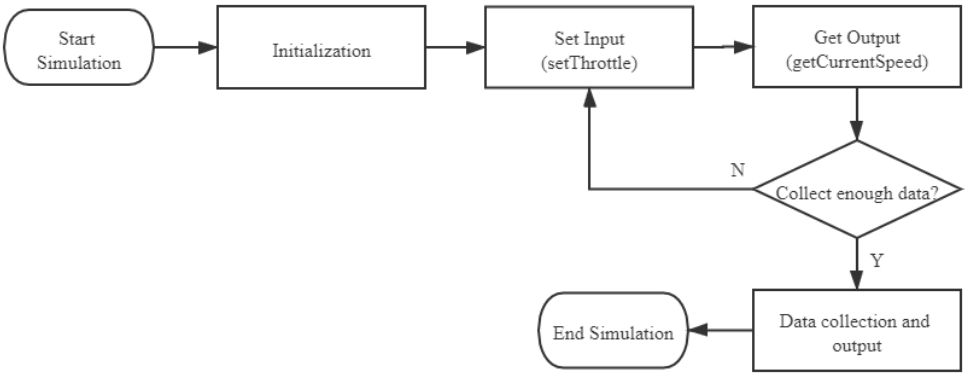


Figure 9.1: Procedures of the controller in Webots

9.2 A traditional solution for reference: PID controller

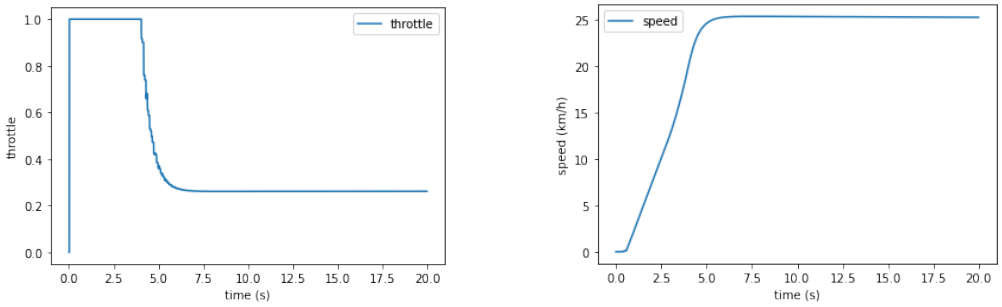
Before applying the NARMA-L2 controller on the system, a traditional solution for the cruise control using PID controller needs to be reviewed for reference and test cases for identification. PID controller (or PD controller) is commonly used in the control of linear time-invariant systems, so it is selected as a traditional solution for cruise control problems.

The PID controller can be represented as follows:

$$\begin{aligned} err(k) &= setpoint - value \\ errI(k) &= \sum_{i=0}^{k-1} errI(i) + err(k) * T \\ output(k) &= P * err(k) + I * errI + D * (err(k) - err(k-1)) \end{aligned} \tag{9.1}$$

In the reference scenario, a `BmwX5` car is placed on a plain surface. The cruise speed (*setpoint*) is set to 25km/h. The initial brake intensity is set to 0.1. The simulation time is 20s (2000 cycles). In Equation 9.1, P equals 150, I equals 2.7, D equals 10. The output (throttle) is divided by 1000 before fed into the system to enhance visibility.

The result of input (throttle) and output (speed) is shown in Figure 9.2.



Throttle Speed

Figure 9.2: Throttle and speed from the PID controller

From Figure 9.2, some features of the controller to be designed can be summarized. First, the process can be divided into the acceleration phase and the cruise phase. In the acceleration phase, the throttle is set to the maximum value (1.0 or otherwise defined), and when the speed

is approaching the cruise speed, the throttle will slowly decrease to the equilibrium point (about 0.25), and the speed will become constant. Therefore, to find the equilibrium throttle point is the target of the controller. Second, the settling time of the PID controller is relatively high, which means that the vehicle will take a longer time to achieve the desired speed.

9.3 NARMA-L2 controller design

As is described in Chapter 7, designing the NARMA-L2 controller is divided into three steps. Firstly, some test controllers are run to collect the data for identification and training. Therefore, using the input-output data, NARMA-L2 model is trained for identification of the model, and the trained model will be verified by test datasets otherwise generated. Then the controller is designed using the trained model.

Firstly, we focus on the most common cases: a `BmwX5` car is placed on a plain road surface. The braking coefficient is set to 0.1 for the whole simulation process. The aim is to design a controller for this system to make the car keep a certain speed.

9.3.1 Data collection

For data collection, the main concern is to find a proper input (excitation signal) dataset that can achieve better accuracy on identification and save the input and output dataset for training. Although the NARMA-L2 controller has no restriction on input and output signals, the selection of the excitation signal will have a significant influence on the accuracy of identification.

There are three choices on the excitation signal, which is:

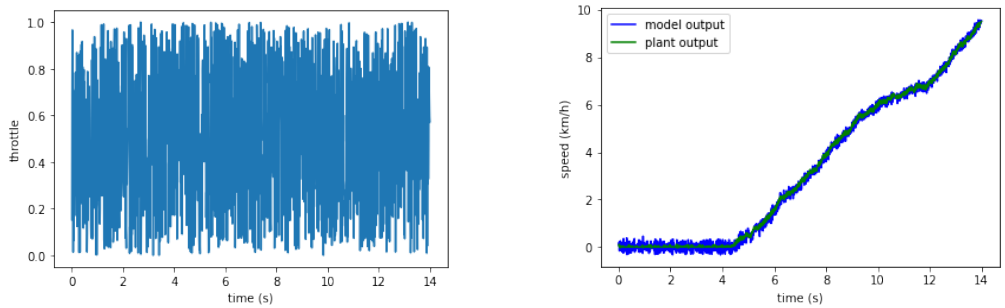
1. Using a pre-defined regular signal like sine functions. It is following the same pattern as the signal used in Chapter 7. In the experiment, the following signal is chosen, where time t is represented in second, and the smallest time step of t is 0.01s:

$$u = sin\pi t \tag{9.2}$$

2. Using random values as regular signals, as is used in MATLAB Simulink. The random number is between 0 and 1, generated by `numpy.random.rand()`.

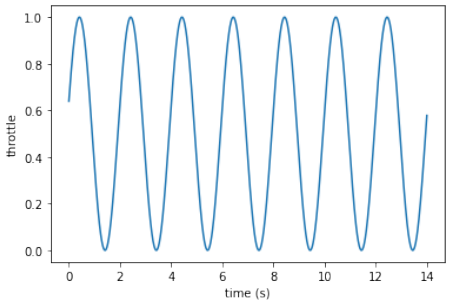
3. Using existing collected datasets like that produced by PID controller in Chapter 9.2.

For comparing the training performance among different inputs, the three inputs listed above are imported into the Webots world with the same properties as that in PID controller, and the datasets generated are used to train a sample NARMA-L2 model. The number of data points of all cases is 1400. The result is shown in Figure 9.3 to 9.5.

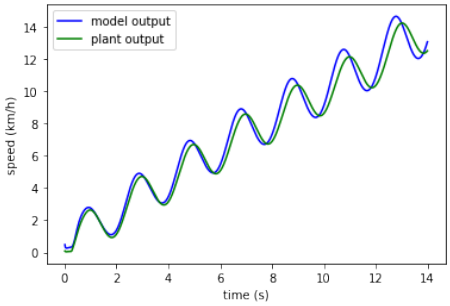


Random input Training result

Figure 9.3: Training result using random inputs

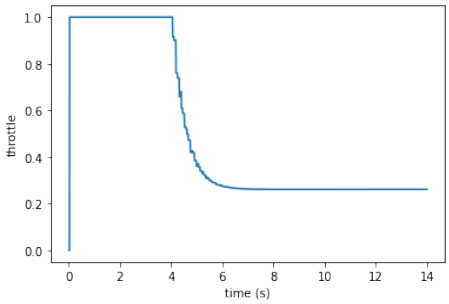


Sine input

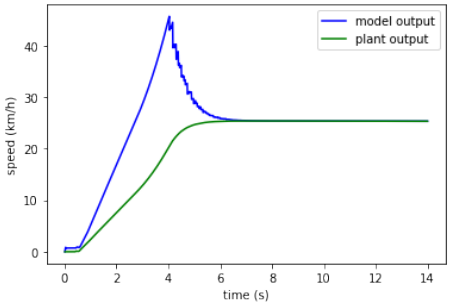


Training result

Figure 9.4: Training result using sine inputs



PID input



Training result

Figure 9.5: Training result using PID inputs

In the figures above, "model output" represents the output of the trained NARMA-L2 model when the input is the signal shown on the left. "Plant output" represents the output of the real cruise control system with the same input. X-axises represent the number of data, while the Y-axises represent either the throttle angle (input, in a ratio between 0 and 1) or the velocity of the vehicle (output, in km/h). From the figures listed above, we can observe that when using random inputs, the model output of the trained NARMA-L2 model appears closely around the output of the real output, while the other two inputs will cause the result to differ apparently between the model output and the plant output. It is apparent that the result using random input has the smallest training error rate compared to sine input and PID input, while the other two inputs show output with larger error rate or even unstable output. Also, because the vehicle has a maximum speed for each gear (as is mentioned in Table 8.1), the collected data is limited to the amount before the vehicle reaches the upper speed limit. So the input that can provide a slower acceleration is preferred, for it can provide more data points and higher accuracy in the training. Therefore, in the training part, the random input is used.

Another issue to be mentioned for random data collection is the situation at the start of the simulation. The default random number range is $[0, 1)$, but sometimes if the random number is lower than the expectation, the car will not get enough torque to start up. If such issue happens, possible solutions include changing the brake intensity, retry the simulation to reset the random seeds and change the lower bound of the random number range to, for instance, $[0.3, 1)$.

The controller for data collection is located in `vehicles\controllers\data_collection`.

9.3.2 Training for identification

After the selection of excitation signals, the datasets are used for training of the NARMA-L2 controller. The training algorithm is the same as Algorithm 1 in Chapter 7.1, while the parameters for training is required to change. The parameter includes learning rate, time-delays of u and y (n in Algorithm 1), size of hidden layers and epoch numbers.

The parameters chosen in training are listed in Table 9.2.

Parameter	Value or Choices
learning rate	1e-6
Input time delays	3
Output time delays	3
network structures	$[6, 10, 1]$
epoch	50
optimizer	SGD

Table 9.2: Training parameters for identification

The training result using the parameter in Table 9.2, along with test result with the PID controller and input signal in Equation 9.2 is shown in Figure 9.6, 9.7 and 9.8. The definition of each line in the figure is the same as that in Figure 9.3. For training results, the MAE of the model input is about 0.15. We can find from these figures that no matter what input signals are used to excite the NARMA-L2 model, the output generated by the model can perfectly follow the output from the real system. Therefore, it can be summarized that the trained NARMA-L2 model has almost the same behaviour as the cruise control system, which proves that the model can fully identify the cruise model and represent the cruise control system precisely.

In the training process, learning rate plays the most important role in shaping the training performance. Because the input is relatively smaller than the output, the optimal learning rate will be smaller than the simulation in Chapter 7. Larger learning rate will cause the model to converge into a non-optimal point, which in NARMA-L2 means that the network output has a similar shape with the model output, but the value differs a lot. Lower learning rate will cause the model to converge less than required, which in NARMA-L2 means that the network output will fail to follow the trend of the model output while the value has a small difference. Figure 9.9

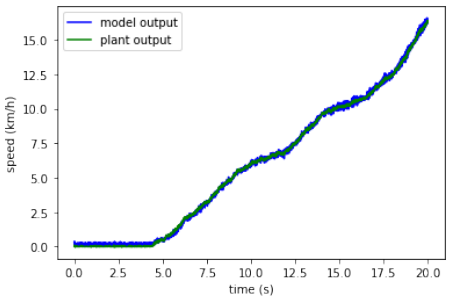


Figure 9.6: Training result for the cruise control

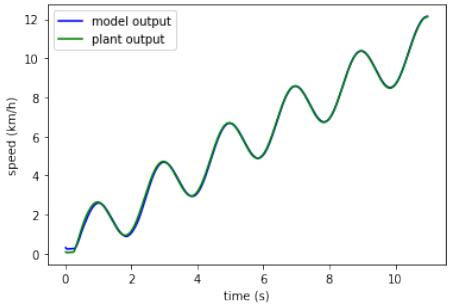


Figure 9.7: Test result for the cruise control using sine input

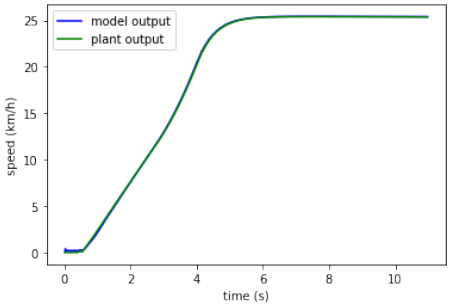


Figure 9.8: Test result for the cruise control using PID input

and 9.10 show the training result with larger (1e-2) and lower (1e-8) learning rate, compared with the adequate (1e-6) learning rate, whose result is shown in Figure 9.6.

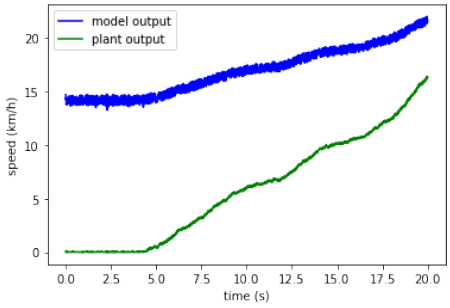


Figure 9.9: Training result with higher learning rate

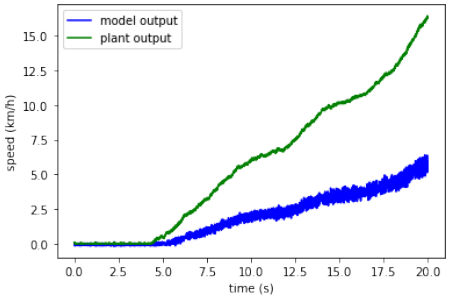


Figure 9.10: Training result with lower learning rate

From Figure 9.9 and 9.10, we can find that when the learning rate is higher than the optimal value, the model output differs significantly from the plant output, while some similarity on the shapes can be found between two plots, like the location to accelerate faster or slower, i.e. they both accelerate around the 500th data point and their acceleration both change on the 1250th data point. When the learning rate is lower, we can find that the model output is much lower than the plant output, but the precision is reasonable at the start of the simulation. The reason is that lower learning rate makes the model converge too slowly to reach the convergence point, so the model stops on the process of convergence when the training ends, and fails to converge to a precise value. These two figures prove the discussion above and prove the value of choosing the learning rate in identification and controller design.

Apart from the learning rate, the input-output time delays will also affect the performance

to some extent. The influence is lower than the learning rate, which means that in a reasonable range like 2 delays, the performance will still be acceptable but require tuning of other parameters. For the size of hidden layer and epochs, changes in these parameters will lead to the difference in the training time, which need to be considered if the update interval of the model is short or the device to run the controller has lower performance or fails to support CUDA.

9.3.3 NARMA-L2 Controller design

In the last section, the NARMA-L2 model is trained to identify the cruise control model. After obtaining an accurate model, the controller can be easily designed using Algorithm 2 mentioned in Chapter 7.1 with constraints of control input (between 0 and 1) and smoothed output as is described in Chapter 6.3. The d in the smoothed controller is $[0.5, 0.2]$.

After designing the controller, the next step is to evaluate the controller to check if it meets the design requirements. Usually, the controller needs to be connected to Webots to test if the vehicle can achieve and keep the desired speed. However, fast and intuitive evaluation methods are still valuable, because the simulation in Webots is slow and hard to visualize, and the odd results can be discarded fast to reduce experiment time.

Therefore, to test the performance of the trained NARMA-L2 controller in the regard of controller performance without running Webots simulation multiple times and give instant feedback about the control input and output, a pre-trained NARMA-L2 model is connected to the newly-trained NARMA-L2 controller and serves as the controlled system. The pre-trained model should be the model with high training accuracy, and it should be replaced by the model ever trained with the highest accuracy. The structure of the controller is shown in Figure 9.11.

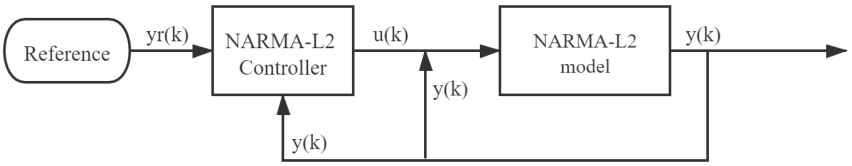
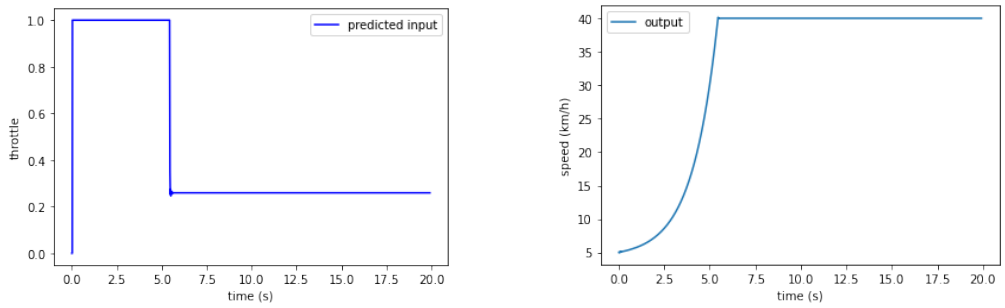


Figure 9.11: Simulation without Webots

In the simulation, the generated control input and the output of the accurate NARMA-L2 model are shown in Figure 9.12. The desired speed is set to 40km/h, and the simulation starts at 5km/h because if the initial speed is under 5km/h, the controller will not converge. The reason is perhaps because of the relatively low precision of the model near 0.

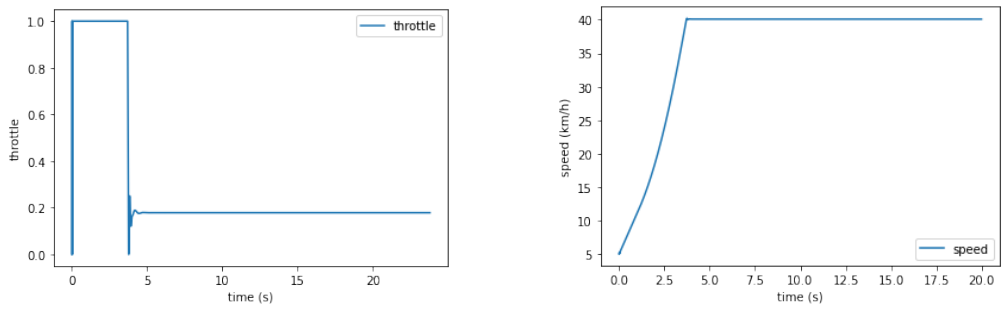


Throttle Speed

Figure 9.12: Throttle and speed for the cruise control system without Webots

Figure 9.12 shows that the vehicle keeps full throttle until the speed reaches 40km/h, the

desired cruise speed. Then the throttle sharply decreases to a balance point (about 0.25 in Figure 9.12), and the speed will be kept at the cruise speed. From the figures above, it can be found that the controller has the proper functionality to accelerate the controller and keep the speed setting in the non-Webots environment. Therefore, the trained model is imported into the Webots controller. Before the car reaches the initial speed, the throttle is set to the maximum value (1). The result is shown in Figure 9.13



Throttle Speed

Figure 9.13: Throttle and speed for the cruise control system in Webots

From the figures, we can find that although the performance between the evaluation system with and without Webots slightly differs, the action of the controller is similar and that the simulation method is proved effective. The NARMA-L2 controller is proved to be effective in Webots, and compared with the PID controller, the change of the throttle is much sharper, which means that the controller finds the equilibrium point much faster than the traditional PID controller.

9.4 Parameter shift and its solution

Firstly, we need to check the performance of the controller without changing it as the environment changes, to find if the adaptability of the NARMA-L2 can compensate for the shift of parameters and keep acceptable performance. Consider the following scenario:

The road is composed of a plain surface and a slope, which goes uphill first and then downhill. The grade of the slope is about 10%. The vehicle runs on this road using the controller trained on plain road, completes acceleration and reaches cruise speed on the plain road, then goes up and down on the slope. The speed and throttle are recorded as follows (cruise speed is 30km/h):

In Figure 9.15, we can discover that the throttle still shows some extent of adaptation and changes as the time and slope grade go, but the fluctuation and vibration of the throttle are high, and the speed is fluctuating between 28km/h and 35km/h. Therefore, Figure 9.15 shows that the NARMA-L2 controller can adapt to the changes to some extent, but the performance of the controller cannot be guaranteed and the result is not satisfactory.

Therefore, a solution for the parameter shifts is raised that the neural networks in the controller are re-trained during the driving process when the environment changes, and when the training accuracy meets the requirement, the model will replace the older one in the controller and complete the update process. The data used for training is collected when driving.

To prove the feasibility of the solution, we need to prove that the performance improved after retraining. Consider an uphill slope with 10% grade, and a `BmwX5` is running on the slope. The default brake intensity is set to 0. The retraining process uses the same network structure as the original training process, which consists of one hidden layer with 10 neurons. The default parameters are listed in Table 9.2.

The performance of the old controller and the controller retrained is shown below. The figure for speed discards the acceleration part and records the output from 2.8s to improve visibility:

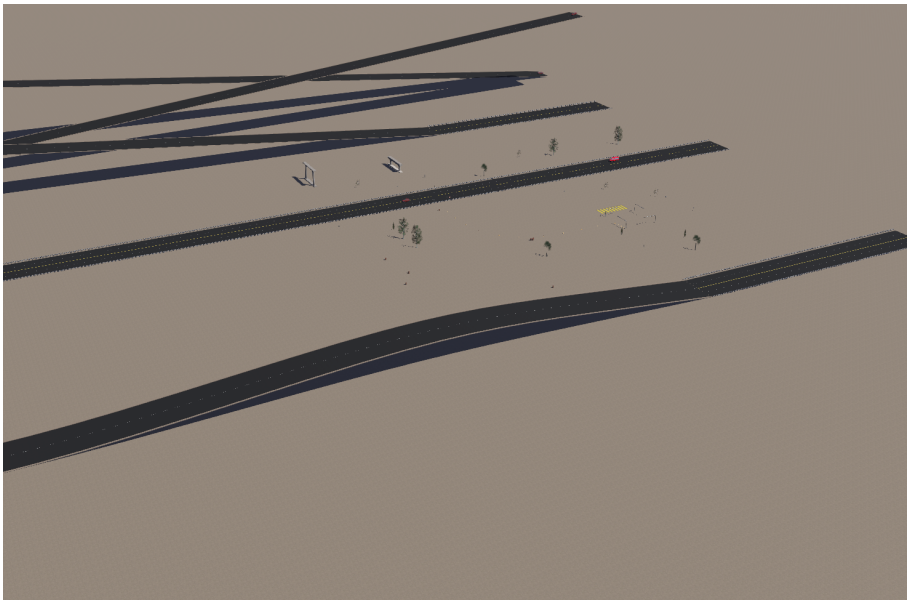
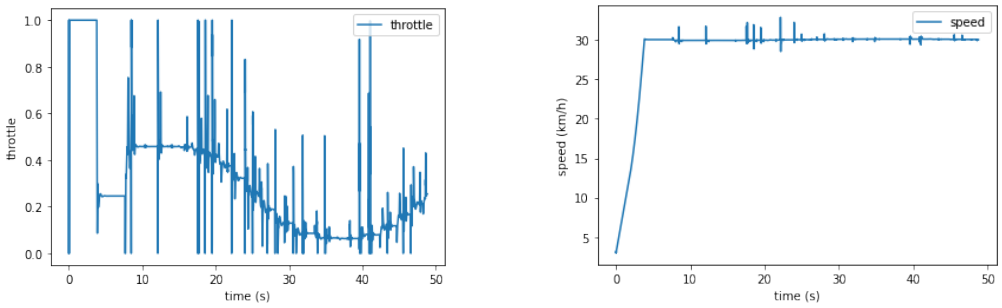
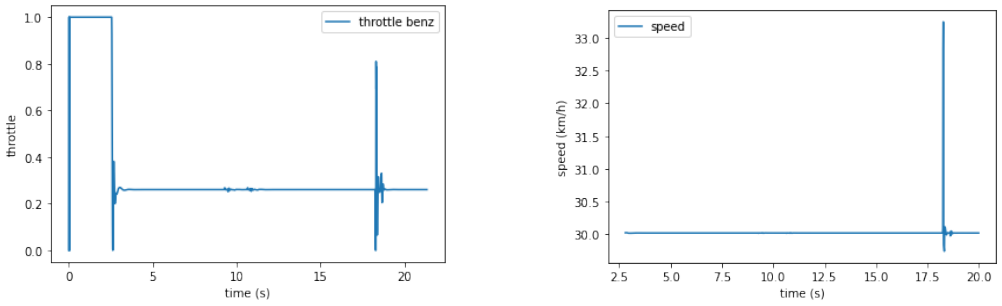


Figure 9.14: A scenario with slopes



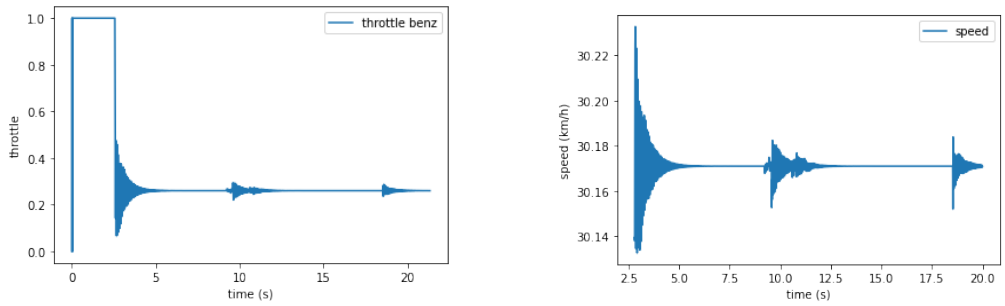
Throttle Speed

Figure 9.15: Throttle and speed on an up-down slope



Throttle Output on an uphill slope using old controller

Figure 9.16: Throttle and speed on an uphill slope using the old controller

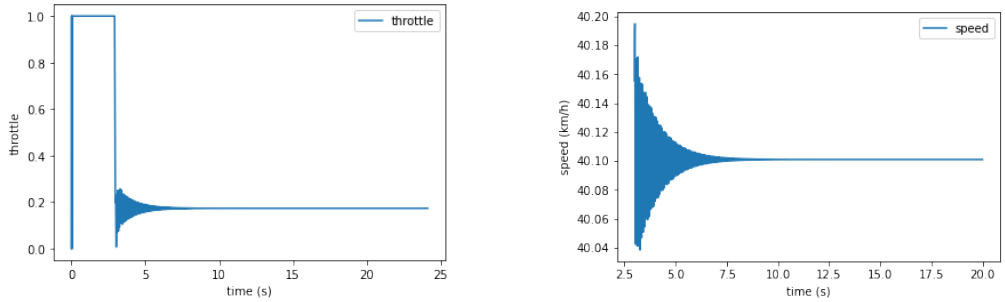


Throttle Speed

Figure 9.17: Throttle and speed on a uphill slope using the new controller

The Mean square error of the old controller, based on the equilibrium point of the speed is 0.0154, while the MSE of the new controller is 3.8e-5. And from the figures, we can find that the fluctuation of speed on the original controller can be up to 3km/h (10%), but in the newly trained controller, the fluctuation can be reduced to 0.08km/h (0.26%). We can summarize from the result that for the systems that change the damping coefficient b, re-training the model has better performance than using the original model.

Consider another scenario, where a `MercedersBenzSprinter` replaces `BmwX5` runs on a plain surface. According to Table 8.1, the `MercedersBenzSprinter` differs from `BmwX5` in mass and the relation between torque and throttle angle. It can simulate the situation when the vehicle load or gear settings change. The speed is set to 40km/h.

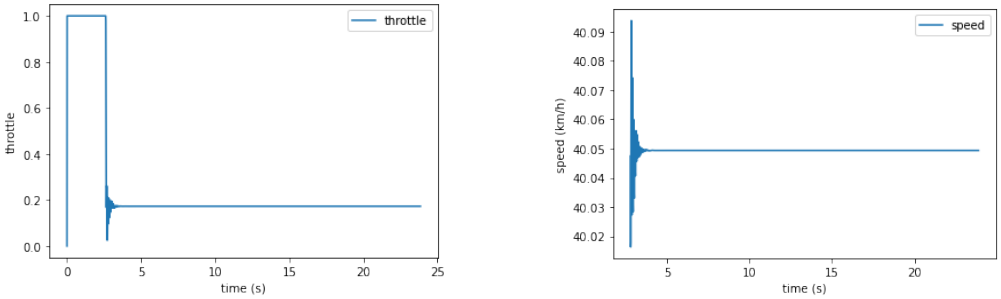


Throttle Speed

Figure 9.18: Throttle and speed on a Benz Sprinter using the old controller

The Mean square error of the old controller is 0.003, while the MSE of the new controller is 9.1e-7. And from the figures, we can find that the response time of the old controller, which is the time for the controller to converge to the equilibrium speed, is much longer than the new controller. We can summarize from the result that for the systems that change the mass m, re-training the model has better performance than using the original model.

We show that the re-training of the model can improve the performance of the controller, and the solution proposed in this thesis is indeed feasible. However, the method requires diverse scenarios and driver operations, and it is hard to implement it in Webots. The implementation will be completed in the future using a comprehensive platform.



Throttle Speed

Figure 9.19: Throttle and speed on a Benz Sprinter using the new controller

Chapter 10

Conclusion

10.1 Conclusion

As is discussed from Chapter 1 to 3, the control performance of a controller will experience fluctuation by parameter changes of the controlled system caused by environmental changes or interaction with other systems. Online model identification and runtime adaptation help to improve the system performance when parameters change by identifying the new system and updating the controller frequently. This thesis discusses the approach of identification and adaptation and elaborates the feasibility of the method with a series of experiments.

In this thesis, we first discuss the NARMA-L2 controller based on system identification and neural network. The mathematical background and detailed implementation of the NARMA-L2 controller are introduced. Then, an XiL simulation framework is designed for validation of the controller, which enables the users to perform simulation using the GUI interface. Furthermore, the NARMA-L2 controller is applied to a cruise control system in Webots simulator. By applying identification and training methods on the controller, the controller for the cruise control system is designed and achieves better performance than the traditional PID controller. For dealing with the parameter changes, online-offline mixed model identification and adaptation is proposed and analyzed.

The work in detail completed in this graduation project is listed below:

1. Do research and literature study about system identification and XiL simulation methods and platforms. Several system identification methods are investigated and evaluated, including the NARMA-L2 controller, which is chosen for the project.

2. Introduce NARMA-L2 model and controller, give mathematical proof for the controller and algorithms for training and controller design of the NARMA-L2, and several cases using NARMA-L2 controller for the linear or nonlinear control system are investigated and evaluated to prove the feasibility of the controller.

3. Apply NARMA-L2 to the cruise control system. Build a verification environment that can do simple simulation without Webots, and realize cruise control in Webots using PROTO nodes integrated with Webots.

4. Propose an online-offline mixed method to deal with the variance of parameters. Do experiments with the changed model to assess the performance of newly trained or previously trained controllers.

5. The XiL simulation platform and toolbox is designed. The toolbox integrated parameter change, platform selection and SDF3 dataflow analysis. Code generation for the controller is also implemented.

From the work conclusion above, it is proved that the NARMA-L2 controller can accurately and effectively identify both linear and nonlinear systems, and its effectiveness has been shown in an actual case, the adaptive cruise control system. The experiments show high potential of adapting the current solution to other control systems with higher complexity.

10.2 Future works

After the completion of the graduation project, there is still some work to do for improvement and further research.

1. To integrate the NARMA-L2 model in the XiL simulation toolbox. Possible methods include calling Python scripts in MATLAB or refactoring the toolbox using Python GUI framework like PyQt.
2. To implement the online solution for dealing with parameter changes described in Chapter 9.4.
3. To extend the controller to other control systems like image-based control.

Bibliography

[1] Chin-Teng Lin, Nikhil R Pal, Shang-Lin Wu, Yu-Ting Liu, and Yang-Yin Lin. An interval type-2 neural fuzzy system for online system identification and feature elimination. *IEEE transactions on neural networks and learning systems*, 26(7):1442–1455, 2014. ix, 4, 5

[2] O. Dubois, J. Nicolas, and A. Billat. Adaptive neural network control of the temperature in an oven. In *IEE Colloquium on Advances in Neural Networks for Control and Systems*, pages 8/1–8/3, 1994. ix, 5, 6

[3] Sajid Mohamed, Dip Goswami, Vishak Nathan, Raghu Rajappa, and Twan Basten. A scenario- and platform-aware design flow for image-based control systems. *Microprocessors and Microsystems*, 75:103037, 2020. ix, 7, 22

[4] Min-Soo Kim, Kwan-Soo Lee, and Sukkee Um. Numerical investigation and optimization of the thermal performance of a brushless dc motor. *International Journal of Heat and Mass Transfer*, 52(5):1589 – 1599, 2009. 1

[5] S. Savin, S. Ait-Amar, D. Roger, and G. Vélu. Aging effects on the ac motor windings: A correlation between the variation of turn-to-turn capacitance and the pdiv. In *2011 Annual Report Conference on Electrical Insulation and Dielectric Phenomena*, pages 64–67, 2011. 1

[6] L. A. Zadeh. From circuit theory to system theory. *Proceedings of the IRE*, 50(5):856–865, 1962. 1, 4

[7] Oliver Nelles. *Nonlinear system identification: from classical approaches to neural networks and fuzzy models*. Springer Science & Business Media, 2013. 1

[8] Shai Ben-David, Eyal Kushilevitz, and Yishay Mansour. Online learning versus offline learning. *Machine Learning*, 29(1):45–63, 1997. 2

[9] V.R. Puttige and S.G. Anavatti. Comparison of real-time online and offline neural network models for a uav. pages 412 – 417, 09 2007. 2

[10] Wittenmark B. Astrom K.J. *Adaptive control*. Dover, 2ed edition, 2008. 2

[11] Wook Hyun Kwon and Seong-Gyu Choi. Real-time distributed software-in-the-loop simulation for distributed control systems. In *Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design (Cat. No.99TH8404)*, pages 115–119, 1999. 2, 3

[12] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan. Modular open robots simulation engine: Morse. In *2011 IEEE International Conference on Robotics and Automation*, pages 46–51, 2011. 3

[13] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004. 3

[14] E. Rohmer, S. P. N. Singh, and M. Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326, 2013. 3, 7

[15] Olivier Michel. Cyberbotics ltd. webots™: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1):5, 2004. 3, 7

[16] K.J. Åström and P. Eykhoff. System identification—a survey. *Automatica*, 7(2):123 – 162, 1971. 4

[17] Karel J Keesman. *System identification: an introduction*. Springer Science & Business Media, 2011. 4

[18] B. A. McElhoe. An assessment of the navigation and course corrections for a manned flyby of mars or venus. *IEEE Transactions on Aerospace and Electronic Systems*, AES-2(4):613–623, 1966. 4

[19] E. A. Wan and R. Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, pages 153–158, 2000. 4

[20] A. U. Levin and K. S. Narendra. Control of nonlinear dynamical systems using neural networks: controllability and stabilization. *IEEE Transactions on Neural Networks*, 4(2):192–206, 1993. 4

[21] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, 1990. 4

[22] K. S. Narendra. Neural networks for control theory and practice. *Proceedings of the IEEE*, 84(10):1385–1406, 1996. 4

[23] Kumpati S. Narendra and Snehasis Mukhopadhyay. Neural networks for system identification. *IFAC Proceedings Volumes*, 30(11):735 – 742, 1997. IFAC Symposium on System Identification (SYSID'97), Kitakyushu, Fukuoka, Japan, 8-11 July 1997. 4, 13

[24] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013. 4

[25] Lu Yingwei, N Sundararajan, and P Saratchandran. Identification of time-varying nonlinear systems using minimal radial basis function neural networks. *IEE Proceedings-Control Theory and Applications*, 144(2):202–208, 1997. 4

[26] Vishwas R Puttige and Sreenatha G Anavatti. Real-time neural network based online identification technique for a uav platform. In *2006 International Conference on Computational Inteligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce (CIMCA'06)*, pages 92–92. IEEE, 2006. 5

[27] Jung-Wook Park, G. K. Venayagamoorthy, and R. G. Harley. Mlp/rbf neural-networks-based online global model identification of synchronous generator. *IEEE Transactions on Industrial Electronics*, 52(6):1685–1695, 2005. 5

[28] Qian Liu, Daqi Zhu, and Simon X. Yang. Unmanned underwater vehicles fault identification and fault-tolerant control method based on fca-cmac neural networks, applied on an actuated vehicle. *Journal of Intelligent & Robotic Systems*, 66(4):463–475, 2011. 5

[29] Kalil Erazo and Satish Nagarajaiah. An offline approach for output-only bayesian identification of stochastic nonlinear systems using unscented kalman filtering. *Journal of Sound and Vibration*, 397:222 – 240, 2017. 5

[30] A. Caponio, G. L. Cascella, F. Neri, N. Salvatore, and M. Sumner. A fast adaptive memetic algorithm for online and offline control design of pmsm drives. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(1):28–41, 2007. 5

[31] Ping Ge and Musa Jouaneh. Tracking control of a piezoceramic actuator. *IEEE Transactions on Control Systems Technology*, 4(3):209–216, 1996. 5

[32] Wei Tang, Lijian Wang, Jiawei Gu, and Yunfeng Gu. Single neural adaptive pid control for small uav micro-turbojet engine. *Sensors*, 20:345, 01 2020. 5

[33] X. Jingjing and L. Jiaoyu. Neural network pid controller auto-tuning design and application. In *2013 25th Chinese Control and Decision Conference (CCDC)*, pages 1370–1375, 2013. 5

[34] Daogang Peng, Hao Zhang, Ping Yang, Yong Wang, and Chumei Xu. Adaptive inverse control for water level of boiler drum based on neural network. In *2006 6th World Congress on Intelligent Control and Automation*, volume 1, pages 2911–2915, 2006. 5

[35] Lucas Nogueira. Comparative analysis between gazebo and v-rep robotic simulators. *Seminario Interno de Cognicao Artificial-SICA*, 2014(5), 2014. 7

[36] Sajid Mohamed, Sayandip De, Konstantinos Bimpisidis, Vishak Nathan, Dip Goswami, Henk Corporaal, and Twan Basten. Imacs: a framework for performance evaluation of image approximation in a closed-loop system. In *2019 8th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–4. IEEE, 2019. 7, 21

[37] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998. 10

[38] Jr. Noman F. Hunter. Analysis of nonlinear systems using arma models. *Eighth International Modal Analysis Conference*, Jan 1990. 12

[39] Kumpati S Narendra and Snehasis Mukhopadhyay. Adaptive control using neural networks and approximate models. *IEEE Transactions on neural networks*, 8(3):475–485, 1997. 13

[40] A. Pukrittayakamee, O. De Jesus, and M. T. Hagan. Smoothing the control action for narma-l2 controllers. In *The 2002 45th Midwest Symposium on Circuits and Systems, 2002. MWSCAS-2002.*, volume 3, pages III–III, 2002. 14

[41] Yue Yang, Cheng Xiang, Shuhua Gao, and Tong Heng Lee. Data-driven identification and control of nonlinear systems using multiple narma-l2 models. *International Journal of Robust and Nonlinear Control*, 28(12):3806–3833, 2018. 17

[42] Sajid Mohamed, Nilay Saraf, Daniele Bernardini, Dip Goswami, Twan Basten, and Alberto Bemporad. Adaptive predictive control for pipelined multiprocessor image-based control systems considering workload variations. In *59th IEEE Conference on Decision and Control (CDC)*, 2020. 22

[43] Sajid Mohamed, Asad Ullah Awan, Dip Goswami, and Twan Basten. Designing image-based control systems considering workload variations. In *58th IEEE Conference on Decision and Control (CDC)*, 2019. 22

[44] Majid Zamani, Soumyajit Dey, Sajid Mohamed, Pallab Dasgupta, and Manuel Mazo. Scheduling of controllers' update-rates for residual bandwidth utilization. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 85–101. Springer, 2016. 22

[45] Sajid Mohamed, Diqing Zhu, Dip Goswami, and Twan Basten. Optimising quality-of-control for data-intensive multiprocessor image-based control systems considering workload variations. In *21st Euromicro Conference on Digital System Design (DSD)*, pages 320–327, 2018. 22

[46] S. De, S. Mohamed, D. Goswami, and H. Corporaal. Approximation-aware design of an image-based control system. *IEEE Access*, 8:174568–174586, 2020. 22

[47] S. De, S. Mohamed, K. Bimpisidis, D. Goswami, T. Basten, and H. Corporaal. Approximation trade offs in an image-based control system. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1680–1685, 2020. 22

[48] S. De, Y. Huang, S. Mohamed, D. Goswami, and H. Corporaal. Hardware- and situation-aware perception for robust closed-loop control systems. In *DATE*, 2021. 22

[49] S. Stuijk, M. Geilen, and T. Basten. Sdf$\hat{3}$: Sdf for free. In *Sixth International Conference on Application of Concurrency to System Design (ACSD'06)*, pages 276–278, 2006. 22

[50] Martin Becker, Sajid Mohamed, Karsten Albers, PP Chakrabarti, Samarjit Chakraborty, Pallab Dasgupta, Soumyajit Dey, and Ravindra Metta. Timing analysis of safety-critical automotive software: The autosafe tool flow. In *Asia-Pacific Software Engineering Conference (APSEC)*, pages 385–392. IEEE, 2015. 22

[51] Write callbacks in app designer. `https://www.mathworks.com/help/matlab/creating_guis/write-callbacks-for-gui-in-app-designer.html`, 2019. 24

[52] UMich. Cruise control: System modeling. `https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=SystemModeling`, 2017. 25

[53] Cyberbotics. Webots documentation: Driver library. `https://cyberbotics.com/doc/automobile/driver-library`, 2020. 26, 29

Appendix A

Source code and documentation

The source code and documentation of the identification model, NARMA-L2 controller and the Webots world, including the controller files and world files, can be found here[1]. The project includes two environments, `train` for training and `vehicle` for simulation and controller design. `train` includes Jupyter Notebook Python files for NARMA-L2 implementation, training and virtual environment for controller design and validation without Webots. `vehicle` includes the world file and built-in libraries and PROTO nodes embedded in each Webots projects. And `vehicle/controller` includes the controller codes in Python that either define a certain input for data collection, or implementation of NARMA-L2 controllers using the model trained before.

The source code and documentation of the XiL framework for validation can be found on the ES Gitlab page[2]. The documentation introduces how to install the framework and configure the environment of the example IMACS controller.

[1] https://git.ics.ele.tue.nl/ecs/mep/chaolun-ma/-/tree/master/cruise_control
[2] https://git.ics.ele.tue.nl/ecs/mep/chaolun-ma/-/tree/master/Control_codes_and_requesting_readme