Hybrid State Machine Model for Fast Model Predictive Control: Application to Path Tracking

Maral Amir, Tony Givargis
Donald Bren School of Information and Computer Sciences, CS Department
University of California, Irvine, California, USA
{mamir, givargis}@uci.edu

***Abstract*—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 which 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 paper, we introduce a model generation framework to transform ODE models of a physical system to Hybrid Harmonic Equivalent State (HES) Machine model equivalents. Moreover, tuning parameters are introduced to reconfigure the model and adjust its accuracy from coarse-grained time critical situations to fine-grained scenarios in which safety is paramount. Machine learning techniques are applied to adopt the model to run-time applications. We conduct experiments on a closed-loop MPC for path tracking using the vehicle dynamics model. We analyze the performance of the MPC when applying our Hybrid 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 a 32% reduction in MPC return time for 0.8% loss in model accuracy.**

***Index Terms*—CPS, Modeling, Simulation, Model-Based Design, Model Generation, State Machine, FFT**

I. 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 [1], HVAC control in electric vehicles [2, 3] and formation flying spacecraft [4]. 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 [5]. 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 [6]. 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 [7] 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 [8] 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 II.

II. 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 [9]. 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 [10]).

Complex ODE models introduce challenges in terms of scalability, performance, power consumption, and accuracy [11]. 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 [12]. The demand for more accurate and mathematically sound CPS solutions, cause an increase in resource utilization and energy consumption [13, 14]. 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 [15]. 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 [16]. A heart-on-a-chip model [17] 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 [7] 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.

Relative to existing literature, our contributions in this work can be summarized as follows:

1) Development of a generic model of a physical system is automated using software development and machine learning techniques. ODE models are employed to train the proposed model at design-time.
2) The model is implemented in state machine representation for fast performance in CPS applications. A global clock conducts the trigger to update the state variables and output actions for the state machine model.
3) The proposed model has a hybrid feature that enables the model to be adapted to coarse-grained time critical or fine-grained safety critical situations. This is because the model is formulated upon frequency harmonics information and outputs of the dynamic model are synthesized from Fast Fourier Transform (FFT) algorithm. Tuning parameters are proposed to reconfigure the model with respect to system requirements.

The rest of the paper is organized as follows. In Section III the architecture of the proposed methodology, tuning parameters, and M-PC formulation is described in details. We demonstrate the workings and effectiveness of our framework for path tracking application in Section IV. Finally, we state our conclusions in Section V.

III. 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:

$$z(t) = f(s, \mathbf{u}) \tag{1}$$

where $\mathbf{u}$ 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.

The State Machine Generator 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 III-B. The inputs to State Machine Generator block are vectors $\mathbf{Re^z}$ and $\mathbf{Im^z}$ 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 $\mathbf{Fr^z}$ 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 $\mathbf{u}$ and the harmonic information vectors $\mathbf{Re^z}$ and $\mathbf{Im^z}$.

A. Model Architecture

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

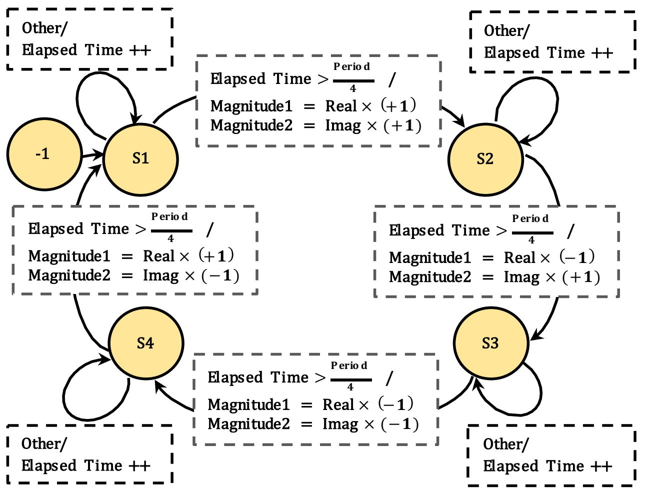


Fig. 1. 5-State synchronous state machine with the inverse of the signal's harmonic frequency as the period.

1) 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 $\mathbf{Re^z}$ and $\mathbf{Im^z}$ and generate concurrent state machines with respective frequency harmonics $\mathbf{Fr^z}$ as the update rates. The synthesis equation of FFT for signal $z[k]$ of size N is employed as presented in Equation 2. In this equation, k stands for the index of samples running from 0 to N-1. The vectors $\overline{\mathbf{Re^z}}[\mathbf{i}]$ and $\overline{\mathbf{Im^z}}[\mathbf{i}]$ 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 [18].

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

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. Figure 1 illustrates the structure of the model generated by the proposed HES framework. 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,

$$\text{StateMachine}{=}(\text{States, AuxVar, Outputs, Update, InitState})$$

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 $\mathbf{Re^z}$ and $\mathbf{Im^z}$ as output values and the harmonic frequencies vector $\mathbf{Fr^z}$ 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 [19] 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 III-B.

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 and optimization. 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 in the synthesis function. A band-pass filter is applied to attenuate the unwanted square wave frequencies. The quality factor of the band pass filter, Q, is considered as a framework parameter to be configured by the user during optimization. In future work, we plan to apply further measurements to compensate for filter error. $(N/2{+}1)$ sinusoidal signals are integrated to synthesize the decomposed signal according to Equation 2. The tool generates an executable C code in state machine representation for the physical signal to be implemented on a target platform.

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

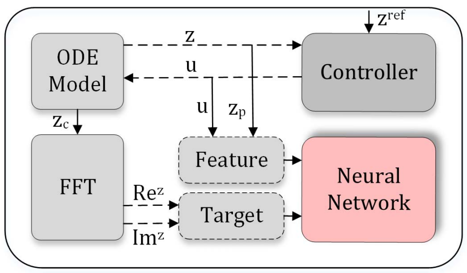
Algorithm 1: Global Tick Function

```
Input: index of the state machine $i$

global variable $HES$
global variable $magnitude1, magnitude2$
const TimeResolution

switch $HES[i].state$ do
    case $-1$ do
        $HES[i].state = \mathrm{S1}$
    case S1 do
        if $HES[i].elapsedTime \geq HES[i].period/4$ then
            $HES[i].state = \mathrm{S2}$
            $HES[i].elapsedTime = 0$
        else
            $HES[i].state = \mathrm{S1}$
    case S2 do
        if $HES[i].elapsedTime \geq HES[i].period/4$ then
            $HES[i].state = \mathrm{S3}$
            $HES[i].elapsedTime = 0$
        else
            $HES[i].state = \mathrm{S2}$
    case S3 do
        if $HES[i].elapsedTime \geq HES[i].period/4$ then
            $HES[i].state = \mathrm{S4}$
            $HES[i].elapsedTime = 0$
        else
            $HES[i].state = \mathrm{S3}$
    case S4 do
        if $HES[i].elapsedTime \geq HES[i].period/4$ then
            $HES[i].state = \mathrm{S1}$
            $HES[i].elapsedTime = 0$
        else
            $HES[i].state = \mathrm{S4}$
    otherwise do
        $HES[i].state = -1$

switch $HES[i].state$ do
    case S1 do
        $magnitude1[i][HES[i].N1] = HES[i].real \times 1.0$
        $magnitude2[i][HES[i].N1] = HES[i].imag \times 1.0$
    case S2 do
        $magnitude1[i][HES[i].N1] = HES[i].real \times -1.0$
        $magnitude2[i][HES[i].N1] = HES[i].imag \times 1.0$
    case S3 do
        $magnitude1[i][HES[i].N1] = HES[i].real \times -1.0$
        $magnitude2[i][HES[i].N1] = HES[i].imag \times -1.0$
    case S4 do
        $magnitude1[i][HES[i].N1] = HES[i].real \times 1.0$
        $magnitude2[i][HES[i].N1] = HES[i].imag \times -1.0$
    otherwise do
        $magnitude1[i][HES[i].N1] = HES[i].real \times 1.0$
        $magnitude2[i][HES[i].N1] = HES[i].imag \times 1.0$

$HES[i].elapsedTime += \mathrm{TimeResolution}$
$HES[i].N1 ++$

return
```

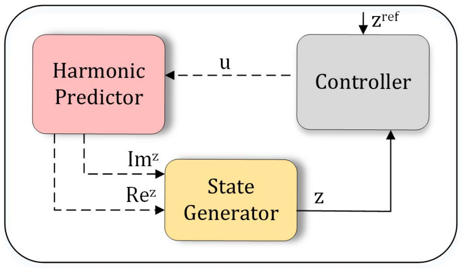
apply a machine learning technique to fit a predictive model that maps the control input vector $\mathbf{u}$ of size N to respective harmonic information vectors $\mathbf{Re^z}$ and $\mathbf{Im^z}$ of size $(N/2{+}1)$. We interpret our input and output vectors for the predictive model as time series data to leverage time series prediction approaches [20].

Neural Networks (NN) have solved time series prediction [21] and hold promising performance to learn linear and non-linear models without prior knowledge of the relation between input and output variables[22, 23]. 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 wights 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 [18]. 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 [24].

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 2.



(a) Training.



(b) Prediction.

Fig. 2. Training and Prediction phases of the Harmonic Predictor block in MPC.

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 2(a), simulation is conducted on the ODE model of the target physical system to record the control inputs $\mathbf{u}$ 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 [18]. The vector data of control inputs $\mathbf{u}$ and the output value from previous time step z_p are considered as the input features and the respective output signals $\mathbf{Re^z}$ and $\mathbf{Im^z}$ 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 $\mathbf{u}$. The predicted harmonic information is fed into the State Machine Generator block for output generation as shown in Figure 2(b). That is, the output of the proposed physical model z can adapt to variations in control inputs u in run-time.

B. 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 (*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 (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.

C. 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[25]:

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[26]. 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 [26] is the solution to the following optimization problem at each time instant

$$\begin{aligned}
\min_{z,u}.\quad & \|z(T_p) - z^r(T_p)\|^2_{P_c} \\
& + \sum_{t=0}^{T_p} \|z(t) - z^r(t)\|^2_{Q_c} + \|u(t) - u^r(t)\|^2_{R_c} && \text{(3a)} \\
s.t.\quad & \\
& z(t) = f(s(t), u(t)), && \text{(3b)} \\
& s(0) = \hat{s}(0), && \text{(3c)} \\
& q(z(t), s(t), u(t)) \geq 0 \qquad t \in [0, T_p] && \text{(3d)}
\end{aligned}$$

Equation (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 (3b), where $z(t)$, $s(t)$ and $u(t)$ represent outputs, state variables and control inputs respectively. Equation (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 (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 [5]. 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 $\mathbf{u}$. 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.

IV. Experimental Results

A. 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 I
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 [27] 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 $\mathbf{u}$ and frequency harmonic components $\mathbf{Re^z}$ and $\mathbf{Im^z}$ respectively.

The experiments are conducted for prediction horizon of size T_p which determines the size for vectors $\mathbf{u}$, $\mathbf{Re^z}$ and $\mathbf{Im^z}$ 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 $\mathbf{Re^z}$ vector of size (T_p/2)+1 followed by the same size vector $\mathbf{Im^z}$. 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 I illustrates the configurations for the neural network and corresponding train and validation error.

For our control application, we adopt the software framework based on the ACADO Toolkit [28]. 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 [29] and BDF [30] 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.

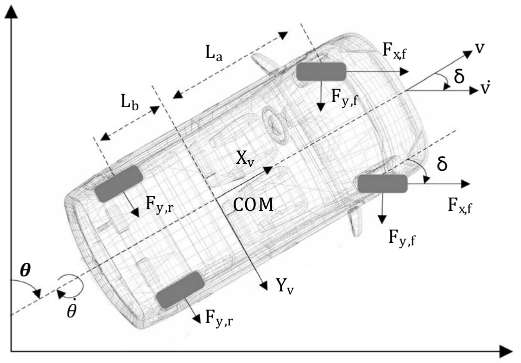


Fig. 3. Schematic view of the vehicle model.

B. 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 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.

$$RMSE = \sqrt{\frac{(\sum(Expected - Evaluated)^2)}{N}} \tag{4}$$

C. 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 [7] shown in Figure 3 is given in equation form as:

$$\dot{x} = v\sin(\theta) \tag{5a}$$

$$\dot{y} = v\cos(\theta) \tag{5b}$$

$$\dot{v} = \cos(\delta)a - \frac{2}{m}F_{y,f}\sin(\delta) \tag{5c}$$

$$\dot{\theta} = \phi \tag{5d}$$

$$\dot{\phi} = \frac{1}{J}(L_a(ma\sin(\delta) + 2F_{y,f}\cos(\delta)) - 2L_bF_{y,r}) \tag{5e}$$

$$\dot{\delta} = \omega \tag{5f}$$

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:

$$F_{y,f} = C_y(\delta - \frac{L_a\phi}{v}) \tag{6}$$

$$F_{y,r} = C_y(\frac{L_b\phi}{v}) \tag{7}$$

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:

$$\min_{x,y} \sum_{t=0}^{T_p} \|\hat{x}(k+1|k) - x^r(k+1|k)\|^2_{Q_c} \tag{8a}$$

$$+ \|\hat{y}(k+1|k) - y^r(k+1|k)\|^2_{Q_c} \tag{8b}$$

$$s.t.$$

$$-0.25 \leq \delta \leq 0.25 \tag{8c}$$

$$-1.25 \leq \omega \leq 1.25 \tag{8d}$$

$$-30 \leq a \leq 30 \tag{8e}$$

The performance of the proposed HES model of the vehicle dynamics in run-time MPC for path tracking application is compared with the model in Equation 5a. The performance of two models in tracking a static path for a certain time horizon is illustrated in Figure 4. 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.

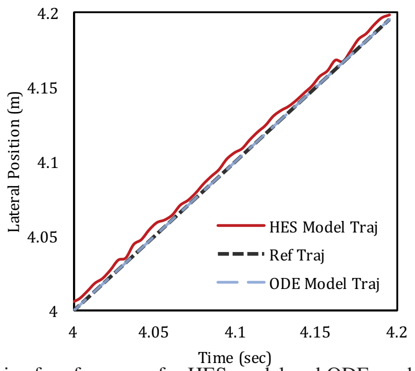


Fig. 4. Analysis of performance for HES model and ODE model in trajectory tracking application.

We compare the error values for ODE model and HES machine for different prediction horizon time steps in Table II. The HES Machine is configured with respect to framework 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. In our future work, we plan to evaluate Machine Learning techniques to consider the step size in the prediction model.

TABLE II
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 III
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

The models are analyzed in terms of execution time over different prediction horizon time steps. The results in Table III 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 5 with respect different number of steps. The results in Figure 5 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.

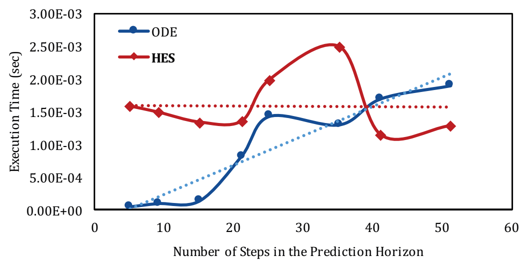


Fig. 5. Comparison and analysis of execution time for ODE model and HES model.

Figures 6 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.

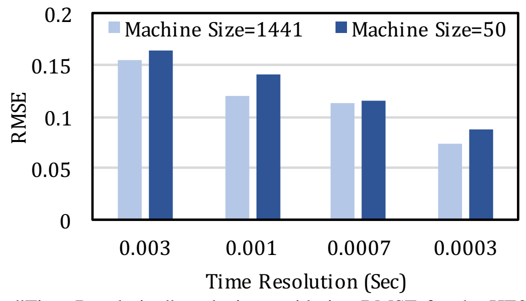


Fig. 6. "Time Resolution" analysis considering RMSE for the HES Model

V. 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.

Acknowledgment

This work was supported in part by the National Science Foundation under NSF grant number 1563652.

References

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

[2] Korosh Vatanparvar and Mohammad Abdullah Al Faruque. Battery lifetime-aware automotive climate control for electric vehicles. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2015.

[3] Korosh Vatanparvar, Al Faruque, and Mohammad 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.

[4] Louis Breger, Jonathan How, and Arthur Richards. Model predictive control of spacecraft formations with sensing noise. In *American Control Conference, 2005. Proceedings of the 2005*, pages 2385–2390. IEEE, 2005.

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

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

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

[8] Erik Narby. Modeling and estimation of dynamic tire properties, 2006.

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

[10] TJ Barlow, S Latham, IS McCrae, and PG Boulter. A reference book of driving cycles for use in the measurement of road vehicle emissions. *TRL Published Project Report*, 2009.

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

[12] Lawrence F Shampine, Ian Gladwell, and Skip Thompson. *Solving ODEs with matlab*. Cambridge University Press, 2003.

[13] Edward 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.

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

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

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

[17] Zhihao Jiang, Sriram Radhakrishnan, Varun Sampath, Shilpa Sarode, and Rahul Mangharam. Heart-on-a-chip: a closed-loop testing platform for implantable pacemakers. 2014.

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

[19] George F Turnpaugh. Programmable interval timer with lock means, November 10 1987. US Patent 4,705,970.

[20] Peter J Brockwell and Richard A Davis. *Introduction to time series and forecasting*. springer, 2016.

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

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

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

[24] Coşkun Hamzaçebi, Diyar Akay, and Fevzi 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.

[25] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

[26] Mario Zanon, Janick V. Frasch, Milan Vukov, Sebastian Sager, and Moritz Diehl. *Model Predictive Control of Autonomous Vehicles*, pages 41–57. Springer International Publishing, Cham, 2014.

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

[28] B. Houska, H.J. 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.

[29] Saul A Teukolsky, Brian P Flannery, WH Press, and WT Vetterling. Numerical recipes in c. *SMR*, 693:1, 1992.

[30] Uri M Ascher and Linda R Petzold. *Computer methods for ordinary differential equations and differential-algebraic equations*, volume 61. Siam, 1998.