UNIVERSITY OF CALIFORNIA,
IRVINE

Resource-Aware Predictive Models in Cyber-Physical Systems

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Maral Amir

Dissertation Committee:
Professor Tony Givargis, Chair
Professor Alex Nicolau
Professor Ian G. Harris

2019

DEDICATION

To my parents, Shahin Sadraie and Morteza Amir, for their unconditional love and support, and for always believing in me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my gratitude to my adviser, Professor Tony Givargis. He has helped me become a better and stronger person and has motivated me to develop skills needed to grow as a successful individual in the society.

I would like to thank the rest of my dissertation committee members, Professor Alex Nicolau and Professor Ian G. Harris for their time and support.

I am forever thankful to my parents who have always given me courage, hope and support and to whom I owe all my success. Shahin Sadraie and Morteza Amir, you two are forever my heroes and I am grateful for all your love and kindness that help me reach my dreams. I am deeply grateful to my sisters, Ghazaleh Amir and Maryam Amir for all their love and protection and for always being there for me.

I am grateful to my friends and colleagues at UC Irvine, who had an impact on me during this significant journey. I would like to thank Korosh Vatanparvar, Laleh Aghababaie Beni, Forough Arabshahi, Sajjad Taheri, Bryan Donyanavard, Majid Namaki Shoushtari, Mahdi Abbaspour Tehrani, and Hamid Mirzaei Buini for their friendship, feedback and guidance.

I would like to thank Dr. Hakan Kardes for his invaluable support and words of encouragement during my hardest times. I want to thank Professor Hamid Mahmoodi for all his advice and for setting a good example as a person.

I must thank my dear friends Mahta Alavi, Prousha Tavakol, Raha Borazjani, Tamoochin Khalajhedayati, and Shervin Zalishahr for their support and the great memories we have created together.

I want to thank UC Irvine graduate division, NSF (Grant Number 1563652) and the school of ICS for providing funding opportunities during my PhD program.

I also thank ACM and IEEE for permissions to include parts of chapters 2, 3, 4, and 5 of my dissertation, which were originally published in IEEE International High Level Design Validation and Test Workshop, IEEE International Conference on Computer-Aided Design, ACM Transactions on Design Automation of Electronic Systems, and IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

CURRICULUM VITAE

Maral Amir

EDUCATION

Doctor of Philosophy in Computer Science **2019**
University of California, Irvine *Irvine, CA*

Master of Science in Computer Science **2019**
University of California, Irvine *Irvine, CA*

Master of Science in Computer Engineering **2014**
San Francisco State University *San Francisco, CA*

RESEARCH EXPERIENCE

Graduate Student Researcher **2014–2018**
University of California, Irvine *Irvine, CA*

Graduate Student Mentor (NASA CIPAIR Program) **Summer 2013-2014**
San Francisco State University *San Francisco, CA*

TEACHING EXPERIENCE

Teaching Assistant: Advanced Operating Systems **Summer 2017-2018**
University of California, Irvine *Irvine, CA*

Teaching Assistant: Embedded Systems **Spring 2016-2018**
University of California, Irvine *Irvine, CA*

WORK EXPERIENCE

Data Science Intern **June 2019 - Dec 2019**
Alignment Healthcare USA, LLC *Orange, CA*

REFEREED JOURNAL PUBLICATIONS

Priority Neuron: A Resource-Aware Neural Network for Cyber-Physical Systems **2018**
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)

Switching Predictive Control Using Reconfigurable State-Based Model **2018**
ACM Transactions on Design Automation of Electronic Systems (TODAES)

REFEREED CONFERENCE PUBLICATIONS

Hybrid State Machine Model for Fast Model Predictive Control: Application to Path Tracking **2017**
International Conference On Computer Aided Design (ICCAD)

HES Machine: Harmonic Equivalent State Machine Model Generation Tool for Cyber-Physical Systems **2017**
International High-Level Design Validation and Test Workshop (IHLDVT)

ABSTRACT OF THE DISSERTATION

Resource-Aware Predictive Models in Cyber-Physical Systems

By

Maral Amir

Doctor of Philosophy in Computer Science

University of California, Irvine, 2019

Professor Tony Givargis, Chair

Cyber-Physical Systems (CPS) are composed of computing devices interacting with physical systems. Model-based design is a powerful methodology in CPS design in the implementation of control systems. For instance, Model Predictive Control (MPC) is typically implemented in CPS applications, e.g., in path tracking of autonomous vehicles. MPC deploys a model to estimate the behavior of the physical system at future time instants for a specific time horizon. Ordinary Differential Equations (ODE) are the most commonly used models to emulate the behavior of continuous-time (non-)linear dynamical systems. A complex physical model may comprise thousands of ODEs that pose scalability, performance and power consumption challenges. One approach to address these model complexity challenges are frameworks that automate the development of model-to-model transformation.

In this dissertation, a state-based model with tunable parameters is proposed to operate as a reconfigurable predictive model of the physical system. Moreover, we propose a run-time switching algorithm that selects the best model using machine learning. We employed a metric that formulates the trade-off between the error and computational savings due to model reduction.

Building statistical models are constrained to having expert knowledge and an actual understanding of the modeled phenomenon or process. Also, statistical models may not produce

solutions that are as robust in a real-world context as factors outside the model, like disruptions would not be taken into account. Machine learning models have emerged as a solution to account for the dynamic behavior of the environment and automate intelligence acquisition and refinement. Neural networks are machine learning models, well-known to have the ability to learn linear and nonlinear relations between input and output variables without prior knowledge. However, the ability to efficiently exploit resource-hungry neural networks in embedded resource-bound settings is a major challenge.

Here, we proposed Priority Neuron Network (PNN), a resource-aware neural networks model that can be reconfigured into smaller sub-networks at runtime. This approach enables a trade-off between the model's computation time and accuracy based on available resources. The PNN model is memory efficient since it stores only one set of parameters to account for various sub-network sizes. We propose a training algorithm that applies regularization techniques to constrain the activation value of neurons and assigns a priority to each one. We consider the neuron's ordinal number as our priority criteria in that the priority of the neuron is inversely proportional to its ordinal number in the layer. This imposes a relatively sorted order on the activation values. We conduct experiments to employ our PNN as the predictive model in a CPS application. We can see that not only our technique will resolve the memory overhead of DNN architectures but it also reduces the computation overhead for the training process substantially. The training time is a critical matter especially in embedded systems where many NN models are trained on the fly.

Chapter 1

Introduction

1.1 Cyber-Physical Systems

The term cyber-physical systems (CPS) refers to a generation of systems that are composed of computing devices interacting with the physical world. The expanded capabilities of the physical world through computation, communication ,and control is the key to many emerging advanced technologies. In today's applications, CPS is designed to control physical plants such as industrial machines, land vehicles, medical equipment, spacecraft, jet engines, etc. The control systems that are implemented to manage these complex physical systems also have a relatively high level of complexity.

Sequential methodologies [50] are well-established techniques to cope with the complexity of designing CPSs. The idea is, first, to select a promising physical system, then define the controller and finally address design challenges of the embedded computer system. To this end, system and control engineers have developed novel engineering methods based on time and frequency domain processing, state space modeling, filtering, prediction, optimization, and advanced control technologies. At the same time, computer scientists have pioneered in

advancing computing techniques, programming languages, embedded system architectures. Such sequential separation of decisions reduces the complexity of the design efforts. However, like most greedy approaches, the overall solution is unlikely to be the best possible design due to missed trade-offs between cyber and physical design knobs.

CPS is effectively developed in a real world environment in which its control system is connected to a "real physical system". However, important criterion for efficient development and testing of CPS may include three factors: cost, time-to-market, and safety. The multi-dimensional and demanding design cycle of CPS motivates developers to leverage techniques that reduce development duration and cost. Moreover, difficulty and safety considerations in online testing of CPS with real systems lead to a need for simulation and offline design methodologies with high accuracy.

CPS research has emerged as solution to integrate the science and engineering principles and requires professionals from multiple fields from computer science and network engineering to automation and control to collaborate closely. Emerging CPS solutions is based on developing computer and network systems while monitoring and controlling the physical processes on the basis of environmental perception. This integration facilitates real-time, safe, reliable development of computing devices interacting with the physical systems through network commutations. The data acquisition modules in physical systems collect data through advanced sensing devices in CPS system and pass data to the computing devices. These computing devices complete a given task such as fault detection, signal processing data security processing ,and feedback control through actuation technologies.

1.2 Model-Based Design

Model-based design is a powerful methodology that utilizes mathematical models in CPS design. Modern CPS design approaches is based on synergistic interaction of software, hardware, and physical aspects with equal importance. Therefore, physical models have to be treated as equally important as cyber models. Fast executable models of physical systems are needed especially for Model-based Predictive Control (MPC) or real-time Hardware-In-the-Loop (HIL) simulations.

1.2.1 Model Predictive Control

Model Predictive Control (MPC), also known as Receding Horizon Control (RHC), is an advanced model-based control method. MPC makes explicit use of a model of the physical system to estimate its behavior for a given stream of inputs in a predetermined prediction horizon. The predicted outputs depend on the past inputs/outputs, and the future control signals [17]. As shown in Figure 1.1, these future control signals are calculated by the optimizer taking into account the cost function and enforced constraints. The cost function usually takes the form of a quadratic function of errors between the predicted output signal and the reference trajectory. In the standard approach, Ordinary Differential Equations (ODE) are employed as the predictive model to represent the dynamic behavior of a physical system. Iterative methods to approximate a solution for non-linear ODEs have introduced challenges in the design of embedded MPCs in terms of scalability, performance, and power consumption [31].

The computational overhead in traditional MPC grows exponentially with the length of the prediction horizon [11]. Research shows that a stable MPC controller requires a sufficiently large prediction horizon [47]. On the other hand, short prediction horizons are preferred

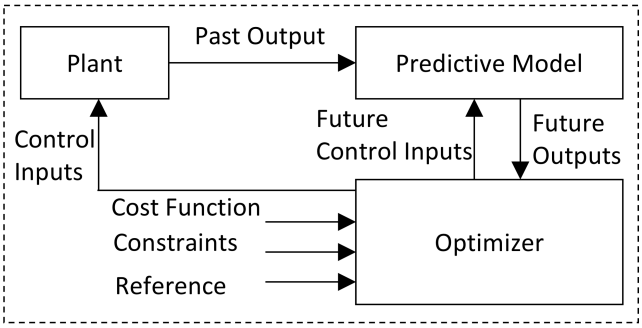


Figure 1.1: Model predictive control loop.

for improved prediction accuracy of predictive models. This is because harmful effects of the poor estimates are amplified over a long prediction horizon time. Here, the problem is addressed by proposing an MPC approach that uses an adaptive prediction horizon with respect to quality measures [23]. However, the numerical effort needed in order to solve the optimal control problem for a long prediction horizon still remains significant.

1.2.2 Physical Models

Physical models that capture and emulate the behavior of real physical systems have gained extensive research attention in CPS design. The dynamic behavior of a physical system is typically calculated from the physical laws governing the mechanical, electrical and thermodynamic attributes of the system. The variations of physical quantities such as motion, velocity, pressure, volume and temperature as a function of time or space, may be captured as a set of equation-driven models, e.g. ODEs. As such, system engineers model physical problems using mathematical equations, and then solve these equations to study the behavior of the targeted cyber physical system.

Complex models of physical systems may be comprised of thousands of non-linear Ordinary Differential Equations (ODE), requiring considerable computing power to execute. These ODE models introduce challenges in terms of scalability, performance, power consumption and accuracy [31, 62]. Discretization methods (Euler, zero-order hold, etc.) are applied

to transform the continuous-time differential equations into discrete-time equivalents, appropriate for numerical computing. The discretized differential equations are solved using numerical algorithms. Iterative solutions are used to solve the non-linear ODE models of physical systems, where a series of linear equations are solved iteratively to converge to the solution for the non-linear ODEs [52]. Therefore, the computation complexity of solving ordinary differential equations may grow with respect to type of the discretization algorithm, numerical ODE solver, number and order of the ordinary differential equations in the physical model. Moreover, the demand for higher accuracy and more mathematically sound CPS solutions cause an increase in resource and energy consumption that must be taken into account during the design cycle [59].

The work in [9] proposes a state space model of a physical system augmented with state and output disturbance variables. The dynamic behavior of a physical system can be described using state machine-based models. The state machines are comprised of states and transitions between those states that are triggered with respect to conditional expressions or predicates. State machines may be used to break complex systems into manageable states and state transitions. Therefore, state-space representation of physical systems can be reformulated as concurrent state machines with time-interval behavior, where a global clock conducts the trigger to update the state variables and output actions.

Another method that is proposed to handle the computational issue associated with MPC systems is to use accelerated predictive models of the physical system. Different variants of NNs (e.g., recurrent neural networks [14]) hold promising performance for time-series prediction as they can easily be built to predict multiple steps ahead all at once. These models are well-known to have the ability to learn linear and non-linear relations between input and output variables without prior knowledge [27]. However, the use of NN models for long prediction horizon MPC problems could raise scalability and computational complexity challenges. The state-of-the-art methodologies are focused on reducing the size of the NN

models without significantly affecting the performance [70, 73, 97]. These methodologies leverage the intrinsic error tolerance property of the NN models due to their parallel and distributed structure. Therefore, model reduction schemes could be exploited to employ the NN as the predictive model in the MPC loop. Several recent studies have focused on rescaling the size of the NN to adjust the resource usage on the embedded platform with respect to response time, power, and accuracy targets [74]. In other words, several sizes of the neural network are available at runtime to manage resources for inference time-, safety-, and energy-constrained tasks. Moreover, continuous learning of neural networks in data-driven modeling [87], transfer learning techniques [44], and adaptive modeling [38] impose significant training-time constraints at *runtime*.

1.3 Reconfigurable Predictive Models

A real physical system is under constant change from the effects of the environment. Moreover, limited hardware resources impose a burden on the development of CPS applications. In model-based design applications such as MPC, the complexity of the model under control has a direct influence on the global performance of the system. Specifically, different levels of complexity for the target physical system shall be provided by the user for a specific application. The work in [100] evaluates the performance of a hybrid controller to steer a car in straight and curved trajectory segments. It suggests employing a relatively more advanced model of the vehicle dynamics [72] in curvature path as opposed to fast and simple kinematic model of the vehicle to follow straight lines on the path. Therefore, it has been recognized that predictive models that can be reconfigured to adjust their accuracy from coarse-grained time critical situations to fine-grained scenarios in which safety is paramount are central in designing resource-constrained cyber-physical systems.

1.4 Thesis Contributions and Organization

In this dissertation, we propose novel abstraction techniques to address trading off granularity against complexity in predictive models for cyber-physical systems. Our proposed abstraction formulations can be dynamically reconfigured to different precision granularities at runtime. The organization of the rest of this thesis can be summarized as follows:

- In Chapter 2 a Harmonic Equivalent State (HES) Machine model generation framework is proposed. This model captures sampled data of a physical signal as the input in respective time windows. HES Machine generates reconfigurable state-machine model of the physical system with an intrinsic disturbance feature to adjust the overall model accuracy with respect to proposed tuning parameters for dynamic accuracy. Moreover, the execution time of the model may be adjusted in tradeoff with accuracy in order to adapt to coarse-grained time critical situations or fine-grained safety critical scenarios. The main contribution of the proposed modeling framework is the inclusion of frequency domain properties in signal synthesis to adopt the reconfigurability feature to the model. Also, as opposed to ODE equivalents, the proposed framework do not perform compute-intensive and iterative tasks to solve the proposed physical model. The input to this model is constrained to be periodic.

- In Chapter 3 machine learning is employed to use the previously introduced *HES Machine* model in non-periodic runtime applications. That is, the output of the proposed physical model can adapt to variations in inputs at runtime. ODE models are employed to train the proposed model at design-time. The effectiveness of the proposed model is evaluated in a closed-loop MPC for path tracking application. The performance of the model is compared with state-of-the-art ODE-based models, in terms of execution time and model accuracy.

- in Chapter 4 a better implementation of HES Machine model is proposed in that its

performance in terms of execution time and model accuracy is improved. The machine learning model is modified to better estimate the dynamic behavior of the physical system. A novel switching model predictive control methodology is proposed based on HES Machine as the predictive model. Machine Learning techniques are employed to design this runtime switching algorithm that determines the optimal granularity level of the current predictive model in use. Simulation experiments are conducted to evaluate the switching controller in a path tracking application containing curved and straight routes.

- in Chapter 5 a reconfigurable Neural Networks (NN) model is proposed. This NN model can be reconfigured to various network sizes at runtime while storing only one set of weight parameters for memory efficiency. a training algorithm is developed that controls the priority of each neuron in the computation of the model's output. Regularization techniques are applied to enforce a priority on each neuron. The neuron's ordinal number is considered as our priority criteria in that the priority of the neuron is inversely proportional to its ordinal number. Therefore, the NN model can be reconfigured to smaller sizes by eliminating low priority neurons. This approach allows the trade-off between the model's computation time and accuracy in resource-constrained systems.

- In Chapter 6, the thesis is concluded and future research directions are addressed.

Chapter 2

HES Machine: Harmonic Equivalent State Machine Model

2.1 Introduction

In this chapter, we present a framework to generate a Harmonic Equivalent State (HES) Machine model of the physical systems. One of the merits of the proposed state machine-based model is that the state machines can eliminate execution of compute-intensive and iterative tasks for describing the behavior of the physical systems. The model accommodates reconfigurable parameters that allow the user to have trade-off between accuracy and execution time in CPS design. For validation purposes, we compare our model performance with state-of-the-art models in terms of execution time and accuracy. The simulation results indicate that our generated HES model executes 38% faster than ODE-based equivalent model with same level of model accuracy.

Model-based design in Cyber-Physical Systems (CPS) provides abstraction and modeling techniques to integrate the dynamics of the physical processes with software and communi-

cation components. As opposed to desktop computing, CPS should be dynamically reconfigurable and adapt to changes in the environment. Application-specific disturbance models may be included to predict the effect of unknown physical disturbances that perturb system behavior and incorporate these effects on the input and state variables for control system design. Complex CPS applications such as in industrial machines, land vehicles, medical equipment, spacecraft, jet engines require new computer-aided methods for modeling, simulation and offline design. These methodologies are influenced by the need for lower time to market and higher quality, reliability and safety for the CPS design.

In the literature, some research has applied Hardware-in-the-Loop (HIL) real-time simulation as a technique to improve estimation accuracy and validate the developed strategies. In real-time simulation methods, the input and output signals show the same time-dependent values as the real dynamic system. The HIL technique aims to model the real world scenarios in an abstracted environment in which the "real physical system (plant)" is replaced with the "simulated physical system". The models of the physical systems may be employed to emulate their real behavior with regards to the laws of physics and enable execution of test scenarios that would be prohibitively dangerous in a real system. Moreover, the physical model should account for the impact of measurable and unmeasurable intruding components caused by the surrounding environment (e.g. wind, noise, etc.) in order to evaluate and verify the robustness of the system under test. Therefore, dynamic model reduction in terms of accuracy may benefit HIL in emulating real-life scenarios during testing and verification.

Figure 2.1 illustrates the application of a real-time HIL simulation to test the performance of the *Controller Unit* in a closed-loop powertrain system model [86]. The *Powertrain* block includes a group of fully assembled components, e.g. engine, transmission, drive shafts, differentials, etc., that generate power and transfer it to the road surface. The power consumption in the loop is also dependent on the speed of the vehicle which may be modeled as a time-varying variable in a *Driving Route Model* block. For this purpose, the standard

driving cycles such as NEDC, ECE and UDDS [90, 91] as a set of sampled data from the environment may be applied.

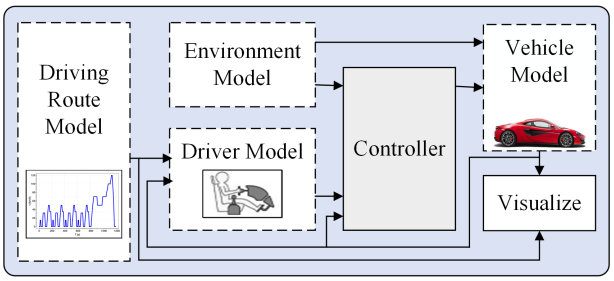


Figure 2.1: Hardware-in-the-loop testing for power train system model.

Another use of model-based design in CPS application is Model Predictive Control (MPC). MPC systems are a class of control algorithms that estimate the behavior of the physical system under control through the use of computational models. The control inputs are optimized to drive the predicted outputs of the model towards the desired trajectory. Closed-loop performance of MPC algorithms is directly correlated with the accuracy of the physical model. Two general techniques to eliminate the steady-state offset error in the closed-loop systems are: 1) including the tracking error in the objective function of the controller, 2) augmenting the predictive model with a data-based disturbance model [71]. Most industrial MPC applications add a constant step disturbance to the output of the physical model to consider the impact of the disturbance in the closed-loop system. The work in [66] proposes a robust MPC algorithm in which a linearized model of a ship is integrated with a wind disturbance model to solve the problem of the ship's control actions in the presence of wind disturbance. This approach requires the user to design a disturbance model and integrate it in the loop with predictive model of the physical system [96]. The state-of-the-art modeling techniques for HIL simulations and MPC applications followed by our contributions are summarized in Section 2.2.

2.2 Related Work

In control system study, the model of the physical system is developed to conform with dynamical system analysis and control system design requirements; that is, simplification and adaptation with respect to state of the system is required [77]. In model-based design applications such as MPC or real-time HIL simulations, the complexity of the model under control has direct influence on the global performance of the system. Specifically, different levels of complexity for the target physical system shall be provided by the user for a specific application. The work in [26] proposes an integrated library of electro-hydraulic models with different complexities. The purpose of the work is to provide the appropriate model with regards to the domain and timing requirements in design-time. However, run-time dynamic disturbance caused by the environment remains neglected.

Complex physical system models may be implemented as thousands of ODEs. The ODE description of a system requires approximations via solver methods such as Euler and Runge-Kutta, to be suitable for computations in computing devices [78]. The demand for more accurate and mathematically sound CPS solutions, cause an increase in resource utilization and energy consumption [59]. Research in model-based design techniques for CPS have introduced solutions to overcome some of the challenges induced by the complexity of ODE models. One approach is to implement the ODEs on Field-Programmable Gate Arrays (FPGA) using Lookup Tables (LUTs) to speed up simulation and enable parallel execution [45]. In general, even though the FPGA implementation of ODE models may improve the execution efficiency for real-time applications, it has implementation challenges regarding limited resources especially for complex ODE models. Hence, a better approach of modeling and solving of ODE may be required to reduce the complexity not only on FPGAs but also on general CPUs.

A state space representation of ordinary differential equations is described in [28] to obtain

a discrete-time solution prior to FPGA implementation. Here, a state space meta model is introduced to model the ordinary differential equation and the respective discrete-time solution. Atlas Transformation Language (ATL) [49] is employed as the framework to implement the model transformation. Here in this work, the experimental results are based upon simple first order differential equations and their technique may not be applicable to more complex ODE physical models. Moreover, the proposed meta models may still carry the complexity of thousands of ODEs, resulting in an implementation overhead that is prohibitive in most constrained system architectures.

The work in [48] proposes a state-based heart model generated from real specifications to be used in a closed-loop system. Implantable cardiac pacemakers monitor and repair the abnormalities in heart rhythms. HIL simulation of a pacemaker is essential to test and verify its functionality with respect to a heart model prior to real implantation. The heart model is implemented in Simulink environment and the HDL coder toolbox is used to generate Verilog code for hardware implementation. The proposed approach is application specific which requires user expertise to implement the model of the heart. Moreover, relying on the HDL coder toolbox for more complex models may require fundamental modifications in the generated Verilog code.

One solution to challenges that arise from the complexity of the ODE-based physical models is frameworks and associated tools that automate model generation and transformation for the target application [75]. Model transformations conduct automated and semi-automated mapping of one or couple of models into another alternative models in order to incorporate flexibility and compatibility in model-based design for CPS [6].

2.3 Contributions

In this work we present an automation framework to generate dynamic state machine model of a physical system augmented with a disturbance feature for Cyber-Physical Systems (CPS) applications. The model accommodates reconfigurable parameters that allow the user to have tradeoff between accuracy and execution time in CPS design. The accuracy of the physical signal may get adjusted during runtime to adopt to the system performance and robustness in the case of sudden changes that may impact the system dynamics. Our contributions in this work can be summarized as follows:

1. Designing a dynamic reconfigurable state machine model for targeted physical systems.
2. Providing tunable parameters to adjust the granularity of the generated model for adaptation to coarse-grained time critical situations or fine-grained safety critical scenarios.
3. Develop an automated framework to generate the model and its executable C code. The code may be implemented in a hardware-in-the-loop for final system testing and integration. Design objectives, model accuracy, and execution time, facilitate the evaluation and verification of the model for embedded systems implementation.

The rest of the chapter is organized as follows; Section 2.4 describes the proposed automated framework that captures physical systems as a set of generated state machine equivalents. We demonstrate the performance of our framework using two benchmarks and present the results in Section 2.5. Finally, we state our conclusions in Section 2.6.

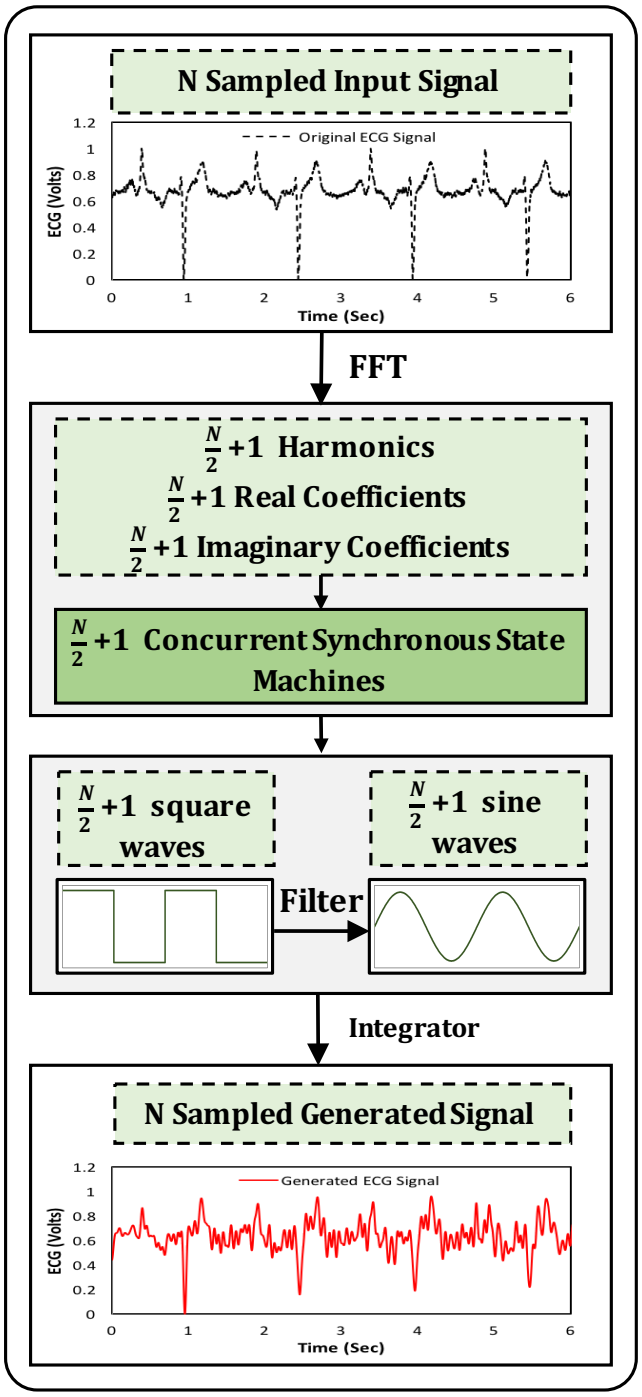


Figure 2.2: High level architecture of the proposed framework illustrates the HES machine model generation process.

2.4 Methodology

The high-level architecture of the proposed framework is depicted in Figure 2.2. The whole framework is based on the concept of signal decomposition and synthesis to generate a reconfigurable state machine model of a target physical system. The input to the framework is sampled signal of size N. Fast Fourier Transform (FFT) algorithm is employed to decompose the signal and derive the frequency information. A synthesis algorithm is presented to integrate the decomposed components of the physical signal in the form of a set of concurrent state machines. The synthesis algorithm employs $(N/2+1)$ inverse of the signal harmonic frequencies and respective FFT coefficient values, as the periods and output magnitudes of concurrent state machines respectively. Band-pass filter is used to translate the output square waves of the concurrent state machine models into sinusoidal signals. The sinusoidal output signals, one per state machine, represent the signal harmonic components. Finally, the harmonic components are integrated to generate a dynamic state machine model for the target physical system. The decomposition (analysis) and synthesis algorithms are described in details in the following sections.

2.4.1 Decomposition

The input to the proposed framework is N number of samples for a given physical signal in time windows of length T. The FFT algorithm is used to derive the frequency spectrum of the physical signal on each time window. Later on, this frequency domain information is employed to synthesize the signal into a state machine model representation.

Frequency Domain Information

The FFT algorithm on a sampled signal of size N decomposes the signal into a series of (N/2+1) sine and cosine wave components which are referred to as basis functions. The process of calculating the frequency domain information of the signal from time domain representation is called decomposition and the inverse process is signal synthesis [79]. The basis functions are a set of sine and cosine waves oscillating at signal harmonic frequencies. For a sample signal represented as array $x[]$ of size N in time domain, the FFT algorithm calculates the frequency domain signals $X[]$ as two arrays of size (N/2+1). The arrays contain the coefficients (amplitudes) of the sine and cosine components as imaginary part $imX[]$ and real part $reX[]$ of $X[]$ respectively for harmonic frequencies $frX[]$. The output values of the FFT algorithm, $frX[]$, $reX[]$, and $imX[]$ are defined as input parameters for the subsequent synthesis process.

2.4.2 Synthesis

The synthesis function for sampled signal $x[i]$ of size N is represented in equation 2.1 [79]. The arrays $re\overline{X}[]$ and $im\overline{X}[]$ are the normalized coefficients of the sine and cosine waves with index k running from 0 to $N/2$ for the respective harmonic frequencies.

$$x[i] = \sum_{k=0}^{N/2} re\overline{X}[k]cos(2\pi ki/N) + \sum_{k=0}^{N/2} im\overline{X}[k]sin(2\pi ki/N) \tag{2.1}$$

HES Machine Synthesis Algorithm

State machines can be used to break complex systems into manageable states and state transitions. Therefore, the state machine model of computation fits the synthesis function

components as concurrent state machines with time-interval behavior. A global clock conducts the trigger to update the state variables and output actions. The components of the physical signal may all be generated by a five-state synchronous harmonic state machine (HES Machine).

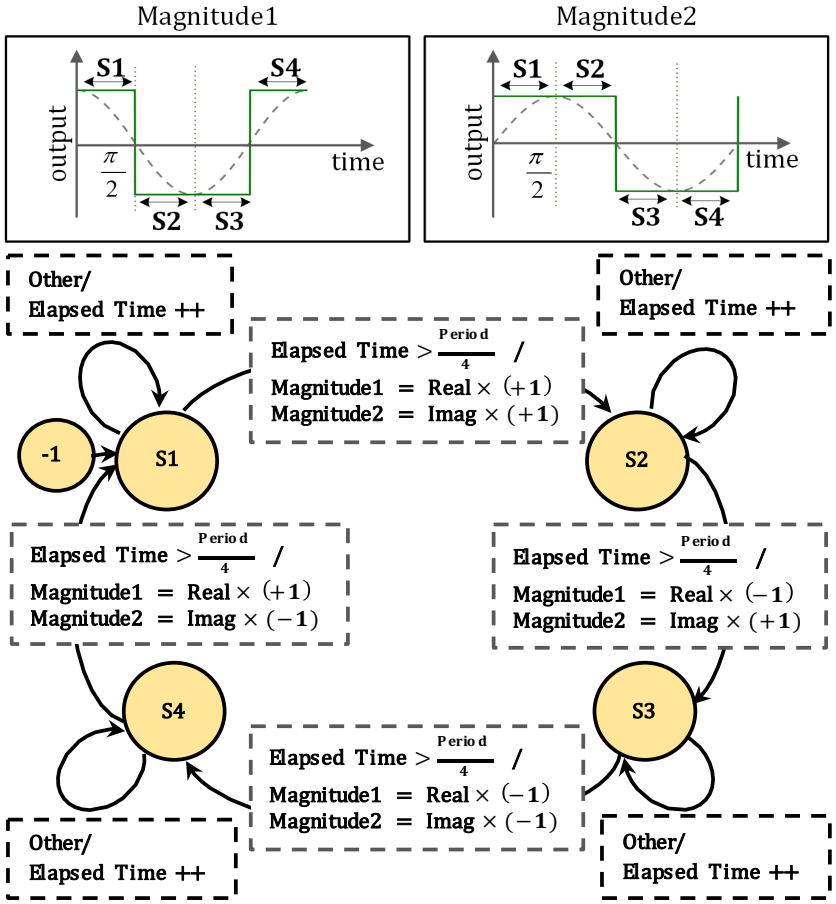


Figure 2.3: 5-State synchronous state machine with the inverse of the signal harmonic frequency as the period.

The architecture of the five-state synchronous harmonic equivalent state machine is depicted in Figure 2.3. Each state machine is designed to represent a harmonic frequency component of the physical signal. Outputs of each state machine are two square wave signals approximating the sine and cosine components in equation 2.1. The square waves will later be integrated into the synthesis function. Inverse of the harmonic frequency is the period and FFT coefficient values are the magnitudes for the corresponding square waves. One period of the square

wave signals is divided into four phases that are represented by the outputs of the states $S1$, $S2$, $S3$, and $S4$. The transitions between theses four states and phases are triggered at each $Period/4$ elapse of time. Finally, $(N/2+1)$ harmonic equivalent concurrent state machines are integrated to synthesize the physical signal. We have utilized this five-state synchronous state machine architecture for our model to highlight the strength of state machines in representation of physical systems.

Algorithm 1 illustrates the structure of the *Tick* function for the executable state machine model of computation. *HES[i]* represents a data structure that includes associated data values per harmonic state machine. Here, i is the index for harmonic state machine ranging from 0 to $(N/2+1)$. The parameter HES_{size} stores the number of concurrent harmonic state machines which are synthesized in a signal synthesis process and may be selected as framework parameters for design configuration. The output array values computed by the FFT algorithm, $reX[]$ and $imX[]$ and $frX[]$, are placed in the *HES* data structure to represent $HES[i].real$, $HES[i].imag$ and $HES[i].period$ respectively. The variable $HES[i].elapsedTime$ is tracked on each call of the *Tick* function. When $(HES[i].elapsedTime \geq HES[i].period/4)$ condition evaluates to true, a state transition occurs and an output action is determined with respect to the current state. N samples of signals are fed into the HES machine model generator in intervening time windows of T. Each execution of the *Tick* function updates the $HES[i].elapsedTime$ variable by adding T_{res} values. The values for the new time window are evaluated when the $HES[i].elapsedTime$ variable surpasses the value T and resets to zero.

The generated square waves are to be translated into sinusoidal equivalents to represent the sine components of the original physical signal. A band-pass filter is applied to attenuate the unwanted square wave frequencies. $(N/2+1)$ sinusoidal signals are integrated to synthesize the decomposed signal according to Equation 2.1. The tool generates an executable C code in state machine representation for the physical signal to be implemented on a target platform.

ALGORITHM 1: Global Tick Function

```
Input: index of the state machine i
global variable HES
global variable magnitude1, magnitude2
const TimeResolution
switch HES[i].state do
    case −1
        HES[i].state = S1
    case S1
        if HES[i].elapsedTime ≥ HES[i].period/4 then
            HES[i].state = S2
            HES[i].elapsedTime = 0
        else
            HES[i].state = S1
    case S2
        if HES[i].elapsedTime ≥ HES[i].period/4 then
            HES[i].state = S3
            HES[i].elapsedTime = 0
        else
            HES[i].state = S2
    case S3
        if HES[i].elapsedTime ≥ HES[i].period/4 then
            HES[i].state = S4
            HES[i].elapsedTime = 0
        else
            HES[i].state = S3
    case S4
        if HES[i].elapsedTime ≥ HES[i].period/4 then
            HES[i].state = S1
            HES[i].elapsedTime = 0
        else
            HES[i].state = S4
    otherwise
        HES[i].state = −1
switch HES[i].state do
    case S1
        magnitude1[i][HES[i].N1] = HES[i].real × 1.0 magnitude2[i][HES[i].N1] = HES[i].imag × 1.0
    case S2
        magnitude1[i][HES[i].N1] = HES[i].real × −1.0 magnitude2[i][HES[i].N1] = HES[i].imag × 1.0
    case S3
        magnitude1[i][HES[i].N1] = HES[i].real × −1.0
        magnitude2[i][HES[i].N1] = HES[i].imag × −1.0
    case S4
        magnitude1[i][HES[i].N1] = HES[i].real × 1.0 magnitude2[i][HES[i].N1] = HES[i].imag × −1.0
    otherwise
        magnitude1[i][HES[i].N1] = HES[i].real × 1.0 magnitude2[i][HES[i].N1] = HES[i].imag × 1.0
HES[i].elapsedTime+ = TimeResolution
HES[i].N1 + +
return
```

HES Machine Tuning Parameters

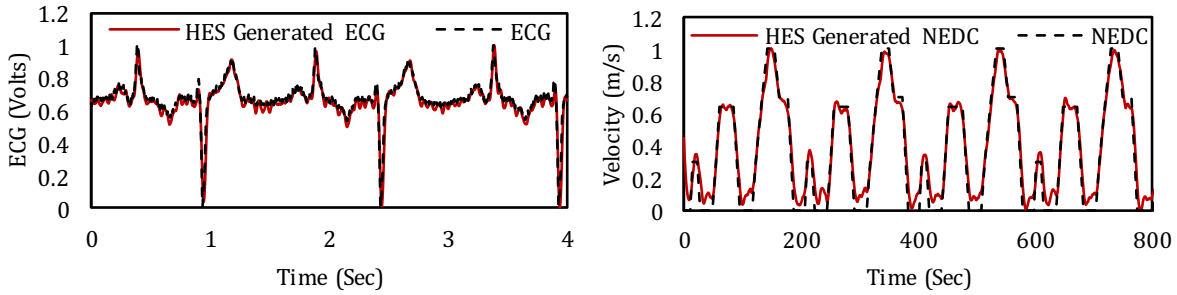
Two tuning parameters HES_{size} and T_{res} are proposed to adjust the HES Machine model in order to meet system requirements (e.g., accuracy and timing).

1. **Machine Size ($\boldsymbol{HES_{size}}$)** is the number of harmonic concurrent state machines to be integrated during the synthesis process ranging from 1 to (N/2+1). The model accuracy may be adjusted with respect to this parameter by inclusion/elimination of certain harmonic frequencies.
2. **Time Resolution ($\boldsymbol{T_{res}}$)** parameter indicates the smallest time unit in the proposed framework by which the generated state machine will be executed. The proposed framework tracks the value of T_{res} as an actual wall-clock (real) time. The parameter T_{res} specifies the timer values for the periodic programmable interval timers to trigger the interrupt service routine (ISR).

2.5 Experimental Results

2.5.1 Implementation and Setup

Simulation experiments are conducted using data for real physical signals and the global clock of the state machine model is updated by interrupt handlers of the operating system. The framework is implemented using C/C++ programming language in order to enable it to be highly portable for compilation and execution. The process from data acquisition to model generation is automated and reconfigurable with respect to model parameters. Our specific experiments were performed on a PC with a quad-core Intel Core i5 and 8 GB of DDR3 RAM. The performance of the proposed model generation framework is evaluated using two examples of ECG signal and NEDC signal. It needs to be noted that one of



(a) Comparing the original ECG signal with the HES Machine generated output signal.

(b) Comparing the original NEDC signal with the HES Machine generated output signal.

Figure 2.4: HES Machine's generated signal of ECG and NEDC.

the merits of our framework is its applicability to any example and application of physical signals.

- **ECG Signal:** The electrocardiogram (ECG) digitized signal is provided by PhysioBank [30] as the reference signal. The signal consists of 7200 double-sized sample values recorded with 720 Hz sampling frequency and 12-bit resolution in T=10-seconds window of time. The model is reconfigured at run-time to serve for HIL testing of implantable medical devices such as pacemaker, smart ECG monitor, etc. [10].
- **NEDC Signal:** The New European Driving Cycle (NEDC) is selected from driving cycle standards (ECE, UDDS, etc.) that are typically employed in model-based vehicle design applications [90]. The NEDC signal contains vehicle velocity data that is captured from a Simulink block [1]. The signal is 8192 double-sized values sampled at frequency of 10 Hz in T=819.2-seconds window of time. The generated dynamic HES Machine model may be reconfigured at run-time to emulate the system behavior in presence of environment disturbance or mis-prediction of trajectory.

The generated signal models of the proposed framework for ECG and NEDC examples are compared in Figures 2.4(a) and 2.4(b) with their original signals. The results justify the validity of our proposed model generation framework for signal synthesis and state machine model generation with average of 0.1% error in model accuracy.

2.5.2 Analysis and Verification

Performance Metrics

Two performance metrics of execution time and precision are considered for comparing the performance of our HES Machine model with state-of-the-art models.

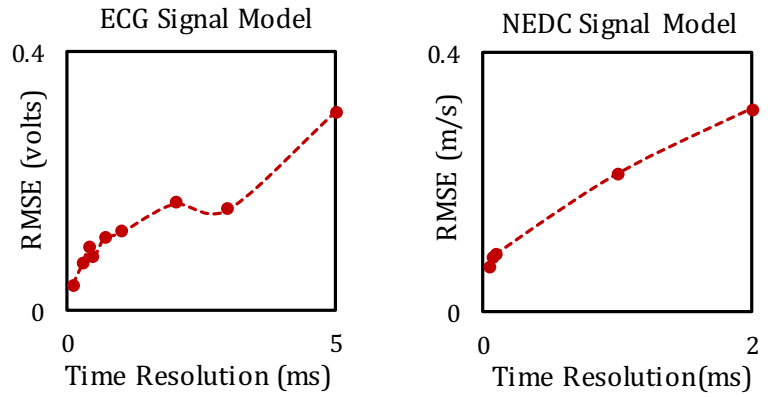
- **Execution Time:** is the time required by the computer to perform a given set of computations.
- **Root Mean Squared Error (RMSE):** is the quality factor to measure the error between the values evaluated by the model and the corresponding expected values for N number of samples.

$$RMSE = \sqrt{\frac{(\sum(Expected - Evaluated)^2)}{N}} \tag{2.2}$$

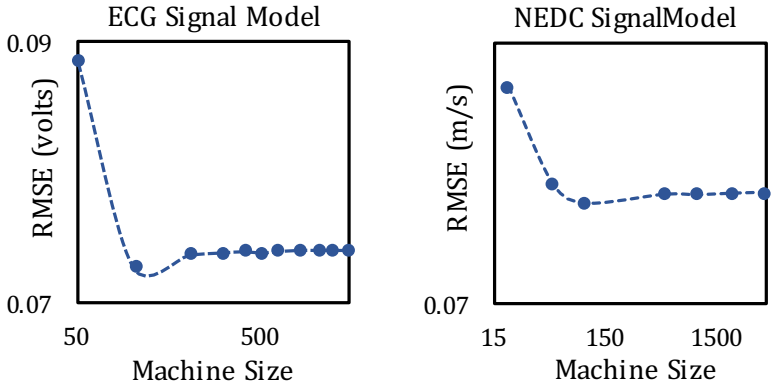
Parameter Analysis

The performance of the proposed state machine model generation framework is evaluated under variations of two parameters by which the model may be configured:

1. **Time Resolution (T_{res}):** Figure 2.5(a) illustrates the change in model accuracy for NEDC and ECG examples with respect to variations in T_{res} parameter values. The results shows improvement in model accuracy for smaller values of T_{res} in case of need for finer physical model.
2. **Machine Size (HES_{size}):** the variations in model accuracy with respect to different values of parameter HES_{size} is illustrated in Figure 2.5(b). Here, The accuracy of the



(a) "Time Resolution" analysis considering the RMSE metric, for ECG and NEDC.



(b) "Machine Size" analysis considering the RMSE metric, for ECG and NEDC.

Figure 2.5: Analyses for different parameter configurations.

model improves for larger values of HES_{size}. In our experiments, the harmonic frequencies and their corresponding state machines are sorted in ascending order to select HES_{size} number of state machines for integration. Other selection algorithms may be applied in accordance with target application which may further improve the precision of the generated model.

Comparison to State-of-the-Art

We evaluated the performance of the proposed framework and generated model in comparison with an ODE-based ECG signal generator, ECGSYN [65]. The model in [65] emu-

lates the quasi-periodic waveform of the ECG signal by tracing around a limit cycle in x-y plane. The ECG signal is generated by using a series of exponentials formulated to follow PQRST-waveform in the z-direction. (P, Q, R, S, T) represent the peaks in ECG signal for one complete heartbeat. The model of motion dynamics is defined as a set of following differential equations

$$\dot{x} = \alpha x - \omega y \tag{2.3}$$

$$\dot{y} = \alpha x + \omega y \tag{2.4}$$

$$\dot{z} = -\sum_{i \in P,Q,R,S,T} a_i \Delta\theta_i \exp(-{\Delta\theta_i}^2/2{b_i}^2) - (z - z_0) \tag{2.5}$$

where ω is the angular velocity, α equals to $(1 - \sqrt{x^2 + y^2})$ and $\Delta\theta$ equals to $(\theta - \ \theta_i)\%2\pi$. Also, $\theta = \arctan(y, x)$ and a_i and b_i are model coefficients. The fourth-order Runge-Kutta method [84] is applied to solve the ordinary differential equation model. The performance of HES and ECGSYN models in terms of execution time and accuracy is evaluated for time interval of T=16 seconds. The functions involved in execution of the HES model are FFT function, state machine, band-pass filter, and integration for signal synthesis. The experimental results in Figures 2.6 and 2.7 illustrate the behavior of the generated HES model in terms of execution time and accuracy based on variations in framework parameters: T_{res} and HES_{size}. The execution time overhead of the generated HES model is attributed to three main functions: state machine, filter and integrator. The figure shows variations in execution time for different model parameter configurations. Table 2.1 shows the execution time values for state-of-the-art model ECGSYN with respect to corresponding accuracy of the model. The frequency f_s represents the step size for the ODE solver and is considered as the model parameter to adjust the accuracy accordingly.

The results in Figure 2.6 shows that the execution time increases as the model accuracy improves with smaller values for T_{res} parameter. Moreover, Figure 2.7 show that smaller values of HES_{size} reduce the execution time since less number of state machines are to be generated and integrated. As shown in the picture, for HES_{size} larger than a certain value, the change in model accuracy will be marginally negligible. We can use this property to improve the execution time for coarse-grained time critical situations. The generated ECG signal with f_s=720 is considered as the reference signal for HES and ECGSYN models to be evaluated; that is, this reference signal is given as the input to our proposed framework

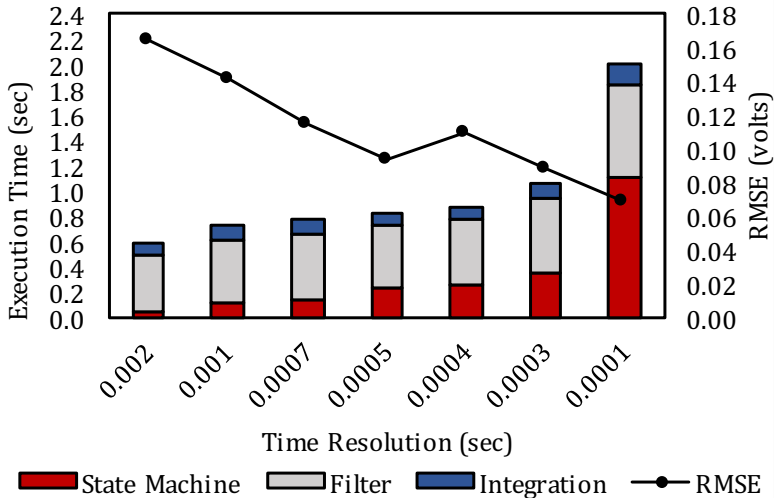


Figure 2.6: Time complexity and error analysis of the proposed HES model with respect to "Time Resolution" parameter.

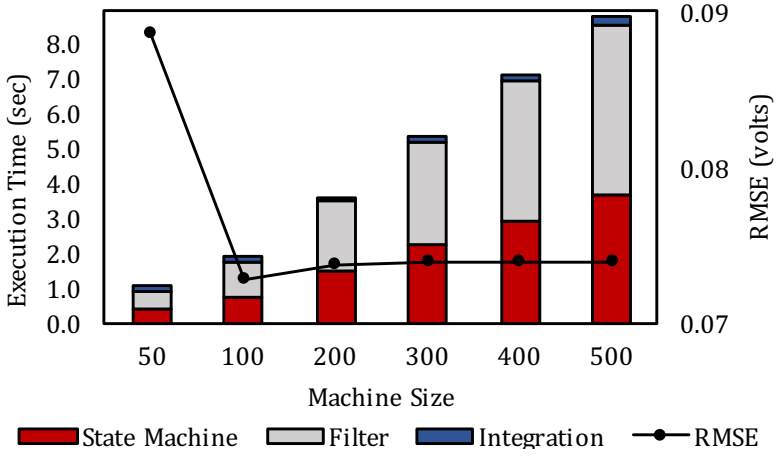


Figure 2.7: Time complexity and error analysis of the proposed HES model with respect to "Machine Size" parameter.

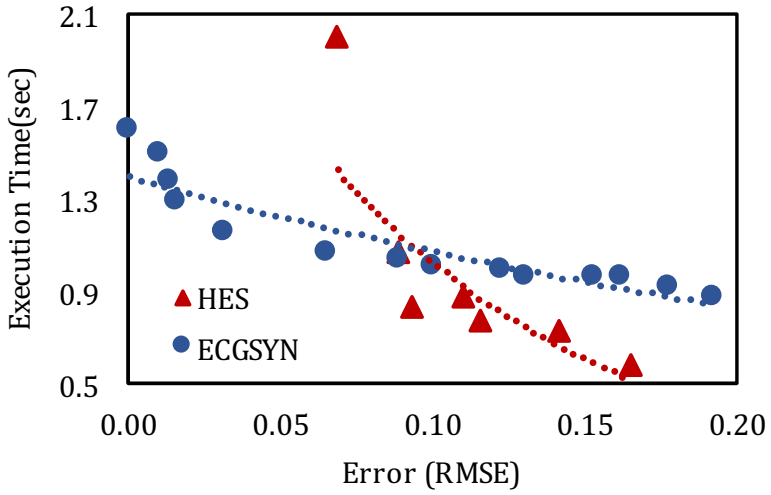


Figure 2.8: Comparison of execution time versus error for the proposed model HES and state-of-the-art ECGSYN.

Table 2.1: Complexity analysis of the ECGSYN state-of-the-art model with respect to "frequency" parameter.

Model	f_s	Execution Time (sec)	RMSE (volts)
ECGSYN	720	0.988	0.000
	718	0.966	0.178
	715	0.960	0.176
	710	0.956	0.164
	705	0.949	0.151
	700	0.949	0.170
	690	0.949	0.179
	650	0.911	0.244
	600	0.919	0.213

to generate the equivalent of reconfigurable HES model. For fairness of comparison, the execution time of the proposed HES model is compared with ECGSYN for the same range of accuracy as shown in Figure 2.8. The results for HES model is derived for HES_{size}=50 which includes 50 harmonics in synthesis of the generated signal. The experimental results show that for same level of model accuracy, the HES model may be executed 38% faster than ECGSYN model.

The improvement in execution time is due to the novel approach to solve the HES model

in contrast to ODE models (ECGSYN). The proposed state machine-based model do not execute compute-intensive and iterative tasks to describe the behavior of a physical system. Moreover, concurrent operation of the state machines are perfectly suitable for intrinsic parallel characteristics of physical systems. In other words, it allows multiple sub-state machines to react to a set of events at the same time. In general, the time complexity of a solver to solve N samples of ordinary differential equations may grow with respect to $c'N$, where c' is a constant factor defined by the type of the discretization algorithm, numerical ODE solver, number and order of the ordinary differential equations in the physical model. On the other hand, the time complexity of the proposed HES model grows with the term cN, where c is determined by the model parameters HES_{size} and T_{res}. Therefore, our HES Machine model is suitable for systems that are more tolerable against model error in tradeoff for reduction in execution time. Here, the ECGSYN includes three simple first-order ordinary differential equations which results in small value of c'. However, we expect that execution of the HES model equivalent to more complex ODE models with larger values of c' will present even smaller values for execution time.

2.6 Conclusion

In this work, we presented an automated model generation framework for physical systems in CPSs. The proposed method utilizes frequency information properties to generate a dynamic state machine model of the physical system. Two tuning parameters are provided to adjust the granularity of the generated model for adaptation to coarse-grained time critical situations or fine-grained safety critical scenarios. Simulation is conducted to evaluate performance of the framework and model using two real physical signals of ECG and NEDC. Moreover, the generated state machine model is compared with ODE-based state-of-the-art equivalent model in terms of accuracy and execution time. The simulation results indicate

that our generated model surpasses the state-of-the-art model by 38% in execution time for same level of model accuracy. The proposed dynamic state machine system may be an excellent replacement for complex ODE solvers when used for testing or embedding CPSs.

Chapter 3

Machine Learning HES Model

3.1 Introduction

Cyber-Physical Systems (CPS) in today's applications are designed to control physical plants such as industrial machines, land vehicles, medical equipment, spacecraft, Unmanned Aerial Vehicles (UAVs), jet engines, etc. The control systems that are implemented to manage these complex physical plants also have relatively high level of complexity. Model-based design is a powerful methodology for the implementation of CPS control systems. For instance, Model Predictive Control (MPC) is typically implemented in CPS applications, e.g., path tracking of autonomous vehicles [76], HVAC control in electric vehicles [90, 91] and formation flying spacecraft [15]. MPC refers to a range of control algorithms in which a dynamic model of the physical system is used to predict the future outputs in a determined horizon [83]. These future outputs of the system are estimated with respect to known input and output values up to the current state and future control signals. An optimization problem is evaluated as a parametric quadratic function to calculate the set of future control inputs subject to constraints enforced by the environment and the dynamic of the system.

Ordinary Differential Equations (ODE) are the most commonly used models to replicate the dynamic behavior of the real physical system in presence of environmental constraints. The ODE models are derived from the conservation laws of physics. A complex physical model may be formulated as thousands of non-linear ODEs which pose scalability, performance, and power consumption issues. Iterative methods are applied to solve non-linear ODEs using quadratic programming paradigms [53]. The execution time of this non-linear programming problem may grow with regards to the algorithms used for discretization and integration of the ODE models and the number or order of ODEs representing the dynamic behavior of the physical system. Development and implementation of techniques to resolve the execution time of non-linear complex ODEs for online control systems are fundamental requirements in CPS design.

A real physical system is under constant change from the effects of the environment. Therefore, we are in need of methodologies to adapt the system to environmental changes and determine the CPS application behavior in respond to such changes. In model-based design applications such as MPC, the complexity of the model under control has a direct influence on the global performance of the system. Specifically, different levels of complexity for the target physical system shall be provided by the user for a specific application. The work in [100] evaluates the performance of a hybrid controller to steer a car in straight and curved trajectory segments. It suggests employing a relatively more advanced model of the vehicle dynamics [72] in curvature path as opposed to fast and simple kinematic model of the vehicle to follow straight lines on the path. The state-of-the-art modeling techniques to design physical models in model-based CPS applications followed by our contributions in this work are summarized in section 3.2.

3.2 Related Work

Cyber-physical systems integrate various engineering areas such as control-, computer-, mechanical-, and network engineering. The complex and heterogeneous design aspects of CPS requires methodologies to combine the corresponding disciplines. Physical models that capture and emulate the behavior of the real physical system have gained extensive research attention in CPS design. A wide variety of physical phenomena such as heart motion, the flow of electric signal and chemical reactions are well described by equations in the literature. Complex physical systems models may be implemented as thousands of ODEs. The mathematical modeling of fuel cells as a power resource in automobile applications is used to explore the reduction in CO2 emissions [64]. The model-based design approach in a vehicle simulation software (ADVISOR) evaluates the operation of fuel cell models under physical settings such as temperature variation in different driving cycles (NEDC, UDDS [12]).

Complex ODE models introduce challenges in terms of scalability, performance, power consumption, and accuracy [31]. The ODE description of a system requires approximations via solver methods such as Euler and Runge-Kutta, to be suitable for computations in computing devices [78]. The demand for more accurate and mathematically sound CPS solutions, cause an increase in resource utilization and energy consumption [59, 88]. Research in model-based design techniques for CPS has introduced solutions to overcome some of the challenges induced by the complexity of ODE models. One approach is to implement the ODEs on Field-Programmable Gate Arrays (FPGA) using Lookup Tables (LUT) to speed up the simulation and enable parallel execution [45]. In general, even though the FPGA implementation of ODE models may improve the execution efficiency for real-time applications, it has implementation challenges regarding limited resources, especially for complex ODE models. Hence, a better approach to modeling and solving of ODE may be required to reduce the complexity not only on FPGAs but also on general CPUs.

Another technique to resolve the challenges raised from complex ODE models is model-to-model transformations and developing frameworks and tools to automate this process. Model transformation introduces flexibility and compatibility in model-based design. Frameworks have been developed to perform automated and semi-automated model transformation [75]. A heart-on-a-chip model [48] is introduced to employ timing behavior of the heart signal and generate different state-based heart conditions as hardware-in-the-loop to test pacemaker software. The heart model is implemented in the Simulink environment and the HDL coder toolbox is employed to generate Verilog code for hardware implementation. The proposed approach is application specific which requires user expertise to implement the model of the heart. Moreover, relying on the HDL coder toolbox for more complex models may require fundamental modifications in the generated Verilog code.

The work in [100] proposes a hybrid MPC method for path planning in path following applications. The technique considers two models of vehicle dynamics with different levels of complexity as predictive models in an MPC application. A metric is introduced based on values of speed and steering angle to select among the two predictive models. The level of complexity for the selected predictive model determines the tradeoff in accuracy for execution time. The technique is application specific and limited to only two levels of complexity for the vehicle model. Moreover, the overhead for complex ODE models remains unresolved.

The observations from state-of-the-art to design physical models in model-based CPS applications, categorize the approaches as follows:

- Application specific models of the physical system are selected at design-time. This approach is bounded to existing mathematical models of the target physical system.
- Ordinary Differential Equation models are commonly used to replicate the dynamic behavior of the physical system. The execution time of non-linear ODE models is often impeding real-time analysis of cyber-physical systems in model-based techniques, e.g.,

MPC.

- Real hardware implementation approaches on FPGA and model-to-model transformation solutions are proposed to overcome the bottlenecks raised from ODE complexity.

3.3 Contributions

Relative to existing literature, we extended our work in Chapter 2. Our contributions in this work can be summarized as follows:

1. We include a harmonic prediction module in our HES [4] model design. This module enables runtime prediction of output signal in physical system given control inputs.
2. We conduct simulation using ODE model of the physical system to collect training data.
3. We evaluate the effectiveness of our predictive machine learning HES model in application of MPC for path tracking.

The Harmonic Predictor block is developed to enable the adaptive feature of the proposed model in run-time applications. The control inputs are given to the Harmonic Predictor block to generate the harmonic information of output signal $z(t)$. Machine learning techniques are applied to develop the Harmonic Predictor block as a prediction model to fit the relation between the control input vector $\vec{u}$ and the harmonic information vectors $\vec{Re}^z$ and $\vec{Im}^z$.

The rest of the chapter is organized as follows. In Section 3.4 the architecture of the proposed methodology, tuning parameters, and MPC formulation is described in details. We demonstrate the workings and effectiveness of our framework for path tracking application in Section 3.5. Finally, we state our conclusions in Section 3.6.

3.4 Methodology

In this work, we introduce the Hybrid Harmonic Equivalent State Machine model of a physical system to be integrated in control systems for fast and dynamic performance to model-based design approaches. The proposed model is the includes frequency domain properties to synthesize the outputs of the dynamic model for hybrid accuracy. Moreover, the execution time of the model may be adjusted in tradeoff with accuracy to adapt the model coarse-grained time critical or fine-grained safety critical maneuvers. The model uses notions of state, input, outputs, and dynamics to describe the behavior of a system as following:

$$z(t) = f(s, \vec{u}) \tag{3.1}$$

where $\vec{u}$ represent the vector data of control inputs for a specific time window which we call the HES Horizon. The variable $z(t)$ stands for the measured output of the system dynamics at time instant t. The state variables of the proposed model are presented as $s \in States$. The high-level architecture for the proposed model contains two main blocks:

1. State Machine Generator.
2. Harmonic Predictor.

This State Machine Generator block is introduced in Chapter 2 Section 2.4. We give a more elaborated explanation here. This block captures frequency information of the output signal for the determined HES Horizon and generates the reconfigurable output signal for the target physical system. The generated output can be reconfigured by the proposed tuning parameters which are introduced in sections 3.4.2. The inputs to State Machine Generator block are vectors $\vec{Re}^z$ and $\vec{Im}^z$ of size $(N/2+1)$ which represent the real and imaginary

components of the frequency spectrum for the output signal $z(t)$. A synthesis algorithm is developed to integrate these imaginary and real components of frequency harmonics $\vec{Fr}^z$ and generate a reconfigurable representation of output $z(t)$ in the form of concurrent state machines. The synthesis algorithm employs $(N/2+1)$ inverse of frequency harmonics and corresponding real and imaginary components, as the periods and output magnitudes of concurrent state machines respectively; this generates N samples of output signal $z(t)$ in the so-called HES Horizon. A band-pass filter is implemented to translate the output square waves of the concurrent state machine models into sinusoidal signals. The sinusoidal output signals, one per state machine, represent the signal harmonic components. Finally, the harmonic components are integrated to generate the output signal for the target physical system. The Harmonic Predictor block is developed to enable the adaptive feature of the proposed model in run-time applications. The control inputs are given to the Harmonic Predictor block to generate the harmonic information of output signal $z(t)$. Machine learning techniques are applied to develop the Harmonic Predictor block as a prediction model to fit the relation between the control input vector $\vec{u}$ and the harmonic information vectors $\vec{Re}^z$ and $\vec{Im}^z$.

3.4.1 Model Architecture

The detailed descriptions of the model sub-blocks are presented in the following sections.

State Machine Generator

The Harmonic Generator block captures frequency spectrum information of the output for the physical system and resynthesizes the signal in the form of state machine model representation. A synthesis algorithm is designed and implemented to integrate the harmonic information vectors $\vec{Re}^z$ and $\vec{Im}^z$ and generate concurrent state machines with respective

frequency harmonics $\vec{Fr^z}$ as the update rates. The synthesis equation of FFT for signal $z[k]$ of size N is employed as presented in Equation 3.2. In this equation, k stands for the index of samples running from 0 to N-1. The vectors $\overrightarrow{Re^z}[i]$ and $\overrightarrow{Im^z}[i]$ are the normalized frequency spectrum coefficients for the sine and cosine waves with index i running from 0 to $N/2$ for the respective harmonic frequencies [79].

$$z[k] = \sum_{i=0}^{N/2} \overrightarrow{Re^z}[i] cos(2\pi ik/N) + \sum_{i=0}^{N/2} \overrightarrow{Im^z}[i] sin(2\pi ik/N) \tag{3.2}$$

The state machines are represented as a set of states and transitions between those states that are triggered with respect to conditional expressions or predicates. Designers use state machines to break complex systems into manageable states and state transitions. Therefore, the state machine model of computation can fit the synthesis function components as concurrent state machines. In this model, the components of the physical signal may all be generated by a five-state synchronous harmonic state machine (HES Machine). Specifically, the proposed harmonic state machine model definition is a 5-tuple,

StateMachine=(States, AuxVar, Outputs, Update, InitState)

where *States*, *Aux Var* and *Outputs* are sets, *Update* is a function, and *Init State* $\in$ *States*. These variables are defined as follows:

- **States (State Variables)**: are state space variables enumerated as -1, $S1$, $S2$, $S3$ and $S4$. The system is always in the "*current*" state.
- **Auxiliary Variables** : refers to the conditional expressions or predicates which trigger the state transition process. The proposed methodology tracks the value of *elapsed time* variable to perform state transition when the conditions are met.
- **Outputs**: is a set of actions per state which assigns FFT coefficients as output values.

- **Update**: is referred to as the $Tick$ function in the proposed methodology. On each call of the $Tick$ function the state machine executes and the current state's outgoing transitions are examined to set the new current state. The actions of the new current state are then executed.
- **Initial State**: is the initial current state and its actions are executed once. The execution of the harmonic state machine is initialized at state -1.

In the integration process, $(N/2+1)$ concurrent state machines are implemented with vectors $\vec{Re}^z$ and $\vec{Im}^z$ as output values and the harmonic frequencies vector $\vec{Fr}^z$ is used to calculate the period of each state machine. These concurrent state machines are executed at a global rate of T_{res} which is configured by the user as a framework parameter. This global clock represents the time resolution of the state machine. It can be measured as an actual wall-clock (real) time by periodic programmable interval timers [94] that call an interrupt service routine (ISR). The global period is designated as the timer value to iterate the ISR calls. We define one global $Tick$ function to execute $(N/2+1)$ concurrent state machines per call of the ISR. In other words, the synthesis components are generated as square waves with user-specified global time resolution. The framework parameters are described in Section 3.4.2.

Harmonic Predictor

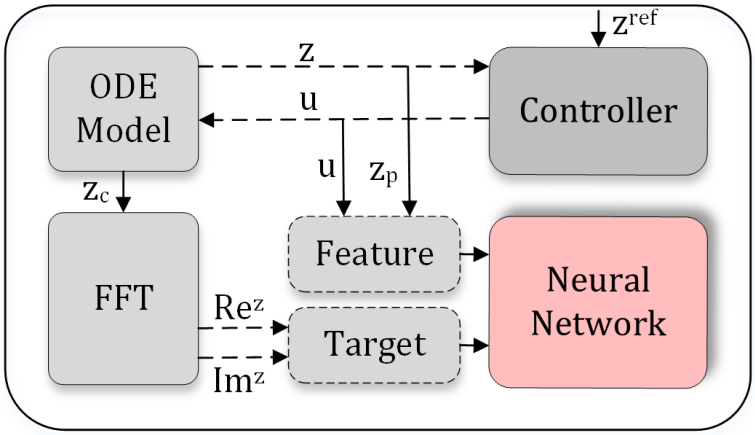
Harmonic Predictor block enables the run-time adaptive feature of the proposed model—it provides a relationship between the control inputs and values of the harmonic components for the respective output signal. Machine learning as non-parametric modeling approaches has gained attention to establish the relation between some measured responses for complex and non-deterministic system behavior. We apply a machine learning technique to fit a predictive model that maps the control input vector $\vec{u}$ of size N to respective harmonic information vectors $\vec{Re}^z$ and $\vec{Im}^z$ of size $(N/2+1)$. We interpret our input and output vectors for the

predictive model as time series data to leverage time series prediction approaches [16].

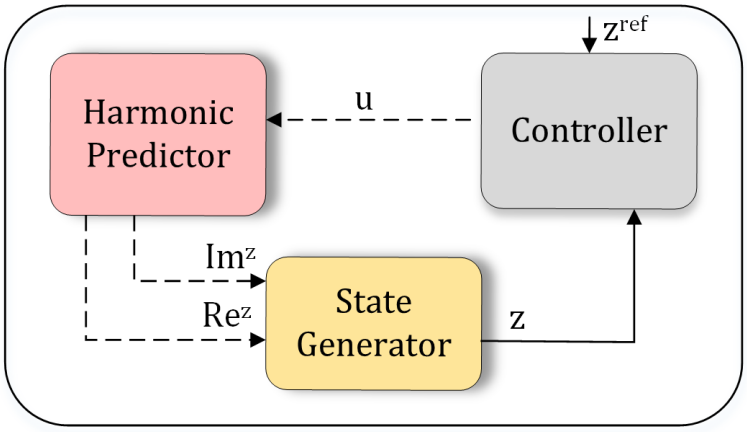
Neural Networks (NN) have solved time series prediction [93] and hold promising performance to learn linear and non-linear models without prior knowledge of the relation between input and output variables[19, 67]. Feedforward networks are a class of neural networks, where the input feeds forward through the network layers to the output. This network is arranged as three input, hidden and output layers. Each layer includes a set of nodes with edges to pass the information. The nodes in the hidden layer and output layer are active and data may be modified as opposed to nodes in the input layer that are passive with no permission to change the data. The edges entering the active nodes are associated with a weight that are factors to inputs of the nodes—these weights are adjusted to yield good performance for the predictive model. A nonlinear mathematical function, e.g., the sigmoid function, is used to limit the node's output [79]. The prediction of the time series data is conducted using the direct NN method in which the time series of output is predicted all at once [33].

The Harmonic Predictor block is implemented in two offline and online phases. The implementation of these phases in for MPC application is illustrated in Figure 3.1.

1. Offline Training Phase: In this phase the weight values are adjusted and determined with respect to the iterative flow of training data through the network. The network learns the pattern that maps the vector of input values to the associated output signal. As shown in Figure 3.1(a), simulation is conducted on the ODE model of the target physical system to record the control inputs $\vec{u}$ and respective output values z to be employed as the data set for the training phase. The recorded current output values z_c are fed into Fast Fourier Transform algorithm in time windows of so-called HES Horizon to derive the frequency information. The frequency domain of a signal carries the same information as the time domain; that is, you can calculate one domain symmetrically from the other one, which is addressed as the duality property [79]. The vector data of control inputs $\vec{u}$ and the output value from



(a) Training.



(b) Prediction.

Figure 3.1: Training and Prediction phases of the Harmonic Predictor block in MPC.

previous time step z_p are considered as the input features and the respective output signals $\vec{Re}^z$ and $\vec{Im}^z$ are the target values of the training data sets.

2. Online Prediction Phase: The mapping that is fitted in the NN layers during the training phase is automatically retrieved in online prediction. The Harmonic Predictor block is used to predict harmonics information of the output signal from the respective run-time values of control inputs $\vec{u}$. The predicted harmonic information is fed into the State Machine Generator block for output generation as shown in Figure 3.1(b). That is, the output of the proposed physical model z can adapt to variations in control inputs u in run-time.

3.4.2 HES Machine Tuning Parameters

In this section three tuning parameters HES_{size}, T_{res} and Q for the proposed framework are described by which the model may be adjusted to meet system requirements (e.g. accuracy and timing).

Machine Size ($\boldsymbol{HES_{size}}$) specifies the number of harmonic concurrent state machines to be integrated during the synthesis process ranging from 1 to ($N/2$+1). The model accuracy may be adjusted with respect to this parameter by inclusion/elimination of certain harmonic frequencies.

Time Resolution ($\boldsymbol{T_{res}}$) parameter indicates the smallest time unit in the proposed framework by which the generated state machine will be executed. The proposed framework tracks the value of T_{res} as an actual wall-clock (real) time. T_{res} specifies the timer values for the periodic programmable interval timers to trigger the interrupt service routines.

Q-Array (Q) is an array of filter quality factors that characterize the band-pass filter response with respect to its center frequency. A filter with a high-quality factor will have a narrow pass-band and vice versa. The quality factor is calculated as the ratio of cut-off frequency to bandwidth. The band-pass filter is required to translate the generated square waves for the harmonic state machine output to sinusoidal equivalents.

The following section describes the application of the proposed framework in model predictive control systems.

3.4.3 MPC Formulation

The proposed model is integrated into the context of model predictive control for CPS applications. Model predictive control designates an ample range of control techniques that

incorporate three elements[17]:

1. Prediction Model: a predictive model to replicate the dynamic behavior of the real physical system with regards to laws of physics.

2. Objective Function: the objective function is usually formulated as a Least Squares (LSQ) objective to obtain the control law. The future output values z should follow the desired reference signal z^r in a determined prediction horizon T_p. Moreover, the deviation from a given reference Δz and the control effort Δu should be penalized.

3. Obtaining the Control Law: the controller employs a mathematical formula called the control law to determine the output u that is sent to the physical model $f(s, u)$.

The predictive model of the physical system is employed to estimate the future outputs $z(t+k|t)$ at time instant t for $k = 1...T_p$. The notation $z(t+k|t)$ refers to value of the output variable z in time instant $t+k$, estimated at time t. The future output values are determined by the past input and output values up to instant t and future control inputs $u(t+k|t)$, $k = 0...T_p - 1$. These future control inputs are calculated in an optimization problem that forces the system to satisfy a determined criterion and follow the reference values for the output signal. This optimization problem is a parametric quadratic function to be solved with analytical or iterative solutions using the linear or non-linear model of a physical system respectively[99]. The optimized control input value for the first instant of the prediction horizon $u(t|t)$ is sent to the physical system under control and the process is repeated for the next sampling time. The MPC formulation taken from [99] is the solution

to the following optimization problem at each time instant

$$\min_{z,u} \quad \|z(T_p) - z^r(T_p)\|^2_{P_c} + \sum_{t=0}^{T_p} \|z(t) - z^r(t)\|^2_{Q_c} + \|u(t) - u^r(t)\|^2_{R_c} \tag{3.3a}$$

$$s.t.$$

$$z(t) = f(s(t), u(t)), \tag{3.3b}$$

$$s(0) = \hat{s}(0), \tag{3.3c}$$

$$q(z(t), s(t), u(t)) \geq 0 \qquad t \in [0, T_p] \tag{3.3d}$$

Equation (3.3a) represents the LSQ objective function where P_c, Q_c and R_c are weight matrices. The model of system dynamics is defined in Equation (3.3b), where $z(t)$, $s(t)$ and $u(t)$ represent outputs, state variables and control inputs respectively. Equation (3.3c) initializes the state variables at time $s(0)$ with current estimates $\hat{s}(0)$. Additional physical limits and constraints may be imposed for system variables through Equation (3.3d).

Ordinary Differential Equations are the most commonly used models to emulate the behavior of continuous-time (non-)linear dynamical systems in response to all possible inputs and initial conditions [83]. Discretization methods (e.g. Euler and zero-order hold) are applied to transform the continuous differential equations into discrete difference equivalents, appropriate for numerical computing. The discretized differential equations are solved using numerical methods with regards to the linearity of the model. The approach to solve non-linear ODEs is iterative methods, where a series of linear equations are solved iteratively to converge to the solution for the non-linear ODE. Therefore, the computation complexity of solving N samples of ordinary differential equations may grow with respect to $c'N$, where c' is a constant factor defined by the discretization algorithm, numerical ODE solver, number

and order of the ordinary differential equations in the physical model.

In an MPC application, the ODE solver method is evaluated per equation to estimate the future control inputs $u(t+k|t)$ at each prediction horizon time instant $k = 0...T_p - 1$; that is, to calculate the control inputs in the next k future steps, one equation in the ODE model of the physical system should be solved k times. Therefore, nk iterations of the solver are computed to solve n equations comprising the ODE model of the physical system.

As mentioned before, The proposed state machine model for the physical system is featured with vector data for control inputs $\vec{u}$. At each simulation time step: 1. The Harmonic Predictor block generates the frequency information of the output signal for the next k time instants for $k = 0...T_p - 1$ all at once where T_p represents the HES Horizon, 2. The State Machine Generator block generates the output $z(t)$ as a signal for time interval T_p. The execution time to generate the output signal in time window T_p is based on the term cN, where c may be adjusted by the proposed tuning parameters HES_{size}, T_{res} and Q in tradeoff with accuracy. Therefore, for $c \ll c'$ the proposed model can surpass the ODE equivalent in terms of execution time. The MPC application may leverage this feature of the proposed model to reduce its return time for fast estimation of future control inputs.

3.5 Experimental Results

3.5.1 Setup

The State Machine Generator block is implemented using the C/C++ programming language in order to enable it to be highly portable and compatible with various platforms for compilation and execution. The global clock of the state machine model is updated by interrupt handlers of the operating system.

Table 3.1: Error analysis for NN with respect to variations in number of steps in the prediction horizon.

# of Steps	Input Size	Output Size	Train Error (MSE)	Validation Error(MSE)
5	13	8	2.59E-10	6.32E-10
9	21	12	4.92E-10	8.96E-10
15	27	18	4.42E-08	1.20E-07
21	45	24	2.58E-07	3.78E-07
25	53	28	8.23E-04	4.91E-03
31	65	34	3.31E-07	6.89E-06
35	73	38	2.50E-07	6.78E-06
41	85	44	2.10E-07	6.90E-06
51	105	54	1.08E-07	1.07E-07

The Harmonic Prediction model is trained by using the Matlab neural networks module (nftool). The training algorithm used in this work is the Levenberg-Marquardt (LM) algorithm also known as damped least-squares (DLS). The LM algorithm is an edition to Gauss-Newton method using a trust region approach [60] which is initially designed as a numerical method to minimize functions that are sums of squares of nonlinear functions. This benefits the neural network training, where the performance metric is the mean squared error. As mentioned before the training phase is offline and applies no additional execution time to the run-time application. The input features and the target outputs for the training phase are the control input vectors $\vec{u}$ and frequency harmonic components $\vec{Re}^z$ and $\vec{Im}^z$ respectively.

The experiments are conducted for prediction horizon of size T_p which determines the size for vectors $\vec{u}$, $\vec{Re}^z$ and $\vec{Im}^z$ as T_p, $(T_p/2)+1$ and $(T_p/2)+1$ respectively. We concatenate the control input vectors with the output values from the previous time step to create the vector for input features. The output values from the previous time step are concatenated to the time series to consider past behavior of the system. The target outputs dataset is a time series of size T_p for $\vec{Re}^z$ vector of size $(T_p/2)+1$ followed by the same size vector $\vec{Im}^z$. The dataset is acquired from 2 seconds simulation of MPC application with 0.01 seconds

sampling time and 2 iterations of FFT algorithm for output signal $z(k)$ for $k = 0...T_p - 1$. Table 3.1 illustrates the configurations for the neural network and corresponding train and validation error.

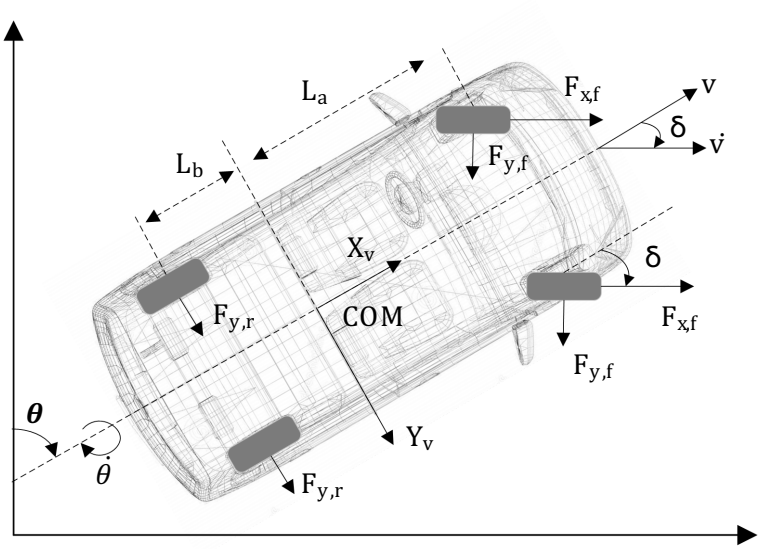


Figure 3.2: Schematic view of the vehicle model.

For our control application, we adopt the software framework based on the ACADO Toolkit [42]. ACADO Toolkit is an open source software written in C++ for automatic control and dynamic optimization. It provides a self-contained environment to implement control algorithms including model predictive control as well as state and parameter estimation. The framework contains efficient implementations for numerical integrators, Runge-Kutta [84] and BDF [8] to solve ODEs and differential algebraic equations(DAEs). ACADO is designed with the object-oriented paradigm and may easily be extended to link external packages and existing algorithms. Our experiments are performed on a PC with a quad-core Intel Core i5 and 8 GB of DDR3 RAM.

3.5.2 Model Performance Metrics

Two performance metrics of computation time overhead and precision are considered for comparing the performance of our HES Machine model with state-of-the-art models.

- **Execution Time:** refers to the processing time required by the operating system and any utility that supports application programs. One of the merits of the proposed state machine-based model is that the state machines do not execute compute-intensive and iterative tasks to describe the behavior of a physical system. Moreover, concurrent operation of the state machines is perfectly suitable for intrinsic parallel characteristics of physical systems. In other words, it allows multiple sub-state machines to react to a set of events at the same time.
- **Accuracy:** is a quality factor to measure the error between the values evaluated by a model and the corresponding expected real values. Root Mean Squared Error (RMSE) is considered to quantify the accuracy as in Equation 3.4. The *Expected* variable holds the sample values of the real physical signal, *Evaluated* variable is the output of the HES Machine model for the respective physical system, and N represents the number of samples.

$$RMSE = \sqrt{\frac{(\sum(Expected - Evaluated)^2)}{N}} \tag{3.4}$$

3.5.3 Implementation for Path Tracking Application

To evaluate the effectiveness of the proposed design, we implement our generated model of vehicle dynamics to be integrated into the MPC closed-loop for path tracking application.

The path tracking problem is dependent on the vehicle modeling to design multi-constraints model predictive control law. As mentioned in the methodology section, the training phase for the Harmonic Predictor block performs the offline simulation with the ODE model of the target physical system to acquire training datasets. The ODE model of the vehicle dynamics [100] shown in Figure 3.2 is given in equation form as:

$$\dot{x} = v\sin(\theta) \tag{3.5a}$$

$$\dot{y} = v\cos(\theta) \tag{3.5b}$$

$$\dot{v} = \cos(\delta)a - \frac{2}{m}F_{y,f}\sin(\delta) \tag{3.5c}$$

$$\dot{\theta} = \phi \tag{3.5d}$$

$$\dot{\phi} = \frac{1}{J}(L_a(ma\sin(\delta) + 2F_{y,f}\cos(\delta)) - 2L_bF_{y,r}) \tag{3.5e}$$

$$\dot{\delta} = \omega \tag{3.5f}$$

where x and y are longitudinal and lateral positions, v is the longitudinal velocity, θ is the azimuth, ϕ and δ represent the angular speed and steering angle respectively. The variable L_a is the distance of sprung mass center of gravity from the front axle, L_b is the distance of sprung mass center of gravity from rear axle and J is te angular momentum. The variables $F_{y,f}$ and $F_{y,r}$ stand for front and rear tire lateral force. These forces are computed from the following equations:

$$F_{y,f} = C_y(\delta - \frac{L_a\phi}{v}) \tag{3.6}$$

$$F_{y,r} = C_y(\frac{L_b\phi}{v}) \tag{3.7}$$

where C_y refers to the lateral tire stiffness. The model is parametrized with respect to real-world specifications. L_a=L_b=1.5m, mass m=1700 kg and tire stiffness data for a 2011 Ford Fusion is applied. The following cost function is considered for tracking a path subject to track and input constraints:

$$\min_{x,y} \quad \sum_{t=0}^{T_p} \|\hat{x}(k+1|k) - x^r(k+1|k)\|^2_{Q_c} \tag{3.8a}$$

$$+ \|\hat{y}(k+1|k) - y^r(k+1|k)\|^2_{Q_c} \tag{3.8b}$$

$s.t.$

$$-0.25 \leq \delta \leq 0.25 \tag{3.8c}$$

$$-1.25 \leq \omega \leq 1.25 \tag{3.8d}$$

$$-30 \leq a \leq 30 \tag{3.8e}$$

The performance of the proposed HES model of the vehicle dynamics in run-time MPC for

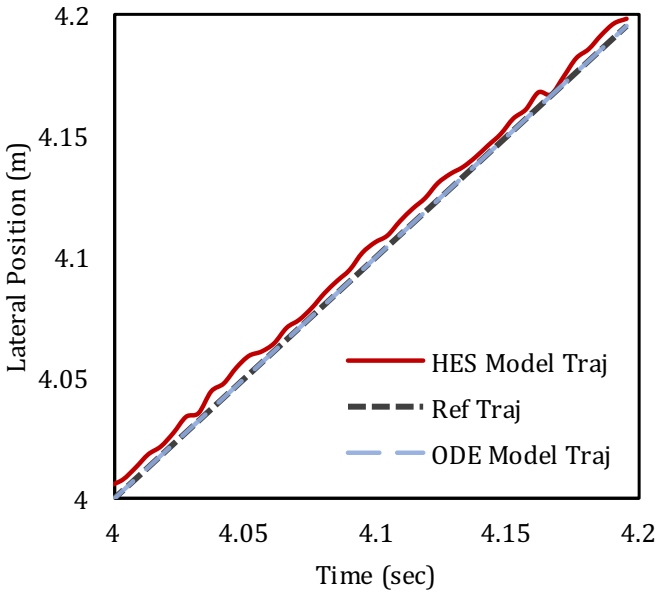


Figure 3.3: Analysis of performance for HES model and ODE model in trajectory tracking application.

Table 3.2: Error comparison of ODE model and HES model for different step size in the prediction horizon.

# of Steps	HES Error	ODE Error	Time Resolution	Machine Size
5	3.40E-03	3.40E-03	0.001	6
9	7.80E-03	7.80E-03	0.001	10
15	3.90E-03	4.00E-03	0.001	16
21	3.60E-03	2.50E-03	0.001	22
25	3.07E-02	2.70E-03	0.001	26
35	1.20E-02	3.13E-04	0.001	36
41	5.20E-03	2.92E-05	0.001	42
51	7.06E-02	1.00E-03	0.001	52

Table 3.3: Execution time comparison of ODE model and HES model for different step size in the prediction horizon.

# of Steps	Predict Harmonic Execution Time (ms)	State Mchine Generator Execution Time (ms)	HES Overall Execution Time (ms)	ODE Execution Time (ms)
5	1.50	0.08	1.58	0.04
9	1.40	0.08	1.48	0.09
15	1.20	0.13	1.33	0.14
21	1.20	0.15	1.35	0.82
25	1.00	0.98	1.98	1.43
35	0.93	1.56	2.49	1.30
41	0.92	0.21	1.13	1.70
51	1.10	0.18	1.28	1.90

path tracking application is compared with the model in Equation 3.5a. The performance of two models in tracking a static path for a certain time horizon is illustrated in Figure 3.3. The HES model is capable to follow the reference path with the average of 1% error in comparison to ODE model with an average error of 0.2%. The small loss in accuracy in using HES model is in the tradeoff for improved performance for applications that are error tolerant.

We compare the error values for ODE model and HES machine for different prediction horizon time steps in Table 3.2. The HES Machine is configured with respect to framework

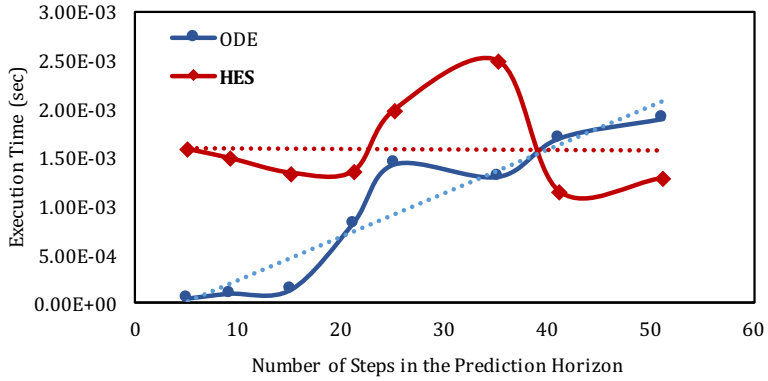


Figure 3.4: Comparison and analysis of execution time for ODE model and HES model.

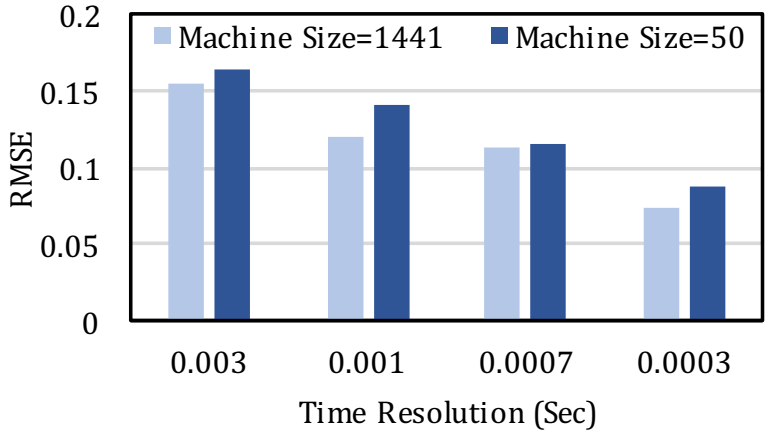


Figure 3.5: "Time Resolution" analysis considering RMSE for the HES Model

parameters, Time Resolution and Machine Size. The Machine Size parameter is set to the maximum value for fair comparison of HES and ODE models. The results indicate comparable error values for two models. The variations in the error for HES model is due to the non-deterministic behavior of the neural network model. For future work, machine Learning techniques could be employed that consider the step size in the prediction model.

The models are analyzed in terms of execution time over different prediction horizon time steps. The results in Table 3.3 indicate that the performance of HES model surpasses the ODE equivalent for large values of step size. The execution time for both ODE and HES models are compared in Figure 3.4 with respect different number of steps. The results in Figure 3.4 illustrate that the performance of ODE model drops below HES model after a

certain cross point; that is, HES model of vehicle surpass the ODE equivalent by 32% in terms of performance for large prediction horizon time steps. Therefore, HES Machine models of physical systems may be an appropriate reconfigurable replacement for ODE equivalents in applications with large prediction horizon requirements that are tolerant to 1% error.

Figures 3.5 depicts the accuracy of the generated model for Pareto-optimal configurations of model parameters. The Pareto-optimal points were explored for the precision metric of RMSE as the optimization cost function. The results justify our claim to reduce the value of HES_{size} parameter for faster execution of the model with minor loss of accuracy. The proposed hybrid state machine system may be an excellent replacement for complex ODE solvers when used for in CPSs.

3.6 Conclusion

We presented a model generation framework to transform ODE models of physical systems to Hybrid Harmonic Equivalent State (HES) Machine model equivalents. The proposed model may be reconfigured to adjust its accuracy and execution time from coarse-grained time critical situations to fine-grained scenarios in which safety is paramount. Experiments on a closed-loop MPC for path tracking application is performed using a model of vehicle dynamics. We analyze the performance of MPC when applying our HES Machine model. The performance of our proposed model is compared with state-of-the-art ODE-based models, in terms of execution time and model accuracy. Our experimental results show 32% reduction in MPC return time for 0.8% loss in model accuracy.

Chapter 4

Switching Predictive Control Using Machine Learning HES Model

4.1 Introduction

With the recent developments in autonomous driving and the futuristic vision offered by automated vehicles, it has been acknowledged that it is just a matter of time before this technology continues to take over humans in driving autonomous and semi-autonomous vehicles [100]. Advanced control methodologies have emerged to empower the development of modern vehicles for path planning and path following applications. As mentioned in Chapter 1 Section 1.2.1 nonlinear Model Predictive Control (MPC) is leveraged to develop path following control systems while handling model uncertainties, constraints and nonlinearities. A predictive model of the physical plant is used to estimate the future outputs for a prediction horizon within a window of time and with respect to known input and output values. The disadvantage of MPC arise from its strong dependence on the model. However, improvements in data-driven modeling and collection of massive amount of sensor data, this may not be as

much of a difficulty.

We are aware that complex models of physical systems may be comprised of thousands of non-linear Ordinary Differential Equations (ODEs), requiring considerable computing power to execute. These ODE models introduce challenges in terms of scalability, performance, power consumption and accuracy [31, 62]. Discretization methods (Euler, zero-order hold, etc.) are applied to transform the continuous-time differential equations into discrete-time equivalents, appropriate for numerical computing. The discretized differential equations are solved using numerical algorithms. Iterative solutions are used to solve the non-linear ODE models of physical systems, where a series of linear equations are solved iteratively to converge to the solution for the non-linear ODEs [52]. Therefore, the computation complexity of solving N samples of ordinary differential equations may grow with respect to the type of the discretization algorithm, numerical ODE solver, and the number and order of the ordinary differential equations in the physical model. Moreover, the demand for higher accuracy and more mathematically sound control solutions causes an increase in resource and energy consumption that must be taken into account during the design cycle [3, 59, 61, 92].

Autonomous behavior in advanced control systems is desirable so they perform well under changing conditions in the physical plant and the environment [4, 5]. Therefore, we proposed a novel switching control methodology to augment the control system to adapt to changes affecting the operating region of the system. In switching predictive control schemes, the controller switches between predictive models of different granularities based on a metric that computes the current dynamic state of the system. An optimal switching control problem consists of: 1) a sequence of switching events, 2) a sequence of modes, 3) a sequence of control inputs associated with each mode [13]. Switched systems are used to model classes of systems with multi-mode features and switching control schemes may be applied as a solution to address the online computational complexity [100, 102]. We reviewed the state-of-the-art strategies to address the computational overhead specifically in MPC systems in

the following section.

4.2 Related Work

Advanced techniques have been proposed to resolve the MPC computational burden. A common approach to reduce the computational complexity of traditional MPC is the switching MPC [57, 100] methodology. Here, the controller combines the use of predictive models of different granularities in a switching control scheme. The controller switches between the predictive models based on a metric that computes the trade-off between the error and computational savings due to model reduction. Zhang et al. [100] proposed a binary switching controller based on two coarse-grained and fine-grained predictive models of the vehicle for path planning and path following application. The proposed method considers only two levels of complexity to be included in the MPC application. Gao et al. [29] designed a hierarchical MPC scheme for path planning and path following application to overcome the computational complexity. The high-level controller is formulated to plan an obstacle free path using a simplified-point mass model of the vehicle. Moreover, a low-level controller is designed based on a nonlinear dynamic model of the vehicle to follow the planned path as the reference. More levels of complexities for the physical model enables the MPC to adapt its performance to a wider range of environmental constraints and uncertainties [102]. Jadbabaie et al. [47] suggested that a stable MPC controller requires a sufficiently large prediction horizon. On the other hand, short prediction horizons are preferred for improved prediction accuracy of predictive models. This is because harmful effects of the poor estimates are amplified over a long prediction horizon time. Here, the problem is addressed by proposing an MPC approach that uses an adaptive prediction horizon with respect to quality measures [23]. However, the numerical effort needed in order to solve the optimal control problem for a long prediction horizon still remains significant.

Erlien et al. [24] adopted variable length time-steps in the prediction horizon as a solution to the computational complexity and cost of nonlinear MPCs. This method can associate different prediction horizons targeted for stabilization and collision avoidance tasks. The approach adjusts the sampling time to allow longer prediction horizon in obstacle avoidance steering task as well as short time steps in the prediction horizon for more detailed dynamic behavior prediction. In [98] the prediction horizon is a function of the vehicle speed and the sample time in path following applications. Mahadevan et al. [63] suggested flattening the non-linear differential equation model of the physical system to reduce the computational overhead. For nonlinear ODE systems, flatness is achieved if all the states and input variables can be written in terms of a set of variables—flat outputs and their derivatives. For the dynamic ODE systems that can be recast as a differentially flat system, the runtime optimization problem is reformulated with simpler constraints, and hence smaller computational complexity. Linear Time-Varying (LTV) MPC is a method that employs a model of the physical system linearized along the simulated path at each time step [25]. Even though successful applications of this approach have been presented in the literature, the overhead raised from the linearization over successive time steps is not resolved. Ferreira et al. [26] proposed a methodology to organize libraries of models for electro-hydraulic elements with variations in terms of model complexity. The purpose of the work is to use the appropriate model with regards to the kind of physical domain and the timing requirements for platform. Specifically, different levels of complexities for the target physical system under test shall be provided by the user for a specific application. Here, the sudden changes to the physical system caused by the environment may not be considered at runtime.

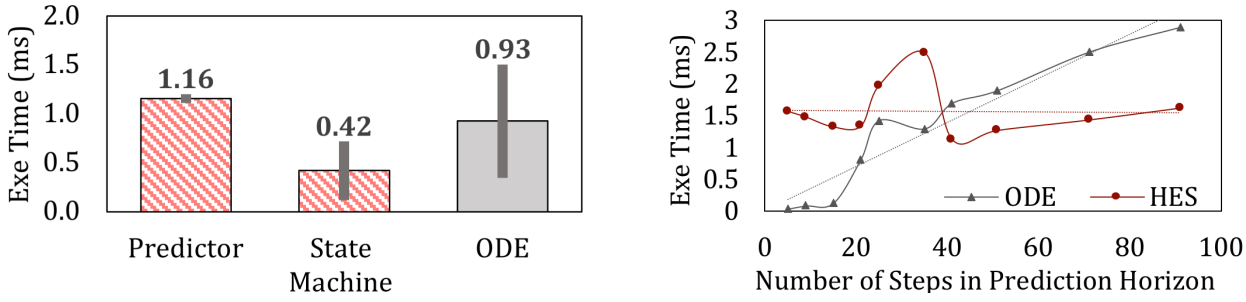
In general, the main burden in managing the computational complexity of nonlinear MPC applications is the concurrent solving of a large number of nonlinear ordinary differential equations. To overcome this computational overhead, we proposed a more general approach that integrates all the above outlined techniques while eliminating the limitations associated with ODE models. The following motivational case study will discuss this further.

4.2.1 Motivational Case Study

The work in [5] presented a framework to generate Harmonic Equivalent State (HES) machine, a state-based model of the physical system. HES is applied as the predictive model in the MPC loop to estimate the behavior of the physical system at future time instants based on the calculated future control inputs. The proposed framework uses the Fast Fourier Transform (FFT) decomposition and synthesis functions to generate a reconfigurable model of various granularities. The granularity is adjusted based on the trade-off between model accuracy and computation time. A machine learning model is trained to estimate the dynamic behavior of the target physical system. MPC simulation is conducted with ODE model of the physical system to collect the training dataset. A Neural Network (NN) model fits the relation between the future control inputs and harmonic frequency information of the predicted outputs in prediction horizon of size T. Then, a state machine generation algorithm uses the harmonic frequency information to produce a reconfigurable representation of the model in the form of concurrent state machines. Each concurrent state machine is executing at the rate of one of the harmonic frequencies to generate a square-wave output. Tuning parameters are provided to reconfigure the model and tune it for the desired execution time and granularity level. A band-pass filter is used to translate the generated square-waves to sinusoidal equivalents. Finally, the sine-wave harmonics are integrated into the final output signal. Figure 4.1 compares the execution time of HES model of a vehicle in comparison with an ODE-based predictive model for MPC in a path following application. The performance of the models is evaluated over different prediction horizon sizes for a constant time step. To further analyze the performance of the proposed HES in comparison with the ODE model, we computed the execution time of the HES model as the sum of its two main components: the Harmonic Predictor block and State Machine Generator block as shown in Figure 4.1(a). The wide bars represent the mean of execution time for each component with respect to changes in the prediction horizon size. The results in the figure indicate that the mean of

execution time for the ODE model is 2 times more than the State Machine Generator block, and the Harmonic Predictor block has the highest mean value for the execution time.

In order to evaluate the performance sensitivity to the size of the prediction horizon, we also computed the variance of execution time for each component shown as in narrow bars. The interesting observation here is that even though the Harmonic Predictor block has the highest mean of execution time, it has a very small variance as opposed to other components and ODE has the highest variance. To be exact, the variance of ODE is 4 times higher than the variance of the Harmonic Predictor block and 2.5 times higher than the state machine component. That is, the small variations in execution time of the Harmonic Predictor block occurs for different values of prediction horizon size. On the other hand, the performance of the ODE model varies more drastically for different prediction horizon sizes. Therefore, it can be concluded that for larger values of prediction horizon, known as long prediction horizon problems, the HES model can outperform the ODE model in terms of execution time. The execution time for both ODE and HES models are compared in Figure 4.1(b) with respect to the common parameter, the prediction horizon size. The dotted trend-lines represent linear changes in the execution time for different values of prediction horizon size. The results show that the HES model outperforms the ODE equivalent with 32% improvement in performance for large prediction horizon. The improvement of performance is in a tradeoff for a minor loss in accuracy for applications that are error tolerant [5].



(a) Mean and variance of execution time.

(b) Performance of ODE vs. HES [5].

Figure 4.1: Comparison of execution time for ODE and HES models.

Conclusion from the observations: The HES model holds promising properties to be employed as the predictive model in novel control methodologies. The model can be reconfigured into various levels of granularities at run-time to be employed as the predictive model in switching MPC approaches. New features may be added to the model to enable run-time reconfiguration subject to current state of the system. Our observations indicate that the error is mostly caused by the filter and the challenges associated with automatic tuning of the filter per harmonic component. Therefore, an alternative solution to replace the filter in the proposed framework is preferred.

4.3 Contributions

Based on existing literature and the motivational example described in Section 4.2.1, we proposed a computationally efficient MPC methodology. Our contributions in this work can be summarized as follows:

1. The performance of the HES model is improved in terms of execution time and model accuracy. The Neural Network model is modified to better estimate the dynamic behavior of the physical system. Furthermore, the band-pass filter is eliminated and a Look-Up Table (LUT) is included to generate sinusoidal signals.
2. A novel switching model predictive control methodology is proposed based on the reconfigurable HES model as the predictive model.
3. Machine Learning techniques are employed to design a runtime switching algorithm that determines the optimal granularity level of the current predictive model in use.
4. Simulation experiments are conducted to evaluate the switching controller in a path following application containing curved and straight routes.

The rest of the chapter is organized as follows. In Section 4.4, the high-level architecture of the proposed switching predictive controller is described. The HES model is defined in Section 4.5. The proposed switching algorithm is described in Section 4.6. We demonstrated the workings and effectiveness of our framework for path following application in Section 4.7. Finally, we stated our conclusions in Section 4.8.

4.4 Switching Model Predictive Control

In general, a switching MPC system can be defined as a family of sub systems and a rule that orchestrates the switching among these subsystems as shown in Figure 4.2. The switching function can be classified into state-dependent or time-dependent based on the function that governs the switching rule [101]. In state-dependent switching, the state space is partitioned into several operating regions and the switching occurs when the system state reaches a certain switching surface. The switching system is defined as time-dependent, when a constant function of time decides the switch among models. The discrete-time linear switched system can be formulated as [101]:

$$\vec{z(k+1)} = A_\sigma \vec{z(k)} + B_\sigma \vec{u(k)} \tag{4.1a}$$

$$\vec{y(k)} = C_\sigma \vec{z(k)} \tag{4.1b}$$

where $\vec{z}$ is the state vector, $\vec{u}$ is the input vector and $\vec{y}$ is the output vector. At any time instant k, the switching function σ formulated in Equation 4.2 may be dependent on time, its past values, the state/output vectors and external signal and takes its value from $I^m = 1, ..., M$ where M is the number of subsystems.

$$\sigma(k_i) = \Phi([k_0, k_N), \sigma([k_0, k_N)), z([k_0, k_N))/y([k_0, k_N)) \qquad i \in 0, ..., N \tag{4.2}$$

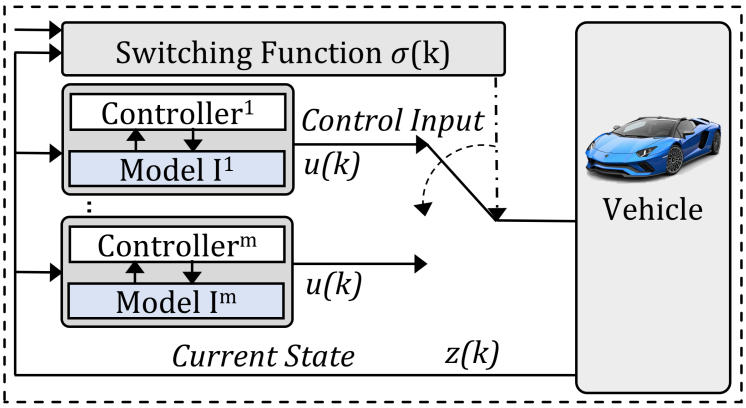


Figure 4.2: General Switching Model Predictive Control Architecture.

We proposed a state-dependent switching predictive control system based on HES models as the predictive model. The switching algorithm monitors the current dynamic state of the system and changes the configuration of the HES as a reconfigurable predictive model at run-time. The high level architecture of the proposed switching predictive control methodology is illustrated in Figure 4.3. MPC employs a predictive model to compute, at each sampling step, an optimal control problem over a finite prediction horizon. In switching predictive control, the controller selects among a library of predictive models with different levels of granularity based on a switching function σ that considers the control performance trade-off [100]. The predictive model in a discrete time domain can be expressed as:

$$z_i(k \vec{+} n|k) = f^m(z_i(\vec{k}|k), u_i(k \vec{+} n|k)) \tag{4.3}$$

where n is the number of time steps in the prediction horizon T and i is the index for number of variables. The notation $z_i(k \vec{+} n|k)$ refers to the value of the state variable z_i in time instant $k+n$, estimated at time k. The index $m \in 1, ..., M$ denotes the level of granularity for the predictive model currently in use. As depicted in Figure 4.3, we employed the reconfigurable predictive model HES^m to estimate the state vector variables of the prediction horizon

T. The HES model is described in Section 4.5. A switching algorithm is designed as a part of the controller to compute the optimal tuning parameters of a HES model based on the switching function σ, which is a function of state variables as described in [100]. This switching approach is elaborated in Section 4.6.

4.5 HES Model as Reconfigurable Predictive Model

The three main merits of the HES model that makes it a valid candidate for a predictive model in switching predictive control are:

- **Adaptive**: The adaptive term means that it can adapt to the behavior of real physical systems for different inputs. The model incorporates machine learning blocks. This empowers the HES model to be adaptive in run-time control applications in that it can fit the relation between any features and targets with proper training. The neural network model implemented in Section 4.5.2 and the experiments illustrated in Section 4.7 validate the working of machine learning models to estimate dynamic behavior of physical systems.

- **Reconfigurable**: Rather than designing a library of models, one HES model can be reconfigured for different levels of granularities at run-time. The tuning parameters introduced in Section 4.5.1 adopt this reconfigurability feature to the model.

- **Computationally Efficient and Handling Model Uncertainty**: HES model can generate multiple outputs as time series data. MPC employs a dynamic model of the physical system to predict the future outputs in a determined prediction horizon. The HES model can advantage MPC application in that the future outputs in the specified prediction horizon are generated all at once. This is as opposed to the ODE predictive models, which are generally required to be solved iteratively to estimate future outputs

in a certain prediction horizon. Moreover, the uncertainty is also handled by HES model as poor estimates are not accumulated over a long prediction horizon. This is as opposed to iterative methods that suffer from amplified noise in long prediction horizons.

The high-level architecture of the reconfigurable HES model in the MPC loop is shown in Figure 4.3. The model is composed of two main blocks: State Machine Generator and Harmonic Predictor. The architecture of HES model is based on the concept of signal decomposition and synthesis to generate a reconfigurable state machine model of a target physical system. The process of calculating the frequency domain information of the signal from time domain representation is called decomposition and the inverse process is signal synthesis. The **State Machine Generator** block captures the harmonic components of the output signal in a prediction horizon T provided by the **Harmonic Predictor**. These harmonic components are integrated into the synthesis function and the future state vector variables are computed as time series data. The granularity level of the final output is determined by the switching algorithm during the integration process.

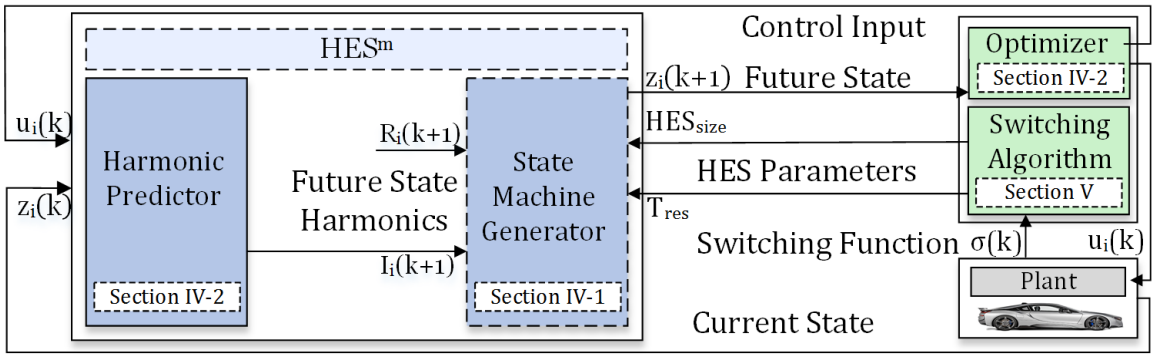


Figure 4.3: Switching model predictive control loop with HES as the predictive model.

4.5.1 State Machine Generator Block

This block captures the harmonic components from the Harmonic Predictor block and the tuning parameters from the switching algorithm to synthesize the future state vector variables $z_i(k \vec{+} n|k)$ in the form of $(N/2+1)$ concurrent state machines. These state machines are updated at the rate of frequency harmonics $\vec{Fr^z}$. The synthesis equation of FFT for signal $z_i(\vec{n})$ of size N is employed as presented in Equation (4.4). In this equation, n stands for the index of samples running from 0 to N-1. The vectors $\overline{R^{\vec{z}}}[i]$ and $\overline{I^{\vec{z}}}[i]$ are the normalized frequency spectrum coefficients for the sine and cosine waves with index i running from 0 to $N/2$ for the respective harmonic frequencies [79].

$$z[n] = \sum_{i=0}^{N/2} \overline{R^{\vec{z}}}[i] cos(2\pi in/N) + \sum_{i=0}^{N/2} \overline{I^{\vec{z}}}[i] sin(2\pi in/N) \tag{4.4}$$

The State Machine Generator block is established based on this synthesis Equation (4.4) to generate the reconfigurable representation of the output signal for models with various levels of granularity. A lookup table (LUT) is employed to collect the sinusoidal values for this equation. The use of LUT has improved the performance of the block drastically. The level of the granularity for the generated signal can be adjusted with respect to following model parameters [4].

Machine Size ($\boldsymbol{HES_{size}}$): defines the number of harmonic concurrent state machines to be integrated in the synthesis Equation (4.4) ranging from 1 to $(N/2+1)$.

Time Resolution ($\boldsymbol{T_{res}}$): is the global period at which rate the generated state machine will be executed.

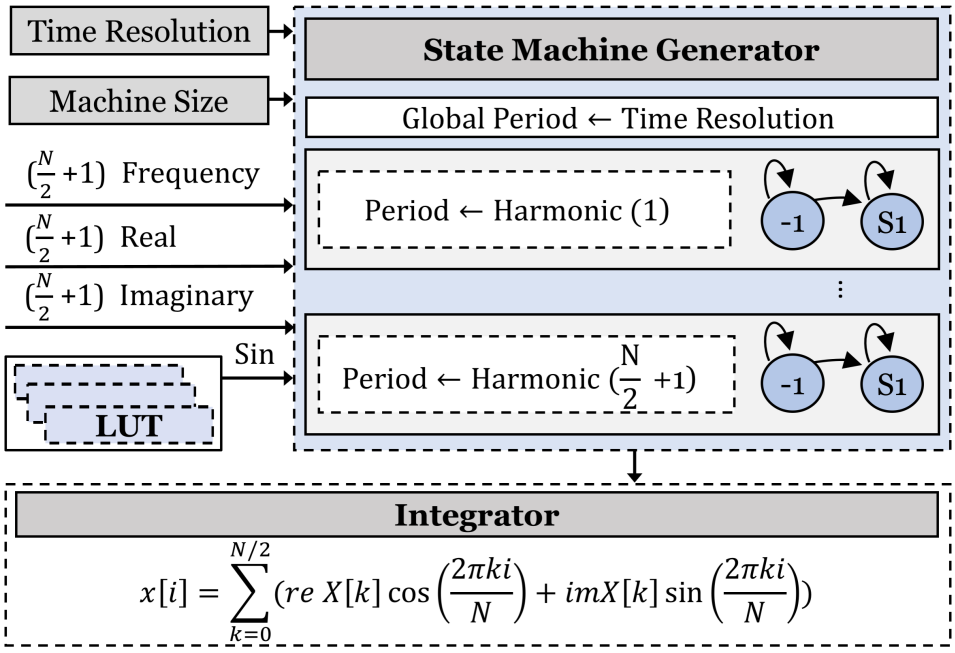


Figure 4.4: Concurrent state machine architecture.

Figure 4.4 illustrates the concurrent state machine architecture for the proposed methodology. HES_{size} concurrent state machines are executed at a global rate T_{res}, which is configured by the switching algorithm. This global clock represents the time resolution of the state machine. The outputs of these state machines are integrated into the synthesis equation 4.4 to compute the future state vector variables $z(k+\vec{n}|k)$ in the form of time series data. The HES model as the predictive model of the physical system expressed in Equation 4.3 should compute future state variables $z(k+\vec{n}|k)$ as a function of current state variables $z(\vec{k}|k)$ and future control inputs $u(k+\vec{n}|k)$. Therefore, the Harmonic Predictor block is designed to fit these variables to a function of machine learning model as described in the following section.

4.5.2 Harmonic Predictor Block

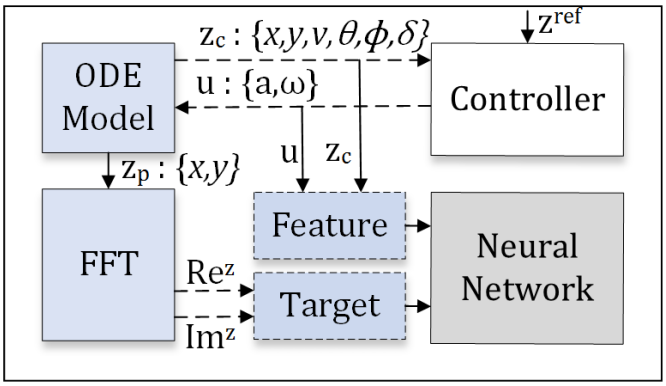
Neural Networks (NN) are capable of solving complex nonlinear relations between the input features and target outputs [7, 68, 89]. Classic NNs have a three layer structure, namely, input, hidden, and output layers. Each layer contains a set of nodes with edges to pass

forward the information. Each node carries an activation function e.g. sigmoid, that limits the variation to output values with respect to changes in NN parameters. The edges entering the nodes are associated with weights that are factors to inputs of the nodes—these weights are selected in the neural network framework using a training algorithm that minimizes a cost function. We applied neural networks to design the proposed Harmonic Predictor block. We mentioned this block design in Chapter 3 Section 3.4. Here, we will elaborate more on the details of the design. This block contributes the most to estimate the dynamic behavior of the physical system as in Equation 4.3. The input features of the NN model are control input $u_i(k \vec{+} n|k)$ and current state $z_i(\vec{k}|k)$ vector variables concatenated respectively. The target outputs are real and imaginary—$Re_i(k \vec{+} n|k)$ and $Im_i(k \vec{+} n|k)$—components of state vector variables $z_i(k \vec{+} n|k)$ in the next n time steps.

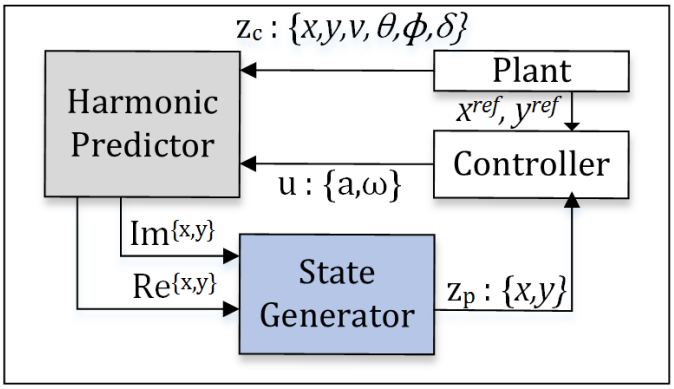
To better represent the behavior of the real physical system, we modified this block in [5] to accept all the current state variables in addition to control inputs as the additional features to the NN model. Moreover, we increased the number of nodes in the hidden layer to mean of input features and target outputs sizes according to an empirically-derived rules-of-thumb [40]. This is to fit a more complex pattern and improve the accuracy which comes as a trade-off for more computational overhead. However, the HES model with better execution time have space for more complex NN with better accuracy. This is due to the replacement of the filter with the lookup table which not only enhances the accuracy but also saves computation time. The results in Section 4.7.2 evaluate the performance of these two architectures. The Harmonic Predictor block is employed in the following training and prediction steps:

1. Training: The process of training the NN is performed in two phases. First, the architecture of NN is determined with respect to the number of hidden layers, hidden neurons and layer types (e.g. Fully-connected). This part of the design of the architecture is usually done empirically. We employed all fully-connected layers with one hidden layer for our architecture. Once the architecture is defined, a training algorithm is employed to adjust the weight

values until the NN reaches the performance objective. The weight adjustment is frequently done using the back-propagation algorithm or some extension of it [58]. For our training algorithm we used the Damped Least-Squares (DLS) which is a combination of Gradient Descent and Gauss-Newton methods [69]. This algorithm is initially designed as a numerical method to minimize computing sums of squares of nonlinear functions. It also benefits the neural network training, where the performance metric is the mean squared error. To collect the input features and target output values for the training datasets, an offline simulation of MPC application is conducted with ODE model of the physical system as shown in Figure 4.5(a). That is, the NN model aims to estimate the behavior of the ODE equivalent. Therefore, this ODE model determines the maximum level of granularity available in the proposed HES model. We assume that mathematical models are well designed to accurately



(a) Training.



(b) Prediction.

Figure 4.5: Training and prediction for Harmonic Predictor block [5].

capture the dynamic behavior of real physical systems. Here, the proposed method is described and evaluated in MPC for path following of autonomous vehicle application. It needs to be noted that the proposed methodology is generic to all MPC applications. The ODE model of the vehicle dynamics [100] as shown in Figure 4.6 is formulated as

$$\dot{x} = v\sin(\theta) \tag{4.5a}$$

$$\dot{y} = v\cos(\theta) \tag{4.5b}$$

$$\dot{v} = \cos(\delta)a - \frac{2}{m}F_{y,f}\sin(\delta) \tag{4.5c}$$

$$\dot{\theta} = \phi \tag{4.5d}$$

$$\dot{\phi} = \frac{1}{J}(L_a(ma\sin(\delta) + 2F_{y,f}\cos(\delta)) - 2L_bF_{y,r}) \tag{4.5e}$$

$$\dot{\delta} = \omega \tag{4.5f}$$

where x and y are longitudinal and lateral positions, v and a are longitudinal velocity and acceleration, θ is the yaw angle, and ϕ is the yaw rate. The variables δ and ω represent the steering angle and angular speed respectively. The variables L_a and L_b are the distance of sprung mass center of gravity from the front and rear axles respectively, and J is the angular momentum. The variables $F_{y,f}$ and $F_{y,r}$ stand for front and rear tire lateral force. More details regarding the model may be found in [5, 100].

These forces are computed from the following equations:

$$F_{y,f} = C_y(\delta - \frac{L_a\phi}{v}) \tag{4.6}$$

$$F_{y,r} = C_y(\frac{L_b\phi}{v}) \tag{4.7}$$

where C_y is lateral tire stiffness. We applied real-world parameters of 2011 Ford Fusion as L_a=L_b=1.5m, mass m=1700 kg and tire stiffness data for our experiments. The MPC formulation to follow the reference trajectory x^r, y^r is the solution to the following optimization problem:

$$\min_{x,y} \quad \sum_{t=0}^{T_p} \|\hat{x}(k+1|k) - x^r(k+1|k)\|^2_{Q_c} \tag{4.8a}$$

$$+ \|\hat{y}(k+1|k) - y^r(k+1|k)\|^2_{Q_c} \tag{4.8b}$$

$s.t.$

$$-0.75 \leq \delta \leq 0.75 \tag{4.8c}$$

$$-3 \leq \omega \leq 3 \tag{4.8d}$$

$$-40 \leq a \leq 40 \tag{4.8e}$$

MPC is simulated to optimize the control input vector variables $u_i(k \vec{+} n|k)$ for the predic-

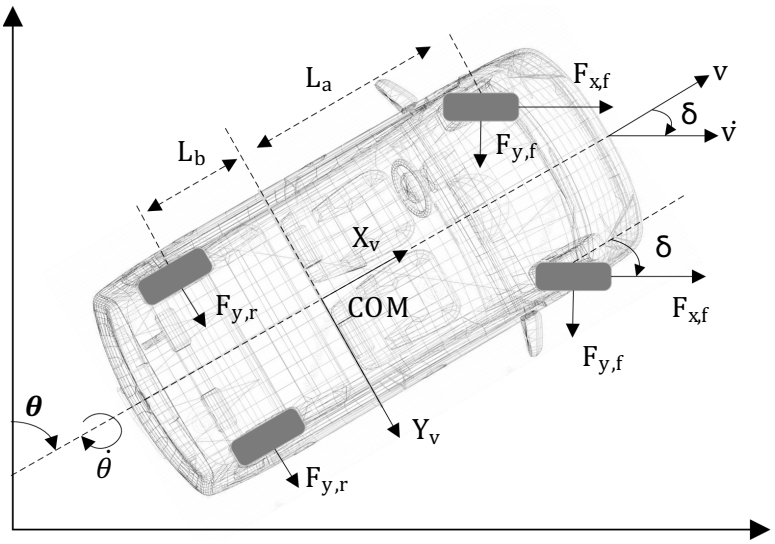


Figure 4.6: Schematic view of the vehicle model [5].

tion horizon T with respect to the cost function and enforced constraints. These control input vector variables are fed as the feature values to the NN model. To be a better representation of Equation (4.3), we included the current state vector variables $z_c(\vec{k}|k)$ from the actual plant as additional features to the NN model. Therefore, for the ODE example formulated in Equation (4.5), we considered the acceleration and steering angular speed as our control input variables $u_i \in \{a, \omega\}$ to predict future states $z_p(k \vec{+} n|k) \in \{x, y\}$. For current state vector variables $z_c(\vec{k}|k)$, we employed all the state variables from Equation (4.5) as $\vec{z_c} \in \{x, y, v, \theta, \phi, \delta\}$.

Next, the State Machine Generator block, accepts only frequency domain components as the input. Therefore, the target outputs of the NN model should be the $Re_i(k \vec{+} n|k)$ and $Im_i(k \vec{+} n|k)$ as the frequency information of state vector variables $z_i(k \vec{+} n|k)$ in the next n time steps. Recall that the Fast Fourier Transform (FFT) algorithm on a sample signal of size N decomposes the signal into real and imaginary components of size $(N/2+1)$. Therefore, here the predicted state vector variables $z_i(k \vec{+} n|n)$ from simulation of ODE are fed into FFT algorithm in time windows of T to derive the frequency information. The training is performed offline and adds no additional computational complexity at run-time to the application.

2. Prediction: The mapping function that is established during the training phase where the NN learns to correctly associate input patterns to output patterns is automatically retrieved during run-time prediction. Therefore, run-time control input vector variables $u_i \in \{a, \omega\}$ in the next n time steps and current state variables $\vec{z_i} \in \{x, y, v, \theta, \phi, \delta\}$ are fed into the NN predictor as shown in Figure 4.5(b) and the harmonic components of the future output vector variables—$Re_i(k \vec{+} n|k)$ and $Im_i(k \vec{+} n|k)$—are estimated. The predicted harmonic information is fed into the State Machine Generator block for output generation—that is, the output of the proposed physical model $z_i(k \vec{+} n|k) \in \{x, y\}$ can adapt to variations in control inputs $u_i(k \vec{+} n|k)$ at run-time as in Equation (4.3).

Here, we must assume that the predictive model given as in Equation (4.5) is suitable for the plant under control. Our approach is neither intended to stabilize systems with a large model mismatch or guarantee if switching is sufficiently slow. Here, we assume that selecting a model will not drive the system to a point of instability since otherwise that model would not be selected a suitable predictive model for our MPC controller.

4.6 Switching Algorithm

We designed a run-time switching algorithm based on the HES model mentioned above. The purpose here, is to choose values for the tuning parameters of HES model that **reconfigures** the model for the desired optimal granularity level **in run-time**. Research shows that, the optimal granularity level for the predictive model in MPC applications varies based on a metric that formulates the trade-off between the error and computational savings due to model reduction [100]. This metric can be associated with state variables of the physical system as in Equation 4.2. For instance, the work in [18] proposed a multi-model switching predictive control strategy that employs the speed variable to schedule the switching rules of the controller. Accordingly, in switching predictive control application, the dynamic state of the system may be monitored to select the optimal granularity level for the predictive model. For that, we formulated two switching functions: σ_{state} and σ_{opt}. The former that we call the state metric is formulated as a function of dynamic state of the system. The latter, optimal granularity metric, is to coordinate the trade-off between the error and computation time for optimal configuration of HES model.

We used the following function of steering angle δ and velocity v from [100] as our switching

function σ_{state} to compute the dynamic state of autonomous vehicles for $z_i(\vec{k}|k) \in \{v, \delta\}$:

$$\sigma_{state}(k|k) = \Phi\ (z_i(\vec{k}|k)) = v(k) - c|\frac{1}{\delta}| \tag{4.9}$$

The optimal granularity metric σ_{opt} is defined as the ratio of execution time e to model divergence d. Model divergence captures the error between the current model and the model with highest level of granularity.

$$\sigma_{opt}(k|k) = \Phi\ (e, d) = \frac{e}{d} \tag{4.10}$$

The current dynamic state of the system $z_i(k|k)$ at time instant k defines the range for σ_{state} switching function. Moreover, the range for σ_{opt} switching function is defined by the available granularity levels $m \in 1, ..., M$ for the predictive models HES^m which are determined by the its tuning parameters. The switching algorithm computes the current dynamic state of the physical system from σ_{state} and associates this value to a range for σ_{opt} as in Equation 4.11. That is, the switching algorithm maps the current dynamic state of the system to an optimal granularity level for efficient performance throughout the reference trajectory. The switching algorithm determines the parameter values for HES model with respect to the computed optimal granularity level.

$$\sigma_{opt}(k|k) = \alpha \times \sigma_{state}\vec{(k|k)} + \beta \tag{4.11}$$

The parameters α and β are adjusted to map the values of state-dependent switching function σ_{state} to the range for optimal granularity metric σ_{opt}. This mapping enables HES model configuration based on the current dynamic state of the system for efficient performance. The value for parameters α and β varies based on the form of the reference path and the number of desired granularity levels. In order to compute the values for α and β, Equation 4.9 is employed to approximate the range for current dynamic state throughout the reference path $r = [\ x^r\ ,\ y^r\ ]$. The value for σ_{state} defines higher optimal granularity levels for large steering angles and small velocity values for the vehicle. On the other hand, the optimal granularity level, decreases with smaller values for steering angle and larger velocities. This relation can associate the optimal granularity level of the predictive model with the reference path's degree of curvature. Research shows that the optimal granularity level needed for curved path where the vehicle is driving with slower speed and larger steering angle value is higher than in straight routes [100].

We used the following equations to estimate the values of velocity v and steering angle δ to travel the target reference path in distance intervals of Δs meters.

$$\Delta s^r = \sqrt{\Delta x_r^2 + \Delta y_r^2} \tag{4.12}$$

$$v^r = \frac{\Delta s^r}{\Delta t^r} \tag{4.13}$$

$$\theta^r = \arctan \frac{\Delta x^r}{\Delta y^r} \tag{4.14}$$

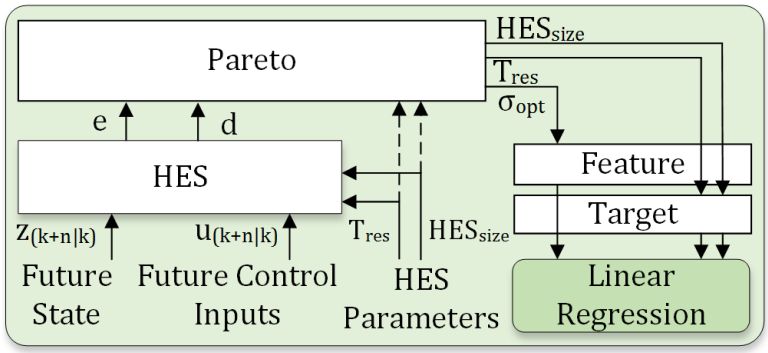
$$\delta^r = \Delta \theta^r \tag{4.15}$$

These values are employed in Equation (4.9) to approximate the vector σ_{state} for the reference path r. To collect values for optimal granularity level as in σ_{opt} , the MPC simulation

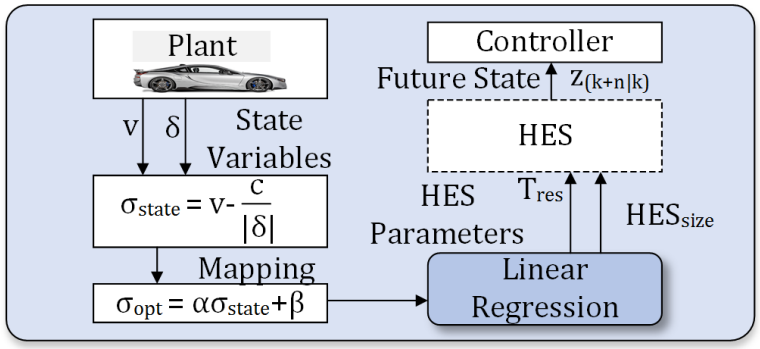
for path following application is conducted for t seconds with ODE model of a vehicle as the predictive model. The predicted output values $z_i(k \vec{+} n|k) \in \{x, y\}$ are recorded for prediction horizon of size T each representing a vector of size n for n is the number of steps in the prediction horizon. Then, these vector values are fed to FFT function to generate their harmonic components which are used as inputs to HES model. The HES model is executed for configurations of tuning parameters HES_{size}=[HES^{min}_{size}, HES^{max}_{size}] and T_{res}=[T^{min}_{res}, T^{max}_{res}] dynamically and generates vectors of predicted outputs $z_i(k \vec{+} n|k) \in \{x, y\}$. The performance metrics—execution time and model divergence— are recorded for each configuration over different prediction horizons in the simulation time. We computed the mean of these performance metrics throughout the simulation time. Design space exploration is performed to select the Pareto optimal points from the possible pairs of model parameters (HES_{size}, T_{res}) considering the trade-off between execution time and model error. The ratio of execution time to model divergence for these Pareto optimal points defines the optimal granularity metric σ_{opt}. Now that ample values for σ_{state} and σ_{opt} are collected, the Equation 4.11 and its respective parameters α and β from can be computed through data-fitting for later usage. The optimal granularity valuesσ_{opt} and respective pairs of model parameters ($HES^{opt}_{size}, T^{opt}_{res}$) are later employed as the training data in the proposed machine learning model.

Machine learning techniques are applied to predict the tuning parameters of the HES model for the desired optimal granularity level σ_{opt}. We employed linear regression machine learning models that use $i \in [1, n]$ number of feature values g_i and their respective weights b_i to predict target outputs s_i as in Equation 4.16. The relation can be fitted on a line using least squares method (LS) that minimizes the sum of the squares of the vertical distance from each data point on the line [37]. The model is implemented in two training and prediction steps as illustrated in Figure 4.7.

$$s_i = b_0 + b_i g_i \tag{4.16}$$



(a) Training.



(b) Prediction.

Figure 4.7: Training and prediction for the switching algorithm.

1. Training: The linear regression model is trained to fit the relation between the optimal granularity level σ_{opt} as the input feature and respective HES model parameters —HES_{size} and T_{res}— as the target outputs. The training data set is collected from the above mentioned design space exploration experiment that collects corresponding optimal granularity values σ_{opt} and respective pairs of model parameters $(HES^{opt}_{size}, T^{opt}_{res})$.

2. Prediction: As shown in the switching Algorithm 2, the values for current state variables —velocity v and steering angle δ— are used to calculate the metric σ_{state} from Equation (4.9) at run-time. These values are captured from the simulation of predictive controller for path following application with current HES model as the predictive model. The value of σ_{state} is inserted in Equation (4.11) to fit in the range of optimal metric σ_{opt}. Then, the σ_{opt} value is fed to the linear regression model as the input feature to predict the corresponding HES

model parameter pair (HES_{size}, T_{res})—that is, the switching algorithm estimates the values for HES model's tuning parameters in that the granularity level of the predictive model is optimal with respect to performance metrics.

ALGORITHM 2: Switching Algorithm for MPC

Input: Current State Variables z
Output: Estimated (HES_{size}, T_{res})

define α, β ▷ equation 4.11
define b_0, b_1 ▷ linear regression training function
$v = z[0]$ ▷ extract velocity and steering angle
$\delta = z[1]$
$\sigma_{state} = v - \frac{c}{\delta}$ ▷ calculate current dynamic state
$\sigma_{opt} = \alpha\ \sigma_{state} + \beta$ ▷ find optimal granularity
$g \leftarrow \sigma_{opt}$
$s_1 = b_0 + b_1\ g_1$ ▷ predict using the regression
$T_{\text{res}} \leftarrow s[0]$ ▷ extract HES parameters
MachineSize $\leftarrow s[1]$
return $[T_{\text{res}}, \text{MachineSize}]$

4.7 Experimental Results

4.7.1 Experimental Setup

Our experiments are performed on a PC with a quad-core Intel Core i7 and 16 GB of DDR3 RAM. The MPC formulation is implemented in software using a framework based on the ACADO Toolkit [42] which is an open source software written in C++ for automatic control and dynamic optimization. It provides a self contained environment to implement control algorithms including MPC as well as state and parameter estimation. The existence of Lyapunov function ensures the stability of autonomous dynamical systems [54]. Therefore, here the so-called LYAPINT integrator in ACADO Toolkit as an explicit Runge-Kutta45 integrator with an appropriate step size control is applied. The State Machine Generator

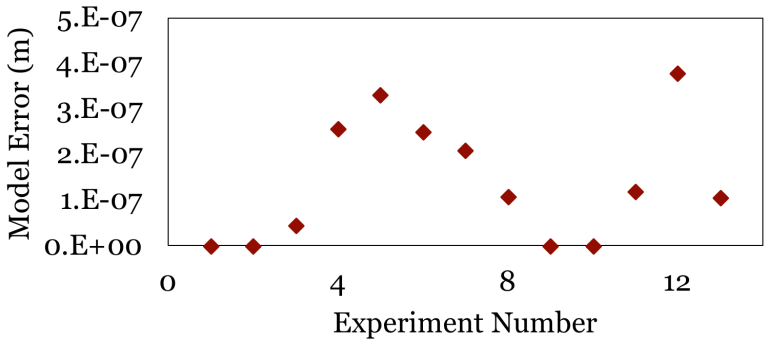


Figure 4.8: Test error for neural network model in Harmonic Predictor block.

block is implemented using the C++ programming language in order to enable it to be highly portable and compatible with various platforms for compilation and execution. The Neural Network model is trained by using MATLAB's neural networks module (nftool). The regression model in the switching algorithm is also implemented in MATLAB. Figure 4.8 illustrates the values for test error of the NN model described in Section 4.5.2 for 13 simulation experiments. The Neural Network described in Section 4.5.2 is trained using 266 training batches for 70 features in the input layer and prediction horizon of size T= 1.55 seconds. The number of neurons in the hidden and output layers are 60 and 68 respectively. As shown in the figure, the value for the error is in micro range which validates the the performance of the NN described in Section 4.5.2 to predict the future dynamic behavior of the physical system.

4.7.2 Comparison to State-of-the-Art

We compared the performance of the HES model described in Section 4.5 with respect to the ODE model of a vehicle formulated in Equation 4.5 in a run-time MPC application in path following. Figure 4.9 illustrates the error and execution time values of MPC using HES and ODE models, simulated for different prediction horizon sizes. As shown in the figure, the mean of execution time for ODE model and HES model are 7.63 ms and 1.25 ms respectively. This improvement in performance is gained at the expense of a minor increase in model error from average of 0.22 m to 0.25 m. The results indicate average of 83% reduction in MPC

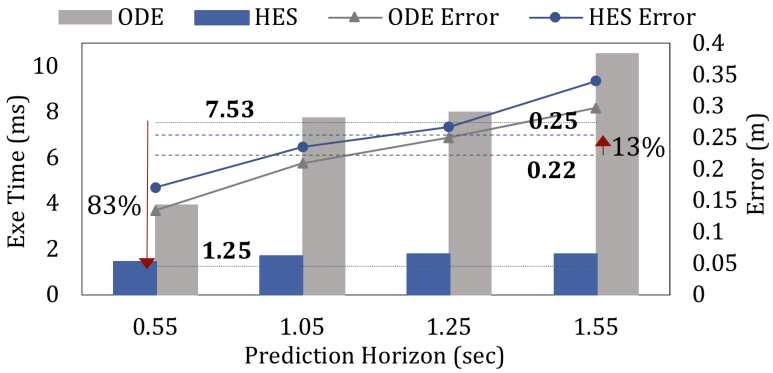


Figure 4.9: Performance comparison of ODE and HES models.

return time using HES model for negligible 13% loss in model accuracy. The improvement in accuracy and execution time compared with the values reported in [5], is due to the removal of the filter and the use of LUT and a more complex NN model in the new design.

Figure 4.10 illustrates the simulation results of MPC application in path following using the HES model as the predictive model. Here, the HES model estimates the dynamic behavior of the vehicle for 1.55 (sec) in the future. The initial velocity of the vehicle is taken as 24 (m/sec) to follow the reference trajectory as shown in the top left. As shown in the figure, the steering control inputs $\{a, \omega\}$ demanded by the controller enables the HES model to track the reference trajectory as the degree of curvature varies.

We conducted simulation experiments to evaluate the performance of the proposed switching predictive control methodology in a run-time path following application of an autonomous vehicle. As described in Section 4.6, we used a linear regression model in the switching algorithm to predict the parameters of the HES model based on the optimal level of model granularity in need. This optimal granularity level is aligned with the current dynamic state of the system. In order to compute the α and β coefficient in Equation 4.11, Equations 4.12-4.15 based on the the reference trajectory $r = [\ x^r\ ,\ y^r\ ]$ are used. Figure 4.11(a) shows the estimated values of σ_{state} throughout the reference trajectory. Figure 4.11(b) shows the Pareto optimal points of HES model parameters that are computed from the design space to collect the training data for the regression model. These Pareto optimal points are associated

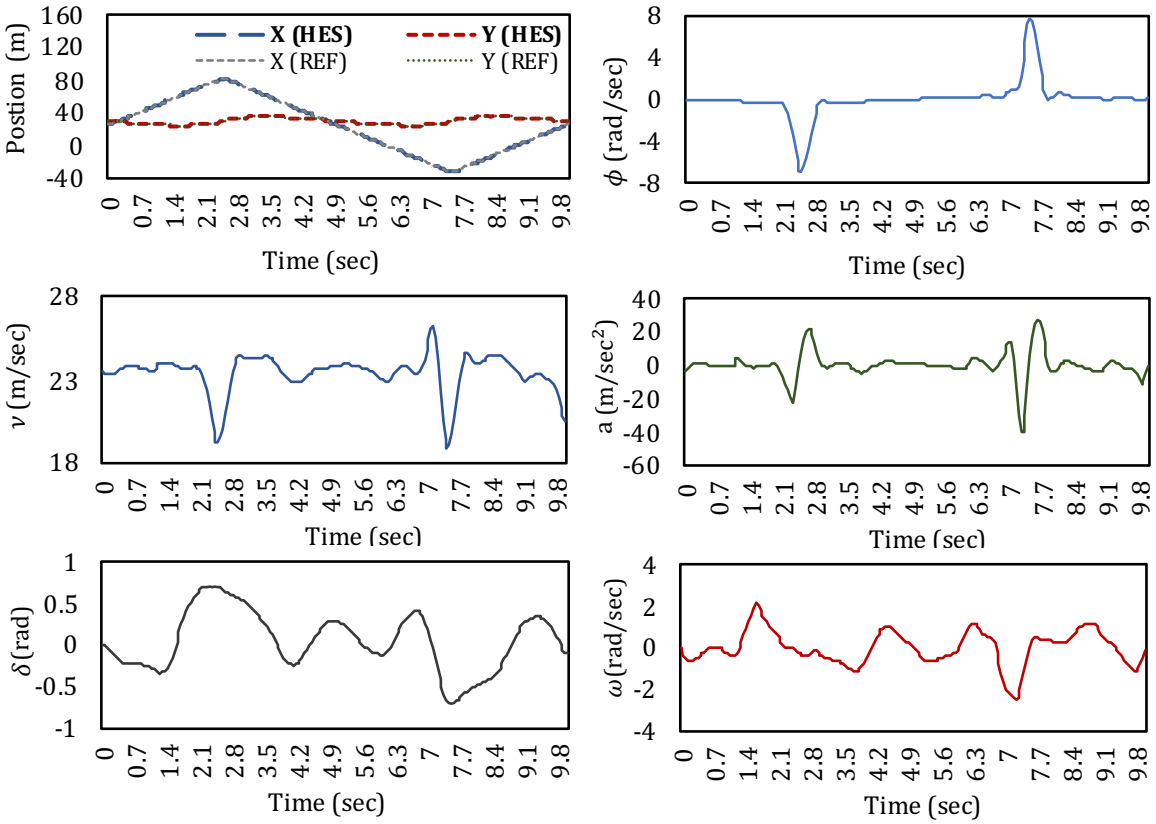
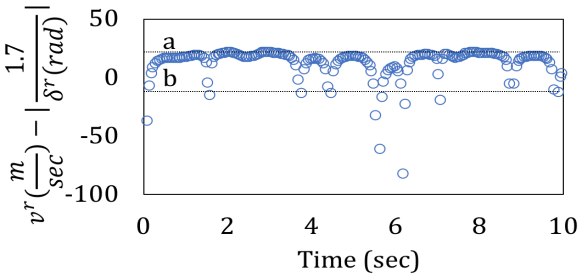


Figure 4.10: The simulation results of the MPC using the HES model as the predictive model.

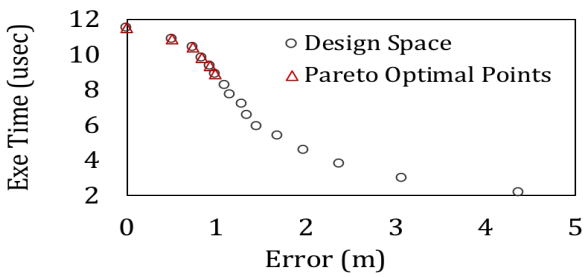
with pairs of $(HES_{size}, T_{res}) \in (13 : 17, 0.05)$ for five levels of granularity. We used these values to compute the optimal granularity metric σ_{opt} as the ratio of execution time e to model divergence d.

In order to compute the parameters α and β in Equation 4.11, we use the data collected in Figure 4.11(a) in the following equation.

$$\sigma_{opt} = \begin{cases} 1 & \sigma_{state} > a. \\ 0 & \sigma_{state} < b. \\ \alpha \times \vec{\sigma_{state}} + \beta & a \leqslant \sigma_{state} \leqslant b. \end{cases} \tag{4.17}$$



(a) σ_{state} estimation for reference trajectory r.



(b) σ_{opt} is computed as the ratio of execution time to model error for pareto optimal points

Figure 4.11: Computing switching functions σ_{state} and σ_{opt}.

Here, we consider $\alpha = 0.025$ and $\beta = 0.5$. As mentioned in Section 4.6, the α and β parameters depend on the form of the reference path and the number of desired granularity levels. Figure 4.12 compares the approximated values of σ_{opt} using Equation 5.10 and actual values computed using Equation 4.10. We can further tune the α and β parameters in order to adjust the over/under estimations shown in the figure.

The linear regression model fits the relation between the optimal granularity level σ_{opt} as the input feature and respective HES model parameters as the target outputs during the training phase.

To better evaluate the effectiveness of the switching controller, we selected the reference path to be a combination of straight and curved routes. For this purpose, we applied the

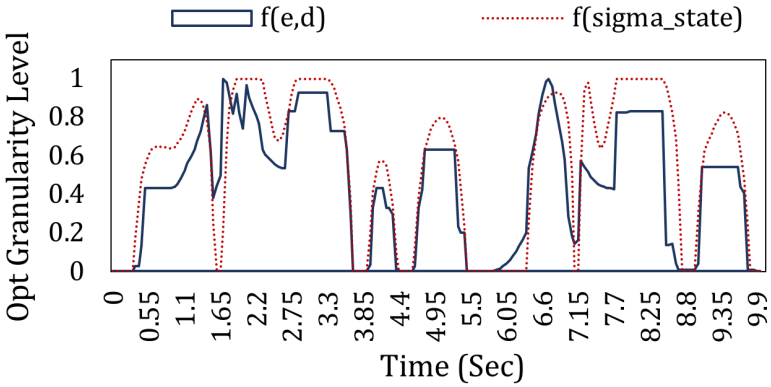


Figure 4.12: Comparing σ_{opt} as a function of (e, d) and as a function of σ_{statet}.

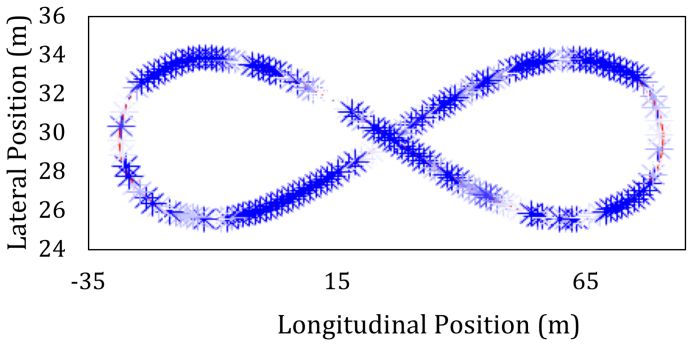


Figure 4.13: Switching predictive control application for path tracking. The red line shows the computed position of HES model in the single granularity control mode and the blue star markers represent the computed position for HES model in the switching control mode.

Lemniscate of Bernoulli function to generate our reference path. Figure 4.13 shows the performance of the proposed switching predictive control methodology for tracking the Bernoulli path. The red line is representing the x, y values for the single granularity control mode and star markers are for switching control mode. The granularity level of the predictive HES model configured by the switching algorithm is associated with the RGB value of the star markers. That is, higher granularity levels are color mapped to higher RGB values, hence, lighter blues. The gradual increase in granularity level (lighter blue) as the vehicle enters the curved route validates the performance of the proposed switching algorithm in selecting the parameters and reconfiguring the HES model appropriately.

The values for the optimal granularity level σ_{opt} calculated from Equation 4.11 and run-time steering angle δ throughout the simulation of MPC are depicted in Figure 4.14. The respective HES model parameters— HES_{size} and T_{res}— are predicted by the regression model to adjust the desired optimal granularity level. The HES parameter T_{res} is set to constant value of 0.05 seconds. Higher values of HES_{size} reconfigures the model for higher granularity levels. The results show the switching of model parameter HES_{size} with respect to the desired σ_{opt} value. As we expected, the optimal granularity level for when the vehicle is driving on a curved route with large steering angle value is higher than when on straight paths.

We compared the performance of the new improved version of the HES model in single

granularity and switching modes in Figure 4.15. The model in the single granularity mode is configured for the highest level of granularity with static parameters (HES_{size}, T_{res})= (17, 0.05) in this example. On the other hand, the switching algorithm is employed to reconfigure the HES model's parameters in run-time. Figure 4.15(a) illustrates the execution time values of the HES model in the switching and single granularity control modes throughout the simulation. These values represent the performance of the HES models in computation of output vectors $z(k+n|k)$ for the next n steps in the prediction horizon of size T. The execution time values reported for the switching mode are computed as the sum of the HES model's execution time and the overhead caused by the switching process. Since the linear regression model is formulated as a function of one input feature, the additional computational overhead in the switching mode is O(1) which is negligible. As shown in the figure, the mean execution time of MPC through the whole path using the HES model in the switching mode is 45% less than single granularity mode, dropping from 10.52 (us) to 7.27 (us). This is due to the presence of the switching algorithm to reconfigure the HES model for optimal performance with respect to dynamic state of the vehicle—that is the switching algorithm selects higher values for (HES_{size} parameter and reconfigures the model to maintain high execution time and granularity level on a curved route with large steering angle value and vice versa.

Figure 4.15(b) compares the error values for the HES model in the switching and single granularity control modes throughout the reference path. The results show 0.5 (m) and 0.62

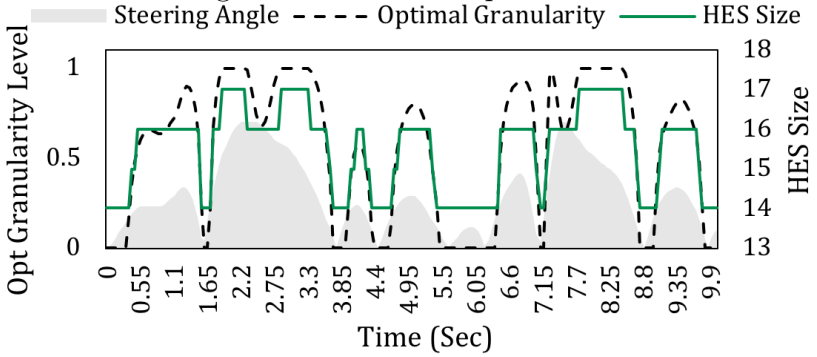
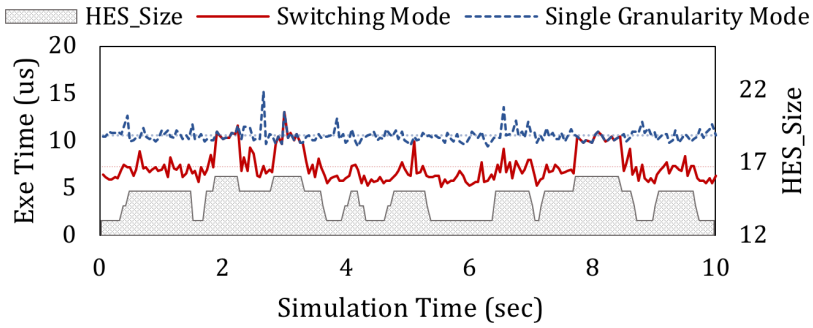
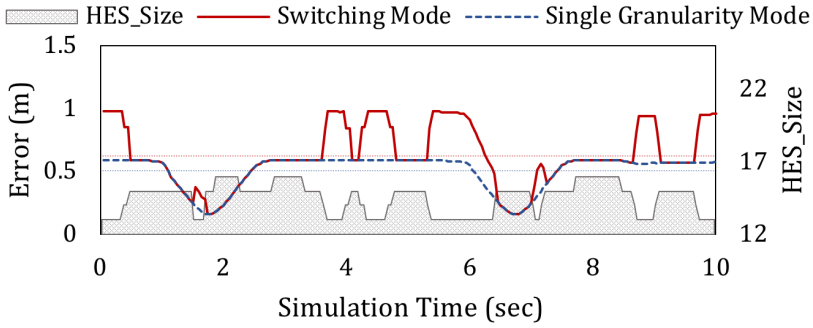


Figure 4.14: HES model granularity levels.



(a) Execution time.



(b) Error.

Figure 4.15: Performance analysis of HES model in two switching and single granularity modes.

(m) as comparable mean of error values for the HES model, in the single granularity and switching control schemes, respectively. This is because the switching algorithm is designed to reconfigure the HES model for lower levels of accuracy *when tolerated*, as in tracking a straight curve with high velocity and low steering angle values. That is, the changes in the range of error values for HES model corresponds with the optimal granularity level. This range is directly related to curvature of the path. It needs to be noted that, the drop in accuracy is only observed in the generation of future output vectors $z(k+n|k)$. However, the error to track the reference trajectory is comparable between the switching and single granularity modes with no drop in accuracy. Our experiments indicate that the use of HES model in the proposed switching scheme acquires 45% decrease in execution time for no loss in trajectory tracking accuracy. Moreover, our proposed switching control method is capable of choosing the optimal model for different velocity and steering angle values.

4.8 Conclusion

In this chapter, a novel switching predictive control methodology is proposed that uses model reduction to achieve a desired performance granularity for autonomous vehicles in path following applications. This method is based on a state-based model of the physical system that is able to adjust its granularity level dynamically. We apply machine learning models to design a switching algorithm. Experimental results show that our proposed switching control method decreases the overall execution time of MPC by 45% for a small 12% loss of accuracy in prediction of future output values and no loss of accuracy in tracking the reference trajectory.

Chapter 5

Priority Neuron: A Resource-Aware Neural Network for Cyber-Physical Systems

5.1 Introduction

In previous chapters we discuss the computational overhead in traditional MPC which grows exponentially with the length of the prediction horizon [11]. Research shows that a stable MPC controller requires a sufficiently large prediction horizon [47]. On the other hand, short prediction horizons are preferred for improved prediction accuracy of predictive models. This is because harmful effects of the poor estimates are amplified over a long prediction horizon time. Here, the problem is addressed by proposing an MPC approach that uses an adaptive prediction horizon with respect to quality measures [23]. However, the numerical effort needed in order to solve the optimal control problem for a long prediction horizon still remains significant. One approach to overcome the computational burden of long horizon

predictions is by implementing multi-rate prediction. In this approach, each look-ahead has a separate weight in the estimation of the steering input, where the furthest look-ahead point has the lowest weight [11].

Another method that is proposed to handle the computational issue associated with MPC systems is to use accelerated predictive models of the physical system. Different variants of NNs (e.g., recurrent neural networks [14]) hold promising performance for time-series prediction as they can easily be built to predict multiple steps ahead all at once. These models are well-known to have the ability to learn linear and non-linear relations between input and output variables without prior knowledge [27]. However, the use of NN models for long prediction horizon MPC problems could raise scalability and computational complexity challenges. The state-of-the-art methodologies are focused on reducing the size of the NN models without significantly affecting the performance [70, 73, 97]. These methodologies leverage the intrinsic error tolerance property of the NN models due to their parallel and distributed structure. Therefore, model reduction schemes could be exploited to employ the NN as the predictive model in the MPC loop. Several recent studies have focused on rescaling the size of the NN to adjust the resource usage on the embedded platform with respect to response time, power, and accuracy targets [74]. In other words, several sizes of the neural network are available at runtime to manage resources for inference time-, safety-, and energy-constrained tasks. Moreover, continuous learning of neural networks in data-driven modeling [87], transfer learning techniques [44], and adaptive modeling [38] impose significant training-time constraints at *runtime*.

5.2 Related Work

Advanced control methodologies have emerged for path planning and path following applications in modern vehicles. Nonlinear MPC is leveraged to develop path following control

systems while handling model uncertainties, constraints and nonlinearities. A predictive model of the physical plant is used to estimate the future outputs for a prediction horizon within a window of time and with respect to known input and output values. Mathematical descriptions in the form of Ordinary Differential Equations (ODE) are used to model the linear/nonlinear behavior of the physical system [90]. ODE solvers are applied to estimate solutions that converge to the exact solution of an equation or system of equations [53]. A runtime optimization routine is evaluated as a parametric quadratic function to calculate a set of future control inputs subject to constraints enforced by the environment and system dynamics. These routines are computationally intensive, and for nonlinear physical models, the computational overhead grows with complexity of the model [57].

One of the challenges in classic MPC is that the computational overhead increases with the length of the prediction horizon [11]. One approach to overcome the computational burden of long horizon predictions is by implementing a multi-rate prediction control strategy, where the prediction horizon is sampled in non-equidistant way [32]. In this approach, for a determined prediction horizon of n time steps, the initial steps have a shorter sampling period than the ones in the more distant future. In other words, fine tuning the control in such a way as to reduce the importance of predictions that contribute to time steps further in the future. Novel approaches are proposed for nonlinear dynamic system modeling and identification, where the NN realizes the behavior of a set of ordinary differential equations with smaller computation overhead [27, 41]. Moreover, data-driven neural networks are increasingly in demand. Data-driven neural networks are based on direct use of input-output observations collected from various real-world processes to perform system optimization, control and/or modeling [80]. Classic NNs have a three-layer structure, namely, input, hidden, and output layers. Each layer contains a set of neurons with edges to pass the information. The edges entering the neurons are associated with weight parameters. The weight parameters are adjusted in a training algorithm (e.g., by back propagation) so that the difference between the network's prediction and the target output is minimized.

Developing resource-efficient neural networks for embedded systems with limited hardware resources is a challenging task. To solve the memory complexity of NN models, many model compression approaches are proposed based on the claim that NN models have natural error tolerance because NNs usually contain more neurons than necessary to solve a given problem [85]. Many network pruning and model reduction techniques are proposed in previous work with promising results [20, 21, 34]. However, finding an optimal pruning solution is NP-hard and requires a costly retraining process [22]. Many works have focused on selecting weight parameters for pruning based on criteria such as magnitude of the weight, activation value for the respective neuron, and increase in training error [35, 39, 95]. Han et. al [36] proposed an iterative pruning method that removes all neuron connections whose weight is lower than a certain threshold. This approach converts a dense fully-connected layer into a sparser layer. The pruning is followed by a retraining process to boost the performance of the trimmed neural network. A common approach to reduce the size of the "parameter intensive" fully-connected layers is to reduce the magnitude of the overall weight parameters by including regularization terms in the model's cost function. Pan et. al [73] exploited regularization terms during the training process to simplify the NN model. At the end of the training, the NN is trimmed by dropping neurons below a certain threshold.

Another approach to address resource-constrained deployment of neural networks for embedded systems is to adapt the size of the neural network model to the performance requirements. Park et. el [74] address the energy complexity of neural networks using a novel big/little implementation, whereby a score margin metric is employed to select between the two sizes. This approach is memory intensive such that it requires storing separate sets of weights for different sizes of neural networks. Tann et. al [82] address the memory complexity problem by proposing a multi-step incremental training algorithm such that the weights trained in earlier steps are fixed. In this method, multiple sub-networks with different sizes are formed while storing and using only one sets of weight parameters. Although this approach is close to ours, our proposed method is more computationally flexible in generating multiple sub-

network sizes and does not suffer from a time-consuming retraining process. In the following section, we describe PNN, our proposed reconfigurable neural network model and its training algorithm.

5.3 Contributions

In this chapter, we propose Priority Neuron Network (PNN), a novel neural network model that is featured with a reconfigurable architecture. Our objective is to design a resource-aware reconfigurable NN model that not only computes the future outputs as time series data in constant time, but is also memory efficient. The summary of our contributions in this work are as follows:

- We develop a reconfigurable neural network model to fit the dynamic behavior of the physical systems for multi-step-ahead prediction in receding horizon problems. Our resource-aware NN model can be reconfigured to various network sizes at runtime while storing only *one set of weight parameters* for memory efficiency.
- We propose a training algorithm that controls the priority of each neuron in the computation of the model's output. We regulate the priority of each neuron using regularization techniques enforced on weight parameters. We consider the neuron's ordinal number as our priority criteria in that the priority of the neuron is inversely proportional to its ordinal number. We can reconfigure our NN model to smaller sizes by eliminating low priority neurons. This approach allows the trade-off between the model's computation time and accuracy in resource-constrained systems.
- We implement our reconfigurable NN model that contains multiple sub-networks using one-time training, hence reducing overall training time.

- Our priority-based training algorithm enforces a sorted distribution on activation values of neurons. This helps to reduce the computation complexity of the model reduction process when searching for n neurons below the pruning threshold, from O(n) to O(logn). It needs to be pointed out that we are not proposing a pruning methodology, but a memory efficient NN model that can be reconfigured to smaller sizes with less computation complexity at runtime.

- We apply our method to train a three-layer fully-connected NN model to be employed as the predictive model of a vehicle in MPC for path tracking application. We conduct closed-loop simulation of MPC using ODE predictive models to collect the training data. To evaluate the efficacy of our methodology, we compare it with two state-of-the-art approaches-Inc [82] and Big/Little [74]- that are targeted for resource-aware NN design in embedded systems. We show that our proposed PNN model outperforms the BL method with 89% reduction in training time and 78% saving in memory storage. The PNN model shows similar results to Inc method in terms of memory and model reduction complexity. However, we show that PNN follows a single training process to adjust weight parameters as opposed to Inc method that is based on multiple retraining. Therefore, the PNN model can cut down the training time by 86% with respect to Inc method while maintaining a better prediction performance from 0.25% to 0.21%.

The rest of the chapter is organized as follows. In Section 5.2, we summarize the state-of-the-art approaches to solve the computational complexity of MPC systems and design resource-efficient neural network models. We describe our proposed method in Section 5.4. We demonstrate the effectiveness of our framework for path following application in Section 5.5. Finally, we give our conclusions in Section 5.6.

5.4 Method

5.4.1 Application of Neural Networks in Model Predictive Control

Model predictive control exploits a predictive model of the physical system to produce an optimized control input sequence. The predictive model computes the output of the system, a number of time steps into the future based on the current output and future control input values. Therefore, the predictive model to estimate future outputs at time k in the next n time steps -$Y(k+n|k)$- can be formulated as a time series prediction function f of future control inputs $I(k+n|k)$ and a vector of current state variables $S(k|k)$ for $S = [S_0, S_1, ..., S_{N_s}]$. Time-series data is a sequence of time-ordered values as measurements of some physical process [81].

$$Y(k+n|k) = f(S(\vec{k}|k), I(k+n|k)) \tag{5.1}$$

The prediction function in Equation 5.1 can be fitted in a multiple input multiple output (MIMO) NN model with future control inputs and current state of the physical system as its input features and the future outputs in the next n time steps as its target outputs. Once the function is learned, the acyclic NN model computes the future outputs as a time-series data in *constant computing time* [27]. We use a three-layer fully connected Feed-Forward Neural Network (FFNN) to fit Equation 5.1 and approximate the dynamic behavior of the physical system. The FFNN is a class of NNs, where the input signal feeds forward through the network layers to the output in a single direction. Here, each layer of the network consists of computing neurons with edges that typically have a weight parameter. The output $\hat{y}_i$ of the neural network model can be computed as follows given x_k input features for $i \in \{1...N_o\}$

and $k \in \{1...N_i\}$:

$$\hat{y}_i = \sum_{j=1}^{N_h} [w_{ji}^2 \ \sigma(\sum_{k=1}^{N_i} w_{kj}^1 \vec{x_k} + \theta_j^1) + \theta_i^2] \tag{5.2}$$

where N_i, N_h, and N_o denote the numbers of input-layer, hidden-layer and output layer neurons, respectively. The parameters w_{kj}^1 and w_{ji}^2 are weights connecting the first layer to hidden layer and connecting the hidden layer to the output layer respectively and are adjusted in the learning process. The threshold offsets for the hidden and output layers are represented as θ^1 and θ^2. The function $\sigma(.)$ represents an activation functions, e.g., sigmoid, or Rectified Linear Unit (ReLU), that limits the variation to output values with respect to changes in NN parameters.

5.4.2 Architecture of Priority Neuron Neural Network as a Predictive Model in MPC

We propose PNN, a resource-aware reconfigurable NN such that the full model can be reconfigured to smaller sizes for less computation time and relatively comparable accuracy. Here, we deploy our proposed NN model for multi-step ahead time-series prediction in constant time for an MPC application. However, the proposed NN model can be generalized for other prediction applications, e.g., computer vision. As stated in Section 5.4.1, the non-linear model in Equation 5.1 is used by MPC to compute future behavior of the physical system can be fitted into a three-layer fully connected FFNN. The future control inputs and current state of the physical system are given as the input features to the FFNN to approximate the future outputs in the next n time steps. The proposed NN model can be described as in Equation 5.2 for $N_i = (\#$ of state variables$(N_s) + N_o)$ and $N_h = N_o = (\#$ of time time steps in the prediction horizon$(n))$. The value for N_h is set empirically equal to N_o. We have

two weight matrices W^1 and W^2 with sizes $(N_i \times N_h)$ and $(N_h \times N_o)$ containing connecting weights of our hidden and output layers, respectively. We use the Rectified Linear Unit (ReLU) activation function which is one of the most widely used activation functions and is defined as:

$$\sigma(z) = max(0, z) \tag{5.3}$$

During the prediction process of the NN, we would ideally want a few neurons in the network to not activate, thereby making the activations sparse and efficient. The ReLU activation function gives us the ability to design a sparser NN model because it outputs 0 for negative input values and imposes no constraint on the positive inputs. Equation 5.2 is broken down into Equations 5.4a and 5.4b to compute the outputs of hidden and output neurons, respectively. Here, for brevity, the bias parameters are deleted.

$$h_j = \sigma(\sum_{k=1}^{N_i} w_{kj}^1 \vec{x_k}) \tag{5.4a}$$

$$\hat{y_i} = \sum_{j=1}^{N_h} (w_{ji}^2 \; h_j) \tag{5.4b}$$

Hereafter, we are seeking a methodology for an architecture of a NN that stores one set of weight parameters yet can be reconfigured to smaller sizes of the NN with small drop in accuracy. To adopt the reconfigurability feature in our model, we exploit the multi-rate prediction idea suggested by [11] that assigns lower accent to further look-ahead points in the computation of the future dynamic behavior of the system. Therefore, the proposed PNN model follows a sequential priority-based architecture. This means we consider the neurons'

ordinal numbers as our priority criteria such that the priority of each neuron is inversely proportional to its ordinal number in the given layer. Therefore, the model can be reduced starting from the neuron with the highest ordinal number. Our goal is to synchronize the priority level of the output and hidden neurons so that the model reduction process is more computationally efficient for runtime applications. We will elaborate more on this in Section 5.4.4. In Figure 5.1 we show the architecture of the proposed PNN as a three-layer FFNN where higher priority neurons are colored darker. We can deploy PNN as a resource-aware predictive model for closed-loop MPC to estimate the future outputs $[Y_0, Y_1, ..., Y_{N_h}]$. Here, we use the future control inputs $[I_0, I_1, ..., I_{N_h}]$ and current state variables $[S_0, S_1, ..., S_{N_s}]$ as input features. In the following section, we describe our proposed training algorithm and the associated cost function to develop the priority-based NN model.

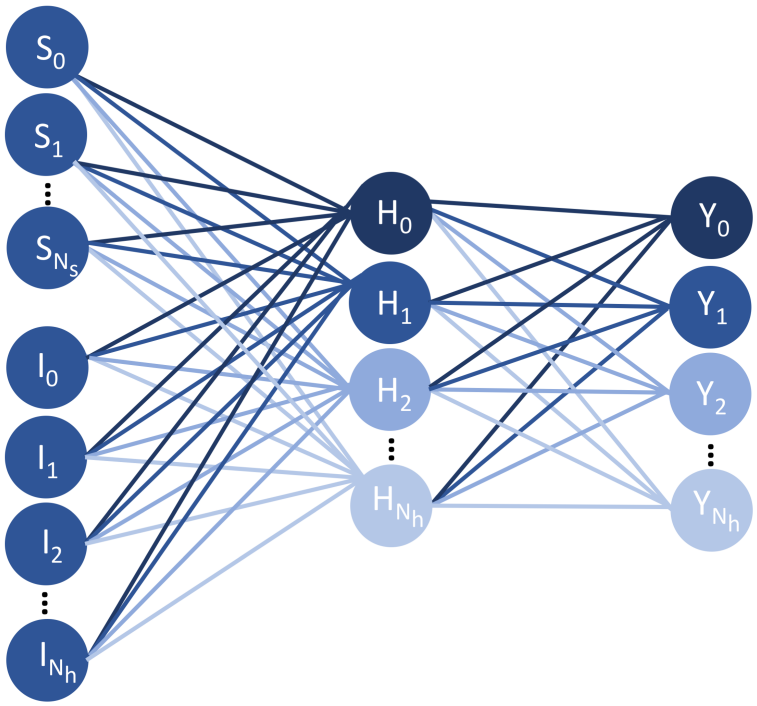


Figure 5.1: Priority Neuron Network (PNN) model.

5.4.3 Training Algorithm to Prioritize Neurons

During the training process of a NN, an optimization algorithm is exploited to minimize an objective function $E_0(.)$, which is simply a mathematical function based on the model's learning parameters (e.g. weights, biases). We might use sum of the squared deviations of our neuron's output $\hat{y}_i$ from the target output y_i as the loss function for N_o number of outputs denoted as:

$$E_0(w, b) = \frac{1}{2N_o} \sum_{i=1}^{N_o} (y_i - \hat{y}_i)^2 \tag{5.5}$$

The learning parameters are optimized and updated in an iterative training process toward a solution that minimizes the loss function. A learning rate η is assigned to the training algorithm that determines the size of the steps we take at each iteration to reach a (local) minimum. For a convex optimization problem like this, we use derivatives of the loss function ∇E. Therefore, the following updating rule is formulated for the weight parameters to be updated after (t+1)-th update iteration:

$$w^{t+1} \leftarrow w^t - \eta \nabla E_0 \tag{5.6}$$

For our optimization algorithm, we employ a variant of gradient descent called Adaptive Moment Estimation (Adam) [56] which computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. *In the proposed PNN model, the priority of the neuron determines how important the value of that neuron is in the overall performance of the NN.* In order to control the priority of each neuron, we enforce

constraints on the computation of its output value. This can be done through regularization techniques that restrain the growth of weight parameters. From Equation 5.4, we see that the weight parameters used to compute the hidden neuron h_j are $W^1[:, j] = [w^1_{1j}, w^1_{2j}, ..., w^1_{N_{ij}}]$. The output neuron $\hat{y_i}$ is computed using weight parameters $W^2[:, i] = [w^2_{1i}, w^2_{2i}, ..., w^2_{N_h i}]$. We call the weight parameters of each neuron its *associated weights*.

Regularization: A common approach to reduce the complexity and size of NN models is to constrain the magnitude of the overall weight parameters by including regularization terms in the model's cost function. The L1 norm is one of the most commonly used regularization techniques that penalizes weight values by adding the sum of their absolutes to the error term. Therefore, the cost function E with the L1 regularization term is:

$$E(w, b) = E_0(w, b) + \frac{1}{2}\lambda \sum_{l=1}^{2} \sum_{i=1}^{N_l} |W^l_i| \tag{5.7}$$

where λ is the weight decay coefficient for which larger values lead to larger cost, and causes the training algorithm to generate small weight values. Existing work sets the same weight decay coefficient for all layers to avoid the computational costs required to manually fine-tune each coefficient. However, to train our priority-based NN model, we penalize each weight with a specific weight decay coefficient so that the value of the corresponding weight is constrained to grow up only to a desired threshold point. Hence, the activation of each neuron is governed by the weight decay coefficients of its *associated weights*. As shown in Algorithm 3, we use a new cost function for our three-layer fully connected feed-forward PNN:

$$E(w, b) = E_0(w, b) + \frac{1}{2} \sum_{k=1}^{N_i} \sum_{j=1}^{N_h} |\lambda^1_{kj} w^1_{kj}| + \frac{1}{2} \sum_{j=1}^{N_h} \sum_{i=1}^{N_o} |\lambda^2_{ji} w^2_{ji}| \tag{5.8}$$

for $\lambda^1 \in \Lambda^1$ and $\lambda^2 \in \Lambda^2$ where Λ^1 and Λ^2 are two weight decay matrices of our hidden and output layers, respectively. Therefore, the new updating rule for weight parameters is:

$$w^{t+1} \leftarrow w^t - \eta(\nabla E_0 + \Lambda^1 W^1 + \Lambda^2 W^2) \tag{5.9}$$

In the following section, we describe our heuristic algorithm used to assign values to weight decay coefficients such that a sorted priority-based architecture is enforced on the proposed NN model.

ALGORITHM 3: Priority Neuron Training Algorithm

Input: input features - x
Input: output targets - y
Output: trained NN - PNN
Output: estimated outputs - $\hat{y}$

`// initialize NN weights`
init_random W

`// estimate outputs given W weights`
$\hat{y} = PNN\,(x)\;[W]$

`// evaluate residual error`
$err = \sum_{i=0}^{N_o} (y_i - \hat{y}_i)^2$

`// evaluate regularization penalty`
$reg = \sum |\Lambda^1_{N_i \times N_h} . W^1_{N_i \times N_h}| + \sum |\Lambda^2_{N_h \times N_o} . W^2_{N_h \times N_o}|$

`// evaluate loss function`
$loss = err + reg$

`// optimize W weights for minimal loss`
$\overline{W} =$ **AdamOptimizer** $(loss)$

`// estimate outputs given optimal` $\overline{W}$
$\hat{y} = PNN\,(x)\;[\overline{W}]$

return $[PNN, \hat{y}]$

5.4.4 Model Reconfiguration of PNN Model

In PNN, we want to force a priority onto each neuron during the computation of model output so that the *accuracy is maintained* after reconfiguring the network to smaller sub-networks by removing low priority neurons. Therefore, we consider larger weight decay coefficients for *associated weights* of neurons that are desired to have lower level of priority and vice versa. We are following the multi-rate prediction scheme that allocates less stress on accuracy of further look-ahead points. We design our weight decay matrices so that a sorted priority-based architecture for our PNN is developed during the training process. The intuition behind the sorted priority-based architecture of the PNN is to reduce the complexity of the model reconfiguration and reduction process. Model pruning approaches to constrain the complexity of NN models by applying regularization techniques, have been around for a while [21, 43]. These approaches are based on an exhaustive search process to remove neurons with activation values below a certain threshold. In our proposed priority-based architecture, we enforce a sorted priority on hidden neurons to compute the overall performance of the model. This helps reduce the time complexity for searching neurons below a certain activation value as we can employ a Binary Search algorithm. Therefore, the worst-case time complexity for the model pruning process in our PNN model with n number of hidden neurons is *O(logn)* as opposed to standard architectures that require *O(n)* worst-case time complexity to prune the network. Moreover, the model can be reduced to smaller sub-networks at constant time O(1) due to its reconfigurability feature that is adopted throughout the training process.

There is always a trade-off between the number of sub-networks and the accuracy of the model. We assign the same level of priority to the number of neurons that are deleted at each level of model reduction. We call this number the *priority size* and denote it as p. Figure 5.2 illustrates the reconfiguration process of the original NN model where neurons are sorted and colored in terms of priority and importance. At each level of reconfiguration, p number of hidden neurons with the least level of priority are deleted from the end of the

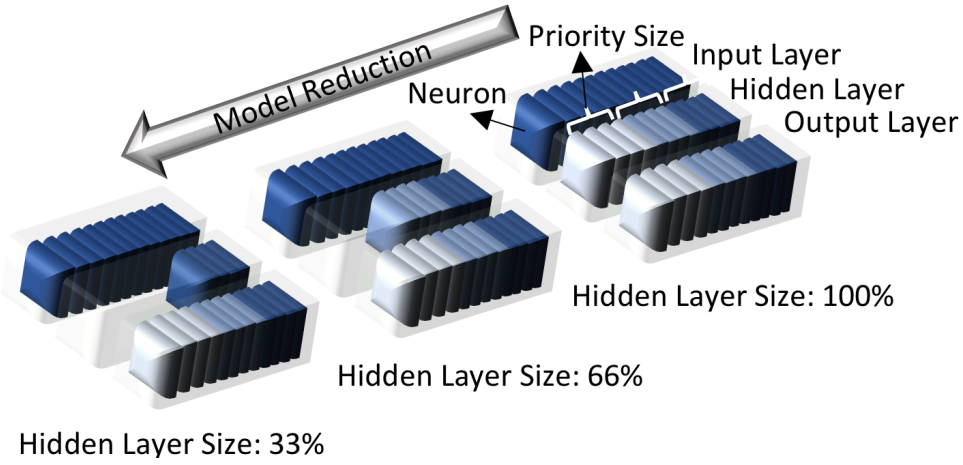


Figure 5.2: The model reduction process for a three-layer fully-connected NN with priority size p=4.

hidden layer. Hence, their input and output weight connections are also removed from the weight space of the neural network. These sub-networks can be deployed separately while reducing the memory complexity to a single network. In other words, only one set of weight parameters are stored for multiple sub-networks of different sizes. We consider neuron's ordinal number as our priority criteria which can be mapped into index values for neuron's associated weights. Therefore, the weight decays vary with respect to row and column indices of the weight matrix where r and c denote the row and column indices, respectively. Equations 5.10 and 5.11 are expanded from Equation 5.4. In Equation 5.11, we see N_o number of output formulas that are used to estimate the future output behavior of the physical system in the next N_o time steps, hence the size of the prediction horizon is N_o. It needs to be noted that, here we do not include the bias terms for simplification purposes.

$$h_0 = w^1_{00}s_0 + w^1_{10}s_1 + ... + w^1_{N_i0}I_{N_i} \tag{5.10a}$$

$$h_1 = w^1_{01}s_0 + w^1_{11}s_1 + ... + w^1_{N_i1}I_{N_i} \tag{5.10b}$$

...

$$h_{N_h} = w^1_{0N_h}s_0 + w^1_{1N_h}s_1 + ... + w^1_{N_iN_h}I_{N_i} \tag{5.10c}$$

$$y_0 = w^2_{00}h_0 + w^2_{10}h_1 + ... + w^2_{N_h0}h_{N_h} \tag{5.11a}$$

$$y_1 = w^2_{01}h_0 + w^2_{11}h_1 + ... + w^2_{N_h1}h_{N_h} \tag{5.11b}$$

...

$$y_{N_o} = w^2_{0N_o}h_0 + w^2_{1N_o}h_1 + ... + w^2_{N_hN_o}h_{N_h} \tag{5.11c}$$

Let us assume that the model is trained for a priority-based architecture where the priority of neurons decreases inversely with their ordinal number. For a pre-trained model with priority size $p = 1$, we want to reduce the size of the model by removing hidden neuron h_{N_h} with the least priority level from the hidden layer. While removing the hidden neuron h_{N_h}, its associated weight connections $W^1[:, N_h] = [w^1_{0N_h}, w^1_{1N_h}, ..., w^1_{N_iN_h}]$ and $W^2[N_h, :\ ] = [w^2_{N_h1}, w^2_{N_h2}, ..., w^2_{N_h(N_o-1)}]$ are removed from W^1 and W^2, respectively. In the next section we describe the selection of weight decay coefficients to enforce a sorted priority on hidden and output neurons. For a simple implementation we use the same number of hidden and output neurons. Therefore, the W^2 weight matrix is squared.

5.4.5 Decay Matrix

A graphical illustration of our W^1 and W^2 weight matrices for hidden and output layers with p=1 is shown in Figures 5.3 and 5.4, respectively. The weight matrices in Figures 5.3 and 5.4 are darker colored based on the value of their corresponding weight decay coefficients. This helps to visualize the selected distribution pattern for weight decay coefficients where a priority-based architecture for our PNN model is developed. In order to maintain the accuracy of the model after the removal of hidden neuron h_{N_h} (computed in Equation 5.10c),

we want the model reduction to affect the least number of output neurons possible. Therefore, we seek to adjust the weight parameters so that removing the hidden neuron h_{N_h} mostly impacts the least priority output neuron y_{N_o}. Hence, we select weight decay coefficients for the weight parameters in the vector $[w^2_{N_h 0}, w^2_{N_h 1}, ..., w^2_{N_h N_o}]$ in a descending order so that the least weight decay value is assigned for $w^2_{N_h N_o}$. Smaller weight decay coefficients push the training algorithm to assign greater values for the weight parameters. In this method, we try to zero out $[w^2_{N_h 0}, w^2_{N_h 1}, ..., w^2_{N_h (N_o-1)}]$ as much as possible such that the removal of h_{N_h} has minimal impact on the values $[y_0, y_1, ..., y_{(N_o-1)}]$.

W^1_{00}	W^1_{01}	W^1_{02}	⋯	$W^1_{0N_h}$
W^1_{10}	W^1_{11}	W^1_{12}		$W^1_{1N_h}$
⋮				
$W^1_{N_s 0}$	$W^1_{N_s 1}$	$W^1_{N_s 2}$		$W^1_{N_s N_h}$
$W^1_{N_{s+1} 0}$	$W^1_{N_{s+1} 1}$	$W^1_{N_{s+1} 2}$		$W^1_{N_{s+1} N_h}$
$W^1_{N_{s+2} 0}$	$W^1_{N_{s+2} 1}$	$W^1_{N_{s+2} 2}$		$W^1_{N_{s+2} N_h}$
$W^1_{N_{s+3} 0}$	$W^1_{N_{s+3} 1}$	$W^1_{N_{s+3} 2}$		$W^1_{N_{s+3} N_h}$
⋮			⋱	
$W^1_{N_i 0}$	$W^1_{N_i 1}$	$W^1_{N_i 2}$		$W^1_{N_i N_h}$

Figure 5.3: Weight parameters of hidden layer.

W^2_{00}	W^2_{01}	W^2_{02}	⋯	$W^2_{0N_o}$
W^2_{10}	W^2_{11}	W^2_{12}		$W^2_{1N_o}$
W^2_{20}	W^2_{21}	W^2_{22}		$W^2_{2N_o}$
⋮			⋱	
$W^2_{N_h 0}$	$W^2_{N_h 1}$	$W^2_{N_h 2}$		$W^2_{N_h N_o}$

Figure 5.4: Weight parameters of output layer.

To expand this idea to other neurons in the hidden layer, we should change the weight decay coefficients above the main diagonal of W^2, in descending order per column and in ascending order per row, so that the least weight decay coefficients are placed on the main diagonal. Moreover, we should adjust the weight decay coefficients below the main diagonal of W^2 in ascending order per column and in a descending order per row. We use ascending order per column so that the priority level of output neurons decreases for larger ordinal numbers and descending order per row forces the weight parameters on the diagonal to contribute the most to the computation of their corresponding output neuron. We propose Equation 5.12

to compute the weight decay coefficient for each weight parameter in order to regulate the sorted priority order of PNN neurons. Here, r and c denote the row and column index of the weight matrix, respectively. The parameter p stands for the number of neurons deleted at each model reduction process, hence the *priority size*.

$$f(x) = \begin{cases} [\lambda_{rc} : \lambda_{r(c+p)}] = \beta f(\frac{r}{c}), & r \geqslant c. \\ [\lambda_{rc} : \lambda_{(r+p)c}] = \beta f(\frac{c}{r}), & r < c. \end{cases} \tag{5.12}$$

Here, $f(.)$ can be considered as a linear, exponential, or logarithmic, etc. growth function considering the target application. The type of function $f(.)$ determines the variance of the priority distribution among various neurons at each layer. The greater the variance of the priority distribution is, the more ways the original NN can be reconfigured into sub-networks. That means less neurons (p) are deleted per model reconfiguration (reduction) process. Larger variance for the priority order of neurons decreases the model accuracy as it enforces more constraints on weight parameters. Therefore, the function $f(.)$ is assigned based on design requirements of the target application and the trade-off between the model accuracy and number of sub-networks embedded in one NN model. The parameter β maps the computed value of weight decay from Equation 5.12 to a range as $\lambda \in [\lambda_{min} : \lambda_{max}]$. This range is empirically selected based on the trade-off between the model accuracy and the number of hidden neurons deleted per reconfiguration of the model-priority size.

5.4.6 Other Types of Neural Networks

The proposed priority-based approach is applied to a fully-connected FFNN architecture. This is because state-of-the-art methods proposed fully-connected FFNN as a predictive model to approximate dynamic behavior of physical systems in a MPC application. Previous

state-of-the-art approaches has mostly focused on reducing the size of the fully-connected layers in other NN architectures because these layers are well known to be parameter intensive and occupy more than 90% of the model size [73]. Another popular architecture of NNs for time series forecasting is Recurrent Neural Network (RNN) which is distinguished from FFNN by having signals traveling in both directions and introducing loops in the network. The RNN architecture can be converted into a FFNN by unfolding over time [14].

5.5 Experimental Results

5.5.1 Experimental Setup

Our implementation is based on the TensorFlow framework [2] executed on a PC with a quad-core Intel Core i7 and 16 GB of DDR3 RAM. The MPC formulation is implemented in software using the ACADO Toolkit framework [42], which is open source software written in C++ for automatic control and dynamic optimization. To evaluate the efficacy of our proposed methodology, we exploit the PNN as a predictive model in a MPC system for the path following application. We describe the process on how we collect our training dataset in the following section.

5.5.2 Simulation to Collect Training Data

As mentioned in Section 5.2, the dynamic behavior of a physical system formulated as ODE can be fitted into a fully-connected FFNN. The future control inputs and current state of the physical system are fed as the input features to the FFNN in order to predict the future outputs in the next n time steps. To collect the training dataset, we exploit the following ODE model of a vehicle [100] as shown in Equation 5.13 and Figure 5.5 to conduct offline

simulation of MPC for a path following application.

$$\dot{s} = \begin{bmatrix} v\sin(\theta) \\ v\cos(\theta) \\ \cos(\delta)a - \frac{2}{m}F_{y,f}\sin(\delta) \\ \phi \\ \frac{1}{J}(L_a(ma\sin(\delta) + 2F_{y,f}\cos(\delta)) - 2L_bF_{y,r}) \\ \omega \end{bmatrix} \tag{5.13}$$

Here, $s = [x, y, v, \theta, \phi, \delta]$ is the vector of state variables with acceleration a and steering angular speed ω as control inputs. The variables x and y stand for longitudinal and lateral positions, v and θ are velocity and the azimuth. The variables δ and ϕ represent the steering angle and speed, respectively. The distance from sprung mass center of gravity to the front and rear axles are denoted as L_a and L_b, respectively, and J is the angular momentum. The variables $F_{y,f}$ and $F_{y,r}$ stand for front and rear tire lateral forces. These forces are computed from the following equations:

$$F_{y,f} = C_y(\delta - \frac{L_a\phi}{v}) \tag{5.14a}$$

$$F_{y,r} = C_y(\frac{L_b\phi}{v}) \tag{5.14b}$$

where C_y is the lateral tire stiffness. We applied real-world parameters of a 2011 Ford Fusion as L_a=L_b=1.5m, mass m=1700 kg and tire stiffness data for our experiments. The MPC formulation to follow the reference path x^r, y^r is the solution to the following optimization

problem:

$$\min_{x,y} \quad \sum_{t=0}^{T_p} \|\hat{x}(k+1|k) - x^r(k+1|k)\|^2_{Q_c} \tag{5.15a}$$

$$+ \|\hat{y}(k+1|k) - y^r(k+1|k)\|^2_{Q_c} \tag{5.15b}$$

$s.t.$

$$\delta_{min} \leq \delta \leq \delta_{max} \tag{5.15c}$$

$$\omega_{min} \leq \omega \leq \omega_{max} \tag{5.15d}$$

$$a_{min} \leq a \leq a_{max} \tag{5.15e}$$

We simulate the MPC to predict 101 time steps in the future with time intervals of 5.05 seconds for a vehicle with an average speed of $v = 10(m/sec)$. The appropriate value for the prediction horizon and step size is bounded by some factors such as stability and accuracy requirements and it varies based on plant dynamic characteristics. We implement a FFNN with input size $N_i = 6 + 102$ for six values of current state variables and future control inputs in the next 101 time steps. We select $N_o = 102$ as the output size for our NN to predict the future output of the physical system in the next 101 time steps. The number of hidden neurons in our three-layer FFNN are $N_h = N_o$.

5.5.3 PNN Training

In order to fine tune the range of weight decay coefficients $\lambda \in [\lambda_{min} : \lambda_{max}]$ and select an appropriate value for the constant factor β in Equation 5.12, we empirically pick the values that yield the best performance on a held-out dataset. Therefore, we conducted experiments based on five different ranges of coefficients. Figure 5.6 shows the error rate of the PNN model with respect to variations in the range of weight decay coefficients. The optimal

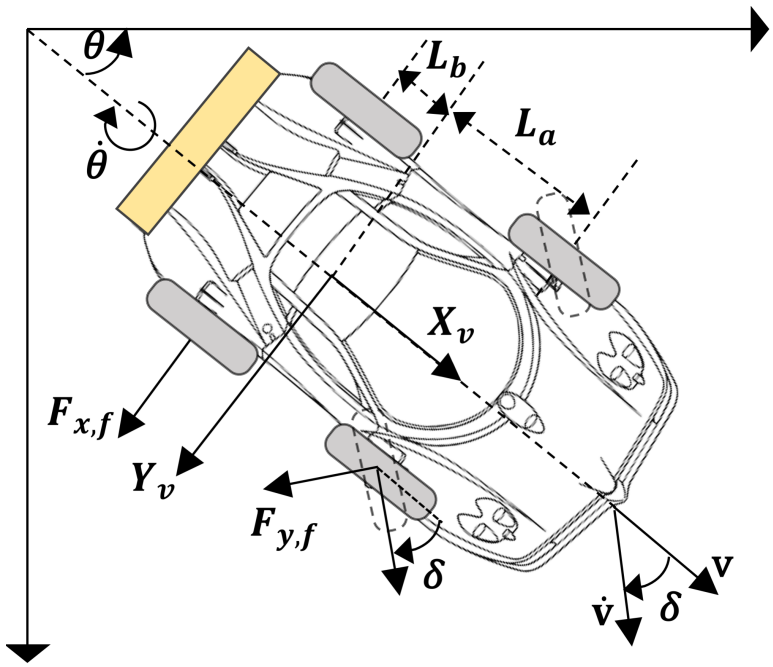


Figure 5.5: Schematic view of the vehicle model.

range of weight decay coefficients for each layer may change with respect to the size of the next layer. In back propagation training, the gradient term in Equation 5.9 is scaled with the size of the next layer [46]. Therefore, to compensate for the rescaling in the gradient term of the update rule, the optimal range for weight decay coefficients might change. These results are derived for priority size of p= 10, which denotes the number of hidden neurons that are removed at each reconfiguration of the model to a smaller sub-network. Greater values of p restrict the original NN model to be reconfigured to less number of sub-networks. Naturally, there is always a trade-off between the accuracy of the model and the number of sub-networks as shown in Figure 5.7. Considering this trade-off, the user might select an optimal priority size based on the design requirements for the target application. The error values in this figure are collected while reducing the size of the NN to 50% of its original size. A trade-off still remains between the number of sub-networks with acceptable error values and the percentage at which the size of the model is reduced. With respect to the application and design requirements, the user may select the appropriate value for the hyper parameter p.

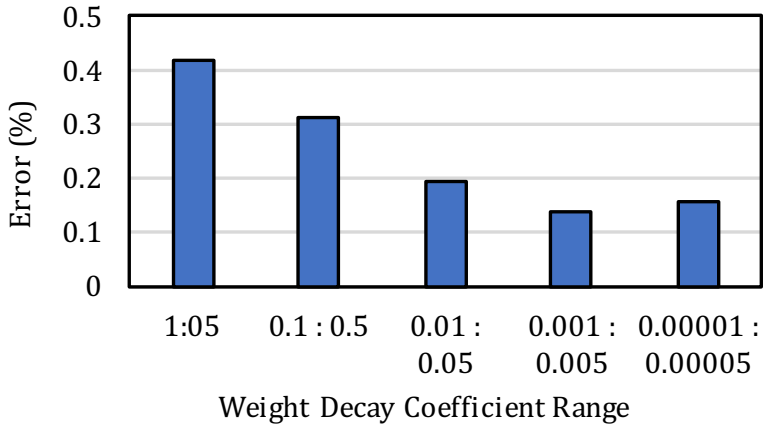


Figure 5.6: Performance of PNN for different ranges of weight decay coefficients.

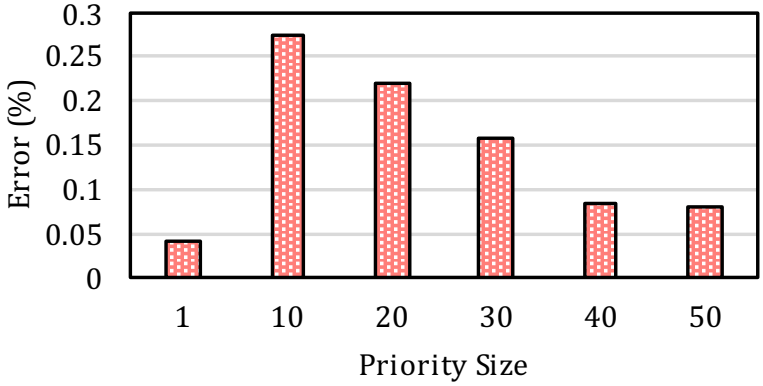


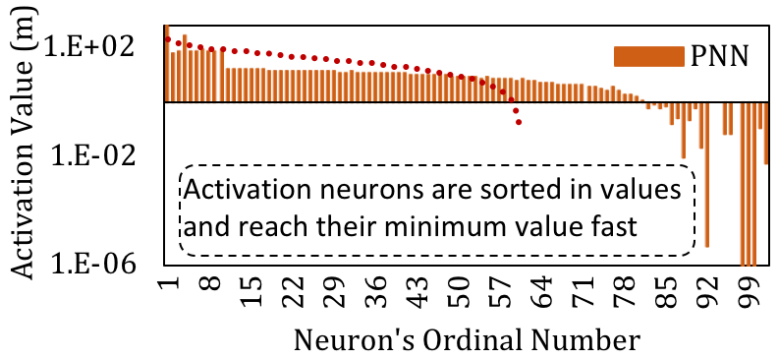
Figure 5.7: Performance of PNN for different priority sizes.

5.5.4 Comparison to State-of-the-Art Methodologies

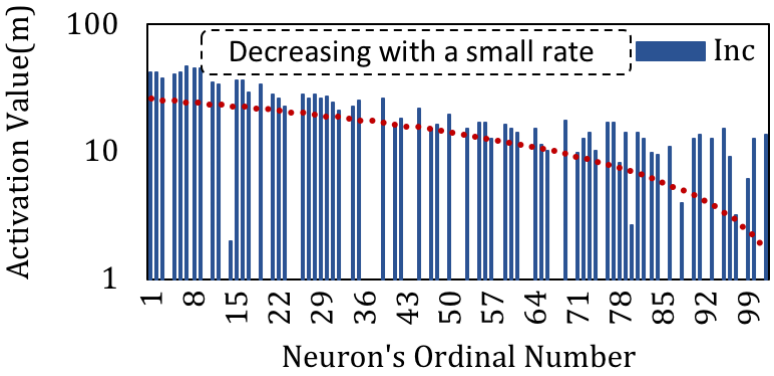
We evaluate the performance of our methodology in training a resource-aware NN model with two state-of-the-art approaches that are proposed as solutions to implement resource efficient NN in embedded system. By using the notation *resource-aware NN model*, we are implying that these NN models are targeted for systems that monitor the resource usage and dynamically manage the allocated resources to the NN model with respect to runtime constraints. The results are collected for a three-layer fully-connected neural network of 108×102 and 102×102 inputs to its hidden and output layers, respectively. The *Big/Little* approach [74], suggests multiple implementations of a NN model with small to bigger sizes. In the *Incremental* method [82], which is the most similar to ours, the NN is trained based on an iteratively incremental training algorithm where the weights computed in the earlier steps are fixed. The *Big/Little* approach would require separate memory storage to hold

model parameters of different sizes. Moreover, a retraining process is mandatory to generate multiple sizes for the NN model. The *Inc* method is more memory efficient such that only one set of model parameters are stored to implement a NN model that can be reconfigured into sub-networks with different sizes. However, this approach suffers from the retraining overhead per increment of size. In today's embedded systems, where runtime continuous learning of neural networks is required, retraining process overhead is prohibitive [87]. Our proposed *PNN* model is memory efficient such that only one set of weights are computed for multiple sub-networks. Furthermore, we compute the model parameters for PNN in a single-training process. Throughout the examples, we use the following abbreviation to indicate the three models. PNN: priority-based, Inc: Incremental, and BL: Big/Little.

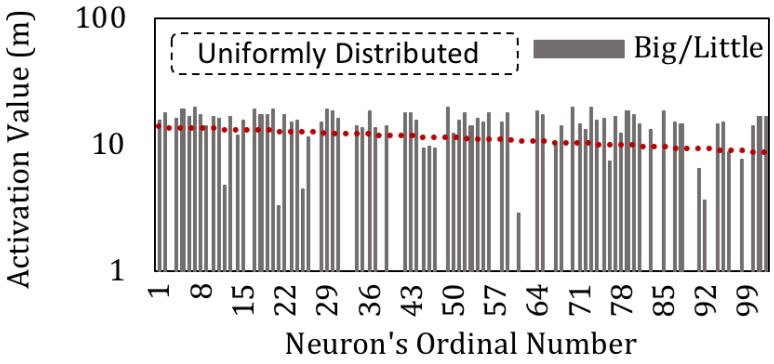
Emerging research is based on developing approaches to estimate the number of neurons and hidden layers required for a neural network [51]. However, these approximations also depend on the type of the database samples for which the network is designed. Therefore, It is still challenging to determine a good network topology for different applications. Therefore, exhaustive pruning and model reduction methodologies are in demand to reduce the over-sized NN models. One advantage of our proposed priority-based training algorithm is that it enforces a relatively sorted distribution to the activation values. We compare the activation value of hidden neurons for our proposed *PNN* model with respect to the *incrementally* trained model and the *Big/Little* model that is trained with no constraint on its weight parameters in Figure 5.8. For fairness of comparison, all experiments are conducted with the same size for all three models. The ordinal number of the neuron denotes the position of the respective neuron in the layer. The dotted red line shows the trend for linear changes in activation values with respect to ordinal number of the neuron. As shown in Figure 5.8(a), the activation values for the hidden neurons in PNN with priority size $p = 10$ is following a sorted order. The trend line shows that the density of the model is mostly populated throughout the first neurons and the activation values for the neurons further in the end of the layer are forced to be very small. This is as opposed to the two other methods that



(a) Activation values for neurons in PNN.



(b) Activation values for neurons in Inc.



(c) Activation values for neurons in Big/Little.

Figure 5.8: Comparing activation values of neurons with respect to their ordinal number.

show a more uniform distributions of activation values for the neurons. The incremental approach in Figure 5.8(b) also shows slight sorted order among activation values. However, as represented by the trend line, the rate of change for neuron's activation value with respect to its ordinal number is very slow compared to PNN method. In other words, in incremental approach, the weight parameters are adjusted more uniformly throughout the network. This decreases the number of sub-networks and the number of hidden neurons that can be pruned

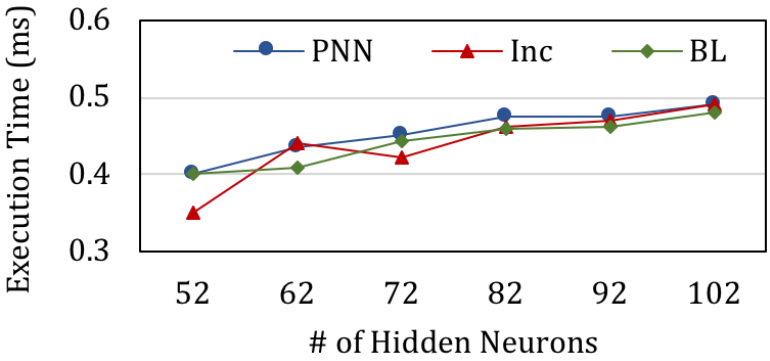
from the model without major drop in accuracy.

Table 5.1: Comparing the training process

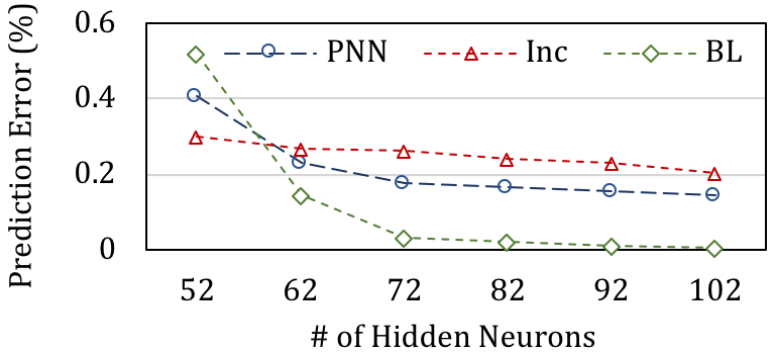
Model	# of Sub-Networks	Retrain	# of Retrain	Train Time (s)
PNN	6	No	0	2627
Inc	6	Yes	6	21534
Big/Little	6	Yes	6	25020

Table 5.1 compares the training process for a three layer fully-connected FFNN using the three aforementioned methods. The data is collected to train 6 separate sub-networks of various sizes using the three methods. As we can see in the table, our proposed method can generate 6 separate sub-networks in single training process. This is as opposed to the two other methods that require retraining for each of the sub-networks. The performance of these 6 sub-networks is evaluated in Figures 10.a and 10.b where the x-axis represents the number of hidden neurons at each sub-network. The retraining process imposes additional computation complexity to re-tune the parameters and hyper parameters. We can see that our proposed model reduces the computation overhead for the training process substantially. The training time is a critical matter especially in embedded systems for CPS applications where many NN models are trained on the fly.

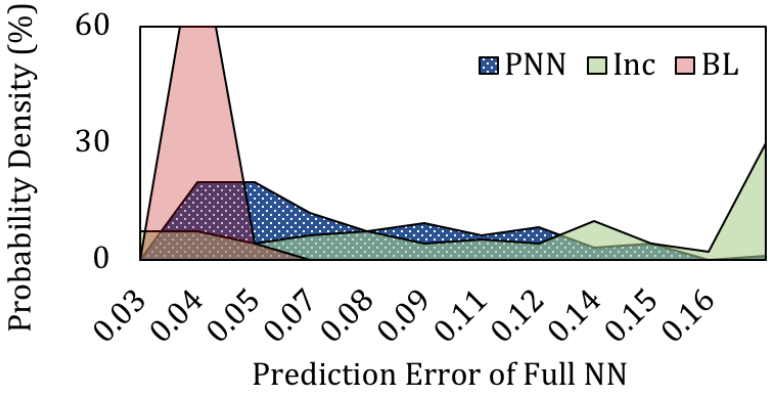
In Figure 5.9(a), we show the prediction time values over 6 different sub-network sizes. The results show similar performance for all three approaches in terms of runtime prediction overhead which increases for larger network size. As shown in the figure, by reducing the number of hidden neurons to half of its original size, we can improve the computation overhead by 30%. However, this saving in computation time comes as a trade-off for model accuracy. Figure 5.9(b) shows the percentage prediction error values for different sub-network sizes. The results for the BL [74] method that trains the sub-networks separately with no additional constraints show that after a certain point the model error does not change with



(a) Execution time.



(b) Prediction error.



(c) Probability distribution of prediction error for full-size NN.

Figure 5.9: Performance comparison of three resource-aware approaches. In Figure 5.9(c) we show the probability distribution of error for test data.

growth in the NN size. This justifies the over-parameterization phenomena in training the neural network that urges pruning and model reduction methodologies. Moreover, the mean of prediction error for 6 different sub-networks using our proposed PNN method and Inc [82] are 0.2% and 0.25% respectively. Therefore, our proposed PNN method outperforms the Inc approach for better prediction performance with no additional retraining process needed.

In order to evaluate the comparability of model accuracy among the three methods, we also show the probability distribution of prediction error values in Figure 5.9(c). These results are collected for a full-size NN with no model reduction process performed. We can see in the figure that the low variation in prediction errors using our proposed PNN model, confirms its stable performance in prediction of various test data. Moreover, the average of prediction errors for the PNN model is very close to that of BL method. This experiment ensures that our proposed model is validated as a memory efficient architecture for NN models with small drop in accuracy and comparable performance can be acquired using all three methods.

Table 5.2: Comparing memory reduction with respect to error.

Model	# of Sub-Networks	# of Parameters	Memory Reduction	Mean Error (%)
PNN	6	21522	78%	0.2
Inc	6	21522	78%	0.25
Big/Little	6	87292	-	0.125

We compare the efficiency of the three resource-aware methods in terms of memory requirements and model reduction complexity in Table 5.2. The PNN and Inc methods are both memory efficient in that they need one set of weight parameters to store multiple sub-network sizes. This is as opposed to the BL method that requires separate memory to store each sub-network. Therefore, we can achieve 78% saving in memory to store 6 sub-networks with very small loss in accuracy.

To summarize, our proposed PNN model outperforms the BL method with 89% reduction in training time and 78% saving in memory storage. Moreover, the computation complexity of the model reduction process to search for n neurons below the pruning threshold is improved from O(n) to O(logn). The PNN model shows similar results to Inc method in terms of memory and model reduction complexity. However, we show that PNN follows a single training process to adjust weight parameters as opposed to Inc method that is based on

multiple retraining. Therefore, The PNN model can cut down the training time by 86% with respect to Inc method while maintaining a better prediction performance from 0.25% to 0.21%.

5.6 Conclusion

In this chapter, we proposed Priority Neuron Network (PNN), a resource-aware neural network model with a reconfigurable architecture. We proposed a training algorithm to exploit regularization constraints on each neuron based on their ordinal number at a given layer. This enforces a sorted order distribution for the activation value of the neurons. We implemented our model for a three-layer fully-connected NN architecture to be employed as the predictive model of a vehicle in MPC for path tracking application. To corroborate the effectiveness of our proposed methodology, we compared it with two state-of-the-art methods for resource-aware NN design. We showed that compared to current state-of-the-art, our approach achieves 75% reduction in memory usage and 87% less training time with no significant drop in accuracy. Moreover, we improve the computational complexity of the model reduction process in order to prune n number of neurons, from $O(n)$ to $O(logn)$.

Chapter 6

Concluding Notes and Future Directions

Advanced control methodologies have helped the development of modern vehicles that are capable of path planning and path following. For instance, Model Predictive Control (MPC) employs a predictive model to predict the behavior of the physical system for a specific time horizon in the future. However, these prediction routines are computationally intensive and the computational overhead grows with the complexity of the model. In general, mathematical descriptions in the form of Ordinary Differential Equations (ODE) are used to mimic the linear/nonlinear behavior of the physical system. However, complex models of physical systems may be composed of thousands of non-linear ODEs, requiring considerable computing power to execute. In general, the main burden in managing the computational complexity of nonlinear MPC applications is the concurrent solving of a large number of nonlinear ordinary differential equations. Switching MPC addresses this issue by combining multiple predictive models, each with a different precision granularity. In switching predictive control schemes, the controller switches between predictive models of different granularities. However, having to store multiple predictive models of the physical system with various granularity levels

introduce memory overhead. This is very crucial specially in embedded systems such as autonomous vehicles where resource is limited. The dynamics of embedded systems is constantly changing, so does the available resources for our computations, therefore the system should be able to adapt to these changes. To address these issues, we propose resource-aware solutions as predictive models and switching algorithms to dynamically manage the granularity level of predictive models and trade performance metrics at runtime.

6.1 Main Contributions

The novel contributions of this thesis are:

- **HES Machine: Harmonic Equivalent State Machine Model Generation Tool for Cyber-Physical Systems:** The main contribution of the proposed modeling framework is the inclusion of frequency domain properties in signal synthesis to adopt the enable multiple granularities and adjust the overall model accuracy at runtime.
- **Hybrid State Machine Model for Fast Model Predictive Control: Application to Path Tracking:** In this work we integrate state machines with machine learning neural networks to develop a predictive model which can adapt to variations in inputs at runtime. ODE models are employed to train the proposed model at design-time.
- **Switching Predictive Control Using Reconfigurable State-Based Model:** A switching algorithm is proposed to change the configuration of the HES Machine as a multi-grained predictive model at run-time. We adopt machine Learning models to determine the optimal granularity level of the current predictive model in use based on the current state and operating region.

- **Priority Neuron: A Resource-Aware Neural Network for Cyber-Physical Systems**: This model can be reconfigured to various network sizes at runtime while storing only one set of weight parameters for memory efficiency. We proposed a training algorithm that adjusts the priority of each neuron in the computation of the models output. We assign the priority of each neuron using regularization techniques.

6.2 Future Research

- In order to further improve the accuracy of the NN in the HES model, we can increase the number of training data. Moreover, a more complex ODE model can be adopted during the simulation process to collect the training data.

- in Chapter 5 we computed the weight decay coefficients empirically based on the trade-off between the model accuracy and the number of hidden neurons deleted per reconfiguration of the model-priority size. This process of selecting optimal ranges for the weight decay coefficients can be automated.

- In this thesis we exploited fully-connected feed forward neural networks architecture. The proposed methods can be expanded to other NN architectures. For instance, Recurrent Neural Networks (RNN) can be employed as a predictive model to estimate the behavior of the physical system. RNN is a popular architecture for time series forecasting and is distinguished from feed forward neural networks by having signals traveling in both directions and introducing loops in the network. Furthermore, a stable adaptive controller [55] may be exploited to reject controllers leading to instability .

Bibliography

[1] Daniel Auger. *Driving-Cycle*. `https://www.mathworks.com/matlabcentral/fileexchange/46777-driving-cycle--simulink-block-`.

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[3] A. Aminifar, P. Tabuada, P. Eles, and Z. Peng. Self-triggered controllers and hard real-time guarantees. In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, pages 636–641. EDA Consortium, 2016.

[4] M. Amir and T. Givargis. Hes machine: Harmonic equivalent state machine modeling for cyber-physical systems. In *2017 IEEE International High Level Design Validation and Test Workshop (HLDVT)*, pages 31–38. IEEE, 2017.

[5] M. Amir and T. Givargis. Hybrid state machine model for fast model predictive control: Application to path tracking. In *Proceedings of the 36th International Conference on Computer-Aided Design*, pages 185–192. IEEE Press, 2017.

[6] M. Amir and T. Givargis. Hybrid State Machine Model for Fast Model Predictive Control: Application to Path Tracking. *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017.

[7] M. Amir and T. Givargis. Priority neuron: A resource-aware neural network for cyber-physical systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[8] U. M. Ascher and L. R. Petzold. *Computer methods for ordinary differential equations and differential-algebraic equations*, volume 61. Siam, 1998.

[9] T. A. Badgwell and K. R. Muske. Disturbance model design for linear model predictive control. In *American Control Conference, 2002. Proceedings of the 2002*, volume 2, pages 1621–1626. IEEE, 2002.

[10] B. Barbot, M. Kwiatkowska, A. Mereacre, and N. Paoletti. Estimation and verification of heart models for personalised medical and wearable devices. In *International Conference on Computational Methods in Systems Biology*, pages 3–7. Springer, 2015.

[11] J. S. Barlow. Data-based predictive control with multirate prediction step. In *American Control Conference (ACC), 2010*, pages 5513–5519. IEEE, 2010.

[12] T. Barlow, S. Latham, I. McCrae, and P. Boulter. A reference book of driving cycles for use in the measurement of road vehicle emissions. *TRL Published Project Report*, 2009.

[13] S. C. Bengea and R. A. DeCarlo. Optimal control of switching systems. *automatica*, 41(1):11–27, 2005.

[14] F. M. Bianchi, E. Maiorino, M. C. Kampffmeyer, A. Rizzi, and R. Jenssen. An overview and comparative analysis of recurrent neural networks for short term load forecasting. *arXiv preprint arXiv:1705.04378*, 2017.

[15] L. Breger, J. How, and A. Richards. Model predictive control of spacecraft formations with sensing noise. In *American Control Conference, 2005. Proceedings of the 2005*, pages 2385–2390. IEEE, 2005.

[16] P. J. Brockwell and R. A. Davis. *Introduction to time series and forecasting*. springer, 2016.

[17] E. F. Camacho and C. B. Alba. *Model predictive control*. Springer Science & Business Media, 2013.

[18] H. Chen, L. Ning, and L. Shaoyuan. Switching multi-model predictive control for hypersonic vehicle. In *Control Conference (ASCC), 2011 8th Asian*, pages 677–681. IEEE, 2011.

[19] S. Chen and S. Billings. Neural networks for nonlinear dynamic system modelling and identification. *International journal of control*, 56(2):319–346, 1992.

[20] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015.

[21] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

[22] X. Dong, S. Chen, and S. Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4860–4874, 2017.

[23] G. Droge and M. Egerstedt. Adaptive time horizon optimization in model predictive control. In *American Control Conference (ACC), 2011*, pages 1843–1848. IEEE, 2011.

[24] S. M. Erlien, S. Fujita, and J. C. Gerdes. Safe driving envelopes for shared control of ground vehicles. *IFAC Proceedings Volumes*, 46(21):831–836, 2013.

[25] P. Falcone, M. Tufo, F. Borrelli, J. Asgari, and H. E. Tseng. A linear time varying model predictive control approach to the integrated vehicle dynamics control problem in autonomous systems. In *Decision and Control, 2007 46th IEEE Conference on*, pages 2980–2985. IEEE, 2007.

[26] J. A. Ferreira, J. E. De Oliveira, and V. A. Costa. Modeling of hydraulic systems for hardware-in-the-loop simulation: A methodology proposal. In *Proc. Int. Mechanical Eng. Congress & Exposition, Nashville, USA*, 1999.

[27] J. Fojdl and R. W. Brause. The performance of approximating ordinary differential equations by neural nets. In *Tools with Artificial Intelligence, 2008. ICTAI'08. 20th IEEE International Conference on*, volume 2, pages 457–464. IEEE, 2008.

[28] A. G. Galeano and F. J. Vargas. Automatic code generation for hardware-in-the-loop simulation of differential equations using model-driven engineering.

[29] Y. Gao, T. Lin, F. Borrelli, E. Tseng, and D. Hrovat. Predictive control of autonomous ground vehicles with obstacle avoidance on slippery roads. In *ASME 2010 dynamic systems and control conference*, volume 1, pages 265–272. American Society of Mechanical Engineers, 2010.

[30] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, 2000 (June 13). Circulation Electronic Pages: http://circ.ahajournals.org/content/101/23/e215.full PMID:1085218; doi: 10.1161/01.CIR.101.23.e215.

[31] A. N. Gorban and D. Roose. *Coping with Complexity: Model Reduction and Data Analysis*, volume 75. Springer Science & Business Media, 2010.

[32] U. Halldorsson, M. Fikar, and H. Unbehauen. Nonlinear predictive control with multirate optimisation step lengths. *IEE Proceedings-Control Theory and Applications*, 152(3):273–284, 2005.

[33] C. Hamzaçebi, D. Akay, and F. Kutay. Comparison of direct and iterative artificial neural network forecast approaches in multi-periodic time series forecasting. *Expert Systems with Applications*, 36(2):3839–3844, 2009.

[34] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. Eie: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 243–254. IEEE, 2016.

[35] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[36] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[37] H. O. Hartley. The modified gauss-newton method for the fitting of non-linear regression functions by least squares. *Technometrics*, 3(2):269–280, 1961.

[38] W. He, Y. Chen, and Z. Yin. Adaptive neural network control of an uncertain robot with full-state constraints. *IEEE transactions on cybernetics*, 46(3):620–629, 2016.

[39] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV)*, volume 2, 2017.

[40] J. Heaton. *Introduction to neural networks with Java*. Heaton Research, Inc., 2008.

[41] N.-B. Hoang and H.-J. Kang. Neural network-based adaptive tracking control of mobile robots in the presence of wheel slip and external disturbance force. *Neurocomputing*, 188:12–22, 2016.

[42] B. Houska, H. Ferreau, and M. Diehl. ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.

[43] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[44] W. Hu, Y. Qian, F. K. Soong, and Y. Wang. Improved mispronunciation detection with deep neural network trained acoustic models and transfer learning based logistic regression classifiers. *Speech Communication*, 67:154–166, 2015.

[45] C. Huang, F. Vahid, and T. Givargis. A custom fpga processor for physical model ordinary differential equation solving. *IEEE Embedded Systems Letters*, 3(4):113–116, 2011.

[46] M. Ishii and A. Sato. Layer-wise weight decay for deep neural networks. In *Pacific-Rim Symposium on Image and Video Technology*, pages 276–289. Springer, 2017.

[47] A. Jadbabaie and J. Hauser. On the stability of receding horizon control with a general terminal cost. *IEEE Transactions on Automatic Control*, 50(5):674–678, 2005.

[48] Z. Jiang, S. Radhakrishnan, V. Sampath, S. Sarode, and R. Mangharam. Heart-on-a-chip: a closed-loop testing platform for implantable pacemakers. 2014.

[49] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. Atl: A model transformation tool. *Science of computer programming*, 72(1):31–39, 2008.

[50] E. Kamburjan, S. Mitsch, M. Kettenbach, and R. Hähnle. Modeling and verifying cyber-physical systems with hybrid active objects. *arXiv preprint arXiv:1906.05704*, 2019.

[51] S. Karsoliya. Approximating number of hidden layer neurons in multiple hidden layer bpnn architecture. *International Journal of Engineering Trends and Technology*, 3(6):714–717.

[52] C. T. Kelley. *Iterative methods for optimization*. SIAM, 1999.

[53] C. T. Kelley. *Solving nonlinear equations with Newton's method*. SIAM, 2003.

[54] S. M. Khansari-Zadeh and A. Billard. Learning control lyapunov function to ensure stability of dynamical system-based robot reaching motions. *Robotics and Autonomous Systems*, 62(6):752–765, 2014.

[55] H. Kim and A. Y. Ng. Stable adaptive control with online learning. In *Advances in Neural Information Processing Systems*, pages 977–984, 2005.

[56] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[57] P. Krauthausen and U. D. Hanebeck. A model-predictive switching approach to efficient intention recognition. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 4908–4913. IEEE, 2010.

[58] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

[59] E. A. Lee. Cyber physical systems: Design challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 363–369. IEEE, 2008.

[60] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

[61] X. Lin, P. Bogdan, N. Chang, and M. Pedram. Machine learning-based energy management in a hybrid electric vehicle to minimize total operating cost. In *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on*, pages 627–634. IEEE, 2015.

[62] Y. Louzoun, S. Solomon, H. Atlan, and I. R. Cohen. Modeling complexity in biology. *Physica A: Statistical Mechanics and its Applications*, 297(1):242–252, 2001.

[63] R. Mahadevan and F. J Doyle III. Efficient optimization approaches to nonlinear model predictive control. *International Journal of Robust and Nonlinear Control*, 13(3-4):309–329, 2003.

[64] C. N. Maxoulis, D. N. Tsinoglou, and G. C. Koltsakis. Modeling of automotive fuel cell operation in driving cycles. *Energy conversion and management*, 45(4):559–573, 2004.

[65] P. E. McSharry, G. D. Clifford, L. Tarassenko, and L. A. Smith. A dynamical model for generating synthetic electrocardiogram signals. *IEEE Transactions on Biomedical Engineering*, 50(3):289–294, 2003.

[66] A. Miller. Model predictive control of the ships motion in presence of wind disturbances. *Zeszyty Naukowe/Akademia Morska w Szczecinie*, 2014.

[67] H. Mirzaei and T. Givargis. Fine-grained acceleration control for autonomous intersection management using deep reinforcement learning. *CoRR*, abs/1705.10432, 2017.

[68] H. Mirzaei and T. Givargis. Fine-grained acceleration control for autonomous intersection management using deep reinforcement learning. *arXiv preprint arXiv:1705.10432*, 2017.

[69] J. J. Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis*, pages 105–116. Springer, 1978.

[70] A. Mujika, F. Meier, and A. Steger. Fast-slow recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 5915–5924, 2017.

[71] K. R. Muske and T. A. Badgwell. Disturbance modeling for offset-free linear model predictive control. *Journal of Process Control*, 12(5):617–632, 2002.

[72] E. Narby. Modeling and estimation of dynamic tire properties, 2006.

[73] W. Pan, H. Dong, and Y. Guo. Dropneuron: Simplifying the structure of deep neural networks. *arXiv preprint arXiv:1606.07326*, 2016.

[74] E. Park, D. Kim, S. Kim, Y.-D. Kim, G. Kim, S. Yoon, and S. Yoo. Big/little deep neural network for ultra low power inference. In *Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis*, pages 124–132. IEEE Press, 2015.

[75] C. Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014.

[76] G. V. Raffo, G. K. Gomes, J. E. Normey-Rico, C. R. Kelber, and L. B. Becker. A predictive controller for autonomous vehicle path tracking. *IEEE transactions on intelligent transportation systems*, 10(1):92–102, 2009.

[77] A. Sassone, D. Shin, A. Bocca, A. Macii, E. Macii, and M. Poncino. Modeling of the charging behavior of li-ion batteries based on manufacturer's data. In *Proceedings of the 24th edition of the great lakes symposium on VLSI*, pages 39–44. ACM, 2014.

[78] L. F. Shampine, I. Gladwell, and S. Thompson. *Solving ODEs with matlab*. Cambridge University Press, 2003.

[79] S. W. Smith et al. The scientist and engineer's guide to digital signal processing. 1997.

[80] D. Solomatine, L. M. See, and R. Abrahart. Data-driven modelling: concepts, approaches and experiences. In *Practical hydroinformatics*, pages 17–30. Springer, 2009.

[81] Z. Tang and P. A. Fishwick. Feedforward neural nets as models for time series forecasting. *ORSA journal on computing*, 5(4):374–385, 1993.

[82] H. Tann, S. Hashemi, R. I. Bahar, and S. Reda. Runtime configurable deep neural networks for energy-accuracy trade-off. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2016 International Conference on*, pages 1–10. IEEE, 2016.

[83] G. Teschl. *Ordinary differential equations and dynamical systems*, volume 140. American Mathematical Society Providence, 2012.

[84] S. A. Teukolsky, B. P. Flannery, W. Press, and W. Vetterling. Numerical recipes in c. *SMR*, 693:1, 1992.

[85] C. Torres-Huitzil and B. Girau. Fault and error tolerance in neural networks: A review. *IEEE Access*, 5:17322–17341, 2017.

[86] K. Vatanparvar and M. A. Al Faruque. Eco-friendly automotive climate control and navigation system for electric vehicles. In *Cyber-Physical Systems (ICCPS), 2016 ACM/IEEE 7th International Conference on*, pages 1–10. IEEE, 2016.

[87] K. Vatanparvar and M. A. Al Faruque. Acqua: Adaptive and cooperative quality-aware control for automotive cyber-physical systems. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017.

[88] K. Vatanparvar and M. A. Al Faruque. ACQUA: Adaptive and Cooperative Quality-Aware Control for Automotive Cyber-Physical Systems. *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017.

[89] K. Vatanparvar, S. Faezi, I. Burago, M. Levorato, and M. A. Al Faruque. Extended range electric vehicle with driving behavior estimation in energy management. *IEEE Transactions on Smart Grid*, 2018.

[90] K. Vatanparvar, A. Faruque, and M. Abdullah. Battery lifetime-aware automotive climate control for electric vehicles. In *Proceedings of the 52nd Annual Design Automation Conference*, page 37. ACM, 2015.

[91] K. Vatanparvar, A. Faruque, and M. Abdullah. Otem: optimized thermal and energy management for hybrid electrical energy storage in electric vehicles. In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, pages 19–24. EDA Consortium, 2016.

[92] K. Vatanparvar and M. A. A. Faruque. Electric vehicle optimized charge and drive management. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 23(1):3, 2017.

[93] A. Waibel. Modular construction of time-delay neural networks for speech recognition. *Neural computation*, 1(1):39–46, 1989.

[94] U. Walter and V. Oberle. μ-second precision timer support for the linux kernel, 2001.

[95] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[96] F. Yakub, A. Abu, S. Sarip, and Y. Mori. Study of model predictive control for path-following autonomous ground vehicle control under crosswind effect. *Journal of Control Science and Engineering*, 2016, 2016.

[97] T.-J. Yang, Y.-H. Chen, and V. Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[98] M. A. Zakaria, H. Zamzuri, R. Mamat, S. A. Mazlan, M. A. A. Rahman, and A. H. A. Rahman. Dynamic curvature path tracking control for autonomous vehicle: Experimental results. In *Connected Vehicles and Expo (ICCVE), 2014 International Conference on*, pages 264–269. IEEE, 2014.

[99] M. Zanon, J. V. Frasch, M. Vukov, S. Sager, and M. Diehl. *Model Predictive Control of Autonomous Vehicles*, pages 41–57. Springer International Publishing, Cham, 2014.

[100] K. Zhang, J. Sprinkle, and R. G. Sanfelice. Computationally aware control of autonomous vehicles: a hybrid model predictive control approach. *Autonomous Robots*, 39(4):503–517, 2015.

[101] L. Zhang, Y. Zhu, P. Shi, and Q. Lu. *Time-dependent switched discrete-time linear systems: Control and filtering*. Springer, 2016.

[102] L. Zhang, S. Zhuang, and R. D. Braatz. Switched model predictive control of switched linear systems: Feasibility, stability and robustness. *Automatica*, 67:8–21, 2016.