Synthesis and real-time implementation of parameterized NMPC schemes for automotive semi-active suspension systems

Karthik Murali Madhavan Rathai

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTE UNIVERSITE GRENOBLE ALPES

Spécialité : **Automatique-Productique**

Arrêté ministériel : 7 août 2006

Présentée par
Karthik Murali Madhavan Rathai

Thèse dirigée par **Olivier SENAME** et
codirigée par **Mazen Alamir**

préparée au sein du **GIPSA-Lab**
dans **Electronique, Electrotechnique, Automatique, Traitement du Signal (EEATS)**

Synthesis and real-time implementation of parameterized NMPC schemes for automotive semi-active suspension systems

Thèse soutenue publiquement le **04 November 2020**,
devant le jury composé de:

M. Didier GEORGES
Professeur, Grenoble-INP, Président
M. Sorin OLARU
Professeur, CentraleSupelec, Rapporteur
M. Michel BASSET
Professor, Université de Haute-Alsace, Rapporteur
M. Jonathan Brembeck
Head of the Vehicle System Dynamics department, German Aerospace Center (DLR), Examinateur
M. Olivier SENAME
Professeur, Grenoble INP, Directeur de thèse
M. Mazen Alamir
Directeur de recherche, CNRS Grenoble, Co-Directeur de thèse

Contents

List of Figures

List of Tables

Table of Acronyms and Notations

MPC	Model Predictive Control
OCP	Optimal Control Problem
pNMPC	Parameterized Nonlinear Model Predictive Control
SS-pNMPC	Scenario Stochastic Parameterized Nonlinear Model Predictive Control
GPU	Graphic Processing Unit
EMPHYSIS	Embedded systems with physical models in the production code software
HiL	Hardware in the Loop
GPGPU	General Purpose computing on Graphic Processing Unit
FMI	Functional Mockup Interface
eFMI	Embedded Functional Mockup Interface
COG	Centre of Gravity
DAQ	Data Acquistion system
PCVC	Probabilistic Constraint Violation Certificate
IP	Intellectual Property
BBO	Black Box Optimization
3AC	Three Address Code
LTI	Linear Time Invariant

$\mathbb{R}$	Real values set
$\mathbb{C}$	Complex values set
A^*	Conjugate of $A \in \mathbb{C}$
A^T	Transpose of $A \in \mathbb{R}$
$A \prec (\preceq) 0$	Matrix A is symmetric and negative (semi)definite
$A \succ (\succeq) 0$	Matrix A is symmetric and positive (semi)definite
$Co(X)$	Convex hull of set X
$A = A^T$	Matrix A is real symmetric
$He(A) = A^T + A$	

Thesis framework and contribution

Contents

0.1 Thesis framework

This thesis is a compilation of the results obtained over three years of doctoral work (beginning from September 2017 to November 2020) on **Embedded modeling and control of vehicle dynamics: application to a small car pilot plant with ER dampers** conducted in the team SAFE (Safe, Controlled and Monitored Systems), formerly known as SLR (Systèmes Linéaires et Robustesse), Automatic department, GIPSA-lab, Grenoble. The thesis was conducted under the guidance of Prof. Mr. **Olivier SENAME** (Director of the thesis, Professor Grenoble INP) and Prof. Mr. **Mazen ALAMIR** (Co-director of the thesis, Research Director CNRS). This work was supported and funded by the ITEA3 European project, 15016 EMPHYSIS (EMbedded systems with PHYSical models In the production code Software) from 2017 to 2020 [EMPHYSIS 2017].

0.2 Introduction and problem statement of the thesis

In the recent years, study on advanced control methods for automotive systems has gained a huge momentum and most of the automotive industries has embarked in research and development and implementation of better control algorithms for improved passenger comfort and safety and as well as vehicle's performance in terms of energy efficiency, pollution reduction etc. This sudden surge in interest is partly in response to the advent of autonomous vehicles and it has also been the key driving factor for automotive companies to strive for better and optimal performance. Conditioned upon the aforementioned requirements both objectively and subjectively, optimal control methods fares much better than other control methods due to the systematic approach embodied in its design to tackle the above issues. In order to implement optimal control in practice, model predictive control (MPC) formulation of the optimal control problem (OCP) provides the necessary feedback control framework to work seamlessly in real-time and real-world. MPC is one of the most efficient and powerful control methodologies and over the last few years, MPC has become commonplace both in industry and academia due to its performance and optimality. Initially, MPC (known as

dynamic matrix control) was restricted to chemical engineering with application for petrochemical industries [Cutler and Ramaker 1980], however comprehending it's potential and performance benefits, the method has gradually pervaded into other streams of engineering such as automotive, aerospace, biomedical, etc. Concomitantly, this has attracted several researchers from different engineering domains and this has positively ensued in multitude of its variants, methods, techniques and theory. Hitherto, some of the well-known extensions of the MPC [Raković and Levine 2018] include (to name a few) explicit MPC [Bemporad et al. 2002] (EMPC), nonlinear MPC [Rawlings, Meadows, and Muske 1994] (NMPC), Linear Parameter Varying MPC (LPV-MPC) [Morato, Normey-Rico, and Sename 2020], stochastic MPC [Mesbah 2016] (SMPC), fault-tolerant MPC [Nguyen et al. 2014] (FTMPC), tube based MPC [Mayne, Seron, and Raković 2005] (TMPC), learning based MPC [Aswani et al. 2013] (LMPC), economic MPC [Ellis, Durand, and Christofides 2014] (eMPC), adaptive MPC [Bujarbaruah et al. 2018] (AMPC) etc. and more exciting extensions to be engendered in years to come and this trend seems to be growing unabatedly. The main reason for this popularity and rapid adoption stems from the idea of receding horizon control, the fact that an online optimization problem (in case of EMPC, an offline optimization problem is precomputed) is solved at every sampling period to obtain the optimal control inputs for the current state of the system. This provides the necessary leeway for the control engineer either to statically or dynamically include the required objectives and system constraints into the optimization problem for control.

With the given advantages of the MPC controller, in today's world, MPC is one of the highly coveted and sought method by the automotive industry due its ability to explicitly shape the performance and safety requirements by means of objective and constraint functions. However, the major hindrance for the method from being pervasive is the need for high computational time for solving the online optimization problem. Over the last few decades with the development of advanced and sophisticated embedded processors, the gap between theory and practice for MPC controller has abridged. At the same time the MPC methods, techniques and its variants have evolved substantially in large proportions and number of research papers on these topics stands as a testimony for this fact [Raković and Levine 2018]. Thus, it can be scarcely denied that with increased complexity of methods, a need for increased computational resource is inevitable to meet the real-time (RT) requirements, especially for fast sampled systems. Under this premise, the multicore hardware architecture of Graphic Processing Unit (GPU) displays a strong potential to provide a boost for unison of theory and practice for several state of the art MPC methods. GPU at its outset was primarily developed for accelerating the 3D graphics rendering pipeline for game engines and other multimedia applications. However in turn of events, due to its unique parallelism encompassed in its hardware (H/W) and software (S/W) architecture, it gained widespread attention in the scientific community and revealed a big deal in solving humongous simulations at faster time scales. In automotive domain, in the light of demanding computational needs, GPU based parallel computing has proven its mettle in solving RT path planning, navigation and control problems for autonomous vehicles [Bojarski et al. 2016].

The work proposed in this thesis combines the utility of MPC methodology as a control S/W paradigm and optionally, the H/W architecture of multi-core processors by the likes of

GPUs and the proposed control scheme is termed Parameterized NMPC (pNMPC). In due course of the thesis work, a code generation S/W tool was developed and implemented in C++ environment with interface to MATLAB/Simulink for embedded implementation of the proposed control scheme for automotive systems. There are several automotive modules where control system play an important role in ensuring comfort and safety such as Electronic Stability Control system (ESC), Anti-lock Braking Systems (ABS), Automatic Emergency Braking (AEB), Collision Avoidance Systems (CAS) etc. [Rajamani 2011]. In this thesis the subject of focus is considered around modeling and control of suspension systems. The thesis was conducted under the purview of the EMPHYSIS project and the outcome of this thesis aligns with the goals of the project. The primary goal of the EMPHYSIS project is to utilize model based methods for design of advanced automotive control systems. The final outcome of the project is to design and develop an automotive standard known as the Embedded Functional Mockup Interface (eFMI), which is used to deploy model based controllers onto the embedded automotive harware or Electronic Control Units (ECUs). GIPSA-lab is one of the academic partners of the project and the bestowed role is project demonstrator. Thus, GIPSA-lab is tasked with implementation of eFMU based controller for control of semi-active suspension system for the INOVE test platform and/or Hardware In the Loop (HiL) implementation in dSPACE MicroAutoBoxII (MABXII). This work is carried out in concerted effort along with its project partners from the French consortium which includes Siemens, SOBEN and Renault.

Vehicle suspension systems play a critical role in guaranteeing safety and comfort for the onboard passengers. There exists plethora of suspension systems depending upon the mode of operation and its technology, however under a bird's eye view, the entire spectrum can be briefly classified into passive, semi-active and active suspension systems. Amongst the three classes, semi-active suspension systems are quite popular in the automotive industry due to multitude of reasons such as negligible power demand, safety, low cost and weight and significant impact on vehicle performance [Savaresi et al. 2010]. Some of the prominent semi-active suspension technologies are a) Electro-Hydraulic (EH), b) Electro-Rheological (ER) and c) Magneto-Rheological (MH) based system. In this thesis, the main theme is considered around modeling and control of ER semi-active suspension system for the INOVE test platform [Vivas-Lopez et al. 2014a]. Thus, the aim of this thesis is to utilize the proposed pNMPC scheme for control of vehicle vertical dynamics by virtue of suspension systems.

In summary, the main contributions of this thesis are

- To propose the pNMPC scheme for control of semi-active suspension system for control of vertical dynamics of vehicle. The method was experimentally tested and valided on the INOVE test platform for a quarter car system and also tested with Hardware In the Loop (HiL) simulations on dSPACE MicroAutoBox II (MABXII).

- The thesis also proposes the plausibility of using multi-core processors by the likes of GPUs for solving the complex simulation based pNMPC scheme for fast sampled systems. As a proof of concept, this was tested in simulation for control of semi-active suspension system for a half car vertical dynamics model with the parallelized pNMPC scheme with INOVE parameters. The method was also augmented to incorporate the

stochastic road profile model and this control scheme is termed as scenario stochastic pNMPC (SS-pNMPC).

- pNMPC - A code generation S/W tool was developed for solving a derivative free pNMPC problem with the capability of generating both CPU and GPU codes. The code generation S/W was tested for a couple of examples for both CPU and GPU versions and also, tests (CPU version) on dSPACE MABXII were conducted for RT verification and validation of proposed code generation tool for a quarter car model with INOVE parameters.

0.3 Structure of the thesis

The organization of the thesis is presented with the following major parts:

- **Part I**: Thesis and theoretical background
- **Part II**: pNMPC with RT applications for control of semi-active suspension system
- **Part III**: pNMPC - A code generation software tool for implementation of derivative free pNMPC scheme for embedded control systems

The first part is dedicated to discuss the fundamentals and provide an outlook on the thesis and theoretical background.

- Chapter 1 provides a general overview on automotive suspension systems and its classifications, along with a brief overview on the mode of operation for each type. The vehicle vertical dynamics modeling is covered for quarter and half car models, which appear recurrently throughout the thesis. The performance and objective requirements of vehicle suspension systems are succinctly explained. The INOVE test platform's system configurations are expounded which includes the physical plant, software (S/W) and hardware (H/W) components. Finally, the chapter is concluded with a simplified explanation of the Functional Mockup Interface (FMI) standard and a brief overview on the EMPHYSIS project and its outcome - embedded FMI (eFMI) standard.
- Chapter 2 provides the theoretical background for the thesis. The first part of the chapter provides a quick overview on nonlinear optimization problem, its classifications, sub-branches and followed by a literature survey on several off the self nonlinear optimization solvers. The second part discusses the fundamentals of NMPC methods and a brief literature survey on several MPC/NMPC toolboxes and finally, the third part is concluded with a gentle introduction to GPGPU computing (General Purpose computing on Graphic Processing Unit) with introduction to parallel programming with CUDA, CUDA memory model, atomics and other CUDA libraries.

The second part discusses about the pNMPC scheme, its application and RT implementation. Also, the implementation of GPU based parallelized pNMPC scheme is explained in detail in Chapter 4.

- Chapter 3 first part introduces the concept of parameterized NMPC scheme and its application for control of semi-active suspension system for the INOVE test platform. The pNMPC controller's design requirements for both objective and constraints for the suspension control problem are explained in detail. The second part explains the experimental implementation of the pNMPC controller for control of semi-active suspension system. The experimental study conducted to model the ER semi-active damper characteristics and parameter estimation are discussed in detail. From the obtained model parameters, the pNMPC controller is experimentally validated on the INOVE platform and also with HiL simulations conducted on dSPACE MABXII embedded target. The core results of this chapter are based on the following papers [Rathai et al. 2018], [Rathai, Alamir, and Sename 2019].
- Chapter 4 discusses the GPU simulation implementation of the pNMPC controller for control of semi-active suspensions system in MATLAB/Simulink for the INOVE test platform model and parameters. In the first part, the parallelized pNMPC scheme is introduced and in the second part, a stochastic version of the parallelized pNMPC is introducted and also, the method was tested in RT on several NVIDIA embedded boards to verify and validate the feasibility of the proposed scheme. The core results of this chapter are based on the following paper [Rathai, Sename, and Alamir 2019].

The third part is completely dedicated for the pNMPC code generation S/W tool, its working, features, implementation and along with few examples on CPU (PC), GPU and embeddeded implementation in dSPACE MABXII.

Chapter 5 introduces the pNMPC code generation S/W tool for implementation of a derivative free pNMPC scheme for embedded control systems. This chapter expounds the features of the developed code generation S/W and also, explains the code generation process in a bird's eye view. With a variety of examples, the proposed S/W was compared against ACADO toolkit and the simulation results are presented. Furthermore, using the proposed S/W, HiL tests were conducted on dSPACE MABXII for control of semi-active suspension system for a quarter car model with INOVE parameters and the simulation results are presented. Results for the GPU version of the code generation codes are also presented in this chapter and finally, the chapter is concluded with future works and conclusions. The pNMPC code generation S/W tool C/C++, MATLAB/Simulink codes are present in the following GitHub repository [Rathai 2020].

Contributions

International conference papers with proceedings

[1] Rathai, K.M.M., Alamir, M., Sename, O. and Tang, R., 2018 *A parameterized NMPC scheme for embedded control of semi-active suspension system*, in **6th IFAC Conference on Nonlinear Model Predictive Control NMPC 2018**, IFAC-PapersOnLine, 51(20), pp.301-306., Madison, Wisconsin, USA.

[2] Rathai, K.M.M., Sename, O. and Alamir, M., November, 2018, *A comparative study of different NMPC schemes for control of semi-active suspension system*, in **VSDIA 2018 - 16th International Conference on Vehicle System Dynamics, Identification and Anomalies**, Budapest.

[3] Rathai, K.M.M., Sename, O. and Alamir, M., 2019, January. *Reachability based Model Predictive Control for Semi-active Suspension System*, in **2019 Fifth Indian Control Conference (ICC)** (pp. 68-73), IEEE, New Delhi, India.

[4] Rathai, K.M.M., Alamir, M. and Sename, O., 2019. *Experimental implementation of model predictive control scheme for control of semi-active suspension system*, in **9th IFAC Symposium on Advances in Automotive Control AAC 2019**, IFAC-PapersOnLine, 52(5), pp.261-266. Orléans, France.

[5] Rathai, K.M.M., Alamir, M. and Sename, O., 2020. *GPU based Stochastic Parameterized NMPC scheme for Control of Semi-Active Suspension System for Half Car Vehicle*, submitted and accepted to **21st IFAC World Congress 2020**, Berlin, Germany (To appear).

Journal papers

[1] Rathai, K.M.M., Sename, O. and Alamir, M., 2019. *GPU-based parameterized NMPC scheme for control of half car vehicle with semi-Active suspension system*, **IEEE Control Systems Letters**, 3(3), pp.631-636, presented in **58th Conference on Decision and Control (CDC)**, Nice, France.

[2] Rathai, K.M.M., Alamir, M. and Sename, O., 2020. *pNMPC-A Code Generation Software Tool for Implementation of Derivative Free Parameterized NMPC Scheme for Embedded Control Systems*, submitted to **IEEE Control Systems Technology** (Under review).

Part I
Thesis and theoretical background

Chapter 1

Thesis background

Contents

1.1 Prelude

This chapter is intended to provide an overview on some of the fundamental concepts on automotive suspension systems, its performance and objective requirements and the INOVE test platform stationed at GIPSA lab, Grenoble, France. Finally, the chapter is concluded with a brief overview on Functional Mock-up Interface (FMI) and the EMPHYSIS project which stands for EMbedded systems with PHYSical models In the production code Software and the outcome of the project i.e. the embedded FMI (eFMI) standard. FMI and eFMI are generic open source software (S/W) standards, yet, mostly used in automotive industry

for model exchange and co-simulation of automotive subsystems as well as its embedded implementation on electronic control units (ECUs) or embedded hardware.

1.2 Automotive suspension system

1.2.1 Introduction to automotive suspension systems

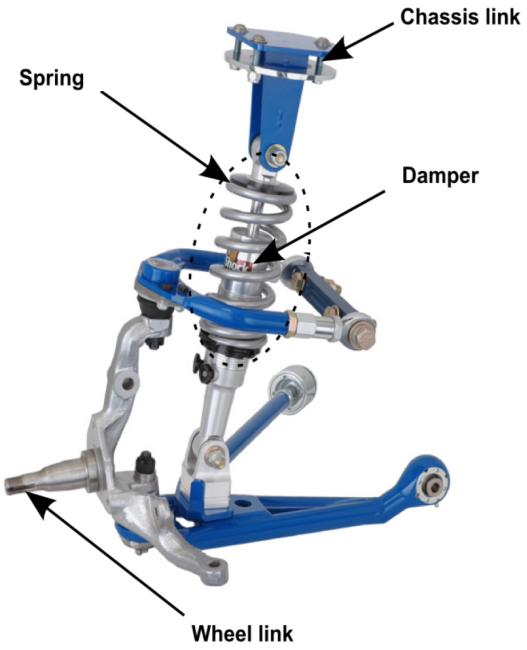

Figure 1.1: CAD design of double wishbone vehicle suspension system

Automotive suspension system is one of the most crucial and an indispensable component of road vehicles. Typically, the tasks of a suspension system are three folds which are a) to provide ride comfort for the passengers (minimize chassis vibrations), b) to provide safety by ensuring that the wheels of the vehicle are in contact with the road (road holding) and c) to minimize the roll angle of the vehicle (ride handling) [Savaresi et al. 2010]. The physical realization of a suspension system is materialized with three main parts and an optional part which are a) the structure of the suspension system, which defines the suspension geometry, b) the spring element, which provides proportional and opposite force to the displacement of the suspension system and also acts as an energy storage element, c) the damper element aka shock absorber which provides proportional and opposite force to the velocity of the suspension system and also used to dissipate energy [Goodarzi and Khajepour 2017], [Gillespie 1992], and d) the optional part, which involves a motion actuator which applies only for active suspension system, which is used to provide additional energy by exerting force on the system. A typical anatomy of a suspension system is illustrated in Fig. 1.1. The suspension system is housed between the wheel (unsprung mass) and the chassis (sprung mass) of the vehicle and

it supports the whole weight of the vehicle. The suspension system also play an important role in describing the lateral, longitudinal and vertical dynamics of the vehicle as the forces and moments along these respective axes are directly affected through the wheels and the suspension system links the wheels to the road, the chassis to the wheels, thereby directly transmits these effects to the chassis. These effects are well pronounced during cornering or evasive maneuvers where vehicle handling becomes an issue of concern.

On the basis of suspension mechanics, the elements that constitute the suspension system can be divided into two parts which are the dynamic elements and kinematic elements. In this thesis, the prime focus lies on the study and control of suspension systems based on the dynamic elements which includes the spring, shock absorbers etc. whereas on the other side of the spectrum i.e. the kinematic elements, which involves the suspension geometry, suspension mechanism etc. and the kinematic effects are not considered in this work. Furthermore, on the basis of technological and dynamic perspective, suspension systems can be broadly classified into three types which are a) passive, b) active and c) semi-active suspension systems. The differences in the these types are well characterized in the force vs deflection velocity plot aka Speed Effort Rule (SER) plot which is illustrated in Fig 1.2. A more detailed exposition into different types of suspension systems are discussed in the following sections.

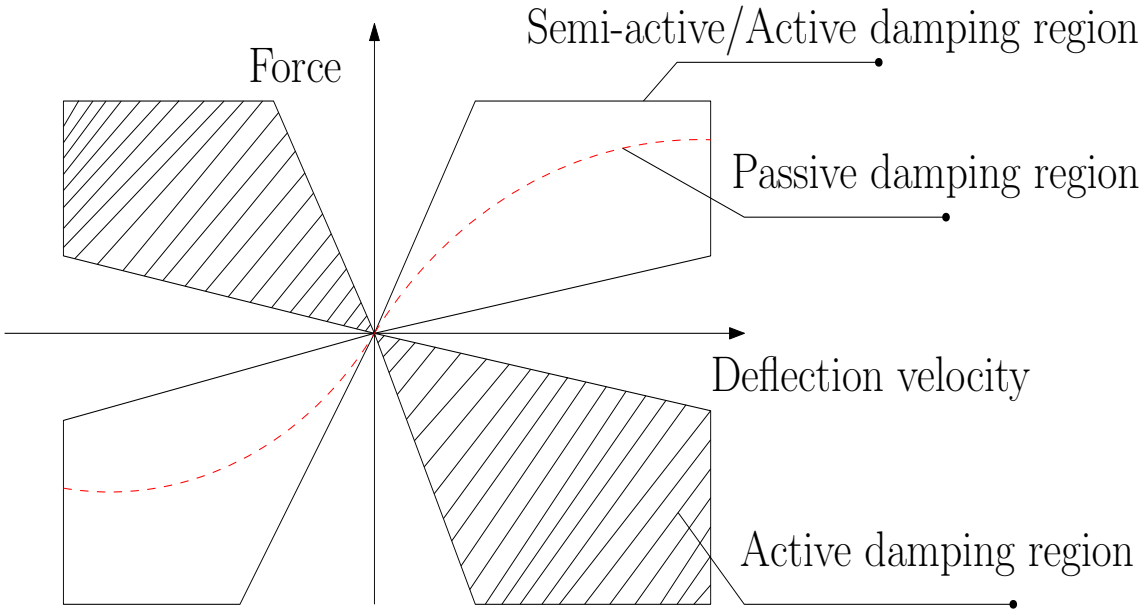

Figure 1.2: Force-deflection velocity characteristics (SER) for passive, semi-active and active suspension systems

1.2.2 Passive suspension system

The passive suspension system is one of the most basic and perhaps a ubiquitous system deployed in several commercial automobiles. Typically, a passive suspension system consists of a spring and a shock absorber and the system is void of any element of control for manip-

ulating the suspension characteristics online [Miller 1988], [Savaresi et al. 2010]. The design parameters such as stiffness coefficient, damping coefficient, suspension structure etc. are optimized and tuned during the research and development phase of the product and the design parameters are chosen based on several requirements such as road condition, rattle space requirements, maximum number of passengers etc. Typically, the spring element is constituted either by coil springs or gas springs which acts as the stiffness component and the dissipative element is constituted by hydraulic shock absorbers which exhibit nonlinear characteristics such as hysteresis, friction etc. The pros of a passive suspension system are: low cost, low weight, free of any electronics or sensor components and the cons are: not suitable for varying road conditions where comfort and safety are of paramount importance and the suspension system is inflexible to adaptation. The SER plot for passive system is pictorially illustrated by the red dashed line in Fig. 1.2 which lies in the first and third quadrant in the SER plot, which typically has non-linear characteristics.

1.2.3 Active suspension system

In contrast to the passive suspension system, the active suspension system can generate, store and dissipate energy which is illustrated in Fig. 1.2, where the SER characteristics lies in all four quadrants [Tora 2012]. Typically, an active suspension system consists of a spring, a shock absorber and an active actuator to impart energy to the system [Fischer and Isermann 2004]. The actuator modulates the displacement between the chassis and wheels by providing input force which ought to be controlled [Sun, Gao, and Shi 2020]. The pros of active suspension systems are: better ride quality, handling and safety and the cons are: exorbitant price, increased power consumption, heavy mechanical components and uncertain management of safety issues [Savaresi et al. 2010].

1.2.4 Semi-active suspension system

Semi-active suspension system is partly an active suspension system, however, the system can not impart additional energy to the system and can only dissipate energy by manipulating the damper characteristics online. The region of operation is illustrated in Fig. 1.2 where the first and third quadrant represents the semi-active damping region. However, the range of operation is much wider than passive suspension system and this provides the necessary latitude for controlling the system characteristics on a broader range. The pros of semi-active suspension system are: negligible power demand, safety characteristics, significant impact on vehicle performance and low cost and weight of the system [Savaresi et al. 2010] and the cons are: reduction in vehicle handling performance, ride quality when compared to active suspension system. Nevertheless, the system is adaptable to road conditions with better controllers. In this thesis, the focus is on control of semi-active suspension system and its application for control of vertical dynamics of vehicle. Also, this thesis follows the work of former PhD theses [Poussot-Vassal 2008], [Do 2011] and [Nguyen 2016a] on control of semi-active suspension systems.

The semi-active suspension system could be classified based upon its technology and some of the most common variants are a) Electro-Hydraulic (EH), b) Magneto-Rheological (MR) and c) Electro-Rheological (ER) damping models. Despite the working mechanism of the semi-active suspension system for different technologies are distinct from each other, the underlying principle of operation, properties and characteristics are virtually the same for all semi-active damper classes. A brief insight into these working principle of these classes are provided below

- Electro-Hydraulic (EH) damper - The EH damper system consists of electronic valves, which are typically controlled by manipulating the solenoid spool valve and therefore, this operates the damper chambers for the fluid to navigate [Savaresi et al. 2010], [Spelta 2008]. By varying the fluid levels, the damping coefficient is controlled continously and linearly with the area of the opening valve [Aubouet 2010].
- Magento-Rheological (MR) damper - The MR damper consists of Magneto-Rheological fluid which is formed by suspending magnetic particles into oil. By varying the magnetic field, the particles aligns itself to the field and this in turn varies the viscocity of the fluid [Rabinow 1948] [J Lozoya-Santos et al. 2012].
- Electro-Rheological (ER) damper - The ER damper shares a similar principle of MR damper, however the damper fluid is a mixture of oil and micron sized electric field sensitive particles. Thus, by varying the electric field, the damping co-efficient is changed and this principle is used for control of the damper [Winslow 1947]. The semi-active dampers utilized for the INOVE test platform are ER dampers (for all four wheels of the vehicle) [Nguyen 2016b] [Vivas-Lopez et al. 2014b].

1.3 Vertical vehicle dynamics modeling

1.3.1 Quarter car vehicle model

The quarter car model illustrated in Fig. 1.3 consists of two mass elements which are the sprung mass element (chassis) and the unsprung mass element (wheel). The vertical dynamics model for the system around equilibrium [Savaresi et al. 2010] is expressed with

$$\begin{aligned} m_s\ddot{z}_s &= -k_s(z_s - z_{us}) + u \\ m_{us}\ddot{z}_{us} &= k_s(z_s - z_{us}) - u - k_t(z_{us} - z_r) \end{aligned} \tag{1.1}$$

where, m_s and m_{us} are the sprung and unsprung masses respectively, k_s and k_{us} are the stiffness coefficients of the damper system and wheel respectively. z_s, $\dot{z}_s$, z_{us} and $\dot{z}_{us}$ are the sprung mass position, velocity and unsprung mass position, velocity respectively and z_r is the vertical road displacement. It is worth to notice that u in this model is the force exerted due to the semi-active suspension system. $\dot{z}_{def} = \dot{z}_{us} - \dot{z}_s$ and $z_{def} = z_{us} - z_s$ are the deflection velocity and deflection position between the sprung and unsprung mass. The dynamics equation (1.1) can be compactly expressed in state space form with

$$\dot{\mathbf{x}} = A_c\mathbf{x}(t) + B_c u(t) + B_c^{dist} d(t) \tag{1.2}$$

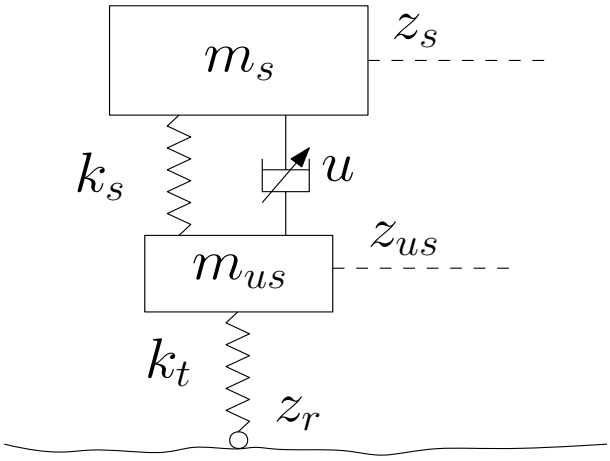

Figure 1.3: Quarter car vehicle model

where, $\mathbf{x} = [z_s\ z_{us}\ \dot{z}_s\ \dot{z}_{us}]^T$ are the system states, $d(t) = z_r$ is the disturbance input from the road profile. $A_c \in \mathbb{R}^{4\times4}$, $B_c \in \mathbb{R}^{4\times1}$ and $B_c^{dist} \in \mathbb{R}^{4\times1}$ are the system matrix, input matrix and disturbance matrix respectively (Listed in equation (1.3)).

$$A_c = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{-k_s}{m_s} & 0 & \frac{k_s}{m_s} & 0 \\ 0 & 0 & 1 & 0 \\ \frac{k_s}{m_{us}} & 0 & \frac{-(k_s+k_t)}{m_{us}} & 0 \end{bmatrix}, B_c = \begin{bmatrix} 0 \\ \frac{1}{m_s} \\ 0 \\ \frac{-1}{m_{us}} \end{bmatrix}, B_c^{dist} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{k_t}{m_{us}} \end{bmatrix} \tag{1.3}$$

1.3.2 Half car vehicle model (Roll oriented model)

The half car vertical dynamics roll oriented model, illustrated in Fig. 1.4 is a 4 degrees of freedom (DOF) model [Mahala, Gadkari, and Deb 2009] [Savaresi et al. 2010], which involves chassis dynamics (heave motion), roll dynamics and dynamics of the two unsprung masses (wheels). The model ought to be viewed as the vehicle being scrunched from the front and rear ends into a single block. Let the left and right corner of the vehicle be indexed with $i \in \{l, r\}$ respectively. The 4 DOF mathematical model is expressed with the following equations

$$\begin{cases} m_s\ddot{z}_s = -\sum_{i\in\{l,r\}} F_{s,i} \\ I_x\ddot{\theta} = (l_l F_{s,l} - l_r F_{s,r}) \\ m_{us,l}\ddot{z}_{us,l} = (-F_{s,l} + F_{t,l}) \\ m_{us,r}\ddot{z}_{us,r} = (-F_{s,r} + F_{t,r}) \end{cases} \tag{1.4}$$

where, m_s, $m_{us,l}$, $m_{us,r}$ represent the chassis mass, unsprung masses for the left and right corners. I_x represents the moment of inertia along the roll axis. l_l and l_r represents the length of the chassis from the left and right corners with respect to centre of gravity (COG). $F_{s,i}$ represents the chassis forces and $F_{t,i}$ represents the wheel forces $\forall i \in \{l, r\}$ which are expressed

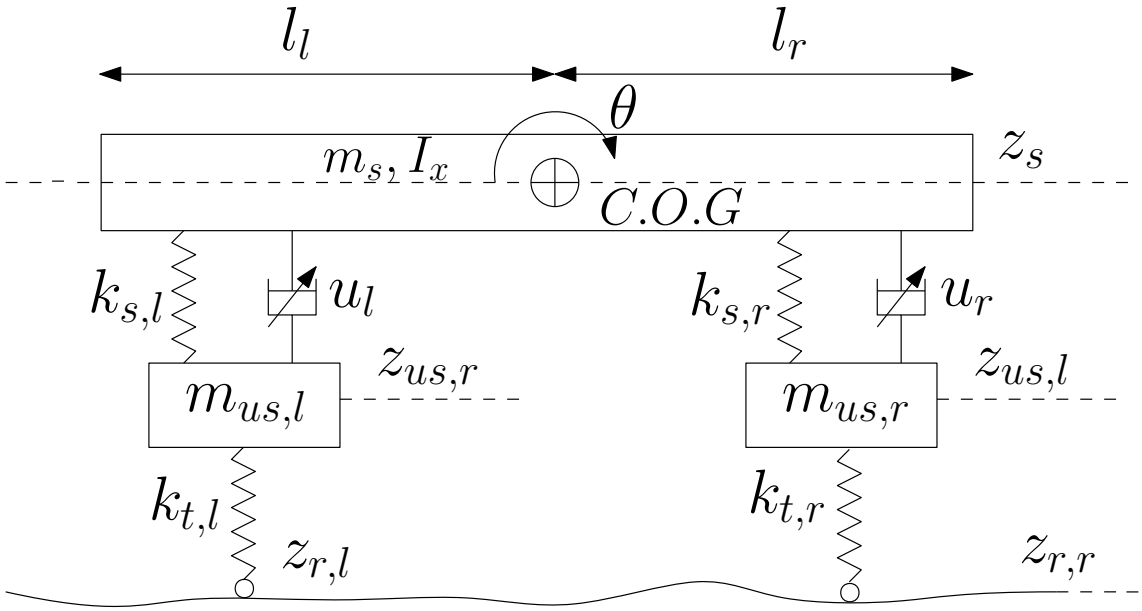

Figure 1.4: Half car roll oriented vehicle model

with

$$\begin{aligned} F_{s,i} &= -k_{s,i}(z_{s,i} - z_{us,i}) + u_i \\ F_{t,i} &= -k_{t,i}(z_{us,i} - z_{r,i}) \end{aligned} \tag{1.5}$$

where, $k_{s,i}$ and $k_{t,i}$ represents the stiffness coefficent of the semi-active suspension system and wheel respectively. $z_{r,i}$ and $z_{us,i}$ represents the vertical road displacement and unsprung mass position $\forall i \in \{l, r\}$. u_i represents the actuation force $\forall i \in \{l, r\}$ exerted due to the semi-active suspension system. $z_{s,i}$, $\forall i \in \{l, r\}$ represents the sprung mass displacement at each corner which are obtained from the following equations

$$\begin{aligned} z_{s,l} &= z_s + l_l \sin\theta \\ z_{s,r} &= z_s - l_r \sin\theta \end{aligned} \tag{1.6}$$

Let $\mathbf{X} = [z_s,\ \theta,\ z_{us,l},\ z_{us,r},\ \dot{z}_s,\ \dot{\theta},\ \dot{z}_{us,l}\ , \dot{z}_{us,r}]$ denote the state vector ($\mathbf{X} \in \mathbb{R}^8$), $\mathbf{U} = [u_l,\ u_r]$ denote the input vector ($\mathbf{U} \in \mathbb{R}^2$) and $\mathbf{D} = [z_{r,l},\ z_{r,r}]$ denote the disturbance vector ($\mathbf{D} \in \mathbb{R}^2$), then the half car model (1.4) can be compactly expressed using the nonlinear state space equation with

$$\dot{\mathbf{X}}(t) = f(\mathbf{X}(t), \mathbf{U}(t), \mathbf{D}(t)) \tag{1.7}$$

1.4 Performance and objective requirements for vehicle suspension control

Despite the fact that the suspension system plays a pivtol role in shaping the lateral, longitudinal and vertical dynamics of the vehicle, automotive engineers study and view the system

predominantly through the lens of vertical dynamics as the effects of road roughness are very palpable on the vehicle body [Goodarzi and Khajepour 2017]. The three important performance and objective requirements for suspension control are:

- Ride comfort - Ride comfort objective is related to the comfort of the on-board passengers of the vehicle. The comfort objective can be quantified by vibration isolation due to road roughness. The spectrum of frequencies from $0-20Hz$ is considered as the comfort zone and ideally, the vertical displacement of the chassis of the vehicle must be same of the road in low frequencies ($< 1Hz$) and attentuated for high frequencies ($> 1Hz$) [Savaresi et al. 2010]. This is achieved by minimizing the vertical acceleration or velocity or displacement of the chassis of the vehicle.
- Road holding - Road holding objective requires the vehicle to remain in contact with the road which is highly sought for safety requirements of the vehicle. The cornering, traction and braking abilities depend on the lateral and longitudinal forces of acting on the tyre which is directly a function of the normal force [Goodarzi and Khajepour 2017]. Thus, the primary goal of this objective is to minimize the fluctuations in the normal force of the wheel and this is assured by minimizing the deflection between the wheel and road position.
- Ride handling - Ride handling is a general requirement which necessitates the vehicle to be stable in every maneuver and also an important condition for stability. The fundamental requirement of ride handling objective is to ensure that the vehicle behavior is predictable and this information is communicated to the driver. The suspension system play a crucial role for ride handling by minimizing the vehicle's roll and pitch motion, controls the wheel angles and decreases the lateral load transfer during cornering maneuvers [Goodarzi and Khajepour 2017].

1.5 INOVE test platform (GIPSA lab)

The INOVE test platform is a GIPSA-lab experimental setup built in collaboration with SOBEN company and the platform is a 1:5 scaled 4 poster testbed which is exclusively dedicated to study and assess the performance of vertical dynamics of vehicle under several road profile scenarios. The test platform consists of three major components which are a) Host PC, b) Target system and c) Physical plant (illustrated in Fig. 1.5). In systems point of view, the test platform is used to verify and validate controllers, observers and also, perform fault diagnosis and condition monitoring [Vivas-Lopez et al. 2014a] of the components. The INOVE platform is shown in Fig. 1.6.

1.5.1 Hardware description

- **Host PC** - The host PC is where the controller/observer/fault diagnosis/condition monitoring modules are designed, developed, deployed and tested to verify and validate the

module's performance on the target system. The front-end S/W is MATLAB/Simulink and by virtue of Simulink Real-Time Workshop (RTW), the Simulink model is converted to embedded C code and deployed on the target.

- **Target system** - There are two target systems for real-time (RT) implementation which are xPC target and dSPACE MABXII. The targets run Simulink RT operating system.
- **Data acquisition system (DAQ)** - The I/O cards include 2x National Instruments PCI-6259 DAQ boards, 1x Quatex QSC 100 serial board. The sampling frequency of the DAQ is 200 Hz, i.e. a sampling period of 5 ms.

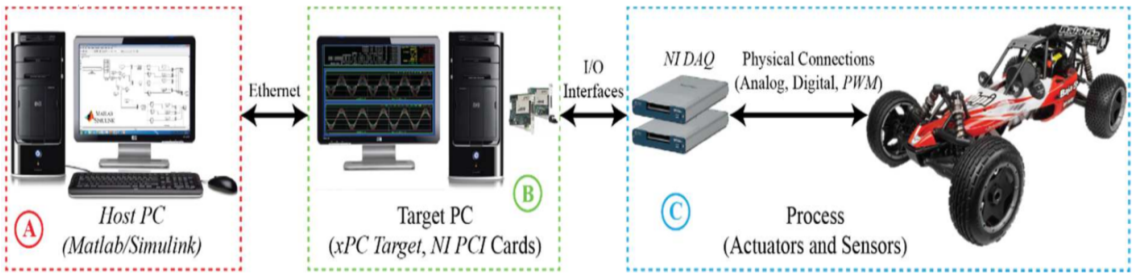

Figure 1.5: Schematic of INOVE test platform [Vivas-Lopez et al. 2014a]

1.5.2 Plant and actuator description

A brief outlook over the plant and actuator systems are presented below

- The physical plant is a 1:5 scaled baja styled racing car which refects a miniature version of a full vehicle which encompasses wheels, engine and braking system. The key component of the plant is the semi-active damping system.
- The semi-active suspension system involves four ER dampers with a force range of $\pm 50N$. The damper is manipulated by varying the input voltage from $0V$ to $5kV$ which is in turn manipulated by varying the pulse width modulation (PWM) duty cycle (DC) signal through CarCon2 driver module. The frequency of PWM-DC signal is $25kHz$.
- The Remote Controlled (RC) car is mounted over four OMRON linear servomotor module over all four corners and this is controlled appropriately to generate user defined input road profile. The servomotors has a bandwidth of $0 - 20Hz$ and a maximum linear velocity of 1.5 m/s.

1.5.3 Sensor description

To capture the complete vertical dynamical behavior of the vehicle, the plant is probed by means of several onboard sensors.

- To measure the vertical acceleration of the unsprung masses (wheels), four Texense 1-axis capacitive accelerometer are used.

- The stroke deflection (i.e. the displacement between the sprung mass and unsprung mass) and the road displacement (motor position) are measured using eight Gefran resistive linear displacement sensors for each corner of the platform.

- To measure the unsprung mass displacement, four Micro-Epsilon draw-wire displacement sensors are used for all four corners of the platform.

- To measure the acceleration and angular velocity of the chassis a SBG MEMS based Attitude and Heading Reference System (AHRS) unit is placed on the sprung mass of the plant. The AHRS unit measures 3 accelerations along lateral, longitudinal and vertical axes and 3 angular velocities along roll, pitch and yaw axes.

- The force exerted by the ER damper is measured via four force sensors mounted on ER dampers across all four corners.

- To measure the tire forces across all four corners, four force sensors are placed at the base of the 4-poster testbed.

Figure 1.6: INOVE test platform GIPSA-lab

1.6 Functional mock-up interface (FMI)

The functional mockup interface (FMI) is a tool independent standard used for model exchange and co-simulation of dynamical systems. The standard was the result of ITEA2 MODELISAR project, where FMI 1.0 was primarily developed for model exchange purpose [Blochwitz et al. 2011]. Ever since the release of its initial version, its popularity surged and several tool vendors adopted the standard and this paved way to FMI 2.0 which is used for both model exchange and as well as co-simulation purpose with improved and new features to ease the use and increase the performance for larger models [Blockwitz et al. 2012]. An instance of FMI is called as Functional Mockup Unit (FMU), i.e. the FMI standard defines the blueprint and the FMU is a container which consists of all the source code files (C-code), meta-data model files (XML files), binaries (static/dynamic libraries) files which are compliant with FMI standard to represent the underlying dynamical model. As mentioned previously, the FMI standard consists of two parts:

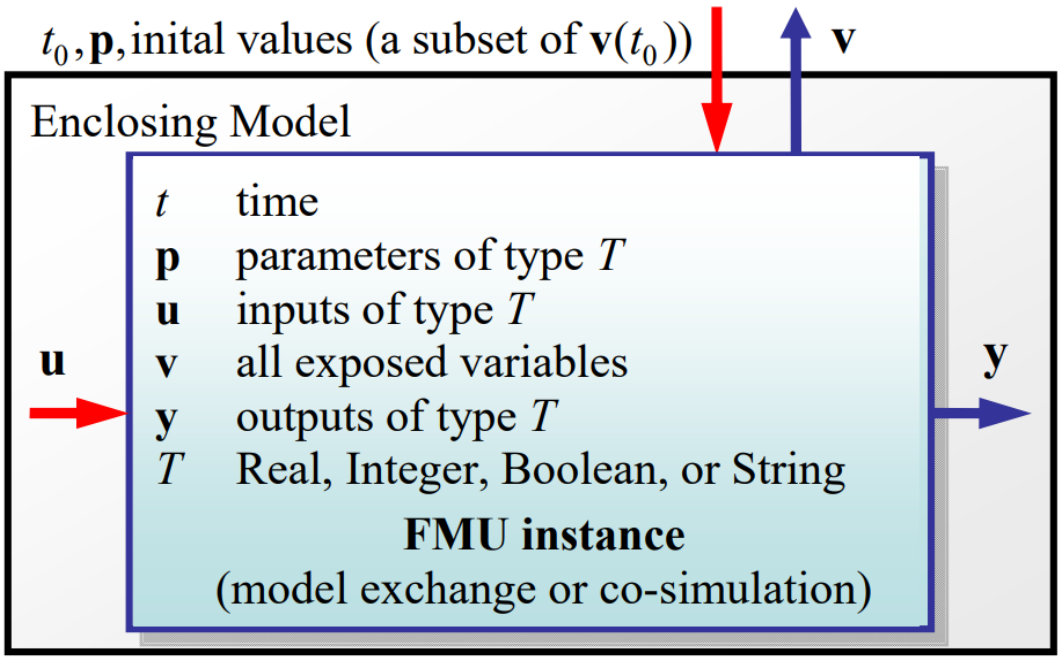

Figure 1.7: Schematic of data flow between the simulation tool and the FMU [Blockwitz et al. 2012]

- FMI for model exchange - FMI for model exchange defines an interface to model the dynamical system which is described by differential, algebraic and discrete-time equations. The model developed in a FMI compliant tool is exported to FMU, which encompasses the details of the dynamic system such as states, inputs, parameters etc. The simulation tool utilizes this information from the FMU and simulates it by means of its inbuilt solvers such as explicit/implicit integrators, fixed/variable step solvers etc. It is also important to note that the quality of solution is contigent over the type of solver used

to simulate the FMUs and this in turn is provided by the simulation environment.

- FMI for co-simulation - The general concept of co-simulation is to simulate multiple interdependent subsystems in the simulation environment. The FMU exported for co-simulation purpose is shipped with the model and as well as the solver to simulate the system independently. Typically, the simulation tool would consist of multiple FMU blocks which interact with each other through input/output (I/O) channels and the primary task of the simulation tool is to handle the communication traffic between the I/O ports of the subsystem, synchronization of data transfer, signal extrapolation and error control. In short, the simulation tool micromanages the whole collection of FMU subsystems and simulates it holistically and seamlessly to reflect the dynamics of the real-world system.

A schematic of the FMUs subsystem is shown in Fig. 1.7, where the block is assumed to be interacting with the simulation environment (For more details, refer [Blochwitz 2014]).

1.7 EMPHYSIS project and embedded FMI (eFMI) standard

1.7.1 EMPHYSIS project

Figure 1.8: Simplified schematic of eFMI workflow [EMPHYSIS 2017]

The goal of the ITEA 3 project EMPHYSIS (EMbedded systems with PHYSical models In the production code Software) is to enhance the production code of embedded systems in automotive vehicles with advanced algorithms based on physical models to improve the performance of the underlying automotive system and to increase the productivity of embedded software developement [EMPHYSIS 2017]. Over the last few years, there has been an increased demand for the need for safe, clean and efficient road vehicles as well the stringent environmental regulations on CO2 and NOx emissions has propelled the automotive industry to address the aformentioned issues with an ironclad measure. In order to circumvent these issues, utilizing physical models for control and diagnosis of automotive systems can improve the performance several folds and also, implementation of model based virtual sensors

(observers) can downsize the production cost. The essence of physics based functions is the ability to predict the dynamic behavior of the system in its whole operation space to achieve significantly better vehicle performance. Despite the several merits of utilization of physical models, there are several challenges such as RT implementation, scalability, efficient operation etc. and these poses a serious concern for engineers. In order to tackle these issues, the EMPHYSIS project was constituted and has marshalled several automotive industries, software companies, academic institutions and research labs spanning across five countries (Germany, France, Sweden, Belgium and Canada) to investigate and develop a generic S/W standard named eFMI (Embedded Functional Mockup Interface). The list of tool vendors, academic partners and ECU S/W developers are shown in Fig. 1.9.

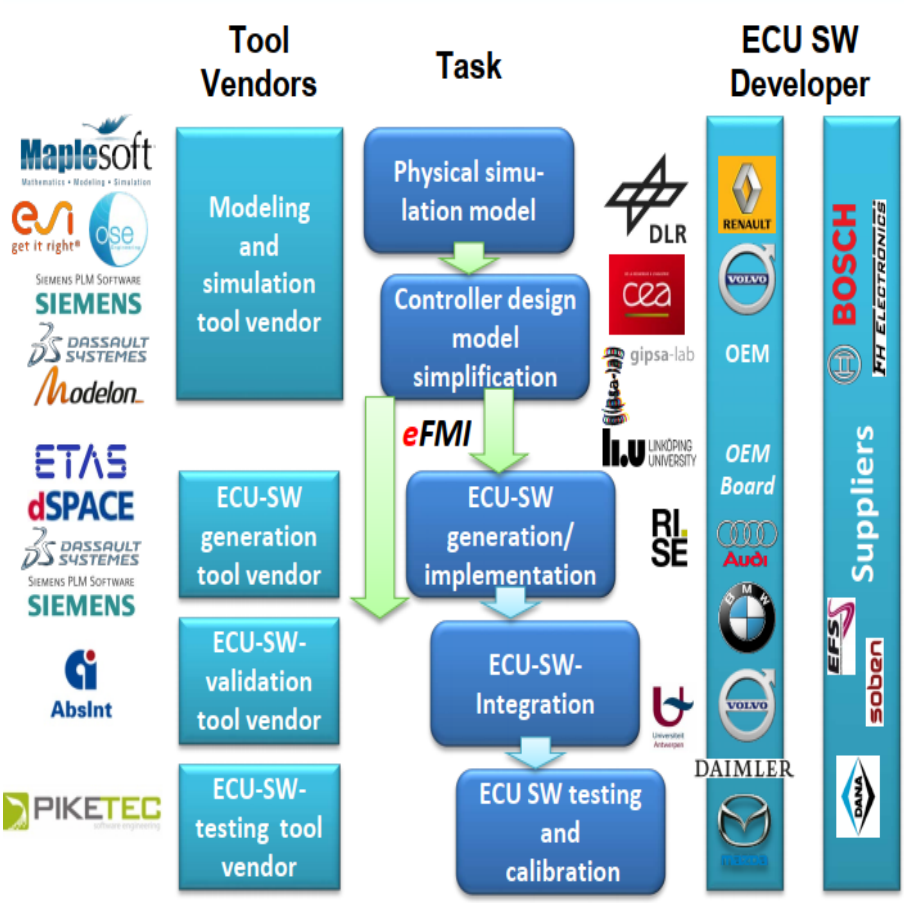

Figure 1.9: Detailed schematic of eFMI workflow [EMPHYSIS 2017]

1.7.2 Embedded Functional Mockup Interface (eFMI)

The eFMI standard provides a systematic framework to implement physics based functions in an automated way on electronic control units (ECU), microcontrollers or embedded systems. The key idea is to provide a standard and seamless interoperability of eFMI via a code generation feature which will transform physics based functions to low level eFMI production

code that fulfils the requirements of ECU S/W and H/W compliant with AUTOSAR standard [Fürst et al. 2009]. Fig. 1.8 illustrates a simplified workflow of eFMI code deployment from simulation S/W to embedded H/W. The eFMU container is typically exported from modeling tools such as AMESIM, OpenModelica, SimulationX etc. and the generated module is deployed either in embedded hardware or ECUs.

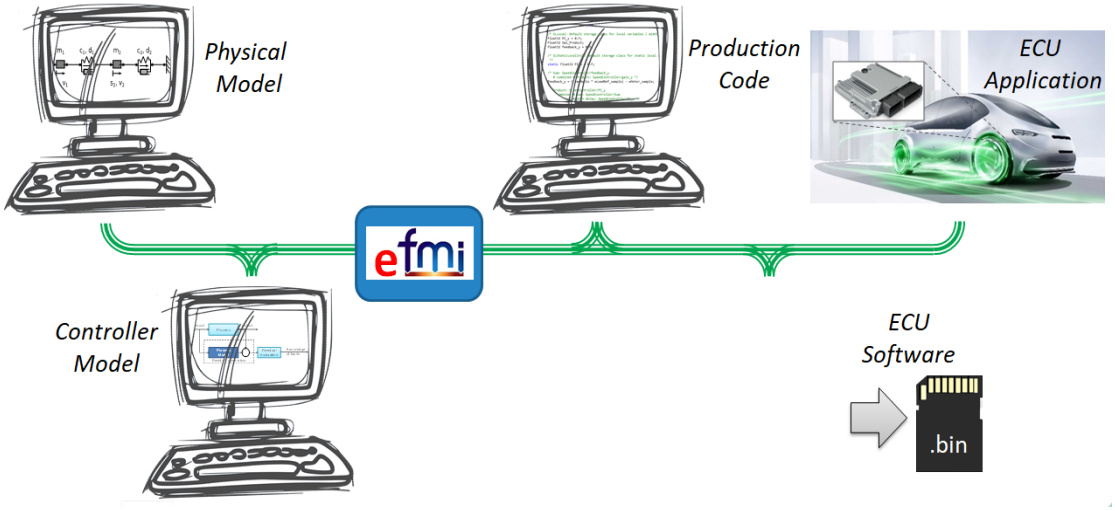

Figure 1.10: Detailed schematic of eFMI workflow [EMPHYSIS 2017]

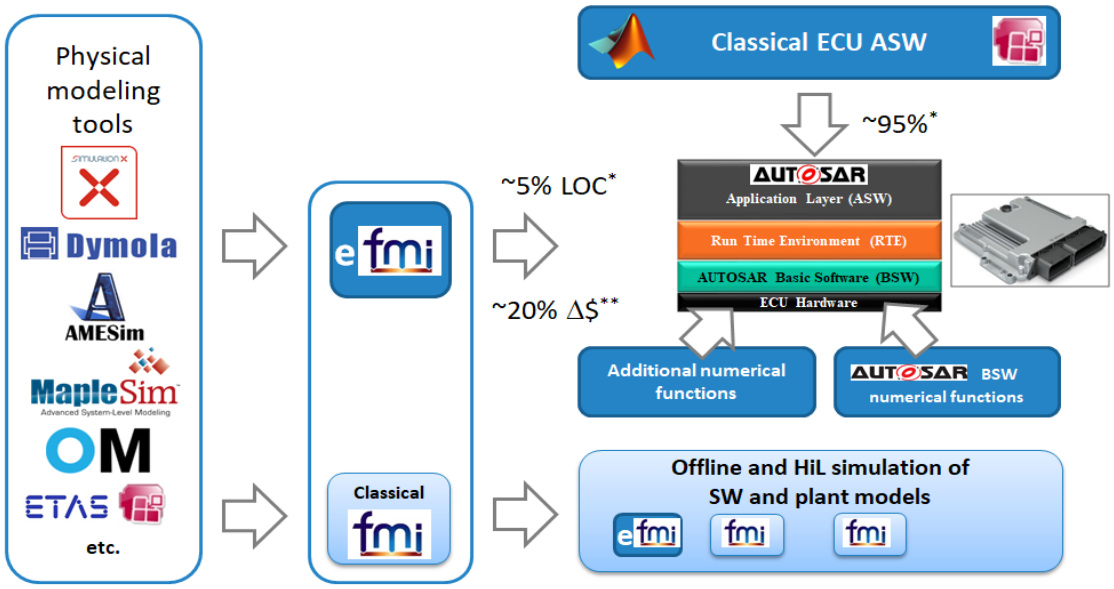

Figure 1.11: Automotive standard compliant production code in eFMI workflow [EMPHYSIS 2017]

Fig. 1.10, illustrates a more detailed version of the eFMI work flow. On a higher level, the underlying physical models as well as the controllers for the automotive systems are modeled using S/Ws such as AMESIM, SimulationX, OpenModelica etc. A specific module in these

S/Ws convert the designed model/model based controllers into eFMU containers. This is further processed to production code which ought to be compliant with other automotive standards such as AUTOSAR, MISRA-C etc. and the final production code is deployed into the embedded H/Ws or ECU. The whole deployment cycle is illustrated in Fig. 1.11.

1.8 Conclusions

In this chapter a brief overview on automotive suspension system is provided which involves the classification of suspension systems and more specifically on classification of ER semi-active damper system based upon its technology. The two vertical vehicle dynamics models - quarter and half car models are presented, which are pertinent to this thesis and recurs in the upcoming chapters. A qualitative explanation of the objective requirements for the suspension system is discussed and this notion is concretized in the following chapters. A quick overview of the INOVE experimental platform is provided and finally, the chapter concludes with a simple explanation on the FMI standard and also, the gravitas for the EMPHYSIS project as well its outcome i.e. the eFMI standard is expressed.

Chapter 2

Theoretical background

Contents

2.1 Introduction

This chapter is dedicated to provide the fundamentals on non-linear optimization problems, different methodologies for implementation of non-linear model predictive control (NMPC) schemes and finally, the chapter is concluded with a brief outlook on general purpose graphic processing unit (GPGPU) computing using CUDA programming framework.

2.2 Nonlinear Optimization

2.2.1 Introduction

A general formulation for an optimization problem is defined by the following form

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & x \in \mathcal{X} \subseteq \mathbb{X} \end{aligned} \tag{2.1}$$

where x is the vector of optimization or decision variables of interest and $\mathcal{X}$ is the admissible or feasible subset on the optimization domain ($\mathbb{X}$) [Borrelli, Bemporad, and Morari 2017]. The optimization problem is carried out with respect to an objective function $f : \mathbb{X} \to \mathbb{R}$ which is used to assign every vector x a cost value with $f(x) \in \mathbb{R}$. Let the optimal cost value for the problem (2.1) be represented with f^* and the optimizer or optimal solution with x^*, then the following equation holds

$$f(x) \geq f(x^*) = f^* \ \ \forall x \in \mathcal{X}, \text{with } x^* \in \mathcal{X} \tag{2.2}$$

In simple words, equation (2.2) states that among all the possible feasible values of x, the solution x^* provides the optimal solution for the objective function $f(x)$. The optimal objective value f^* is also defined as the greatest lower bound for the problem (2.1). If $f^* = -\infty$, then the problem is regarded as unbounded below and if $f^* = \infty$, then the problem is regarded as infeasible and if $\mathcal{X} = \mathbb{X}$, i.e. the whole search space, then the problem is regarded as an unconstrained optimization problem. The problem of determining the existence of optimal solution is called as a feasibilty problem. The set of optimizers x^* (the solution could either be singleton or set-valued) that optimizes problem (2.1), i.e. $f^* = f(x^*)$ is defined with the following form

$$\underset{x \in \mathcal{X}}{\operatorname{argmin}} \ f(x) = \{x \in \mathcal{X} : f(x) = f^*\} \tag{2.3}$$

The optimization problem defined in equation (2.1) is generic in nature, however depending upon the structure of the objective function $f(x)$ and the search space domain $\mathcal{X}$, there exists plethora of classifications such as linear and non-linear, deterministic and stochastic, continuous and discrete, convex and non-convex, finite and infinite dimensional and also, the combination of all the aforementioned divisions. In this thesis the prime focus lies on a specific class of optimization problem known as non-linear optimization or non-linear programs (NLP), which is defined by

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g(x) = 0, \\ & h(x) \geq 0 \end{aligned} \tag{2.4}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^p$, $h : \mathbb{R}^n \to \mathbb{R}^q$ are the objective function, equality constraints and inequality constraints respectively and the involved functions are assumed to be continuous and smooth in nature. In case when the knowledge of these functions are available, one can make further assumptions such as the functions are once or twice differentiable. In other cases, such as black box models which might involve computer codes, compiled binaries, eFMU/FMUs models or functions etc., only the function values could be queried and the derivative information may not be available for solving the pertinent NLP problem. Thus, it is important to address the nonlinear optimization problems for both these

cases which appears in several real-world applications. Typically, all the methods utilized to solve the NLP problem (2.4) are iterative in nature and the solution converges either to the optimal or suboptimal values depending upon various factors such as accuracy, relative tolerance, number of iterations etc., which are the tuning parameters for the NLP solver.

2.2.2 Classification of nonlinear optimization problems

As mentioned previously, in this thesis the prime focus lies on nonlinear optimization problems with the assumption that the involved functions are continuous and smooth in nature. Under this condition, the whole domain of NLP could be broadly classified into convex and non-convex problems. The key property of a convex problem is that the local optimizer is the global optimizer, thereby the obtained solution is the global optimimum for the optimization problem. Convex optimization involves optimization of convex functions over convex sets [Boyd, Boyd, and Vandenberghe 2004]. Most of the convex problems could be broadly classified into the following classes: Linear Programming (LP), Quadratic programming (QP), Quadratically Constrained Quadratic Programming (QCQP), Second Order Conic Program (SOCP), Semi-Definite Programming (SDP) and Conic Program (CP) as illustrated in Fig. 2.1. It is also important to note that each of the aforementioned convex problems form a hierarchy chain wherein each class is a subset of another under specific assumptions on the structure of the objective and constraints set i.e. LP $\subset$ QP $\subset$ QCQP $\subset$ SOCP $\subset$ SDP $\subset$ CP. In spite of the hierarchy in this classification, the methods involved for solving these convex problems are compositionally distinct from each other as the individual solvers for each of these classes exploit the structure of the functions or constraints set in a different way in order to compute the solution efficiently within a fewer iterations. It is also important to note that LP is a subclass of convex NLP where the objective and constraints sets are linear and polytopic in nature and the rest follows suit.

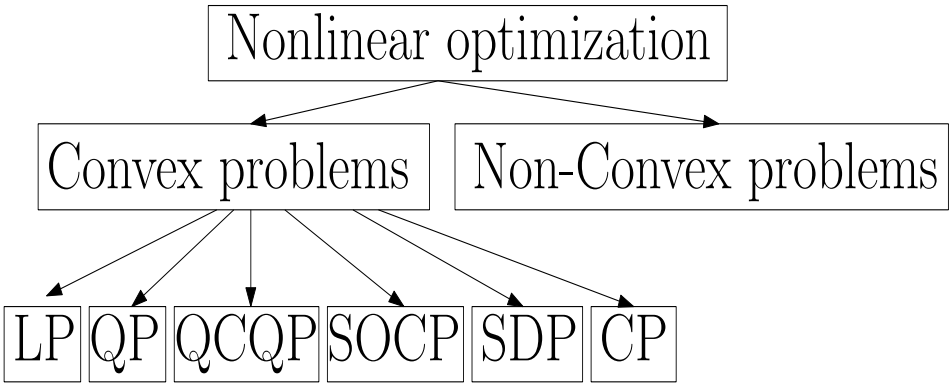

Figure 2.1: Classification of nonlinear optimization problems

The problems which are not convex in nature are deemed as non-convex problems, therefore by definition only a local minimizer could be obtained and there isn't any guarantee or method to establish the global optimum of the obtained solution. There exists variety of methods for solving both convex and non-convex problems, however in a broader sense the methods

could be classified as derivative based methods and derivative free methods. The derivative based optimization methods relies upon the zeroth order (function values) and first order (Jacobian)/second order (Hessian) derivative information of the functions involved in the NLP problem. The derivative free methods relies only upon the zeroth order information i.e. only function values can be queried for a given argument list or input vector. Typically, it is always recommended to utilize the derivative information whenever it is available, however, in situations where the details of the model or the functions are concealed from the end user due to security reasons or privacy reasons or protection of intellectual property rights, then it becomes highly impractical and cumbersome to utilize derivative based optimization methods. Thus, the natural recourse is to resort to derivative free methods for solving NLP problems.

A brief outlook into derivative based and derivative free methods are provided in the following sections. It is important to note that the primary intention of this chapter is only to provide an overview and not delve into the implementation details of any of the methods. Nevertheless, appropriate references are cited to benefit the reader to study more in-depth on these topics.

2.2.3 Derivative based methods

The state of the art derivative based methods can further be classified into the following types

- First order derivative (Jacobian/gradient) based methods
- Second order derivative (Hessian) based methods
- Quasi-second order derivative based methods

First order derivative (Jacobian/gradient) based methods: Over the last few years, the popularity of gradient based methods has increased in several folds and the main reason for this sudden surge in interest is attributed to recent developments in the field of machine learning (ML) [Goodfellow et al. 2016]. When dealing with surfiet of training data, the first order derivative based methods provide faster and better solutions for large scale ML problems. There exists variety of gradient based methods such as classic gradient descent, batch gradient descent, stochastic gradient descent, gradient descent with momentum, Nesterov accelerated gradient (NAG), adagrad, adadelta, RMSprop, adams, adamax etc. The survey paper [Ruder 2016] provides a quick glimpse into the several gradient based methods utilized for solving NLP problems in the ML community. However, the aforementioned gradient based methods are mostly designed for unconstrained NLP problems. In optimal control, one is obliged to include the constraints into the NLP problem formulation. Some of the constrained gradient based methods include proximal gradient methods [Parikh and Boyd 2014], Alternating Direction Method of Multipliers (ADMM) [Boyd, Parikh, and Chu 2011], operator splitting methods [Stathopoulos et al. 2016] etc. and also, several variants of these methods. These are commonly adopted for online optimization in optimal control community. The prime benefit of first order method is, it is not computationally taxing compared to other methods, however, the major

downside is that the convergence rate is either superlinear or linear, which is slow when compared to second or quasi-second order methods.

There are several first order derivative based solvers for solving the unconstrained NLP problems and these are predominantly utilized in the ML community [Ruder 2016]. However, for the constrained case and as well as in the context of optimal control and RT implementation, this is currently in its inchoate state and there are only a handful of promising solvers such as OSQP [Stellato et al. 2020], FiOrDOs [Ullmann 2011a], TFOCS [Becker, Candes, and Grant 2011], QPgen [Giselsson 2018], GPAD [Patrinos and Bemporad 2012], POGS [Fougner and Boyd 2018], SCS [O'donoghue et al. 2016], FORCES PRO [Zanelli et al. 2020]. For more details into the inner workings of first order methods, refer [Beck 2017].

Second order derivative (Hessian) based methods: The whole class of second order derivative methods relies on multiple variants of the classic Newton's method for solving the roots of non-linear equations. In short, the NLP formulation in (2.4) is converted to a set of nonlinear equations by means of Karush–Kuhn–Tucker (KKT) conditions and at every Newton iteration of the NLP problem, a linear algebra subroutine is solved and this is repeated until convergence to the optimal solution. Despite the simplicity of the above explanation, the implementation methods are quite involved and requires special techniques such as matrix pre-conditioning, determining the right step size, computational complexity etc. [Nocedal and Wright 2006]. The most commonly used second order derivative based methods for solving NLP problems are a) Sequential Quadratic Programming (SQP) methods and b) Interior-Point (IPM) methods. The SQP method can be further classified based on its working principle which are a) SQP Active Set (AS) methods and b) SQP Interior Point Quadratic Programming (IPM-QP) methods. A detailed explanation on the aforementioned methods are discussed in [Bazaraa, Sherali, and Shetty 2013]. Typically, second order methods possesses quadratic rate of convergence and the optimal solution is generally obtained within a couple of iterations. Despite the fast convergence of the second order methods, one of the main issues is the demand for high computational need to compute the Hessians and this doesn't scale well for large scale problems. However, in optimal control, the structure of the NLP problem is exploited due to the specialized functional form of the objective and constraints functions and the Hessians are pre-computed and either cached or stored offline (the Hessians are mostly sparse matrices). This provides a significant boost for RT implementation of optimal control problems in embedded systems.

Quasi-second order derivative based methods: As mentioned previously, one of the major issues with second order methods is the high computational complexity involved in computing the Hessian matrices. The conventional methods used to compute the derivatives for Hessians are finite differences method and algorithmic differentiation. However, the finite differences method is highly prone to numerical inaccuracies and algorithmic differentiation may not be possible for implementation for large scale NLP problems. Quasi-second order methods set a middle ground by retaining the computational complexity of first order methods and convergence property of second order methods [Dennis and Moré 1977]. The quasi-second order methods approximates the Hessians using the zeroth order information and first order derivative (Jacobians) information and these approximations are plugged into

the IPM or SQP solver. The approximation is improved at every Newton iteration and this procedure is repeated until convergence to optimal solution for the NLP problem. Some of the well known Hessian update rules under the quasi-second order methods are Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm, Broyden's rank-1 update and family of its method, Davidon–Fletcher–Powell (DFP) algorithm, Symmetric rank-one methods etc. A detailed exposition into these methods are provided in [Nocedal and Wright 2006].

The list of commonly used second order and quasi-second order NLP solvers are listed in Table 2.1. For more details on the classification of the NLP solvers, refer [Leyffer and Mahajan 2010]. The references for the respective solvers are: CVXOPT [Vandenberghe 2010], IPOPT [Wächter and Biegler 2006a], KNITRO [Byrd, Nocedal, and Waltz 2006], LOQO [Vanderbei 1999], CONOPT [Drud 1994], FilterSQP [Fletcher and Leyffer 1998], LINDO [Lin and Schrage 2009], LRAMBO [Griewank, Walther, and Korzec 2007], NLPQLP [Schittkowski 2006], NPSOL [Gill et al. 1986], SNOPT [Gill, Murray, and Saunders 2005], SQPlab [Gilbert 2009], qpOASES [Ferreau et al. 2014a], FORCES PRO [Domahidi and Jerez 2014].

Table 2.1: Second and quasi-second order NLP solvers

NLP solver	Method	Interfaces	Programming language
CVXOPT	IPM	Python	Python
IPOPT	IPM	AMPL, CUTEr, C, C++, f77	C++
KNITRO	IPM SQP	AIMMS, AMPL, GAMS Mathematica, MATLAB, MPL C, C++, f77, Java, Excel	C++
LOQO	IPM	AMPL, C, MATLAB	C
CONOPT	SQP	AIMMS, GAMS	Fortran
FilterSQP	SQP	AMPL, CUTEr, f77	Fortran77
LINDO	SQP	C, MATLAB, LINGO	Proprietary S/W
LRAMBO	SQP	C	C/C++
NLPQLP	SQP	C,f77,MATLAB	Fortran77
NPSOL	SQP	AIMMS, AMPL, GAMS MATLAB, C, C++, f77	Fortran77
SNOPT	SQP	AIMMS, AMPL, GAMS MATLAB, C, C++, f77	Fortran77
SQPlab	SQP	MATLAB	MATLAB
fmincon	IPM SQP	MATLAB	MATLAB
qpOASES	SQP	MATLAB/Simulink, Octave, SCILAB Python	C++
FORCES PRO	IPM	MATLAB, Python	C/C++

2.2.4 Derivative free methods

Derivative free methods can be broadly classified into a) direct search methods, b) model-based methods, c) heuristics based approach and d) randomized algorithms. A detailed exposition into derivative free methods are propounded in the following literatures [Nocedal and Wright 2006], [Beck 2014], [Larson, Menickelly, and Wild 2019], [Conn, Scheinberg, and Vicente 2009]. A comparative study of several state of the art derivative free optimization methods are presented in [Pham, Malinowski, and Bartczak 2011], [Moré and Wild 2009], [Powell 2007]. It is also important to note that the field of derivative free optimzation is not as mature as derivative based methods and most of the research in the past have been conducted only for the unconstrained case. Only a few methods exists for constrained case and this field is still being researched for new directions. Table 2.2 lists the commonly used derivative free optimization solvers, for more details refer [Rios and Sahinidis 2013], [Custódio, Scheinberg, and Nunes Vicente 2017]. The references for the respective solvers are: ASA [Ingber 1989], BOBYQA [Powell 2009], CMA-ES [Hansen, Müller, and Koumoutsakos 2003], COBYLA [Powell 1994], HOPSPACK [Plantenga 2009], IMFIL [Kelley 2011], DAKOTA [Adams et al. 2009], DFO [Conn, Scheinberg, and Toint 1998], NEWUOA [Powell 2006], NOMAD [Le Digabel 2011], PSwarm [Vaz and Vicente 2009], SID-PSM [Custodio and Vicente 2008], SNOBFIT [Huyer and Neumaier 2008] , TOMLAB [Holmström, Göran, and Edvall 2010], GLODS [Custódio and Madeira 2015], PDF-MPC package [Alamir 2017].

2.3 Fundamentals of Nonlinear Model Predictive Control

2.3.1 Introduction

The general formulation for a non-linear optimal control problem (OCP) is described by the following form

$$\begin{aligned} \min_{x(.),u(.)} \quad & \int_0^T L(x(t),u(t))\,dt + \psi(x(T)) \\ \text{subject to} \quad & \dot{x} = f(x(t),u(t)),\ \forall t \in [0,T] \\ & g(x(t),u(t)) \leq 0,\ \forall t \in [0,T] \\ & h(x(t),u(t)) = 0,\ \forall t \in [0,T] \\ & x(0) = x_0,\ r(x(T)) \leq 0 \end{aligned} \tag{2.5}$$

where $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$ represents the state vector and input vector respectively, $L : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}$ and $\psi : \mathbb{R}^n \mapsto \mathbb{R}$ represents the stage cost and terminal cost respectively, $f : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$, $g : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^p$, $h : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^k$ represents the dynamics of the system (Ordinary Differential Equations (ODEs)/Differential Algebraic Equations (DAEs)), inequality and equality constraints respectively. x_0 and $r : \mathbb{R}^n \mapsto \mathbb{R}^r$ represents the initial and final state conditions (terminal set) on the state variable. The inequality constraints

Table 2.2: Derivative free optimization solvers

Solver	Constraints	Bounds	Programming language
ASA	no	required	C
BOBYQA	no	required	Fortran
CMA-ES	no	optional	MATLAB
COBYLA	yes	required	Fortran
DAKOTA/DIRECT	yes	required	C++
DAKOTA/EA	yes	required	C++
DAKOTA/PATTERN	yes	required	C++
DAKOTA/SOLIS-WETS	yes	required	C++
DFO	yes	required	Fortran
fminsearch	no	no	MATLAB
GLOBAL	no	required	MATLAB
GLODS	yes	required	MATLAB
HOPSPACK	yes	optional	C++
IMFIL	yes	required	MATLAB
MCS	no	required	MATLAB
NEWUOA	no	no	Fortran
NOMAD	yes	optional	MATLAB
PSWARM	yes	required	MATLAB
SID-PSM	yes	optional	MATLAB
SNOBFIT	no	required	MATLAB
TOMLAB/GLCCLUSTER	yes	required	MATLAB
TOMLAB/LGO	yes	required	MATLAB
TOMLAB/MULTIMIN	yes	required	MATLAB
TOMLAB/OQNLP	yes	required	MATLAB
PDF-MPC Package	yes	required	MATLAB

arises from the physical, performance or path constraints imposed on the system and the equality constraints arises from ODE/DAE equations, point constraints and mostly are added for merely the convenience of solving the nonlinear OCP problem via NLP solvers. It is important to note that the non-linear OCP (2.5) is an infinite dimensional problem. Nonlinear model predictive control (NMPC) implements the non-linear OCP (2.5) in practice by means of receding horizon control technique over the prediction horizon T. The optimization problem in (2.5) is solved at every sampling period (τ) and only the optimal input over one sampling period is injected into the system i.e. $u^*([0,\tau])$. In short, NMPC is non-linear OCP put in practice and it is also important to note that an implicit feedback is formed in the NMPC problem formulation as with receipt of new state information at every sampling period, the inputs are recomputed. Therefore, the optimal input is a function of the initial state as well ($u^*(x_0)$) [Maciejowski 2002].

Fundamentally, there are three methods (illustrated in Fig. 2.2) for solving the non-linear

OCP problem (2.5), which are

- Hamilton-Jacobi-Bellman (HJB) equation or Dynamic programming (DP)
- Direct methods or Direct transcription methods
- Pontryagin minimum/maximum principle (PMP)

In this chapter, only the direct methods are described with considerable details. However, for the sake of completeness, a brief overview on the HJB equation/DP and PMP methods are given below.

HJB equation or DP method is based on the principle of optimality where the non-linear OCP is solved from the terminal (final) cost to the initial cost in a backward fashion. The cost optimization method is divided into several stages (starting from terminal to initial stage costs) and the optimal input function or policy is computed for each stage recursively and finally, the optimal input policy is recovered for the initial stage and the initial state. It is important to note that HJB equation is used in the context of continuous time systems and DP method is used in the context of discrete time systems. The DP method in practice is typically solved by means of tabulation method and HJB equation is solved by solving a constrained Partial Differential Equation (PDE). The major downside of the method is it suffers from the curse of dimensionality and doesn't scale for higher dimensional systems [Bertsekas et al. 1995]. However, recently DP method has received considerable attention due to reinforcement and deep reinforcement learning community and the method in these fields are termed as approximate dynamic programming (ADP) or bootstrapping [Bertsekas 2019].

The Pontryagin minimum/maximum principle (PMP), also known as indirect method is based on the calculus of variations, where the objective is to minimize/maximize the Hamiltonian of the non-linear OCP. The method follows the philosophy of first optimize and then discretize and is solved by means of gradient methods or shooting methods or collocation methods. The method can treat large scale systems and also solve boundary value problems (BVPs). Some of the major downsides include instability of the nonlinear ODE obtained for the state and co-state equations, requirement of explicit expression for the input and singular arcs are difficult to deal with and also, only necessary conditions for local optimality is obtained [Liberzon 2011]. However, the method is mostly used in the field of aerospace engineering for computing the optimal orbit trajectory for satellites and spacecrafts. For more details, refer [Naidu 2002].

2.3.2 Direct methods

The direct methods or direct transcription methods transcribes the problem into a finite dimensional problem, which yields an approximate solution to non-linear OCP (2.5) by virtue of NLP solvers. Direct methods follow the philosophy of first discretize and then optimize. The direct methods are classified into a) Single shooting methods, b) Multiple shooting methods and c) Direct collocation methods [Rathai, Sename, and Alamir 2018]. Single shooting

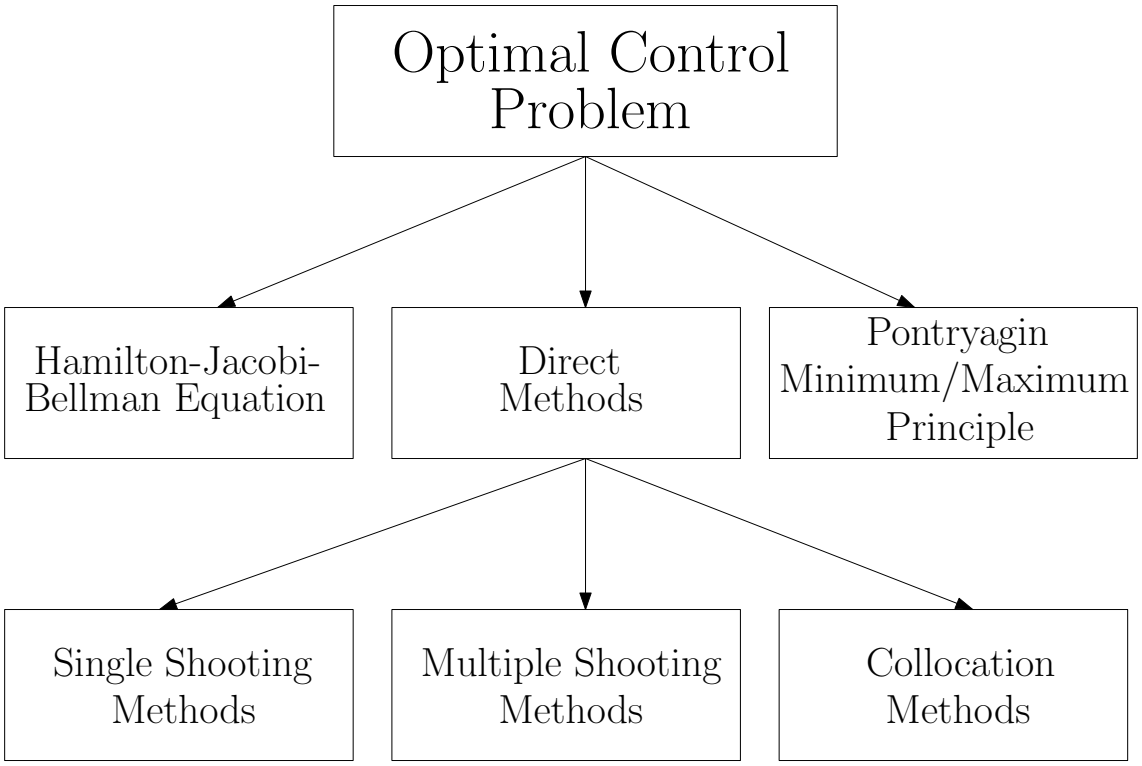

Figure 2.2: Classification of optimal control problems

methods falls under the category of sequential methods and multiple shooting and direct collocation methods falls under simultaneous methods. In order to set the transcription procedure, the following assumptions are taken into consideration [Alamir 2006], [Rawlings, Mayne, and Diehl 2017].

- The input $u(t)$, $\forall t \in [0, T]$ is finitely parameterized by piecewise constant vector set $\mathcal{U}$ at an integer multiple of the sampling period (τ) over the prediction horizon. With this representation, the input can be expressed with $u(t) = \mu(t; \mathcal{U})$, which is a piecewise continuous input signal.

- The dynamics (defined by the ODE/DAE in (2.5)) of the system is numerically simulated for the given input signal $u(t) = \mu(t; \mathcal{U})$, and the solution is compactly expressed with $x(t) = \phi(t; \mathcal{U}, x(0))$, which is evaluated at N discrete time instants $T_d = \{t_0, t_1, \ldots, t_N\} \subset [0, T]$. The time stamps T_d typically corresponds to the integer multiple of the sampling period and it is utilized to discretize the dynamics, objective function and the constraint functions listed in (2.5). A ODE/DAE solver is utilized to simulate the system over the prediction horizon [Ascher and Petzold 1998]. With the aforementioned assumptions, the non-linear OCP is transcribed to a generic NLP

framework, which is expressed with

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & F(w) \\ \text{subject to} \quad & G(w) \leq 0, \\ & H(w) = 0 \end{aligned} \tag{2.6}$$

where, w is the optimization variable which depends upon the direct method formulation, and $F(w)$, $G(w)$ and $H(w)$ represents the objective, inequality constraints and equality constraints respectively. The transcribed NLP in equation (2.6) solves the non-linear OCP (2.5) and this is the crux of direct methods.

Remark - The ODE/DAE solver mentioned in the assumption is to be considered as a computer code and not in terms of algebraic equations. The ODE/DAE solver is a simulator which takes the numerical input trajectory $u(t) = \mu(t;\mathcal{U})$ and outputs the numerical state trajectory which is utilized in the objective and constraint functions for the NLP problem mentioned in (2.6). Thus, the objective and the constraint functions is embedded with the ODE/DAE solver code and is ought to be deemed as computer codes (function code). Under conditions of twice differentiability of all the functions (codes) listed in (2.5), the Jacobians and Hessians for the NLP solver are numerically obtained by methods such as finite differences, algorithmic differentiation etc. [Griewank and Walther 2008] (also known as oracles in optimization parlance) and this information aids the optimization procedure.

2.3.2.1 Direct single shooting method

The direct single shooting method also known as sequential method eliminates the dynamics equality constraint in the non-linear OCP (2.5) by forward simulation and thus, removing the state variables from the OCP NMPC problem [Rathai, Sename, and Alamir 2018]. This reduces the optimization problem only to the input variables, which is obtained from the following NLP problem.

$$\begin{aligned} \min_{\mathcal{U}} \quad & \sum_{k=0}^{N} L(\phi(t_k;\mathcal{U},x(0)),\mu(t_k;\mathcal{U}))\,\Delta t + \psi(\phi(t_N;\mathcal{U},x(0))) \\ \text{subject to} \quad & g(\phi(t_k;\mathcal{U},x(0)),\mu(t_k;\mathcal{U})) \leq 0,\ \forall t_k \in T_d \\ & h(\phi(t_k;\mathcal{U},x(0)),\mu(t_k;\mathcal{U})) = 0,\ \forall t_k \in T_d \\ & r(\phi(t_N;\mathcal{U},x(0))) \leq 0 \end{aligned} \tag{2.7}$$

The objective is discretized by means of Riemann sum at time stamps T_d and Δt is the temporal difference between t_{k+1} and t_k, which usually coincides with the sampling period (τ). The states are replaced with the ODE/DAE simulator evaluated at these time stamps in the objective as well as the constraints. The optimal solution $\mathcal{U}^*$ obtained from (2.7) and the first input $\mathcal{U}^*(0)$ is injected into the system and the process is repeated in receding horizon manner. The prime benefit of the method is, it only require few degrees of freedom, however

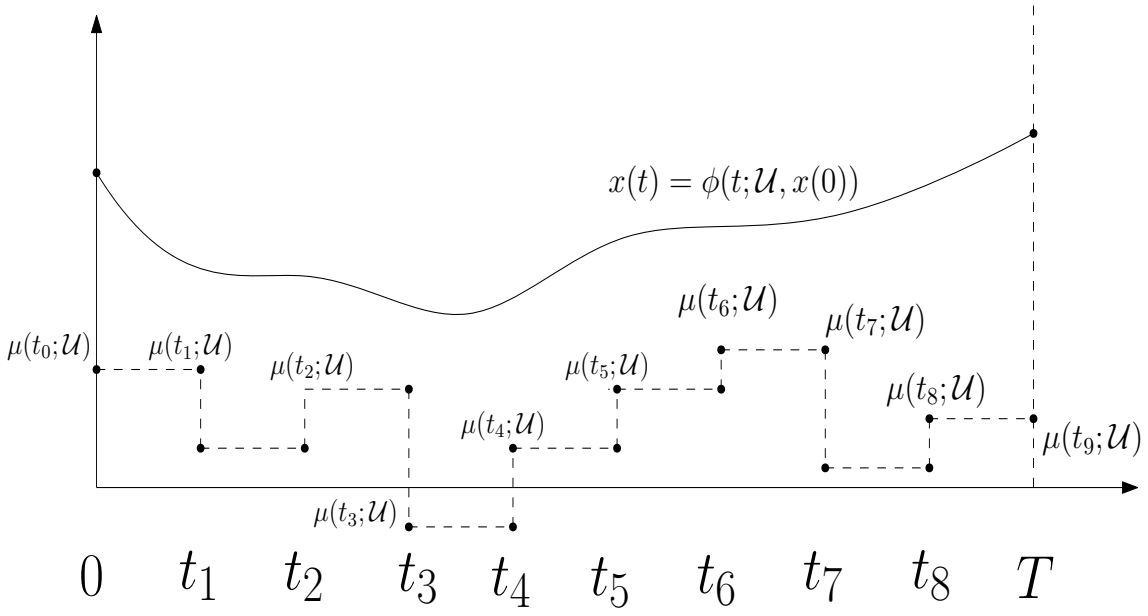

Figure 2.3: Direct single shooting illustration

the major downside of the method is, it performs poorly on unstable ODE/DAE systems and applies only for short prediction horizons. A pictorial illustration of the method is shown in Fig. 2.3. For more details, refer [Gros et al. 2020].

2.3.2.2 Direct multiple shooting method

The direct multiple shooting method, also a part of simultaneous methods, retains the state variables as optimization variables and this increases the number of decision variables in the optimization formulation in the non-linear OCP (2.5). The ODE/DAE solver simulates the system over multiple time intervals i.e. $[t_k, t_{k+1}]$, $\forall k = \{0, 1, \ldots, N-1\}$ simultaneously and the final state value $x(t_k)$ for the simulation in the each interval $[t_k, t_{k+1}]$ is stipulated to obey the dynamics of the system, which is enforced by equality constraints [Rathai, Sename, and Alamir 2018]. This method is implemented when the dynamics of the system (ODE/DAE equations) is numerically ill-conditioned and for larger prediction horizon. By dividing the prediction horizon into sub-intervals and simulating the system with respect to each sub-interval, the stability property of the simulation is retained. The NLP optimization problem is formulated as

$$
\begin{aligned}
\min_{\mathcal{U},\{x(t_1),\ldots x(t_N)\}} \quad & \sum_{k=0}^{N} L(x(t_k),\mu(t_k;\mathcal{U}))\,\Delta t + \psi(x(t_N)) \\
\text{subject to} \quad & g(x(t_k),\mu(t_k;\mathcal{U})) \leq 0,\ \forall t_k \in T_d \\
& h(x(t_k),\mu(t_k;\mathcal{U})) = 0,\ \forall t_k \in T_d \\
& x(t_{k+1}) - \phi(t_{k+1};x(t_k),\mu(t_k;\mathcal{U})) = 0,\ \forall t_k \in T_d \\
& x(0) = x_0,\ r(x(t_N)) \leq 0
\end{aligned}
\tag{2.8}
$$

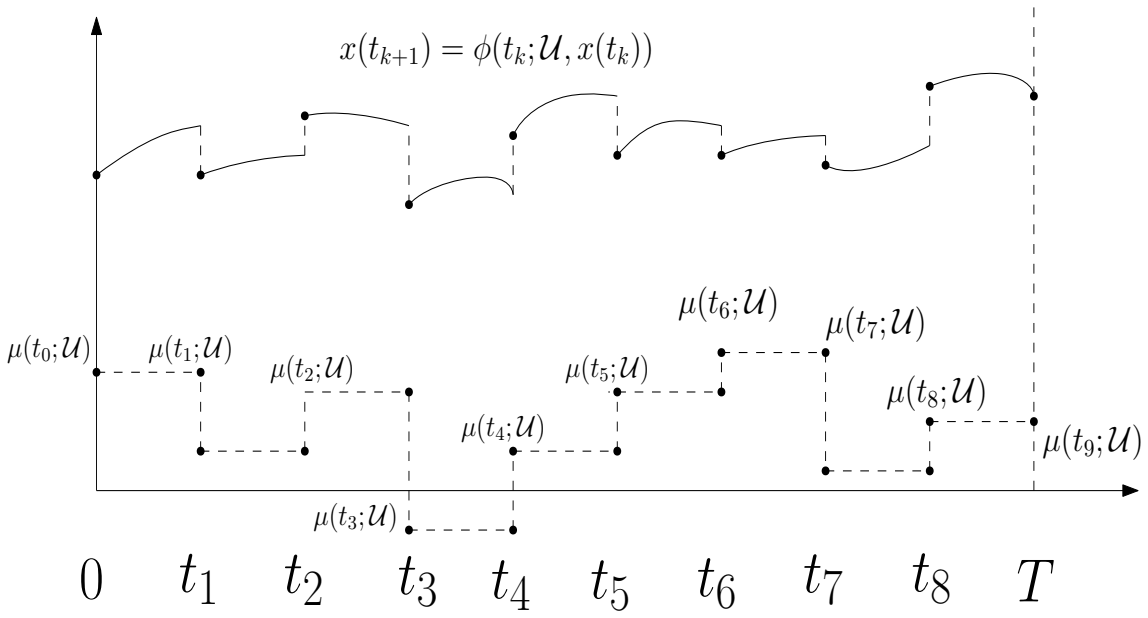

Figure 2.4: Direct multiple shooting illustration

The optimal solution for the above optimization problem yields both the optimal state trajectory and optimal input sequence. As per the standard receding horizon policy, the first input $\mathcal{U}^*(0)$ is applied to the system and this procedure is repeated in the future. The benefits of utilizing direct multiple shooting methods include a) Better simulator stability with unstable system, b) Parallelizability of ODE/DAE simulations, c) Structural properties of the Hessian matrices aid the optimization routine. The Jacobians and Hessians for the ODE/DAE simulator are obtained via either forward or adjoint sensitivity analysis methods However, the downside is that the optimization is carried out over an increased number of variables and a good initialization for the NLP solver is required for faster convergence to the optimal/suboptimal solution (this can be ameliorated by warm start procedure). A pictorial illustration of the method is shown in Fig. 2.4. For more details, refer [Gros et al. 2020].

2.3.2.3 Direct collocation method

Direct collocation methods is a part of simultaneous methods, where the ODE/DAE simulator is expunged from the multiple shooting formulation (2.8) and is replaced with algebraic equal-

ity constraints enforced at the collocation points [Rathai, Sename, and Alamir 2018]. The solution for the dynamics (ODE/DAE) system is typically represented by means of orthogonal polynomials such as Lagrange polynomials and the infinite ODE/DAE system is finitely discretized at the orthogonal collocation points defined by the polynomials. Therefore, the ODE/DAEs are gridded and transcribed into several equality constraints enforced at these collocation points and this method is also known as orthogonal collocation method. With this transformation, the NLP problem is expressed with

$$\begin{aligned} \min_{\mathcal{U},\{x(t_1),\dots x(t_N)\}} \quad & \sum_{k=0}^{N} L(x(t_k),\mu(t_k;\mathcal{U}))\,\Delta t + \psi(x(t_N)) \\ \text{subject to} \quad & g(x(t_k),\mu(t_k;\mathcal{U})) \leq 0,\ \forall t_k \in T_d \\ & h(x(t_k),\mu(t_k;\mathcal{U})) = 0,\ \forall t_k \in T_d \\ & \Psi(x(t_{k+1}),x(t_k),\mu(t_k;\mathcal{U})) = 0,\ \forall t_k \in T_d \\ & x(0) = x_0,\ r(x(t_N)) \leq 0 \end{aligned} \tag{2.9}$$

The fundamental difference between (2.8) and (2.9) is Ψ, which defines the dynamics of the system at collocation points and also, unlike multiple shooting method, there are no online simulations of the ODE/DAE system. Direct collocation methods are typically suited for stiff ODE/DAE systems. The benefits of direct collocation methods are a) suitable for large scale problems and also, the NLP is sparse in nature, b) suitable for unstable systems, however the downside is that the method is not adaptive for new grid and changes in NLP dimensions. For more details, refer [Gros et al. 2020].

2.4 A Gentle Introduction to GPGPU Computing for Control Engineers using CUDA Programming Paradigm

Part II
pNMPC with RT applications for control of semi-active suspension system

Chapter 3

Experimental implementation of pNMPC scheme for control for semi-active suspension system

Contents

3.1 Introduction

As mentioned previously in Chapter 1, suspension systems play a vital role for control of vertical dynamics of vehicles for guaranteeing comfort and safety for the on-board passengers. The seemingly simple task of control poses to be daunting under the presence of multiple non-linearities, physical constraints and specification over objective requirements for the system. Thereby, it is of paramount importance to account for these issues during control design for better efficiency and prolonged endurance of the suspension system. Along with these control design requirements, it is also important to address the issue under the ambit of the EMPHYSIS project [EMPHYSIS 2017], where the semi-active suspension system based vehicle models are exported from modeling tools such as AMESIM, OpenModelica, SimulationX etc. as eFMU containers. Typically, the eFMU encapsulates the model of the system either as C source codes or binary executables and therefore, the generated eFMU could be considered as a black box model. Under such circumstances, where the knowledge of the system is completely obscured from the control engineer, the control design problem becomes more complicated in nature. Predicated upon these requirements, in this thesis a simulation-optimization based control strategy is proposed namely parameterized NMPC (pNMPC) scheme to handle this control problem. The pNMPC controller is tested, verified and validated for the INOVE test platform model by presuming the availability of the knowledge of the system, so that the method is scalable and copes up with black box model oriented eFMUs for future use case.

MPC is indisputably one of the most advanced and efficient control design methodology. However, despite its enormous advantages in terms of optimal performance and constraint satisfaction, one of the major shortcoming is that the entire MPC controller hinges upon the model utilized in the control design. Due to the predictive nature of the controller, utilizing an erroneous model would ensue poor performance due to the mismatch of models between the plant and the controller. Thus, it is important to build a high fidelity model such that the mismatch is reduced and tangible performance benefits from the MPC controller are obtained. Despite the mathematical model could be a black-box model or white-box model, the accuracy of the model with respect to the ground truth play a pivotal role in determining the performance of the MPC controller. Given this prelude and in the same spirit, in this chapter, the first part addresses the problem of modeling and parameter identification for the ER semi-active suspension system for the INOVE test platform of a quarter car system. The obtained model could be conceptualized as a black-box model such as binary executables or eFMU/FMUs and the pNMPC scheme could be applied for this model. In the second part, the obtained model parameters are utilized for design and implementation of the proposed pNMPC scheme via HiL simulations on dSPACE MABXII and the INOVE test platform.

3.1.1 Related works

There have been several semi-active suspension system control design methods developed in the past such as Skyhook proposed in [Karnopp, Crosby, and Harwood 1974], Acceleration Driven Damping (ADD) proposed in [Savaresi, Bittanti, and Montiglio 2005], Mixed Skyhook-

ADD (SH-ADD) proposed in [Savaresi and Spelta 2007], LPV/H-∞ based control methods proposed in [Do, Sename, and Dugard 2010], [Sename et al. 2012]. A detailed literature review of different control strategies are presented in [Tseng and Hrovat 2015] and [Poussot-Vassal et al. 2011]. Despite the several mentioned control strategies provide good performance, these methods adopt the state and input constraints in an ad-hoc fashion and not completely into control design. In [Nguyen et al. 2016b], a robust control approach is applied by taking into account state and input constraints into control design, however, the method has several limitations and conservativeness. This exclusion of fully incorporating the constraints into control design might deteriorate the system performance and not fully utilize the potential of the semi-active damper system. To circumvent this problem, MPC based approach provides an elegant way of tackling the system constraints and objectives in control system design. Another key advantage of MPC based approach is the ability to incorporate future road information (road preview) into control design which could improve the performance of the system in many folds.

Over the last decade, there has been several research contributions on MPC based approach for control of semi-active suspension systems. In [Canale, Milanese, and Novara 2006], a Fast MPC method is proposed where the optimal control input is computed offline by means of set membership approximation technique, however the model can not incorporate dynamic information into problem formulation such as road profile, variation in system parameters etc. In Hybrid MPC approach proposed in [Giorgetti et al. 2006], the system is modeled as hybrid dynamical system and the optimal control input is computed offline by solving a multi-parametric program for a mixed-integer quadratic program (MIQP) and the method suffers from similar shortcoming as for Fast MPC method. In [Cseko, Kvasnica, and Lantos 2010], a detailed analysis of explicit MPC for semi-active suspension system is conducted. In [Gohrle et al. 2012], a preview information based MPC scheme is proposed for control of suspension system for a full car model and the model is presumed to be a LTI system. In [Nguyen et al. 2016a], MPC for semi-active suspension system is implemented for full car model, where a MIQP problem is solved online, however, the sampling period is too high for practical implementation. In [Morato et al. 2019], a fast RT Linear Parameter Varying (LPV) MPC scheme is proposed for control of semi-active suspension system for a full vehicle to overcome the computational issues in [Nguyen et al. 2016a] by modeling the system by means of LPV model.

3.1.2 Chapter contributions

The main contribution of this chapter are

- System identification and parameter estimation of ER semi-active damper system for the INOVE test platform is explained in detail along with the design of experiments. The obtained model parameters are utilized for implementation of the proposed pNMPC method for a quarter car system. It is also important to note that the estimated model and its parameters are utilized for all ER damper models throughout the thesis.

- The working principle of pNMPC controller is explained in detail along with its application for control of semi-active suspension system. The design requirements such as objectives, constraints and the pNMPC OCP formulation for the suspension control system are laid down firmly.

- The proposed method is implemented in HiL simulation on dSPACE MABX II and also, compared against a linearization based MPC using CVXGEN [Mattingley and Boyd 2012] solver. The proposed method is experimentally implemented on the INOVE test platform and the performance was compared against other controllers.

3.2 Control oriented ER semi-active damper modeling and parameter identification

In general, any semi-active damper modeling exercise can be broadly classified into a) parametric and b) non-parametric based approach. Under the former regime, the structure of the model is dictated by the physics of the system, which is modeled by means of first principles techniques and by contrast, the latter method obscures the underlying physics of the system and this leads to a flexible model structure, which is modeled by means of empirical techniques. [Butz and Von Stryk 2002] provides a detailed survey on different types of parametric/non-parametric modeling for semi-active suspension systems. In this work, the parametric method is of primary interest and some of the popular parametric models (to name a few) include the Bingam model [Stanway, Sproston, and Stevens 1987], the phenomenal Bouc-Wen model [Spencer Jr et al. 1997], the nonlinear viscoelastic-plastic model [Kamath and Wereley 1997], the nonlinear bi-viscous model [Stanway, Sproston, and El-Wahed 1996], Guo's damper model [Guo, Yang, and Pan 2006] etc. In this work, Guo's damper model is utilized to describe the ER semi-active suspension system for the INOVE test platform due to its simplicity and parsimonity to describe the damper characteristics.

3.2.1 Vehicle modeling - Quarter car model

The vertical dynamics model for the quarter car system equipped with ER semi-active damper system (around the equilibrium) is defined with the same quarter car model described in Chapter 1, Section 1.1 which is defined by

$$\begin{aligned} m_s\ddot{z}_s &= -k_s(z_s - z_{us}) - u \\ m_{us}\ddot{z}_{us} &= k_s(z_s - z_{us}) + u - k_t(z_{us} - z_r) \end{aligned} \tag{3.1}$$

where, m_s, m_{us} are the sprung mass and unsprung mass respectively, k_s, k_{us} are the stiffness coefficients of suspension system and the tire respectively, z_s, $\dot{z}_s$ are the sprung mass position and velocity respectively, z_{us}, $\dot{z}_{us}$ are the unsprung mass position and velocity

respectively, z_r is the vertical road position or disturbance and u is the force exerted due to the ER semi-active damper system. The state vector is represented with $\mathbf{x} = [z_s\ z_{us}\ \dot{z}_s\ \dot{z}_{us}]^T$.

3.3 Quasi-static nonlinear ER damper model

The ER semi-active damper force u is expressed using the quasi-static nonlinear damper model (Guo's damper model) [Guo, Yang, and Pan 2006] with

$$u = f_c\phi\tanh(a_1\dot{z}_{def} + a_2 z_{def}) + c_0\dot{z}_{def} \tag{3.2}$$

where, c_0, f_c, a_1 and a_2 represents the viscous damping coefficient, dynamic yield force of the fluid, and hysteresis coefficient of the damper model respectively. $z_{def} = z_s - z_{us}$ and $\dot{z}_{def} = \dot{z}_s - \dot{z}_{us}$ represents the deflection position and velocity $\forall i \in \{l, r\}$ between the chassis and wheel respectively. ϕ is the PWM-DC input for the system such that $\phi \in \mathbb{U}$, where $\mathbb{U} := [\phi_{min}, \phi_{max}]$ such that $0 \leq \phi_{min} < \phi_{max} \leq 1$. For reasons of safety and operational requirements, the minimum and maximum bounds for the PWM-DC signal were set to $\phi_{min} = 0.1$ and $\phi_{max} = 0.35$.

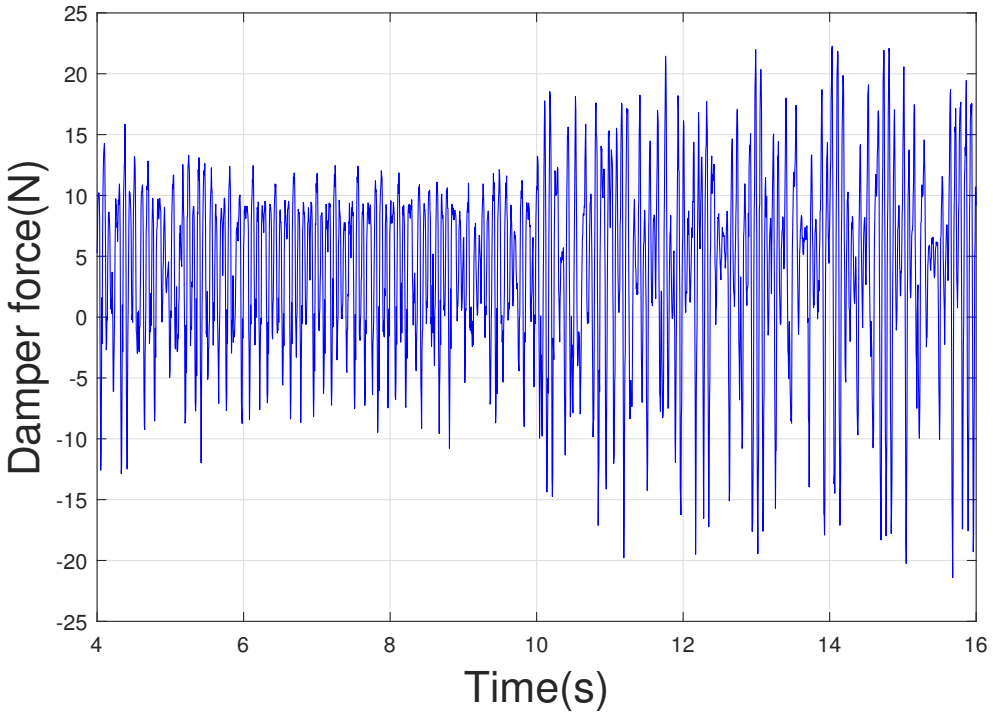

Figure 3.1: Measured damper force (u) for PRBS based road profile

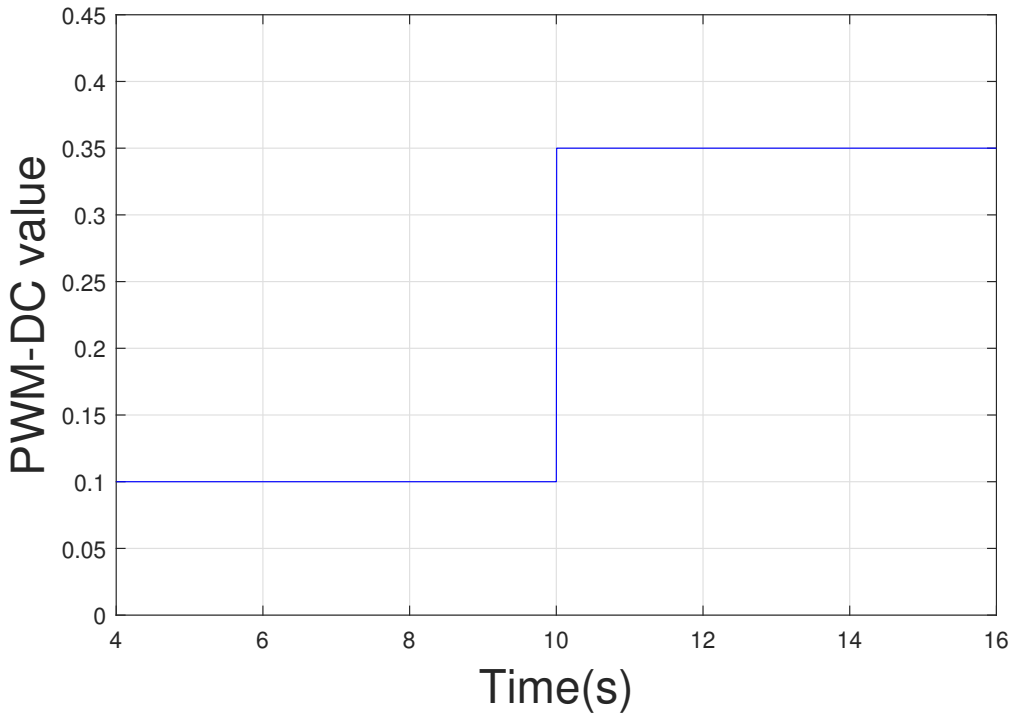

Figure 3.2: PWM-DC signal (ϕ)

3.4 Parameter estimation

3.4.1 ER semi-active damper response time estimation

To study the dynamic characteristics of the ER-SA damper system, it is important to estimate the response time of the system, which is indirectly estimated by finding the peak response time of the system (T_r). The following experiment was conducted to estimate T_r:

- The platform was excited with a pseudo binary random sequence (PRBS) road profile with an amplitude of 5 mm for a duration of 20 s, illustrated in Fig. 3.1.
- The PWM-DC signal was flipped from ϕ_{min} to ϕ_{max} at time 10 s (a step change in input) and the ER-SA damper force (u) was measured, illustrated in Fig. 3.2.

In order to zero-in the point of transition (high frequency content), which provides the necessary cues for T_r estimation, time-frequency analysis was performed by means of wavelet transform. Fig. 3.3 illustrates the wavelet analysis performed on the damper force signal (u) with Morlet wavelet basis functions [Tangirala, Mukhopadhyay, and Tiwari 2013]. It is evident from the contour plot that the peak in the frequency and energy content at time 10.23s, provides the necessary cue to approximate the peak response time, i.e. $T_r \approx 230\ ms$. This inference serves three purposes which are

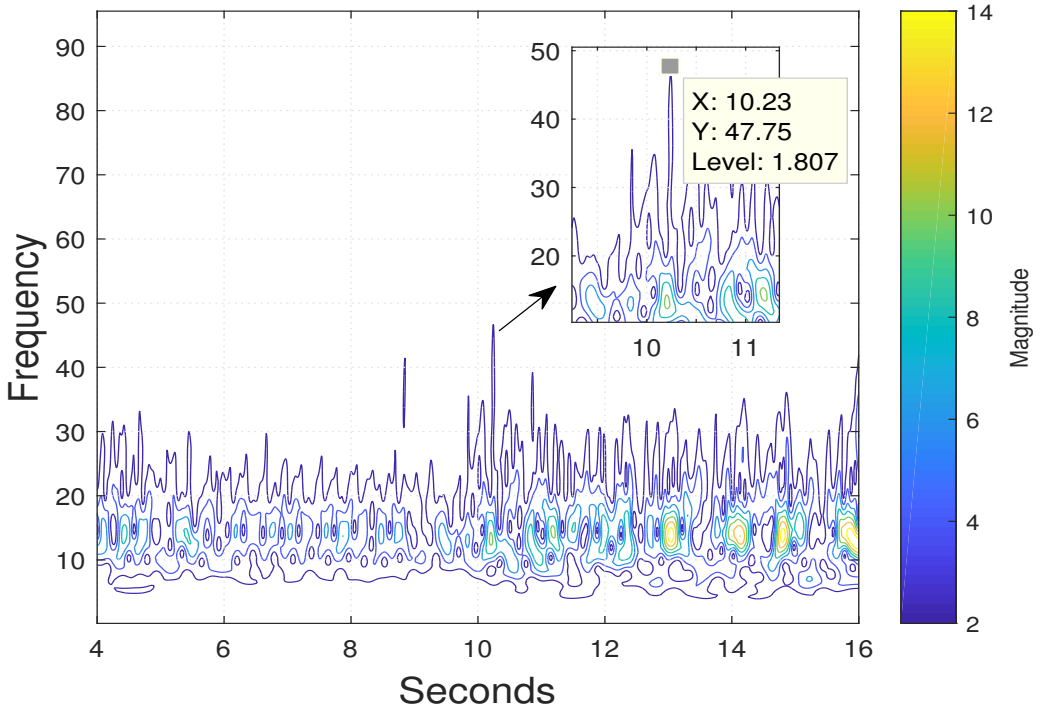

Figure 3.3: Wavelet analysis of ER semi-active damper force (u) signal

- The at-most period for PWM-DC transition to completely capture the dynamical behavior of the ER-SA damper system is ascertained.
- The look ahead period or the prediction horizon for the MPC controller is computed.
- Estimation of sampling time (T_s) for the damper system, which is computed using the general measure with $T_s \in [\frac{T_r}{10}, \frac{T_r}{5}]$ is computed [Astrom and Wittenmark 1982]. This also accounts for all the delays in the system i.e. sensors and actuator delays. However, the natural sampling time of the system is 5 ms.

3.4.2 Design of experiments

In order to obtain the best model parameters for the ER semi-active damper system, it is imperative to conduct informative experiments and collect the input/output data that captures the dynamic behavior of the system. Conditioned upon the previous requirement, the test involved the following scenario:

- PRBS signal based road excitation with an amplitude of 5 mm for a duration of 20 s.
- A PRBS based input PWM-DC signal between the interval $[\phi_{min}, \phi_{max}]$ with a holding period of T_r.

The rationale to adopt this scenario is to induce persistent excitation and minimize the crest factor for input design [Ljung 1987]. The ER semi-active damper system was operated upon the aforementioned scenario and all the input/output data were collected for parameter estimation stage. For the purpose of illustration, Fig. 3.4 and Fig. 3.5 illustrates the force vs deflection position and velocity respectively for selected PWM-DC values which are $\{0.1, 0.21, 0.27, 0.35\}$.

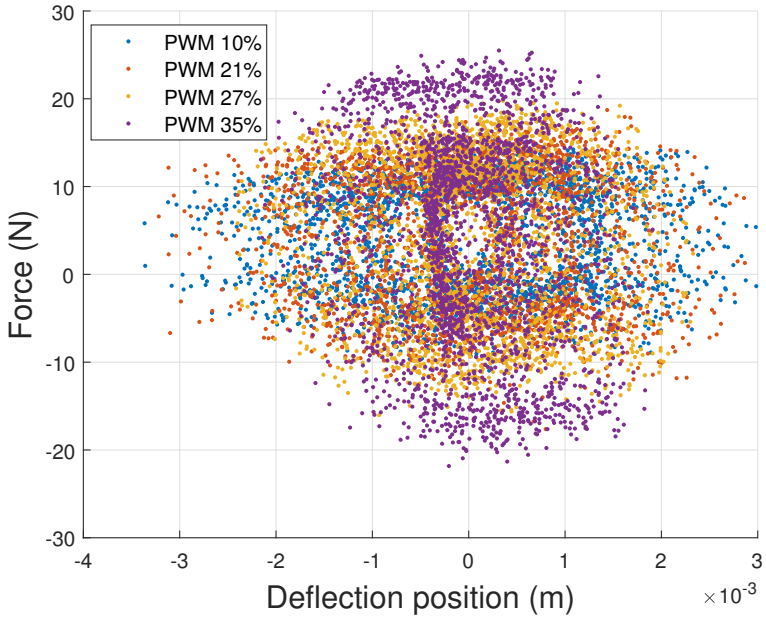

Figure 3.4: F_{ER} vs z_{def} plot for different PWM-DC signals

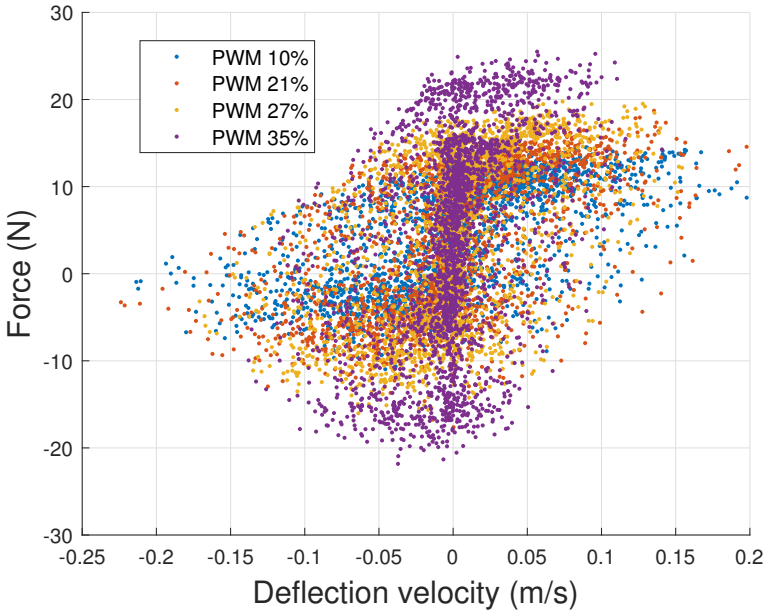

Figure 3.5: F_{ER} vs $\dot{z}_{def}$ plot for different PWM-DC signals

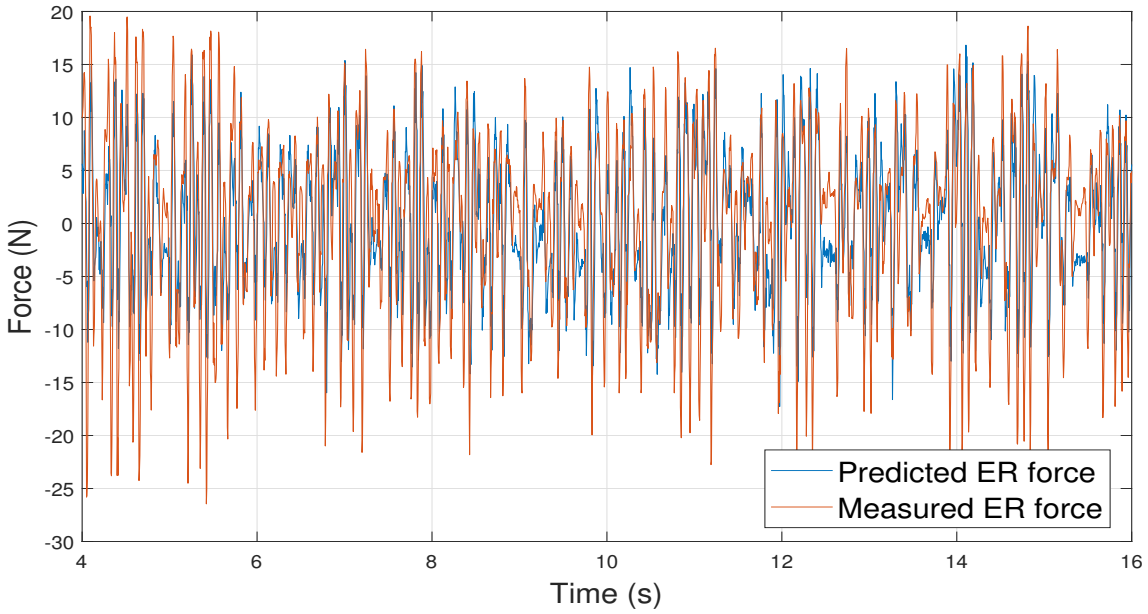

Figure 3.6: Predicted damper force and measured damper force u

3.4.3 Non-linear least squares (NLS) based data fitting

The input/output N-sample dataset are expressed with $\{\mathcal{D}^i_{\mathcal{X}}\}_{i=1}^N$ and $\{\mathcal{D}^i_{\mathcal{Y}}\}_{i=1}^N$, where $\mathcal{X} = [z_{def}\ \dot{z}_{def}\ \phi]$ and $\mathcal{Y} = u$. Let the unknown parameters be represented with $\theta = [f_c\ a_1\ a_2\ c_0]$, then the NLS objective function is defined with

$$\chi^2(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(\mathcal{D}^i_{\mathcal{Y}} - \hat{\psi}(\mathcal{D}^i_{\mathcal{X}}, \theta)\right)^2 \tag{3.3}$$

where, $\hat{\psi}$ is the estimated function for the data fitting problem for the output dataset, which in this case is the ER semi-active suspension force, i.e. u. The optimal parameters are computed by solving the following nonlinear optimization problem

$$\theta^* = \underset{\theta\in\Theta}{\operatorname{argmin}}\ \chi^2(\theta) \tag{3.4}$$

where, $\Theta \subset \mathbb{R}^4$ is the constraint set for the parameters, which is defined with $\Theta := \{\theta \in \mathbb{R}^4 \mid \{\theta_1, \theta_4\} \in \mathbb{R}_{\geq 0},\ \{\theta_2, \theta_3\} \in \mathbb{R}\}$. The estimated model was validated using K-fold cross validation method with $K = 5$ and the accuracy of the model was estimated to 4.65 units. The estimated model parameters are listed in the Table 3.1. Fig. 3.6 illustrates the predicted vs measured ER semi-active damper force (u) for a single track of input/output data.

Table 3.1: Estimated ER semi-active damper parameters

Parameter	Symbol	Value (SI unit)
Force parameter	f_c	$21.38(N)$
Deflection position parameter	a_1	$178.93(1/m)$
Deflection velocity parameter	a_2	$23.21(s/m)$
Nominal damping coefficient	c_0	$71.03(Ns/m)$

3.5 pNMPC design requirements for semi-active suspension system

3.5.1 Objective requirements

The dichotomy of the objective design for the semi-active suspension system for a quarter car system could be both qualitatively and quantitatively classified into a) Comfort objective and b) Road Holding objective [Savaresi et al. 2010].

- Comfort objective (J_t^{com}): Qualitatively, the prime goal of the comfort based objective design is to guarantee the comfort for the on-board passengers. The human body is sensitive to certain frequencies and it is of paramount importance to mitigate the effects of vibrations at these spots of the spectrum. Quantitatively, this tantamount to minimizing the vertical acceleration of the chassis ($\ddot{z}_s$). The comfort objective for the given look ahead period T_l is expressed as

$$J_{T_l}^{com} = \int_0^{T_l} (\ddot{z}_s(t))^2 dt \tag{3.5}$$

- Road holding objective (J_t^{rh}): Qualitatively, the prime goal of the road holding based objective design is to guarantee that the wheel is always in contact with the road. The requirement of this objective is crucial in control of longitudinal and lateral dynamics of the vehicle. Quantitatively, this objective corresponds to the requirement of minimizing the displacement between the road and the wheel ($z_{us} - z_r$). The road holding objective for the given look ahead period T_l is expressed as

$$J_{T_l}^{rh} = \int_0^{T_l} (z_{us}(t) - z_r(t))^2 dt \tag{3.6}$$

It is also important to note that both the objectives are conflicting in nature. Thus, the objective for the semi-active suspension system holistically at the current time instant t is expressed as

$$J_{T_l}^{obj} = \theta_1 J_{T_l}^{com} + \theta_2 J_{T_l}^{rh} \tag{3.7}$$

Where, θ_1 and θ_2 are the weighting coefficients between comfort and road holding objective and also, the coefficients form a convex combination between the two objectives such that $\theta_1 + \theta_2 = 1$ and $\theta_1, \theta_2 \geq 0$.

3.5.2 Constraint requirements

The constraints for the semi-active suspension system primarily arises from the physical limitations of the system. These are hard constraints and must be handled systematically to prevent weariness of the system components. For the pNMPC design considered, six constraints are included in the problem formulation which are

- Semi-active ER damper input constraints:
  - PWM input constraint: $\phi(t) \in \mathbb{U}$.
  - Max/Min damper force constraint: This forms a non-linear mixed state-input constraint such that $u(t) \in [\underline{u}, \overline{u}]$, where $\underline{u}$ and $\overline{u}$ are the minimum and maximum saturation forces for the semi-active suspension system.
- State limitations constraints:
  - Max/Min deflection between the chassis and wheel position: This forms a linear state constraint such that $z_s - z_{us} \in [z_{def}^{min}, z_{def}^{max}]$, where $z_{def}^{min}, z_{def}^{max}$ are the minimum/maximum deflection position between the chassis and the wheel.
  - Max/Min deflection between the chassis and wheel velocity: This forms a linear state constraint such that $\dot{z}_s - \dot{z}_{us} \in [\dot{z}_{def}^{min}, \dot{z}_{def}^{max}]$, where $\dot{z}_{def}^{min}, \dot{z}_{def}^{max}$ are the minimum/maximum deflection velocity between the chassis and the wheel.
- Dynamics constraint: The nonlinear equality constraints due to dynamics of the system defined in (3.1) and (3.2). It is important to note that this is eradicated via simulation aka direct single shooting method.
- Road disturbance assumption: Under the consideration without road preview information, it is not uncommon to presume a constant road disturbance input measured at the current time instant t for the entire future horizon for the NMPC problem, i.e. $d^+ = d(t)$.

3.6 Parameterized NMPC

The proposed parametrized NMPC approach is based on simulation methods, i.e. an explicit/implicit ODE solver is utilized to simulate the non-linear system in equation (3.1), (3.2) to determine the evolution of the states for a set of input sequences over the horizon. In the context of eFMI/FMU, the underlying system could be treated as black-box models and the solvers could include either Ordinary Differential Equations (ODEs) or Differential Algebraic

Equations (DAEs) solvers [Ascher and Petzold 1998], which could either be provided by the simulation environment or inbuilt within the eFMU/FMU containers. The optimal input sequence is elicited from the simulations which minimizes the objective function and satisfies the constraint requirements, which are handled algorithmically [Rathai et al. 2018]. The proposed parametrized NMPC algorithm is sequentially presented as follows

Algorithm:

1. The input ϕ of the non-linear system in equation (3.1), (3.2) is finitely parameterized in time with N_δ equidistant points over the look ahead period T_l with $\{\delta_0 \dots \delta_{N_\delta - 1}\}$ time stamps with an interval of $\frac{T_l}{N_\delta}$ and $T_l = \delta_{N_\delta - 1}$ and in space, the set $\mathbb{U}$ is discretized with N_s points such that $\phi \in \{\phi_1, \dots \phi_{N_s}\} \subset \mathbb{U}$, where ϕ_i is a discretization point in $\mathbb{U}$. The input sequence over the horizon is compactly represented with $\mu(\delta_j | \{\phi_i(\delta_j)\}_{i=1}^{N_s}, t)$, $\forall j \in \{0, \dots N_\delta - 1\}$, i.e. at a given time instant δ_j, there exists N_s possible input values and this spans for all given time stamps.

2. The explicit/implicit ODE solver for the non-linear system in equation (3.1), (3.2) is simulated for all input sequences along space and time.

3. The optimal control sequence is computed with respect to the objective and constraints by plugging the simulated trajectory onto the cost function and the constraint functions. The constraints are handled algorithmically that if a particular input sequence violates the constraints, then the input sequence is discarded and the solver is proceeded with another control sequence until the minimum cost is obtained.

4. In case, if no input sequence satisfies the constraints, then the input sequence which least violates the constraints is considered as the optimal input sequence.

5. This procedure is repeated in receding horizon policy method at every sampling period (T_s) and the optimal control input is $\phi^*(0) = \phi^*(\delta_0)$.

For the considered case of quarter car semi-active suspension system, N_s is assumed as variable (space discretization) and $N_\delta = 1$ (time discretization) and the solver utilized is a simple fourth order explicit Runge-Kutta (RK) method with fixed integration step $h = 1\ ms$. The model parameters for INOVE quarter car platform and proposed MPC design are listed in Table 3.2. It is important to note that the ascertained sampling period from the system identification tests is $T_{sest} = 0.023$ and the natural sampling period of the system or the sampling period of the DAQ is $T_s = 0.005$. For experiments on the test bench the former was used and for tests on dSPACE MABXII the later was used.

Table 3.2: Model parameters for INOVE quarter car platform and proposed MPC design

Parameter	Symbol	Value (SI unit)
Chassis quarter car mass	m_s	$2.27(kg)$
Unsprung mass	m_{us}	$0.25(kg)$
Suspension stiffness	k_s	$1396(N/m)$
Tyre stiffness	k_t	$12270(N/m)$
Max/Min damper force	$\overline{u}, \underline{u}$	$\pm 21(N)$
Max/Min deflection position	$z_{def}^{max}, z_{def}^{min}$	$\pm 0.005(m)$
Min PWM duty cycle	ϕ_{min}	0.1
Max PWM duty cycle	ϕ_{max}	0.35
Look ahead period	T_l	$0.23(s)$
Estimated sampling period for the INOVE platform	T_{sest}	$0.023(s)$
Natural sampling period	T_s	$0.005(s)$

3.7 RT HiL implementation of pNMPC controller and Linearization based MPC controller on dSPACE MABXII

The proposed pNMPC controller was compared against a linearization based MPC controller. The performance and computation time were compared with each other to gauge the potential of the proposed pNMPC method.

3.7.1 Linearization based MPC design

The fundamental assumption for the linearization based MPC design is to linearize the nonlinearities present in the system (i.e. constraints and dynamics) by means of first order Taylor series expansion and then, the problem is casted as a linear MPC problem, i.e. a convex QP problem. This procedure is repeated at every operating point and a linear MPC is solved at every operating point in receding horizon fashion. The first order linearization of the quasi-static nonlinear damper model (3.2) under the modified input at a given operating point $P_i = (\mathbf{x}_i, \phi_i)$ is expressed as

$$u_{P_i}(\mathbf{x}_k, \phi_k) = u(P_i) + \nabla_\phi u|_{(P_i)} \Delta\phi + \nabla_x u|_{(P_i)} \Delta\mathbf{x} \tag{3.8}$$

Where, $\Delta\mathbf{x} = \mathbf{x}_k - \mathbf{x}_i$ and $\Delta\phi = \phi_k - \phi_i$ are the state and input deviation variables with respect to the operating point P_i. The first order linearization of the nonlinear dynamics at the operating point P_i in continuous time is expressed as

$$\Delta\dot{\mathbf{x}}(t) = A_c^i \Delta\mathbf{x}(t) + B_c^i \Delta\phi(t) + B_{cd} d(t) \tag{3.9}$$

Where, $A_c^i \in \mathbb{R}^{4\times4}$ and $B_c^i \in \mathbb{R}^{4\times1}$ are the linearized system and input matrices at P_i. The obtained continuous time matrices are converted to discrete time matrices by means of zero

order hold (ZOH) method with a sample time T_s. The discrete-time linearized state space equation at the point P_i is expressed as

$$\Delta x^+ = A_d^i \Delta \mathbf{x}(k) + B_d^i \Delta \phi(k) + B_{dd} d(k) \tag{3.10}$$

Where, $A_d^i \in \mathbb{R}^{4\times4}$, $B_d^i \in \mathbb{R}^{4\times1}$ and $B_{dd} \in \mathbb{R}^{4\times1}$ are the discrete-time system matrix, input matrix and disturbance matrix. The linearized MPC finite time optimal control problem (FTOCP) at the point P_i with with $\mathbf{x}_0 = \mathbf{x}(0)$ and with $d_0 = d(0)$ is casted as a convex QP problem with horizon length N corresponding to the look ahead period T_l which is described as

$$\begin{aligned} J_{P_i}^* = \min_{\phi_{0:N-1}, \mathbf{x}_{1:N}} \quad & \sum_{k=0}^{N-1} J_k^{obj} \\ \text{s.t.} \quad & z_s - z_{us} \in [z_{def}^{min}, z_{def}^{max}],\ \forall k = 1 \dots N \\ & d_{k+1} = d_k,\ \forall k = 0 \dots N-1 \\ & u_{P_i}(\mathbf{x}_k, \phi_k) \in [\underline{u}, \overline{u}],\ \forall k = 0 \dots N-1 \\ & \phi_k \in \mathbb{U},\ \forall k = 0 \dots N-1 \\ & (3.10),\ \forall k = 0 \dots N-1 \end{aligned} \tag{3.11}$$

Where, $J_{P_i}^*$ is the optimal objective function. The optimal control input at point P_i to the actual system is $\phi^*(0) = \phi_0^*$ and this procedure is repeated in receding horizon policy method. For initialization of the input for the next linearization point, the solution of the previous program of equation (3.11) is utilized, i.e. $P_{i+1} = (\mathbf{x}_{i+1}, \phi_1^*)$, where ϕ_1^* is the solution at time step 1 at P_i point. The linearization is performed by precomputing the Jacobians a priori and is evaluated at every time instant.

3.7.2 Simulation analysis for pNMPC method

A detailed analysis is conducted for the proposed parameterized NMPC method for the quarter car semi-active suspension system for different cases. The parameterized NMPC method is simulated in MATLAB/Simulink environment and its closed-loop performance characteristics are investigated for different complexity factors i.e. different space discretization points (N_s) and computational time i.e. the control update period (τ_ϕ). The considered acid test is for the following scenario

- A chirp road profile with a frequency sweep between 0.1 Hz to 25 Hz with an amplitude of 1 mm for a duration of 10 s.
- The control objective is selected to be comfort oriented design i.e. $\theta_1 = 1$ ($\theta_2 = 0$).
- The control update period is a variable which is expressed with $\tau_\phi = \gamma N_s T_s$, where γ is the computational scale factor and $T_s = 0.5 \times 10^{-4}$ s.

The rationale behind this heuristic and the analysis is to comprehend the behavior of the proposed controller when executed in different computational resources for different complexities (N_s), control update period (τ_ϕ) and also, the analysis provides insight over the optimal

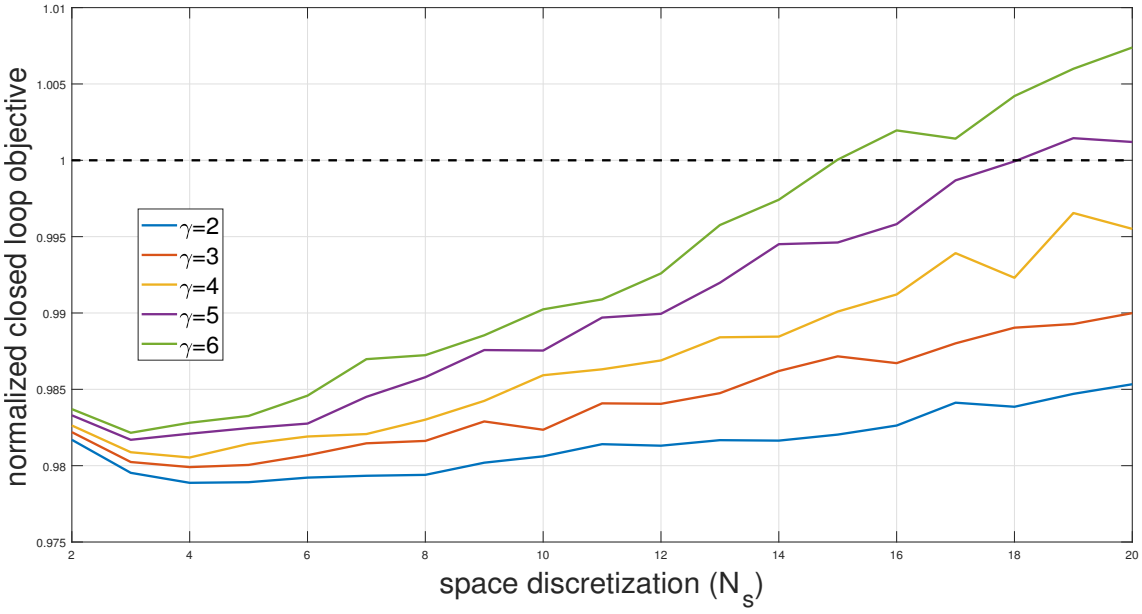

Figure 3.7: N_s vs J_{CL}^{norm} for different values of computational scale factor γ

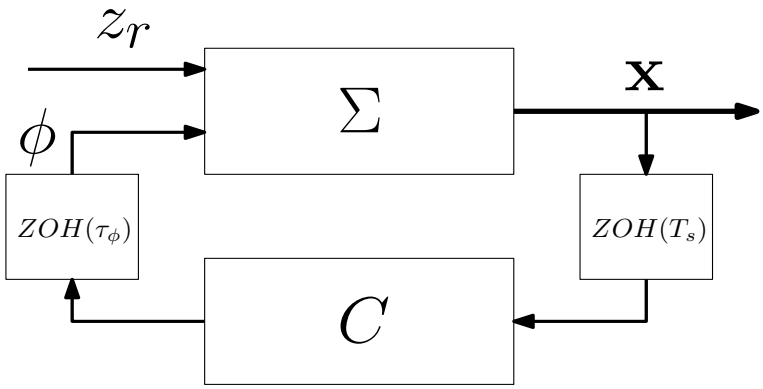

Figure 3.8: Control scheme for the proposed analysis

complexity factor N_s^* to be utilized for a given computational resource. Fig. 3.7 illustrates the normalized closed loop performance of the system for complexity factor (N_s) vs normalized closed loop objective (J_{CL}^{norm}) for different computational scale factor (γ). The normalized closed loop objective (J_{CL}^{norm}) is defined with respect to the objective of nominal passive suspension system defined as

$$J_{CL}^{norm} = \frac{J_{CL}^{obj}}{J_{CL}^{pass}} \tag{3.12}$$

where, J_{CL}^{pass} corresponds to the the closed loop objective for the nominal passive suspension system and J_{CL}^{obj} corresponds to the closed loop objective of the parameterized NMPC method. The curves in plot Fig. 3.7 illustrates the fact that the normalized closed loop objective (J_{CL}^{norm}) for a given computational scale factor (γ) declines as the complexity factor gradually increases, however as the complexity factor increases more than a certain threshold, the normalized closed

loop objective (J_{CL}^{norm}) increases due to the fact that the computational load is elevated and consequently, the control update period (τ_ϕ) is increased, which results in poor performance of the controller. The abscissa of the optimal point for the curves indicates the best/optimal complexity factor N_s^* for a given computational resource or computational scale factor (γ).

Remark - The proposed parametrized NMPC is of high interest for practical applications for a large set of semi-active dampers. Indeed it is worth noting that N_s defines the set of damping coefficients than can be used in real-time control. When that N_s tends to infinity this corresponds to a continuously-variable damper. When $N_s = 2$ this corresponds to a 2-states damper or to a min-max suspension control approach (as for SkyHook, and ADD and SH-ADD methods). This offers a large flexibility for the implementation of several control methods for different damper types.

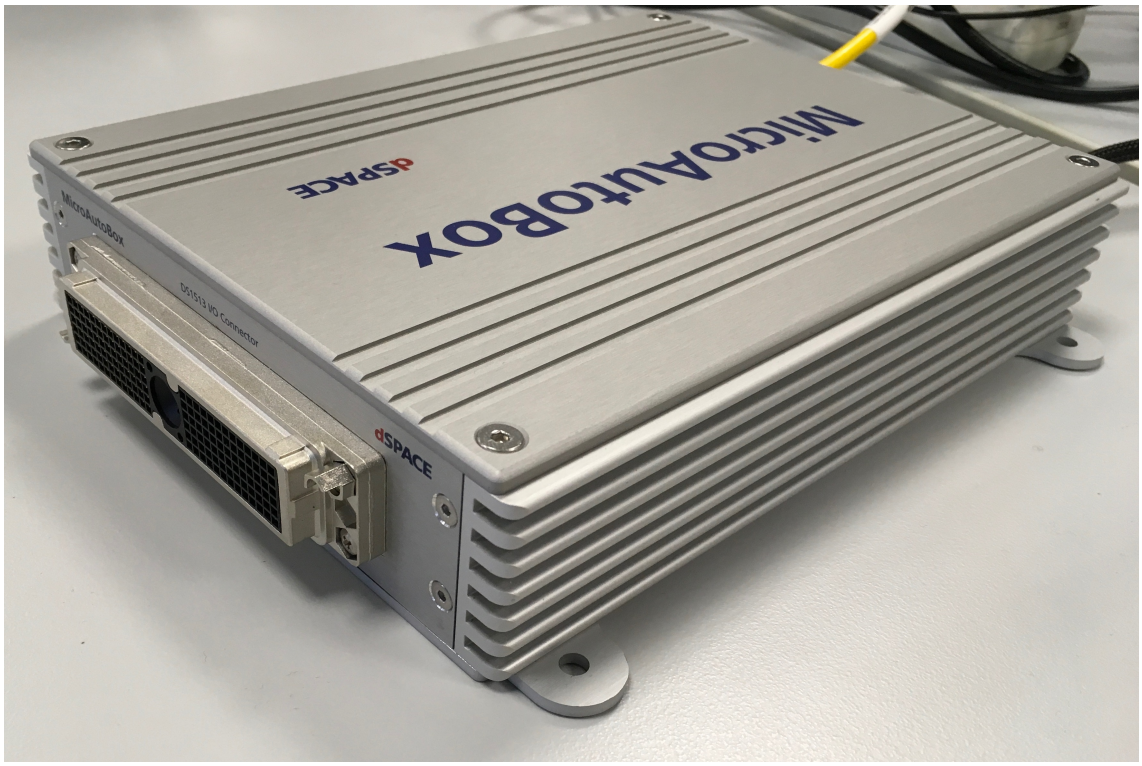

Figure 3.9: dSPACE MicroAutoBox II - 1401/1511

3.8 Real-time Implementation

The proposed method and linearization based MPC method are implemented on RT conditions on dSPACE MicroAutoBox II hardware with MATLAB/Simulink interface. HIL tests are conducted for different scenarios and the performance characteristics of the controllers are investigated. The linearization based MPC is implemented using CVXGEN solver by [Mattingley and Boyd 2012], in which the optimization problem in equation (3.11) is programmed and

the generated C-code is patched with Simulink using S-function builder block. The parameterized NMPC is programmed using Simulink-MATLAB function block. The generated Simulink files are deployed to the dSPACE hardware and the results were obtained from ControlDesk environment.

3.8.1 Computational efficiency test

To test the computational efficiency of the two methods, the maximum execution time is recorded for different complexity parameter of the controller. The complexity parameter for the linearization based MPC is selected to be the number of Newton steps/iterations for the QP solver and for the proposed approach, the number of space discretization points (N_s) is considered. The road profile is a chirp signal with amplitude of 1 mm and frequency sweep from 5 Hz to 25 Hz with comfort objective i.e. $\theta_1 = 1$ (also applies for road holding objective $\theta_2 = 0$).

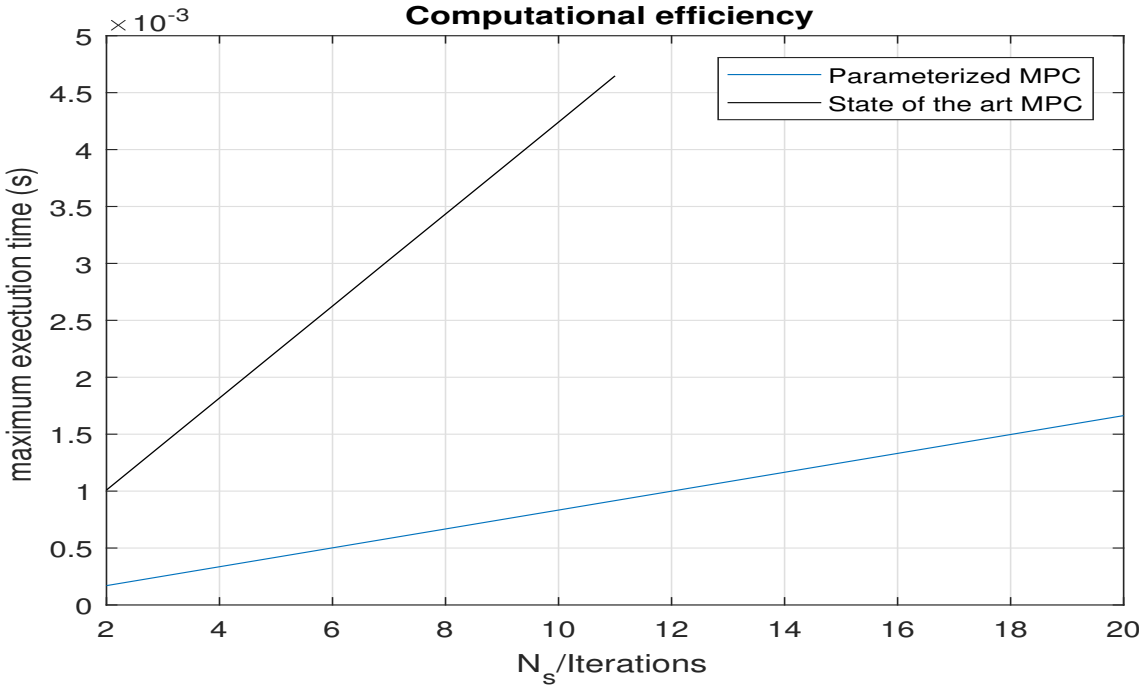

Figure 3.10: Computational efficiency between the pNMPC and linearization based MPC controller

Fig 3.10 shows a linear relationship between the complexity and the maximum execution time for both controller, however the scale of the proposed controller is roughly three times lesser than the linearization based MPC using CVXGEN. This observation leverages the plausibility to utilize smaller sampling time (T_s) and complexity factor N_s for the proposed method.

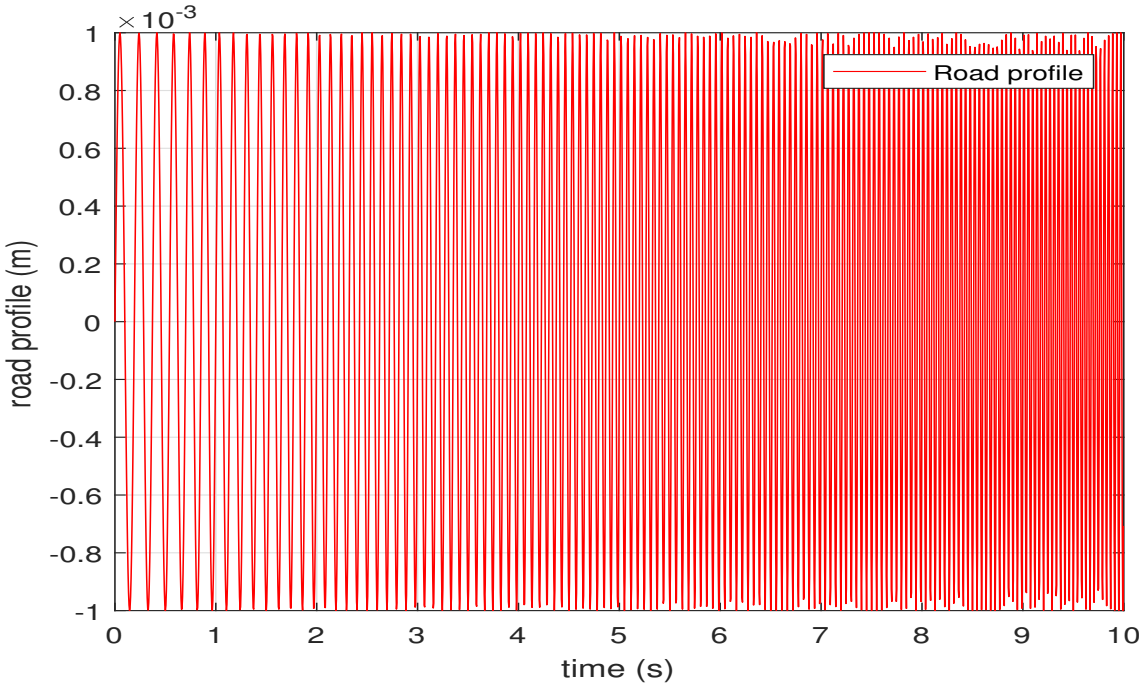

Figure 3.11: Chirp Road profile with amplitude of 1 mm and frequency sweep from 5 to 25 Hz

Table 3.3: RMS values for comfort objective for chirp road profile

Objective	Linearized MPC	Parameterized NMPC
Comfort (m/s^2)	2.7701	2.2643

3.8.2 Chirp test with comfort objective

The test involves a chirp road profile (shown in Fig. 3.11) with amplitude of 1 mm and frequency sweep from 5 Hz to 25 Hz with comfort objective i.e. $\theta_1 = 1$ (also applies for road holding objective i.e. $\theta_2 = 0$). The complexity factor for linearization based MPC, i.e. number of iterations is 11 and for parameterized NMPC N_s is 20. The root mean square (RMS) values for the simulations are listed in Table 3.3. For the comfort objective, the chassis acceleration is shown in Fig. 3.12. The dissipativity constraint due to non-linear modeling of ER damper is shown in Fig. 3.13. The results demonstrate better performance of the proposed approach compared to linearization based MPC for the considered scenario in RT considerations.

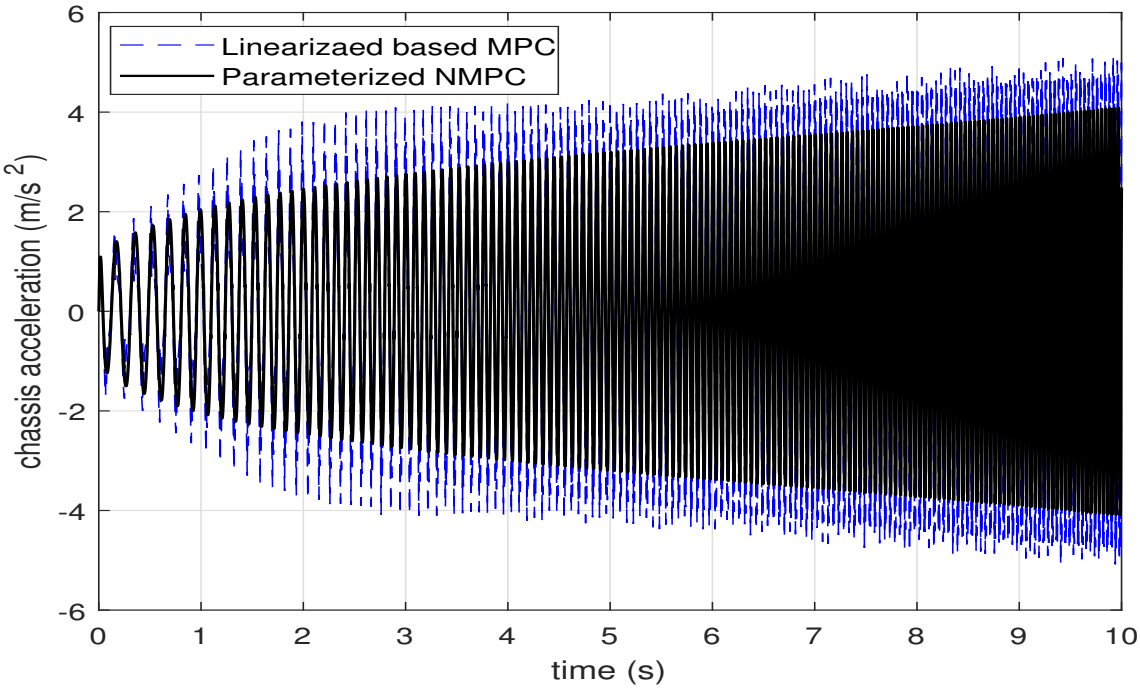

Figure 3.12: Chassis acceleration for the quarter car system

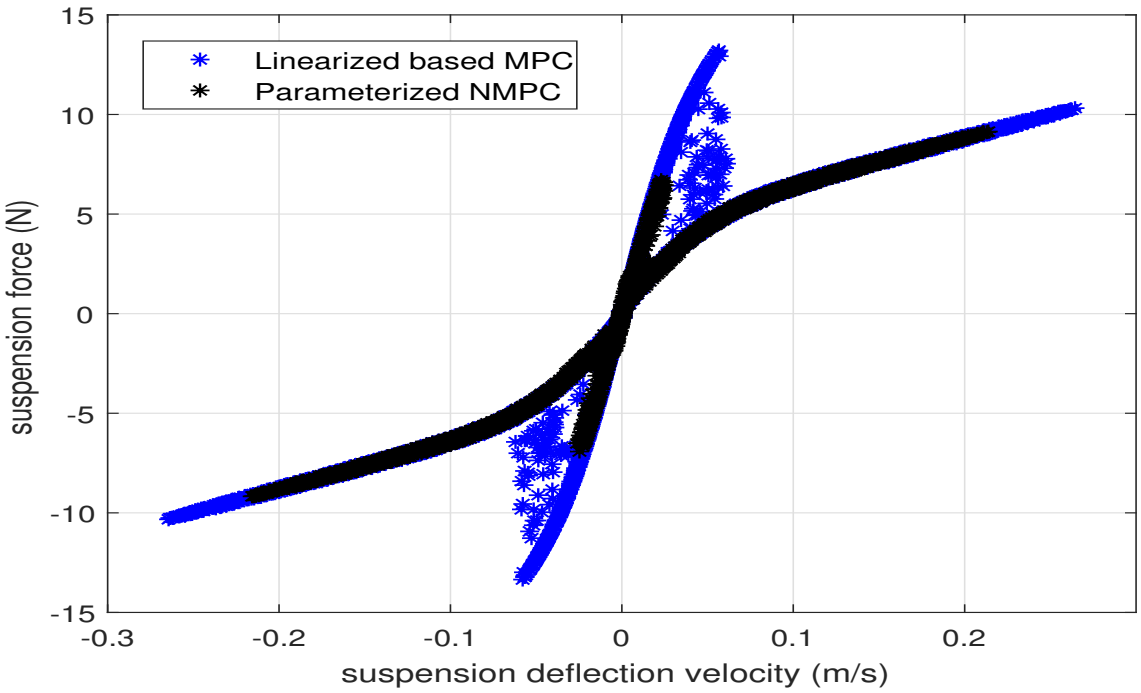

Figure 3.13: Damping force vs suspension deflection velocity

3.9 Experimental implementation of pNMPC controller on INOVE test platform

The proposed pNMPC controller was tested on the INOVE test platform for a series of tests along with a few comparison controllers.

3.9.1 Comparison controllers

- **Modified Skyhook controller** - Skyhook controller is one of the most prominent and well known controller for semi-active suspension system [Karnopp, Crosby, and Harwood 1974]. The modified skyhook controller is an extension to the skyhook controller where the ER-SA damper system's PWM-DC signal swings between minimum and maximum value conditioned upon a switch condition. Mathematically, the controller is expressed with

$$\phi = \begin{cases} \phi_{max}, \text{ if } \dot{z}_s \dot{z}_{def} \geq 0 \\ \phi_{min}, \text{ if } \dot{z}_s \dot{z}_{def} < 0 \end{cases} \tag{3.13}$$

- **Nominal passive suspension** - The nominal passive suspension is a typical passive suspension system, however the term nominal indicates that the PWM-DC for the ER-SA damper system is fixed to the mean value of the minimum and maximum values of the PWM-DC, i.e. $\phi_{nom} = \dfrac{\phi_{min} + \phi_{max}}{2}$. The value is held constant over the entire period of operation, which is 0.225.

3.9.2 Results and Implementation

The proposed MPC controller and the comparison controllers were programmed in MATLAB/Simulink environment and was implemented on the INOVE test platform. Two road profile tests were conducted to validate the performance of the proposed MPC controller, which are a) Chirp road profile test and b) Bump road profile test.

3.9.2.1 Chirp road profile test

The test involved a chirp road profile with amplitude of 2.5 mm and frequency sweep from 5 Hz to 22 Hz (this corresponds to the comfort frequency range for the INOVE test platform). The road profile is shown in Fig. 3.14. The PWM-DC control inputs for different controllers is shown in Fig. 3.15. It is clearly evident that the proposed MPC utilizes the control authority in a judicious manner such that to minimize the vertical chassis acceleration. The RMS values of the chassis acceleration for the test and the percentage gain with respect to nominal passive damping are listed in Table 3.4. The RMS values clearly evinces the fact that the proposed MPC method fares better the nominal passive damping and modified skyhook controller.

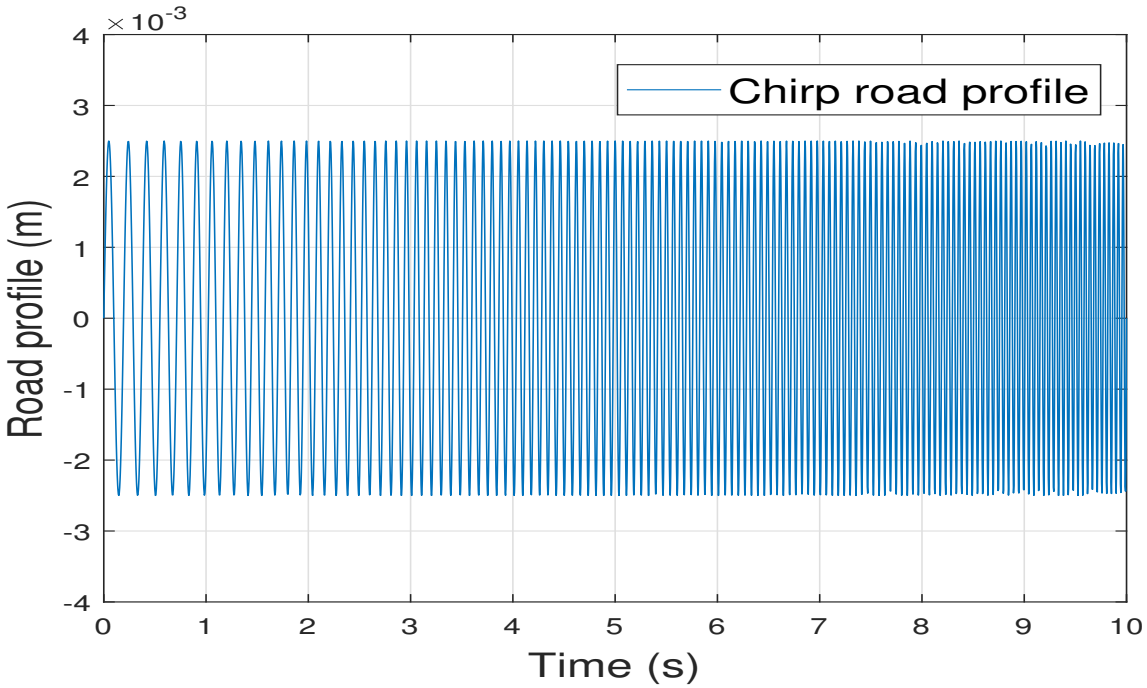

Figure 3.14: Chirp road profile

Table 3.4: RMS values for comfort objective for chirp road profile

Controller	RMS (m/s^2)	% Gain
Nominal passive damping	6.87	0
Modified Skyhook controller	6.66	3.05
Proposed pNMPC controller	6.42	6.5

3.9.2.2 Bump road profile test

The INOVE test platform was excited with bump road profile, shown in Fig. 3.16 with peak amplitude of 7 mm and duration of 10 s. The recorded chassis acceleration is shown in Fig. 3.17. From the chassis acceleration plot, it is evident that the proposed MPC method mitigates the peak chassis acceleration at bump points. The PWM-DC control input is shown in Fig 3.18.

3.10 Conclusions

This chapter has presented the pNMPC scheme for control of vertical dynamics of vehicle via ER semi-active suspension system for a quarter car model. The method was tested through HiL simulations as well as the INOVE test platform to validate and verify the performance with respect to the objective and constraints requirements. The method was compared against

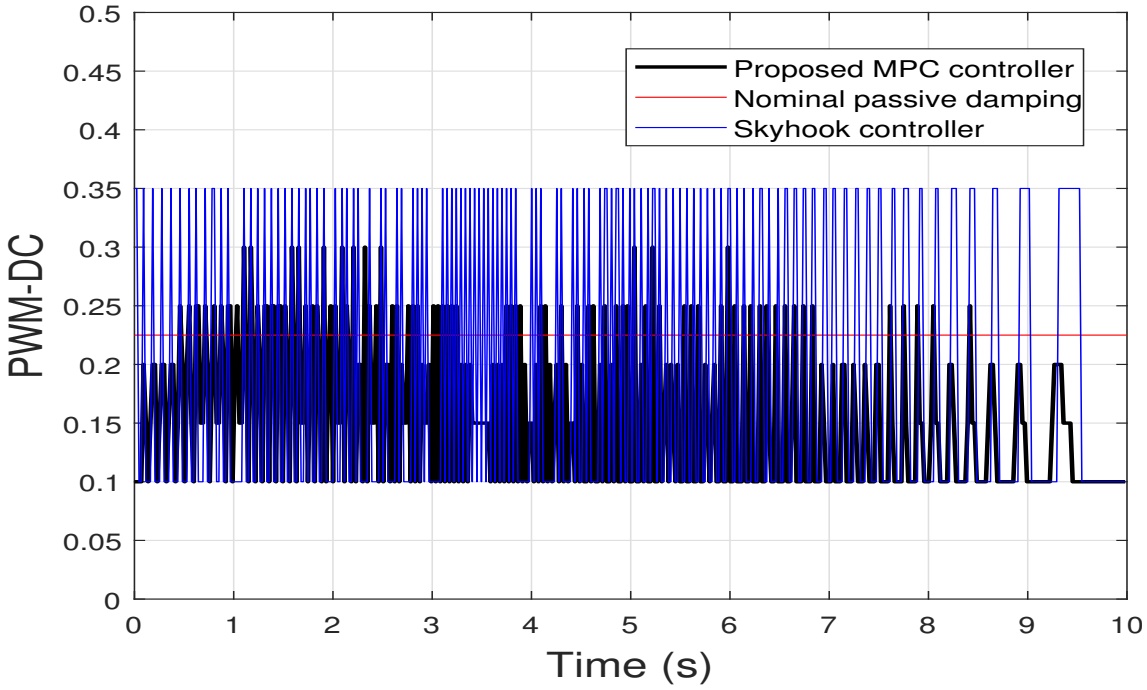

Figure 3.15: PWM-DC input for different controllers for chirp road profile

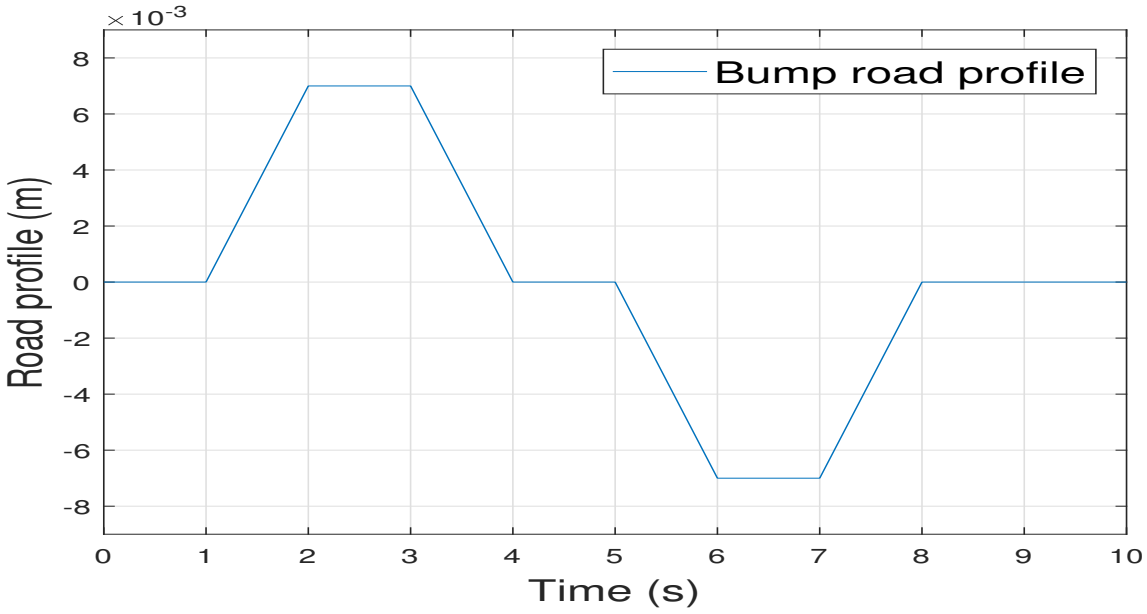

Figure 3.16: Bump road profile

CVXGEN based linearized MPC controller in dSPACE MABXII and control schemes such as skyhook and passive damping setup on the INOVE test platform. Overall, the proposed pNMPC method fares well and looks promising for practical implementation on real-world systems. It is also important to note that the crux of the EMPHYSIS project is to utilize model based controllers for embedded control of automotive systems and at the same time,

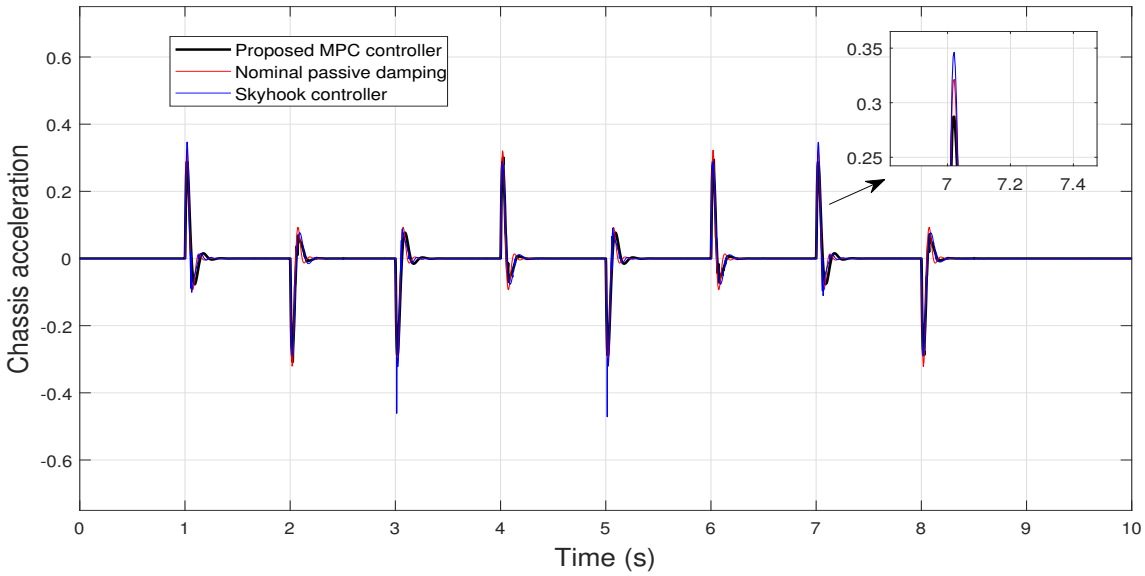

Figure 3.17: Chassis acceleration for bump road profile

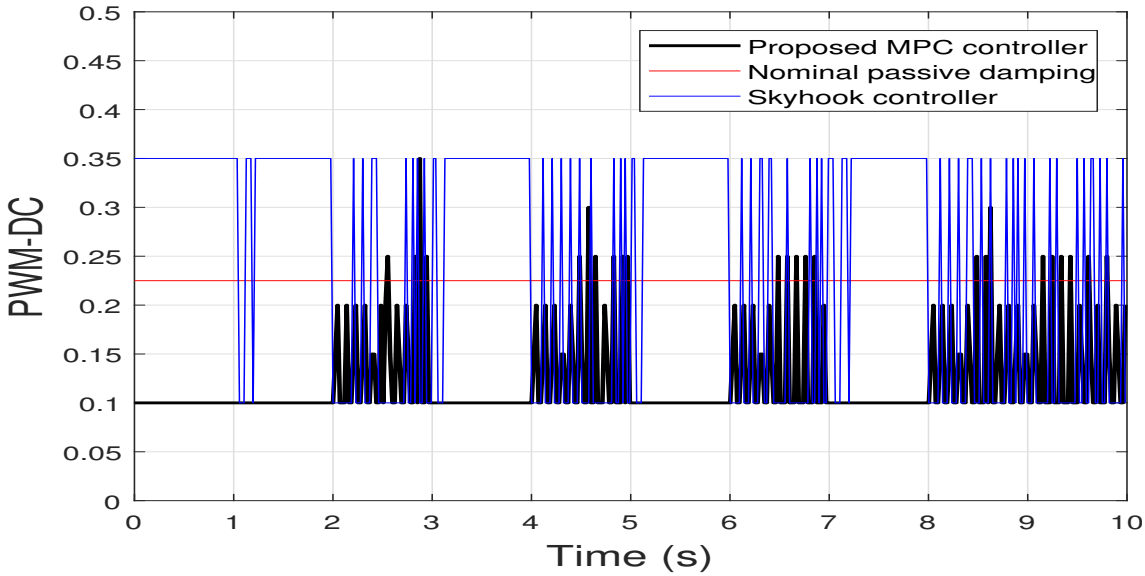

Figure 3.18: PWM-DC input for different controllers for bump road profile

the dynamical models are exported as eFMU containers, where at times, the knowledge of the system is not explicitly provided to the end user. The proposed pNMPC method does not require any internal details of the system rather only the model ought to be simulated to obtain the stae trajectory of the system. Thus, the proposed pNMPC method is scalable with eFMU containers and also, can cope up with black-box models.

Chapter 4

GPU based parallelized pNMPC scheme for control of semi-active suspension system

Contents

4.1 Introduction

This chapter introduces the parallelized pNMPC scheme for control of semi-active suspension system for a half car vehicle. The method taps the power of GPU computing for parallelizing the pNMPC control scheme across several GPU multi-core processors. The method is generic in nature, however in thesis the proposed method is utilized for solving the automotive suspension control problem. The chapter is divided into two parts, where in the first part the road model is not accounted into the OCP formulation and in the second part the road profile is modeled via a stochastic process and the resulting method is termed scenario stochastic pNMPC scheme (SS-pNMPC).

Some of the key aspects of the proposed methods are

- **Automotive standard** - The method is amenable with the existing automotive standards as only simple math operations are required and also, independent of any back-end optimization solver.
- **Modeling flexibility** - The proposed SS-pNMPC method requires no a-priori assumptions on the system such as linear time invariant (LTI) dynamics, stationary property, Gaussian distribution of noise etc. This feature is highly sought these days as proposed in [Guanetti and Borrelli 2017], the road model stochastic information can be dynamically obtained from cloud servers and this could be easily embedded in the OCP formulation.
- **Black-box model compatibility** - When dealing with black box models such as FMUs [Blockwitz et al. 2012] or eFMUs, the model implementation details such as dynamical equations, system parameters etc. are concealed away from the end user due to protection of intellectual property (IP) rights. In the such a situation, designing a model based controller poses to be a serious challenge, however the proposed pNMPC controller is based on a simulation-optimization method and this obviates any need to stipulate the structure of dynamics, objective or constraint functions. The proposed approach is implemented on GPU to increase the computation throughput for the controller.
- **Efficient RT operability** - The fast computation of the control input by means of GPU renders the method to be RT operable, especially for fast systems.
- **Validation/Verification on Embedded platforms** - The proposed SS-pNMPC method was tested on multiple GPU based embedded boards to calculate the computation time and also to assess the RT feasibilty of the method. The method was tested on the NVIDIA Jetson embedded boards - Nano, TX1, TX2 and Xavier and the results looks promising for RT implementation.

4.1.1 Related works

There has been several research conducted in a recent past on solving the MPC problem which harnesses the potential of GPUs. A brief tutorial on different parallel architectures for MPC is

proposed in [Koegel and Findeisen 2012]. In [Gade-Nielsen, Dammann, and Jørgensen 2014], several GPU based interior point methods were developed for linear MPC framework. There have been several research contributions on utilization of GPUs for solving stochastic control problems and in [Abughalieh and Alawneh 2019], a detailed survey on various types of parallel implementation of MPC methods are described. In [Sampathirao et al. 2017], a scenario based Stochastic MPC (SMPC) method is proposed where the structure of the system is exploited and the problem is solved using proximal gradient method which is parallelized on GPU. In [Ohyama and Date 2017], a sampling based parallelized nonlinear MPC (NMPC) scheme is proposed and experimentally validated for control of inverted pendulum system. In [Williams et al. 2016], a path integral based MPC method is proposed and experimentally validated for a Remote Controlled (RC) car, the paper derives an input update rule for a stochastic optimal control problem based on information theoretic concepts and at every sampling period, multiple random scenarios are generated to update the input sequence. In [Rogers 2013], a guidance law for guided projectiles is proposed, where the GPUs are utilized to generate RT scenarios to predict the impact point and probability of violating impact area constraints. In [Hyatt and Killpack 2020], a GPU based RT NMPC scheme for control of robots is proposed where the method is termed as Nonlinear Evolutionary MPC (NEMPC) and is practically implemented for control of a 24 state pneumatically actuated continuum soft robot.

Concerning the control of semi-active suspensions system, as mentioned previously in Chapter 3, [Savaresi et al. 2010] provides a comprehensive collection of all classical and modern control methods such as (to name a few) Skyhook, SH-ADD, Hybrid MPC with preview, H_∞ and LPV methods etc. However, in the line of research of application of Stochastic MPC for control of suspension systems, not many research have been conducted in the past. To the best of knowledge of the authors, in [Guanetti and Borrelli 2017] a cloud aided SMPC method is proposed for control of active suspension system for a quarter car vehicle. However, the method is proposed only for LTI systems and also, the second order conic program (SOCP) solver utilized in the method is computationally not tractable within the prescribed sampling period for higher order systems such as half/full car.

4.1.2 Chapter contribution

The main contributions of this chapter are

- The pNMPC method proposed in Chapter 3 is augmented by parallelizing the method over the several multi-core streaming processors in the GPUs. By parallelizing the pNMPC method over several multi-core processors, one can attain significant reduction in computation time due to several simultaneous simulations of the dynamical system under several control configurations. The proposed parallelized pNMPC method is applied for control of semi-active suspension system for a half-car model with two configurations.
- In the first configuration, the road model is not included into the control design and the proposed controller is simply termed as parallelized pNMPC scheme. In the second configuration, the road model is modeled via a stochastic model using the ISO road pro-

file standand and the proposed controller is termed Scenario-Stochastic pNMPC scheme (SS-pNMPC).

- The proposed methods were tested in simulation as well as in NVIDIA embedded boards to test, verify and validate the performance both in simulation as well as in RT scenario. The methods were developed using C++ with CUDA-C compiler support and MATLAB/Simulink environment.

4.2 Half car model with semi-active suspension system

4.2.1 Half car mathematical model without road model

As decribed in Chapter 1, Section 1.3.2, the half car vertical dynamics model is a 4 degrees of freedom (DOF) model which involves chassis dynamics, roll dynamics and dynamics of the two unsprung masses (wheels). Let the left and right corner of the vehicle be indexed with $i \in \{l, r\}$ respectively. The 4 DOF mathematical model is expressed with the following equations

$$\begin{cases} m_s \ddot{z}_s = -\sum_{i\in\{l,r\}} F_{s,i} \\ I_x \ddot{\theta} = (l_l F_{s,l} - l_r F_{s,r}) \\ m_{us,l} \ddot{z}_{us,l} = (-F_{s,l} + F_{t,l}) \\ m_{us,r} \ddot{z}_{us,r} = (-F_{s,r} + F_{t,r}) \end{cases} \tag{4.1}$$

4.2.2 Half car mathematical model with stochastic road model

Here the objective is to incorporate the road model in to the problem formulation. The mathematical model of the vehicle is comprised of a) Dynamical model of vertical motion of half car vehicle extended with road model and b) Kinematics model for longitudinal motion of the vehicle. Let the left and right corners of the vehicle be indexed with $i \in \{l, r\}$ respectively. The equations of the model is expressed as follows

$$\begin{cases} m_s \ddot{z}_s = -(F_{s,l} + F_{s,r}) \\ I_x \ddot{\theta} = (l_l F_{s,l} - l_r F_{s,r}) \\ m_{us,l} \ddot{z}_{us,l} = (-F_{s,l} + F_{t,l}) \\ m_{us,r} \ddot{z}_{us,r} = (-F_{s,r} + F_{t,r}) \\ \dot{z}_{r,l} = -\alpha v_x z_{r,l} + \xi_l \\ \dot{z}_{r,r} = -\alpha v_x z_{r,r} + \xi_r \\ \dot{v}_x = a_x \end{cases} \tag{4.2}$$

4.2.3 ISO road profile

The ISO road profile is primarily dependent upon two factors which are a) Longitudinal velocity of the vehicle and b) Road roughness coefficient, i.e. the road surface [Tyan et al. 2009]. In accordance with the ISO-8608 standard [ISO 1995], the factored power spectral density (PSD) of road surface is defined with

$$H(j\omega)\Psi_w H^*(j\omega) = \frac{2\alpha v_x \sigma^2}{(\alpha v_x + j\omega)(\alpha v_x - j\omega)} \tag{4.3}$$

where, ω is the excitation frequency of the road input (in rad/s), $\Psi_w = 2\alpha v_x \sigma^2$ represents the spectral density of the Gaussian white noise and σ^2 denotes the road roughness variance. $H(j\omega) = \frac{1}{(\alpha v_x + j\omega)}$ is the first order shaping filter. This frequency response function can be recasted as a time-varying first order auto regressive process, which is expressed with

$$\dot{z}_r(t) = -\alpha v_x(t) z_r(t) + \xi(t) \tag{4.4}$$

where, $z_r(t)$ is the road profile as defined in (4.2) and road disturbance drawn from a normal distribution defined with $\xi(t) \sim \mathcal{N}(0, \Psi_{z_r}(t))$. The parameters for different road surfaces are listed in Table 4.1. It is also important to note that the road roughness variance listed in Table 4.1 is scaled accordingly to suit the INOVE test platform.

Table 4.1: ISO road roughness parameters

Road surface	Road roughness variance (σ^2)	α (rad/s)
ISO A (Very Good)	4×10^{-6} m	0.127
ISO B (Good)	16×10^{-6} m	0.127
ISO C (Average)	64×10^{-6} m	0.127
ISO D (Poor)	256×10^{-6} m	0.127
ISO E (Very Poor)	1024×10^{-6} m	0.127

4.2.4 Mathematical terminology

z_s, θ represents the heave/chassis position and roll angle of the vehicle w.r.t. the centre of gravity (COG) respectively. $z_{us,i}$, $z_{r,i}$, ξ_i $\forall i \in \{l, r\}$ represents the wheel/unsprung mass position, vertical road profile of the vehicle and random disturbances respectively [Guanetti and Borrelli 2017]. v_x and a_x denotes the longitudinal velocity and acceleration of the vehicle. m_s, $m_{us,l}$, $m_{us,r}$ represents the chassis mass, unsprung masses for the left and right corners. I_x represents the moment of inertia along the roll axis. l_l and l_r represents the length of the chassis from the left and right corners with respect to COG. α represents the ISO road profile parameter. $F_{s,i}$ represents the chassis forces and $F_{t,i}$ represents the wheel forces $\forall i \in \{l, r\}$ which are expressed with

$$\begin{aligned} F_{s,i} &= -k_{s,i}(z_{s,i} - z_{us,i}) + u_i \\ F_{t,i} &= -k_{t,i}(z_{us,i} - z_{r,i}) \end{aligned} \tag{4.5}$$

where, $k_{s,i}$ and $k_{t,i}$ represents the stiffness coefficent of the SA suspension system and wheel respectively. $z_{r,i}$ and $z_{us,i}$ represents the vertical road displacement and unsprung mass position $\forall i \in \{l, r\}$. u_i represents the actuation force obtained from the nonlinear SA damper model (see Section 4.2.5). $z_{s,i}$ represents the sprung mass displacement at each corner which are obtained from the following equations

$$\begin{aligned} z_{s,l} &= z_s + l_l \sin\theta \\ z_{s,r} &= z_s - l_r \sin\theta \end{aligned} \tag{4.6}$$

4.2.5 Nonlinear quasi-static SA damper model

The SA damper force u_i (4.4) is expressed by utilizing the Guo's damper force model [Savaresi et al. 2010] $\forall i \in \{l, r\}$ with

$$u_i = c_0 \dot{z}_{d,i} + f_c \phi_i \tanh(a_1 \dot{z}_{d,i} + a_2 z_{d,i}) \tag{4.7}$$

where k_0, c_0, f_c, a_1 and a_2 represent the damper stiffness coefficient, viscous damping coefficient, dynamic yield force of the fluid, hysteresis coefficient due to velocity and position respectively. ϕ_i, $\forall i \in \{l, r\}$ represents the duty cycle (PWM-DC) input signal which manipulates the damper characteristics online by changing the input voltage. $z_{d,i} = z_{s,i} - z_{us,i}$ and $\dot{z}_{d,i} = \dot{z}_{s,i} - \dot{z}_{us,i}$ represents the deflection position and velocity $\forall i \in \{l, r\}$ between the chassis and wheel respectively. The values for the parameters are listed in Chapter 3, Table 3.1

4.3 Parallelized pNMPC scheme for control of semi-active suspension system without road model

4.3.1 Mathematical model notations

Let $\mathbf{X} = [z_s,\ \theta,\ z_{us,l},\ z_{us,r},\ \dot{z}_s,\ \dot{\theta},\ \dot{z}_{us,l}\ , \dot{z}_{us,r}]$ denote the state vector, $\mathbf{U} = [\phi_l,\ \phi_r]$ denote the input vector and $\mathbf{D} = [z_{r,l},\ z_{r,r}]$ denote the disturbance vector. $\mathbf{X} \in \mathbb{R}^8$, $\mathbf{U} \in \mathbb{R}^2$ and $\mathbf{D} \in \mathbb{R}^2$, then the half car model in equation (4.1) can be compactly expressed with

$$\dot{\mathbf{X}}(t) = f(\mathbf{X}(t), \mathbf{U}(t), \mathbf{D}(t)) \tag{4.8}$$

4.3.2 Parallelized pNMPC design requirements

4.3.2.1 Objective requirements for Parallelized pNMPC without road model

The objective design can be briefly classified as a) comfort and b) ride handling objective [Savaresi et al. 2010].

- **Comfort objective**: The goal of the comfort based objective is to minimize the vertical acceleration of the chassis ($\ddot{z}_s$), obtained from equation (4.1). The comfort objective for a given look ahead period T_l is expressed with

$$J_{T_l}^{acc}(\mathbf{X}(.), \mathbf{U}(.), \mathbf{D}(.)) = \int_0^{T_l} (\ddot{z}_s(t))^2 dt \tag{4.9}$$

- **Ride handling objective**: The goal of the ride handling objective is to minimize the roll angle (θ) of the vehicle. The ride handling objective for a given look ahead period T_l is expressed with

$$J_{T_l}^{roll}(\mathbf{X}(.), \mathbf{U}(.), \mathbf{D}(.)) = \int_0^{T_l} (\theta(t))^2 dt \tag{4.10}$$

4.3.2.2 Constraint requirements for Parallelized pNMPC without road model

The constraints for the SA suspension system primarily arise from the physical limitations and secondarily from performance requirements [Baumal, McPhee, and Calamai 1998]. For the MPC design, the included constraints are

1. **ER-SA damper input constraints**:
   (a) Damper force constraint (Physical): The ER-SA damper force is bounded, i.e. $u_i \in [u_{min,i},\ u_{max,i}]$, $\forall i \in \{l, r\}$.
   (b) PWM-DC input constraints: The operating DC for the PWM signal is constrained to $\phi_i \in [\phi_{min,i},\ \phi_{max,i}]$, $\forall i \in \{l, r\}$.
2. **State constraints**:
   (a) Stroke deflection constraint (Physical): This forms a linear state constraint i.e. $z_{d,i} \in [z_{dmin,i},\ z_{dmax,i}]$, $\forall i \in \{l, r\}$.
   (b) Wheel rebound constraint (Performance): This bounds the deflection position between the wheel and road. This is to ensure the tyre deflection forces are bounded i.e. $z_{us,i} - z_{r,i} \in [z_{rebmin,i},\ z_{rebmax,i}]$, $\forall i \in \{l, r\}$.
   (c) Unsprung mass displacement constraint (Performance): This bounds the displacement of the unsprung mass. This reinforces the road holding condition for the vehicle i.e. $z_{us,i} \in [z_{usmin,i},\ z_{usmax,i}]$, $\forall i \in \{l, r\}$.
3. **Road disturbance assumption**: The road disturbance is assumed to be constant over the prediction horizon (T_l) with $\mathbf{D}(0) = \mathbf{D}_0$ i.e. the road profile measured at the current time instant by means of observers [Doumiati et al. 2017]. There is no loss of generality with this formulation as it can be easily extendable with any road models such as ISO road profiles or from road preview sensors.

The mixed input-state constraint set is compactly expressed with $(\mathbf{X}, \mathbf{U}) \in \Omega_{(\mathbf{X},\mathbf{U})} \subset \mathbb{R}^8 \times \mathbb{R}^2$, the input constraint set is compactly expressed with $\mathbf{U} \in \Omega_{\mathbf{U}} \subset \mathbb{R}^2$ and the state constraint set is compactly expressed with $\mathbf{X} \in \Omega_{\mathbf{X}} \subset \mathbb{R}^8$.

4.3.3 MPC problem formulation

In summary, with the proposed objective and constraints function, the NMPC OCP is casted as

$$\begin{aligned} J_{obj}(\mathbf{X}_0, \Gamma_0, \mathbf{D}_0, \mathbf{U}(.)) = \min_{\mathbf{U}(.)} \Gamma_0^1 J_{T_l}^{acc} + \Gamma_0^2 J_{T_l}^{roll} \\ \text{subject to } \dot{\mathbf{X}}(t) = f(\mathbf{X}(t), \mathbf{U}(t), \mathbf{D}(t)) \\ \mathbf{X}_0 = \mathbf{X}(0), \mathbf{D}(t) = \mathbf{D}_0 \\ \mathbf{X}(t) \in \Omega_{\mathbf{X}}, \mathbf{U}(t) \in \Omega_{\mathbf{U}} \\ (\mathbf{X}(t), \mathbf{U}(t)) \in \Omega_{(\mathbf{X},\mathbf{U})} \end{aligned} \tag{4.11}$$

where $\Gamma_0 = [\Gamma_0^1, \Gamma_0^2]$ with Γ_0^1 and Γ_0^2 being the convex combination weights between the two objectives i.e. $\Gamma_0^1 + \Gamma_0^2 = 1$. Once the optimal input trajectory is computed, the first control action in $\mathbf{U}^*$ is injected into the system and this procedure is repeated in receding horizon fashion. In this work, a constant input profile is assumed over the control horizon.

4.3.4 Parallelized pNMPC Method

The crux of the method is to finitely parameterize the input constraint set $\Omega_{\mathbf{U}}$ and by virtue of parallel computing methods, the system in equation (4.8) is simulated for every parameterized input given the current state, disturbance and objective specification information. The optimal input is elicited which minimizes the objective and satisfies the constraint in equation (4.11) (see [Rathai et al. 2018] for more details). The pseudo-code for the implementation of parallelized pNMPC is shown in Implementation 1.

The pseudo-code deserves some explanation to elucidate its working principle. The details enclosed in this section provides the reader with the need to know basics to handle parallel computing using the NVIDIA CUDA GPUs for implementation of the proposed parallelized pNMPC method. The pseudo-code is written in a manner to benefit the reader to easily connect with CUDA-C implementation. For the respective application programming interfaces (APIs) and other syntax details refer to [Sanders and Kandrot 2010].

Explanation of Implementation:

1. **Initialization, I/Os and Syntax declaration**:

   (a) The first step is the data initialization where the model/constraint parameters and GPU parameters are initialized. b_s, g_s represents the size of the grid and blocks respectively [Sanders and Kandrot 2010]. n_{ϕ_l}, n_{ϕ_r} represents the number of quantized levels of the PWM-DC input signal for the left/right corner of the vehicle respectively.

   (b) The input variables for the method are $\mathbf{X}_0, \mathbf{D}_0, \Gamma_0$ and the output variable is $\mathbf{U}^*$.

(c) The decorators __HOST__, __GLOBAL__, __DEVICE__ denotes the function call made from host (CPU) to host, host to device (GPU) and device to device respectively. The first function invoked is the **MAIN**.

2. **MAIN** function:

   (a) The lines 2-3 dynamically allocates memory (DMA) in the host and the device for the objective and inputs for every input combination.

   (b) In line 8, the **KERNEL** function is launched with appropriate launch parameters g_s and b_s.

   (c) The line 9 transfers the computed objective and input data from the device to the host.

   (d) The lines 6-8 returns the optimal input $\mathbf{U}^*$ by finding the index of the minimum objective or constraint violation.

3. **KERNEL** function:

   (a) The line 11 sets the thread indices for each PWM-DC input combination. From this point onward, each thread parallely computes the solution for each input combination.

   (b) The line 12 serializes the 2D grid into a single vector for objective and input vector designation.

   (c) The line 13 is a check condition to make sure the thread access is not exceeded.

   (d) The lines 16 and 17 obtains the input combination and objective value from the functions **GRID2D** and **ODESIM2OBJ**.

4. **GRID2D** function:

   (a) The line 20-19 sets the PWM-DC input values for left and right corner of the vehicle.

   (b) The function assigns the input values for every thread call, i.e. for all input combinations.

5. **ODESIM2OBJ** function:

   (a) The line 25 initializes the objective and constraint violation variable to zero and line 26 invokes the **GRID2D** function for specific thread indices which corresponds to an input combination.

   (b) The lines 27-35 runs the Euler integration scheme for the system (4.8) until the look ahead period T_l with an integration step of h. In case the constraints are violated, the objective is set to a very high value `MAX` and the constraint violation is quantified with `N` function, which computes the 2-norm for the constraints. Otherwise, the objective in (4.11) is computed numerically.

   (c) The line 36 returns the sum of objective and constraint violation for the given thread indices.

Implementation 1 Parallelized pNMPC implementation

Data Initialization: Model/Constraint parameters, b_s, g_s, n_{ϕ_l}, n_{ϕ_r}
Input: $\mathbf{X}_0, \mathbf{D}_0, \Gamma_0$; **Output**: $\mathbf{U}^*$

function __HOST__ **MAIN**$(\mathbf{X}_0, \mathbf{D}_0, \Gamma_0)$
  `HostDMAObj`(h_{obj}) ; `HostDMAInp`(h_{inp})
  `DeviceDMAObj`(d_{obj}); `DeviceDMAInp`(d_{inp})
  KERNEL$_{\{g_s,b_s\}}(d_{obj}, d_{inp}, \mathbf{X}_0, \mathbf{D}_0, \Gamma_0)$
  `DevicetoHostCpy`(h_{obj}, d_{obj}); `DevicetoHostCpy`(h_{inp}, d_{inp})
  $k_{opt} \leftarrow$ `indexmin`(h_{obj})
  $\mathbf{U}^* \leftarrow h_{inp}(k_{opt})$
  return $\mathbf{U}^*$
end function

function __GLOBAL__ **KERNEL**$_{\{b_s,g_s\}}(d_{obj}, d_{inp}, \mathbf{X}_0, \mathbf{D}_0, \Gamma_0)$
  $i \leftarrow$ `bIdx.x*bDim.x+tIdx.x`; $j \leftarrow$ `bIdx.y*bDim.y+tIdx.y`
  $r \leftarrow n_{\phi_l} i + j$
  if $i \geq n_{\phi_l} \vee j \geq n_{\phi_r}$ **then**
    return
  end if
  $d_{inp}[r] \leftarrow$ **GRID2D**(i, j)
  $d_{obj}[r] \leftarrow$ **ODESIM2OBJ**$(\mathbf{X}_0, \mathbf{D}_0, \Gamma_0, i, j)$
end function

function __DEVICE__ **GRID2D**(i, j)
  $\phi_{i,l} = \phi_{min,l} + \frac{i}{n_{\phi_l}-1}(\phi_{max,l} - \phi_{min,l})$
  $\phi_{j,r} = \phi_{min,r} + \frac{j}{n_{\phi_r}-1}(\phi_{max,r} - \phi_{min,r})$
  return $\{\phi_{i,l}, \phi_{j,r}\}$
end function

function __DEVICE__ **ODESIM2OBJ**$(\mathbf{X}_0, \mathbf{D}_0, \Gamma_0, i, j)$
  `Obj` $= 0$, `Con` $= 0$
  $\mathbf{U}_{i,j} \leftarrow$ `GRID2D`(i, j)
  for $t_{loop} = 0 : h : T_l$ **do**
    $\mathbf{X}^+ \leftarrow \mathbf{X}_0 + hf(\mathbf{X}_0, \mathbf{U}_{i,j}, \mathbf{D}_0)$
    if $\mathbf{X}^+ \notin \Omega_{(\mathbf{X},\mathbf{U})} \vee \mathbf{X}^+ \notin \Omega_{(\mathbf{X})}$ **then**
      `Obj` $=$ `MAX`; `Con` $=$ `Con` $+$ `N`$(\Omega_{(\mathbf{X},\mathbf{U})}, \Omega_{(\mathbf{X})})$
    else
      `Obj` $=$ `Obj` $+ h(\Gamma_0^1 J_{l_l}^{acc} + \Gamma_0^2 J_{l_l}^{roll})$
    end if
    $\mathbf{X}_0 = \mathbf{X}^+$
  end for
  return {`Obj` + `Con`}
end function

4.4 Analysis and Simulation results

The conducted simulation study can be broadly classified into three parts which are

1. Computational time (CT) analysis b/w CPU (serial) and GPU (parallel) for the proposed pNMPC method.
2. Comparative analysis b/w ACADO-qpOASES NMPC controller [Houska et al. 2009] and the proposed parallelized pNMPC method.
3. Performance analysis with a road profile test. Both methods were implemented in MATLAB/Simulink on a Intel Core i7 PC and NVIDIA GTX 1050Ti with 768 CUDA cores

Code generation option was utilized for ACADO-qpOASES NMPC controller and the proposed parallelized pNMPC was programmed in CUDA C and patched into Simulink with S-function.

4.4.1 Computational time analysis b/w CPU and GPU

The raison d'être for conducting this analysis is to emphasize the significance of GPUs for solving huge simulations and viability of the approach for control of real SA suspension system. From Fig. 4.1, it is evident that the pNMPC method fares well in GPU compared to CPU in terms of mean CT per sampling period. The abscissa indicates the number of discretized inputs for the PWM-DC signal for both left (n_{ϕ_l}) and right (n_{ϕ_r}) corner of the vehicle (i.e. $n_{\phi_l} \times n_{\phi_r}$ input combinations). The road profile involved a chirp signal (same profile for both corners) with an amplitude of 2.5 mm and frequency sweep from $1 - 14$ Hz for a duration of 10 s. The objective was comfort ($\Gamma_0^1 = 1, \Gamma_0^2 = 0$).

4.4.2 Comparative analysis

A comparative analysis was conducted b/w ACADO-qpOASES NMPC controller and the proposed parallelized pNMPC method. The basis for conducting this study was to analyze computational time (CT), normalized closed-loop objective (NCLO), i.e. with respect to nominal damping (see Section 4.4.3) and feasibility analysis (FA), i.e. constraint satisfaction. The study involved a gradual increase in the complexity parameter and the aforementioned criteria were recorded. The complexity parameter for ACADO-qpOASES NMPC controller was the number of Newton iterations (N_s) i.e. `IMPLICIT_INTEGRATOR_NUM_ITS` (see [Houska et al. 2009]) and other settings were set to default and the number of discretization points $\{n_{\phi_l}, n_{\phi_r}\}$ (left/right corner of the vehicle) for parallelized pNMPC method. The road profile utilized was the same as mentioned in Section 4.4.1 and the objective was comfort ($\Gamma_0^1 = 1, \Gamma_0^2 = 0$).

The ✗and ✓indicates the infeasibility and feasibilty of the imposed constraints for the system. The recorded readings are listed in Table 4.2 and 4.3 and from the tables, it is evi-

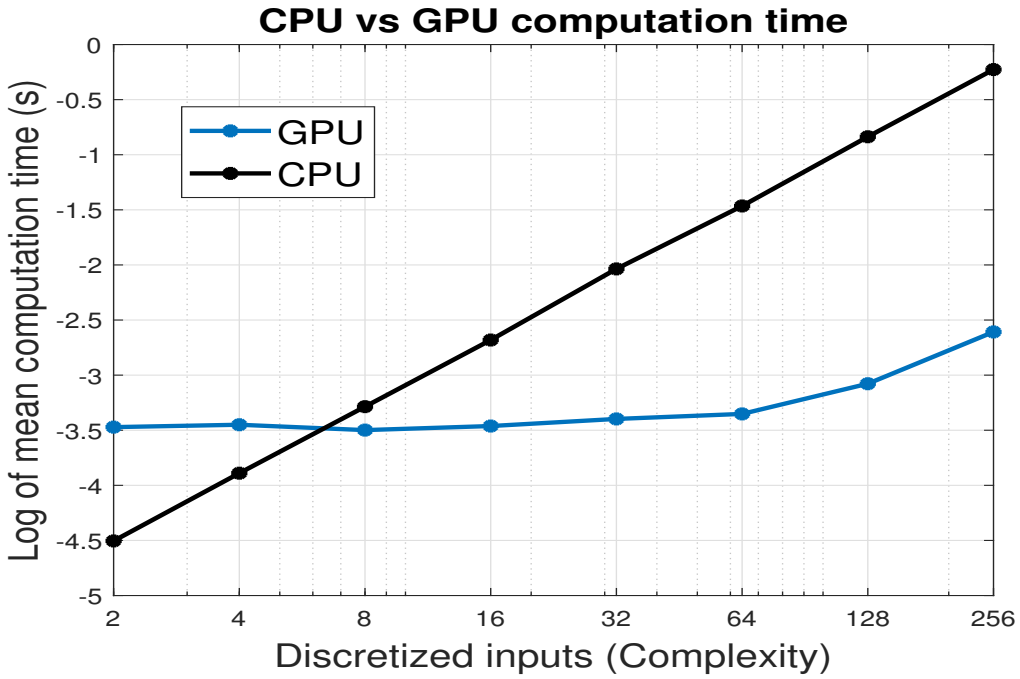

Figure 4.1: CPU vs GPU pNMPC computation time

Table 4.2: ACADO-qpOASES NMPC controller

N_s	FA	Mean CT (ms)	Max CT (ms)	NCLO
5	✗	0.70	1.6	—
10	✗	0.95	2.2	—
15	✓	1.3	2.9	0.6484
20	✓	1.5	3.0	0.6412
25	✓	1.9	3.9	0.6317

dent that the proposed parallelized pNMPC method performs better than ACADO-qpOASES NMPC controller in all criteria.

4.4.3 Road profile simulation test - Ride handling

The experiment involved a lopsided bump road profile with an amplitude of $4mm$ with ride handling objective ($\Gamma_0^1 = 0, \Gamma_0^2 = 1$) and the duration of simulation was $10s$. The road profile is illustrated in Fig. 4.2. Three different methods were adopted to analyze the performance of the system which are 1) Nominal damping, 2) ACADO-qpOASES NMPC controller and 3) Parallelized pNMPC method. The settings for each method are listed below

1. Nominal damping (Passive): The PWM-DC input for both left and the right corner was set to a constant value 0.225, i.e. the case when the damper control is switched off and

Table 4.3: Parallelized pNMPC method

$\{n_{\phi_l}, n_{\phi_r}\}$	FA	Mean CT (ms)	Max CT (ms)	NCLO
{2, 2}	✓	0.35	0.62	0.4679
{4, 4}	✓	0.35	0.60	0.4640
{8, 8}	✓	0.35	0.61	0.4646
{16, 16}	✓	0.36	0.55	0.4588
{32, 32}	✓	0.41	0.67	0.4568

the system is passive.

2. ACADO-qpOASES NMPC controller: The corresponding best setting was utilized, **Integrator** - 4th order Runge Kutta integrator, **QP solver** - qpOASES, **Hessian approximation** - Gauss-Newton, **Discretization** - Multiple shooting, **Discretization intervals** - 5 and for the rest, default parameters were utilized.

3. Parallelized pNMPC controller: The number of discretization points (complexity) was $\{n_{\phi_l}, n_{\phi_r}\} = \{8, 8\}$ for both left and right corners of the vehicle.

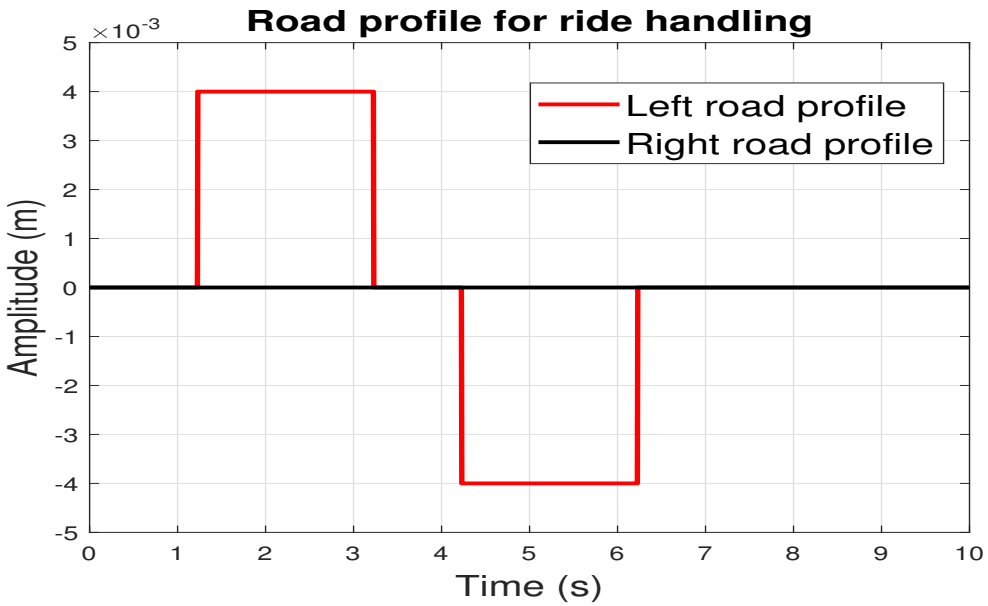

Figure 4.2: Road profile for ride handling

Fig. 4.3 illustrates the roll angle (θ) and it is clearly seen that the overshoot in the nominal damping is larger than the other two methods. In comparing the ACADO-qpOASES NMPC and proposed parallelized pNMPC method, the latter performs slightly better than the former. However the key aspects here are the computation time and the PWM-DC input profile.

The histogram of computation time for both the methods are displayed in Fig. 4.4. The mean computation time are 0.28 ms and 0.48 ms and maximum computation time are

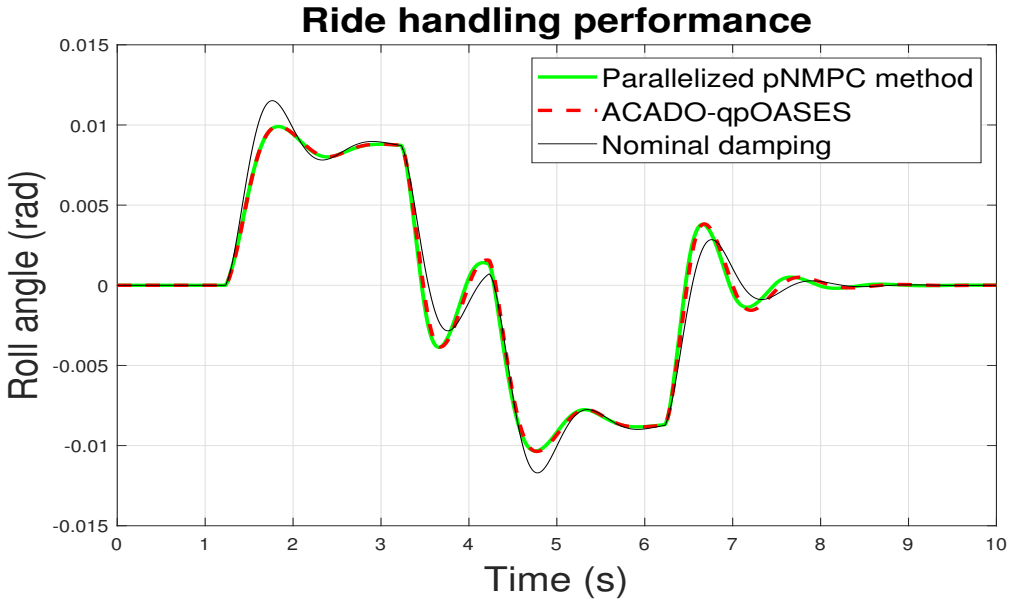

Figure 4.3: Ride handling - roll angle θ

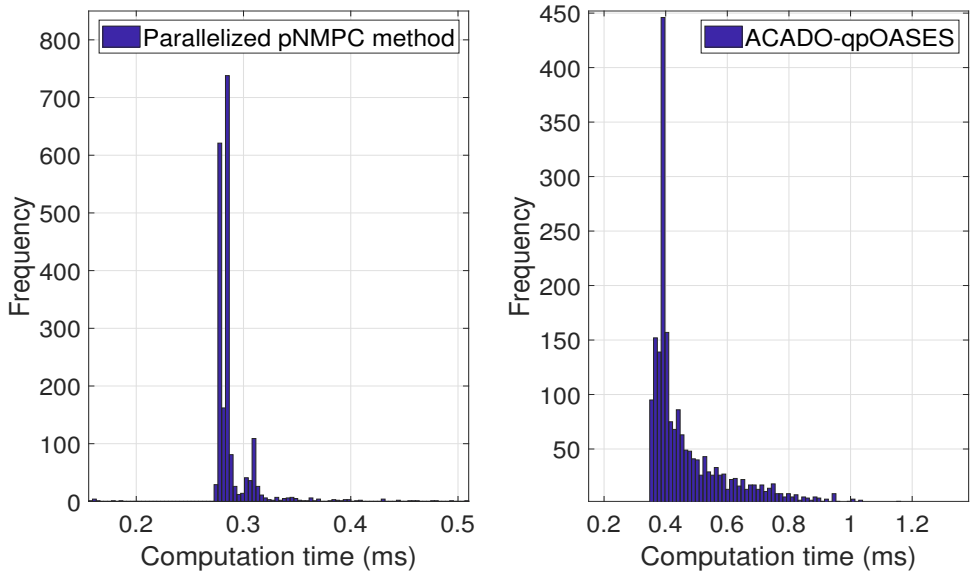

Figure 4.4: Computational time distribution

0.51 ms and 1.6 ms for parallelized pNMPC method and ACADO-qpOASES NMPC controller respectively. A significant reduction in the computation time is observed for nearly the same performance of the system. The PWM-DC input for parellelized pNMPC method and ACADO-qpOASES NMPC controller are displayed in Fig. 4.5 and Fig. 4.6. A key point to be noted is that the input profile rendered by ACADO-qpOASES NMPC controller is unrealistic and cannot be practically injected into the real SA suspension system (INOVE test platform) due to the limitations of the set of operational input levels of the PWM-DC signal, i.e. quan-

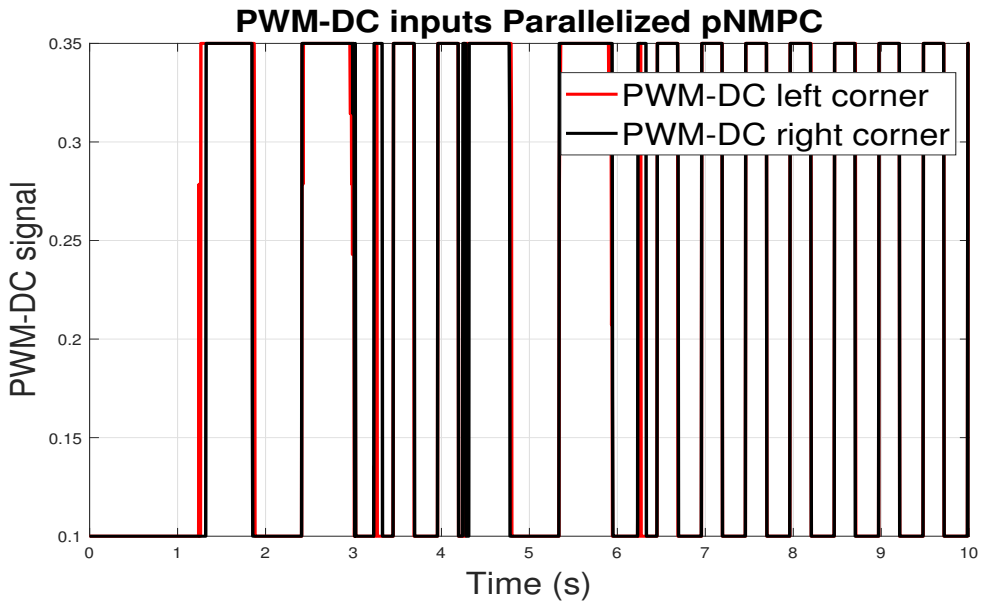

Figure 4.5: PWM-DC input for Parallelized pNMPC method

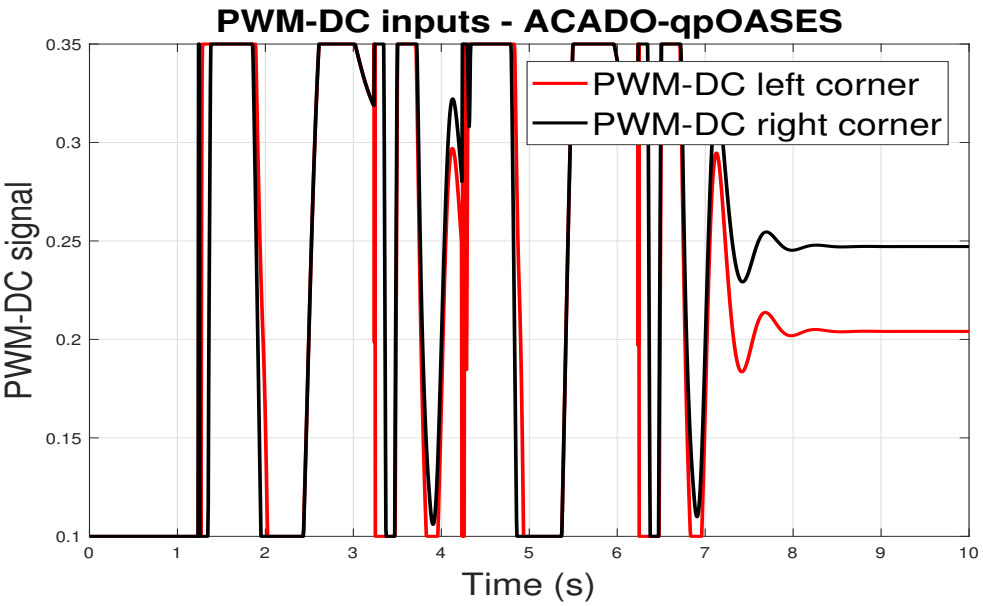

Figure 4.6: PWM-DC input for ACADO-qpOASES NMPC controller

tized levels of operation. From practical experience obtained by working on the INOVE test platform, only quantized values of PWM-DC inputs provide a significant difference in the system's output. The input control profile obtained from the ACADO-qpOASES controller in Fig. 4.6, assumes the inputs to be in continuum, which in practice might not render the required performance for the system. On the contrary, this can be easily included into the proposed parallelized parameterized NMPC (pNMPC) method and this is illustrated in Fig. 4.5.

4.5 Scenario-stochastic pNMPC scheme for control of semi-active suspension system

4.5.1 Mathematical model notations

Let $\mathbf{X} = [z_s, \theta, z_{us,l}, z_{us,r}, \dot{z}_s, \dot{\theta}, \dot{z}_{us,l}, \dot{z}_{us,r}, z_{r,l}, z_{r,r}, v_x]$ denotes the state vector, $\mathbf{U} = [\phi_l, \phi_r]$ denotes the input vector and $\mathbf{\Xi} = [\xi_l, \xi_r]$ denote the disturbance vector. $\mathbf{X} \in \mathbb{R}^{11}$, $\mathbf{U} \in \mathbb{R}^2$ and $\mathbf{\Xi} \in \mathbb{R}^2$, then the half car model in equation (4.2) can be compactly expressed with

$$\dot{\mathbf{X}}(t) = f(\mathbf{X}(t), \mathbf{U}(t), \mathbf{\Xi}(t), a_x(t)) \tag{4.12}$$

It is important to note that all the state variables are assumed to be measured and this also includes the road profile $[z_{r,l}, z_{r,r}]$ at every sampling period.

4.5.2 SS-pNMPC design requirements

4.5.2.1 Objective requirements for SS-pNMPC controller

The objective design in time domain can be briefly classified as a) comfort and b) ride handling objective [Savaresi et al. 2010].

1. **Comfort objective**: The goal of the comfort based objective is to minimize the vertical acceleration of the chassis ($\ddot{z}_s$), governed by equation (4.2). The comfort objective for a given look ahead period T_l is expressed with

$$J^{acc}_{T_l}(\mathbf{X}(.), \mathbf{U}(.), \mathbf{\Xi}(.), a_x(.)) = \int_0^{T_l} (\ddot{z}_s(t))^2 dt \tag{4.13}$$

2. **Ride handling objective**: The goal of the ride handling objective is to minimize the roll angle (θ) of the vehicle. The ride handling objective for a given look ahead period T_l is expressed with

$$J^{roll}_{T_l}(\mathbf{X}(.), \mathbf{U}(.), \mathbf{\Xi}(.), a_x(.)) = \int_0^{T_l} (\theta(t))^2 dt \tag{4.14}$$

4.5.2.2 Constraint requirements for SS-pNMPC controller

The constraints for the SA suspension system primarily arise from the physical limitations and secondarily from performance requirements [Baumal, McPhee, and Calamai 1998]. For the MPC design, the included constraints are

1. **ER-SA damper input constraints**:

(a) Damper force constraint (Physical): The ER-SA damper force is bounded, i.e. $u_i \in [u_{min,i},\ u_{max,i}]$, $\forall i \in \{l, r\}$.

(b) PWM-DC input constraints: The operating DC for the PWM signal is constrained to $\phi_i \in [\phi_{min,i},\ \phi_{max,i}]$, $\forall i \in \{l, r\}$.

2. **State constraints**:

(a) Stroke deflection constraint (Physical): This forms a linear state constraint i.e. $z_{d,i} \in [z_{dmin,i},\ z_{dmax,i}]$, $\forall i \in \{l, r\}$.

(b) Wheel rebound constraint (Performance): This bounds the deflection position between the wheel and road. This is to ensure the tyre deflection forces are bounded i.e. $z_{us,i} - z_{r,i} \in [z_{rebmin,i},\ z_{rebmax,i}]$, $\forall i \in \{l, r\}$.

3. **Road variable assumption**: The vertical road displacement at the current time instant is assumed to be measured by means of adaptive road profile observers [Doumiati et al. 2017] or from cloud servers [Zhang et al. 2017].

4. **Longitudinal acceleration assumption**: The longitudinal acceleration a_x is assumed to be constant over the prediction horizon (T_l). In a real vehicle setting a_x is typically obtained from Inertial Measurement Unit (IMU) of the vehicle.

The mixed input-state constraint set is compactly expressed with $(\mathbf{X}, \mathbf{U}) \in \Omega_{(\mathbf{X},\mathbf{U})} \subset \mathbb{R}^{11} \times \mathbb{R}^2$ and the input constraint set is compactly expressed with $\mathbf{U} \in \Omega_{\mathbf{U}} \subset \mathbb{R}^2$. The disturbance has a probabilistic support $\mathbb{P}$ which is normally distributed with $\boldsymbol{\Xi} \sim \mathcal{N}(0, \Sigma(\mathbf{X}(t)))$, where $\Sigma(\mathbf{X}(t)) = \text{diag}(\Psi_{z_{r,l}}(\mathbf{X}(t)), \Psi_{z_{r,r}}(\mathbf{X}(t)))$. The variance is dependent on longitudinal velocity (v_x) of the vehicle, which is a state variable of the system (see Section 4.2.3).

4.5.3 SNMPC problem formulation

In summary, with the proposed objective and constraints function, the SNMPC OCP is casted as

$$\begin{aligned} J^*_{obj}(\mathbf{X}_0, \Gamma_0, a_{x,0}, \mathbf{U}^*(.)) = \min_{\mathbf{U}(.)} \mathbb{E}(\Gamma^1_0 J^{acc}_{T_l} + \Gamma^2_0 J^{roll}_{T_l}) \\ \text{s.t. } \dot{\mathbf{X}}(t) = f(\mathbf{X}(t), \mathbf{U}(t), \boldsymbol{\Xi}(t), a_x(t)) \\ \mathbf{X}(0) = \mathbf{X}_0, a_x(.) = a_{x,0}, \mathbf{U}(.) \in \Omega_{\mathbf{U}} \\ \mathbb{P}[(\mathbf{X}(.), \mathbf{U}(.)) \notin \Omega_{(\mathbf{X},\mathbf{U})}] \leq \eta \end{aligned} \tag{4.15}$$

where $\mathbf{X}_0$ represents the initial state vector and $a_{x,0}$ represents the constant longitudinal acceleration over the prediction horizon. $\Gamma_0 = [\Gamma^1_0, \Gamma^2_0]$ with Γ^1_0 and Γ^2_0 being the convex combination weights between the two objectives comfort and ride handling respectively. The stochastic measure adopted for the total objective is the expectation operator ($\mathbb{E}$). The mixed state-input constraints ($\Omega_{(\mathbf{X},\mathbf{U})}$) are encapsulated in a probabilistic framework with a finite level of violation $\eta \ll 1$. Once the optimal input trajectory is computed, the first control action is injected into the system and this procedure is repeated in receding horizon fashion.

It is also important to note that the two objectives are conflicting in nature (See [Savaresi et al. 2010]). In this work, a constant control input profile is assumed over the control horizon. As the semi-active suspension system is inherently stable the foregoing assumption is apposite for performance requirements for fast sampled systems.

4.5.4 SS-pNMPC method

4.5.4.1 Method description

The method is an extension of the work proposed in [Rathai et al. 2018] and [Rathai, Sename, and Alamir 2019], where the stochastic road model is accounted in the model dynamics. The fundamental idea of the method is to parameterize the input set ($\Omega_{\mathbf{U}}$) into finite number of control inputs (similar to finite control set MPC) and for each parameterized control input, the SMPC is solved by performing MC simulations with several scenarios of the road profile. From the simulations, the expected objective function is numerically obtained by empirical mean and a probabilistic constraint violation certificate (PCVC) is numerically obtained by computing the ratio between constraint violation and scenarios generated. The optimal input is selected by finding the minimum expected objective along with the consideration of PCVC less than or equal to the specified level (η). If none of the input satisfies the above criteria, then the input with the least PCVC is selected. The pseudo-code for the method is shown in Algorithm 2. The algorithm is explained in the following part. It is important to note is that the pseudo-code is an abstraction of the parallel programming paradigm. Several threads are spawned in parallel to achieve this task (See [Sanders and Kandrot 2010]).

Explanation of Implementation:

1. **Initialization, I/Os and Syntax declaration**:
   (a) The data initialization step sets the parameter values for the half-car model and the constraints.
   (b) The input variables are $\mathbf{X}_0, \Gamma_0, a_{x,0}, n_g, \gamma$, where n_g and γ are the number of input parameterization and number of scenarios respectively. The output variable is $\mathcal{U}_{i^*}$, i.e. the optimal input vector injected into the system.
   (c) The qualifiers __CPU__ and __GPU__ denotes the function operation in CPU and GPU respectively. The entry point is **SSpNMPC** function.
2. **SSpNMPC** function:
   (a) In line 2, the input set $\Omega_{\mathbf{U}}$ is finitely discretized into n_g points and collected in the grid $\mathcal{U}_{1:n_g}$.
   (b) From line 3-8, the **parfor** i.e. parallel **for** function is utilized to dispatch each and every discretized input from $\mathcal{U}_{1:n_g}$ to GPU and for each input the **SIM** function is utilized to conduct N_s number of MC simulations. The respective objective function $\texttt{Obj}_i[l]$ and constraint violation $\texttt{CV}_i[l]$ for the l^{th} MC simulation and i^{th} input are

Implementation 2 SS-pNMPC pseudocode

Data Initialization: Model/Constraint parameters
Input: $\mathbf{X}_0, \Gamma_0, a_{x,0}, n_g, \gamma := N_s \times N_\tau$
Output: $\mathcal{U}_{i^*}$

function __CPU__ **SSpNMPC**$(\mathbf{X}_0, \Gamma_0, a_{x,0})$
  $\mathcal{U}_{1:n_g} \leftarrow \texttt{grid}(\Omega_{\mathbf{U}}, n_g)$
  parfor $i \leftarrow 1 : n_g$ **do**
    parfor $l \leftarrow 1 : N_s$ **do**
      $(\texttt{Obj}_i[l], \texttt{CV}_i[l]) \leftarrow \textbf{SIM}(\mathbf{X}_0, \Gamma_0, a_{x,0}, \mathcal{U}_i)$
    end parfor
    $\texttt{EObj}_i \leftarrow \frac{\sum_l \texttt{Obj}_i[l]}{N_s}$; $\texttt{PCVC}_i \leftarrow \frac{\sum_l \texttt{CV}_i[l]}{N_s}$
  end parfor
  if $(\texttt{EObj}_{i^*} \leq \texttt{EObj}_{\forall i \setminus \{i^*\}}$ & $\texttt{PCVC}_{i^*} \leq \eta)$ **then**
    $i^* \leftarrow \texttt{indexmin}(\texttt{EObj})$
    $J^*_{obj} \leftarrow \texttt{EObj}_{i^*}$
  else
    $i^* \leftarrow \texttt{indexmin}(\texttt{PCVC})$
  end if
  return $\mathcal{U}_{i^*}$
end function
function __GPU__ **SIM**$(\mathbf{X}_0, \Gamma_0, a_{x,0}, \mathcal{U}_i)$
  $\texttt{CV} \leftarrow 0$; $\texttt{Obj} \leftarrow 0$
  for $j \leftarrow 0 : N_\tau$ **do**
    $\texttt{BO} \leftarrow 1$; $\texttt{TObj} \leftarrow 0$; $\mathbf{X}_{em} \leftarrow \mathbf{X}_0$
    for $t_{loop} \leftarrow 0 : h : T_l$ **do**
      $\boldsymbol{\Xi} \sim \mathcal{N}(0, \Sigma(\mathbf{X}_{em})h)$
      $\mathbf{X}_{em} \leftarrow \mathbf{X}_{em} + hf(\mathbf{X}_{em}, \mathcal{U}_i, \boldsymbol{\Xi}, a_{x,0})$
      if $(((\mathbf{X}_{em}, \mathcal{U}_i) \notin \Omega_{(\mathbf{X},\mathbf{U})})$ & $\texttt{BO})$ **then**
        $\texttt{CV} \leftarrow \texttt{CV} + 1$; $\texttt{BO} \leftarrow 0$
      end if
      $\texttt{TObj} \leftarrow \texttt{TObj} + h(\Gamma_0^1 J_{l_l}^{acc} + \Gamma_0^2 J_{l_l}^{roll})$
    end for
    $\texttt{Obj} \leftarrow \texttt{Obj} + \texttt{TObj}$
  end for
  return $\{\frac{\texttt{Obj}}{N_\tau}, \frac{\texttt{CV}}{N_\tau}\}$
end function

obtained as return arguments. Finally, the expected objective function $\texttt{EObj}_i$ and probabilistic constraint violation certificate $\texttt{PCVC}_i$ are numerically obtained for the i^{th} input in $\mathcal{U}_{1:n_g}$.

(c) The lines 9-15, looks for the optimal input i^* index with minimum expected objective i.e. $\texttt{EObj}_{i^*}$ and also satisfies the violation condition $\texttt{PCVC}_{i^*} \leq \eta$. If the aforementioned condition is true, i^* is obtained from the function **indexmin** over the vector `EObj`, otherwise, i^* with the least violating constraint is obtained from the function **indexmin** over the vector `PCVC`. Once the index for the optimal input (i^*) is obtained, $\mathcal{U}_{i^*}$ is injected to the system. It is important to note that the optimal objective (J^*_{obj}) and input ($\mathbf{U}^*(.)$) in (4.15) are $\texttt{EObj}_{i^*}$ and $\mathcal{U}_{i^*}$.

3. **SIM** function:

(a) The lines from 19 to 30 executes the MC simulation for the i^{th} input - $\mathcal{U}_i$. The simulation consists of two **for** loops. The outer loop runs the MC simulation N_τ ($N_\tau = \frac{\gamma}{N_s}$) times to maximize the efficiency of CUDA cores to simulate several scenario instances. The inner loop is dedicated for problem (4.15), where the stochastic ODE is simulated by means of Euler-Maruyama integration and in due course of simulation, if the constraints are violated, the counter variable `CV` registers the violation and also, the objective is numerically approximated by means of Riemann sum. The inner loop objective is stored in a temporary variable `TObj` and the outer loop objective `Obj` is a N_τ times accumulation of the inner loop objective. It is also important to note that the total number of MC simulations for each input is $\gamma = N_s \times N_\tau$ and the total number of simulations for all the inputs is $N = \gamma \times n_g$.

(b) The line 31 returns the average objective and constraint violation certificate with respect to N_τ simulations.

4.5.4.2 Scenario generation

As the SNMPC optimization problem in (4.15) is numerically solved by means of MC simulations, it is of paramount importance to sample enough number of scenarios to approximate the solution for the problem. Thanks to the theory of statistical learning and randomized algorithms which provides a systematic framework to derive the minimum number of scenarios required to achieve a probabilistic bound over the quality of the solution. The following content of this section is a summary of the core result presented in the seminal paper [Vidyasagar 2001]. To present the result more concretely, the context of the problem is laid down firmly with the following setup. Consider the task of approximating a stochastic function defined by

$$h(\mathbf{y}) = \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_\mathbf{x}}[\psi(\mathbf{x}, \mathbf{y})] \tag{4.16}$$

where, $\mathbf{x} \in \mathcal{X}$ is distributed w.r.t. the distribution $\mathbb{P}_\mathbf{x}$ over $\mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$. Let the empirical mean approximation of the function in (4.16) be defined with $\hat{h}(\mathbf{y})$. Let the parameters for accuracy, probability level and confidence level be defined with $\epsilon, \beta, \delta \in [0, 1]$. Given these parameters, the study of randomized algorithms is to derive a lower bound for the number of

scenarios required to achieve the following requirement.

$$\mathbb{P}[\mathbb{P}[|\hat{h}(\mathbf{y}) - h(\mathbf{y})| > \epsilon] \leq \beta] \geq 1 - \delta, \ \forall \mathbf{y} \in \mathcal{Y} \tag{4.17}$$

In simple words, the above (4.17) two layered probabilistic statement implies that the difference between the empirical approximation and real function must be not greater than ϵ with a probability less than β and this must be certified with a probabilistic confidence level of $1-\delta$. Given this prelude, the task of optimizing the function $\hat{h}(\mathbf{y})$ by means of scenarios is defined by the following Theorem 4.5.1.

Theorem 4.5.1. *Choose the integers n and m defined with*

$$n \geq \frac{ln(\frac{2}{\delta})}{ln(\frac{1}{1-\beta})} \ and \ m \geq \frac{1}{2\epsilon^2} ln \frac{4n}{\delta} \tag{4.18}$$

Generate i.i.d. samples $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n \in \mathcal{Y}$ and $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m \in \mathcal{X}$ according to $\mathbb{P}_{\boldsymbol{x}}$. Define

$$\hat{h}(\boldsymbol{y}_i) = \frac{1}{m} \sum_{j=1}^{m} \psi(\boldsymbol{x}_j, \boldsymbol{y}_i), i = 1 \ldots n \ and \ \hat{h}^* = \min_{1 \leq i \leq n} \hat{h}(\boldsymbol{y}_i)$$

Then with confidence $1-\delta$ it can be said that $\hat{h}^*$ is a probably approximate near the minimum $h(.)$ to accuracy ϵ and level β. The result is universal and applicable for all family of functions [Vidyasagar 2001].

Using the results from Theorem 4.5.1, the empirical means in the SNMPC problem (4.15) both in the objective as well as the chance constraints can be numerically approximated. An important point to note is that the chance constraint can be recasted with expectation formulation with

$$\mathbb{P}[(\mathbf{X}(.), \mathbf{U}(.)) \notin \Omega_{(\mathbf{X},\mathbf{U})}] = \mathbb{E}[\mathbb{1}\{(\mathbf{X}(.), \mathbf{U}(.)) \notin \Omega_{(\mathbf{X},\mathbf{U})}\}] \tag{4.19}$$

where, $\mathbb{1}\{A\}$ represents the indicator function over the set $\{A\}$. Thus, by leveraging the results propounded in Theorem 4.5.1, the number of scenarios γ for Algorithm. 2 can be derived. Setting $\beta = \delta = 0.05$ yields the total number of input parameterization to $n_g = 64$. Utilizing the previous result and setting $\epsilon = 0.125$, the number of scenarios ($m = \gamma$) for each parameterized input is $\gamma \approx 270$. In total, the GPU simulates approximately 270×64 simulations over the prediction horizon at every sampling period. In this study ϵ is assumed to be fixed, however it is really important to note that the parameter ϵ is implicitly a function of the computing resource such that the method is RT doable.

4.5.5 Results and simulations

4.5.5.1 RT embedded tests on NVIDIA boards

The test was conducted to study the RT viability of the proposed method and it involved execution of the proposed SS-pNMPC method (Algorithm. 2) with the aforementioned scenario parameters on multiple NVIDIA embedded boards as listed in Table 4.4. The key

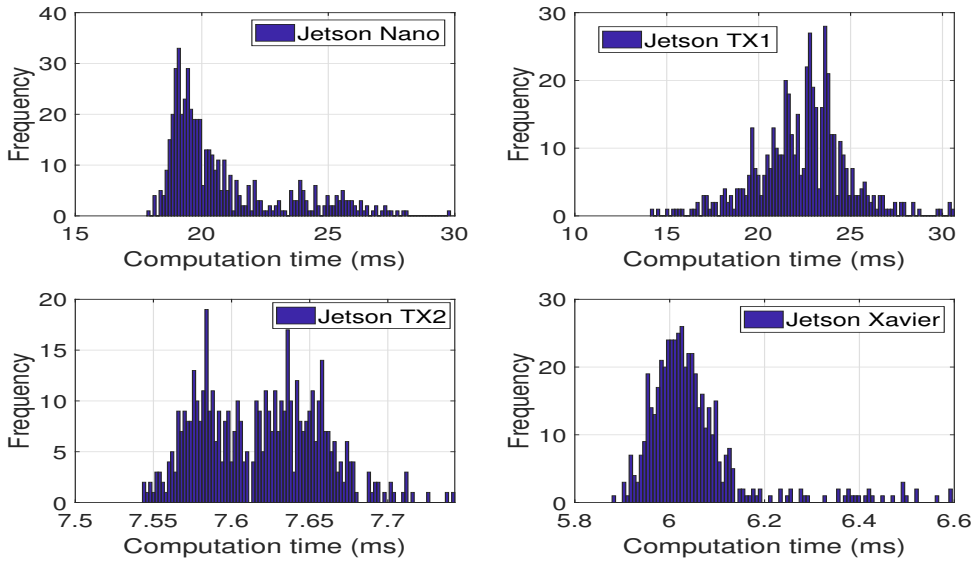

Figure 4.7: Histogram of computation time on different embedded GPU plaftforms

metrics which define the H/W performance of the board are a) Number of CUDA cores and b) Compute capability (CC) of the board i.e. the underlying architecture of the board. The performance of the method is gauged with the mean computation time (CT) and maximum computation time (CT) in terms of milliseconds (ms). The method was programmed in C++ using CUDA libraries.

Table 4.4: NVIDIA boards H/W configuration and results

Board	#Cores	CC	Mean CT (ms)	Max CT (ms)
Jetson Nano	128	5.3	20.81	29.83
Jetson TX1	256	5.3	22.37	30.64
Jetson TX2	256	6.2	7.62	7.74
Jetson Xavier	512	7.2	6.04	6.59

As from Table 4.4, it clearly evident that with increase in the hardware configuration a considerable reduction in the mean and maximum CT is observed. However, the natural sampling period (T_s) of the platform is 5 ms and the best of embedded NVIDIA boards - Jetson Xavier hovers around 6 ms. Despite the boards doesn't compute the optimal input within T_s, the scenario parameters ϵ, β, δ (mentioned in Section 4.5.4.2) can be tweaked to meet RT requirements, however the quality of the solution would be deteriorated. The histogram for the computational time for different NVIDIA embedded platforms is shown in Fig. 4.7.

4.5.5.2 Pareto optimality of objectives

As mentioned in Section 4.5.3, the two objectives (comfort and ride handling) are conflicting in nature. Thus, the convex weights Γ_0^1 and Γ_0^2 ought to be tuned in order to strike a proper balance between the two objectives. To study the effects on variations of the convex weights, a Pareto optimality analysis was performed for several simulations. The simulation involved the vehicle moving at a constant velocity $v_x = 20m/s$ on a IS0-C road profile for a duration of 10s. In Fig. 4.8, the Pareto optimal front is plotted and this aids in detecting the right weights for the best results.

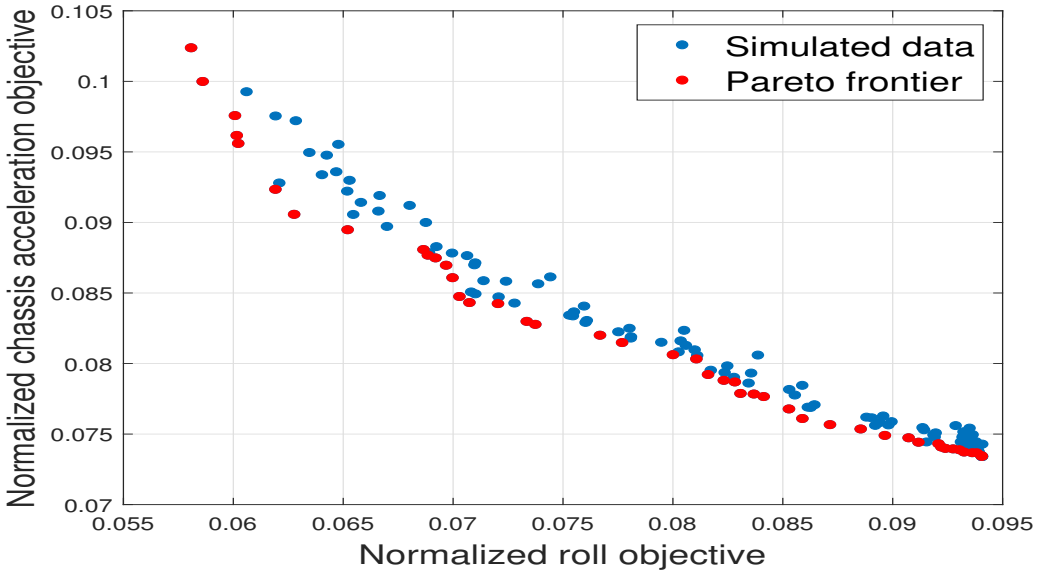

Figure 4.8: Pareto optimal front between comfort and ride handling objective

4.5.5.3 Road profile simulation test

The simulation test involved the vehicle moving with longitudinal acceleration and velocity profile as shown in Fig. 4.9 and Fig. 4.10 on all the ISO road profiles. These profile correspond to the scenario where the vehicle is subjected to sudden braking. The convex weights used in the study are $\Gamma_0^1 = 0.75$ and $\Gamma_0^2 = 0.25$, shown in Fig. 4.8. Three passive controllers - $\underline{u_i}$, $\overline{u_i}$ and nominal $u_{n,i} = \frac{\underline{u_i}+\overline{u_i}}{2}$, $\forall i \in \{l, r\}$ were chosen to compare the performance with the proposed SS-pNMPC controller.

The RMS values of the chassis acceleration for all the systems are listed in Table 4.5. It is clearly evident that the RMS value of the proposed method is less than the nominal passive system and in par with the minimum passive system (Comfort design). For illustration purpose, Fig. 4.12 displays the chassis acceleration plot for ISO-C road profile. Also, at the same time, the proposed method minimizes the ride handling objective judiciously in comparison with other passive systems. For illustration purpose, Fig. 4.11 displays the roll

angle plot for ISO-E road profile.

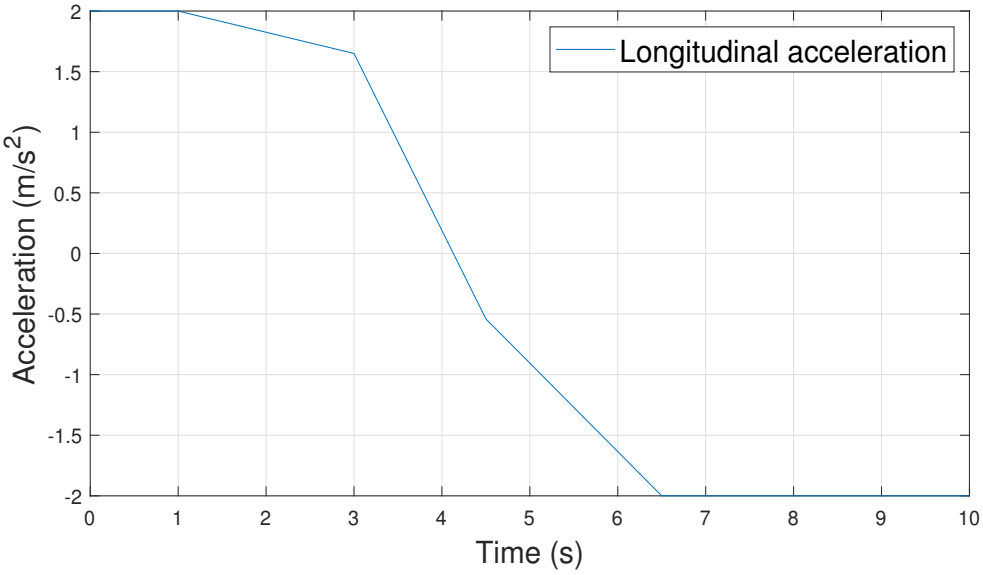

Figure 4.9: Longitudinal acceleration profile (a_x)

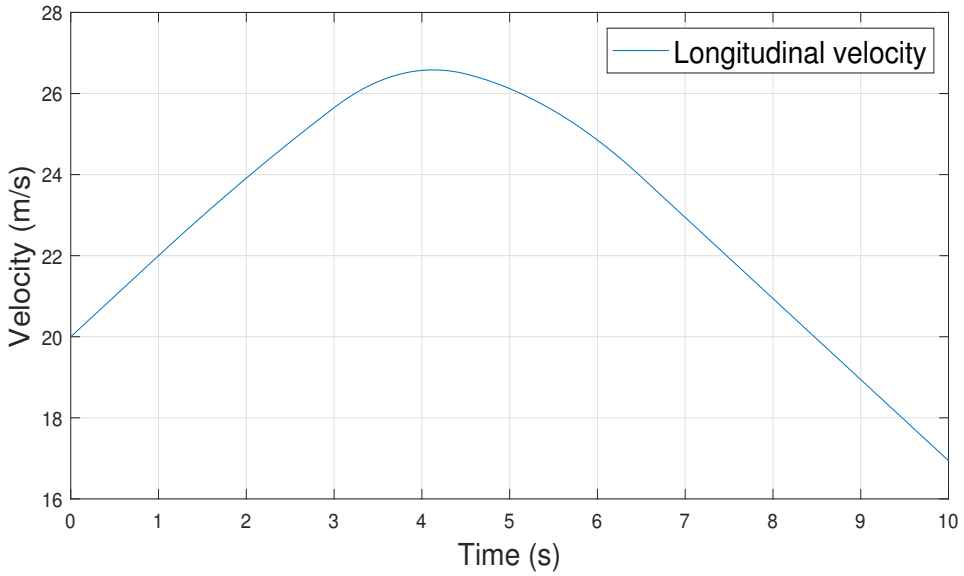

Figure 4.10: Longitudinal velocity profile (v_x)

4.6 Conclusions

This chapter has scaled the pNMPC scheme proposed in the previous chapter (Chapter 3) by parallelizing the method on GPUs. The utilization of GPU could be perceived as an instance

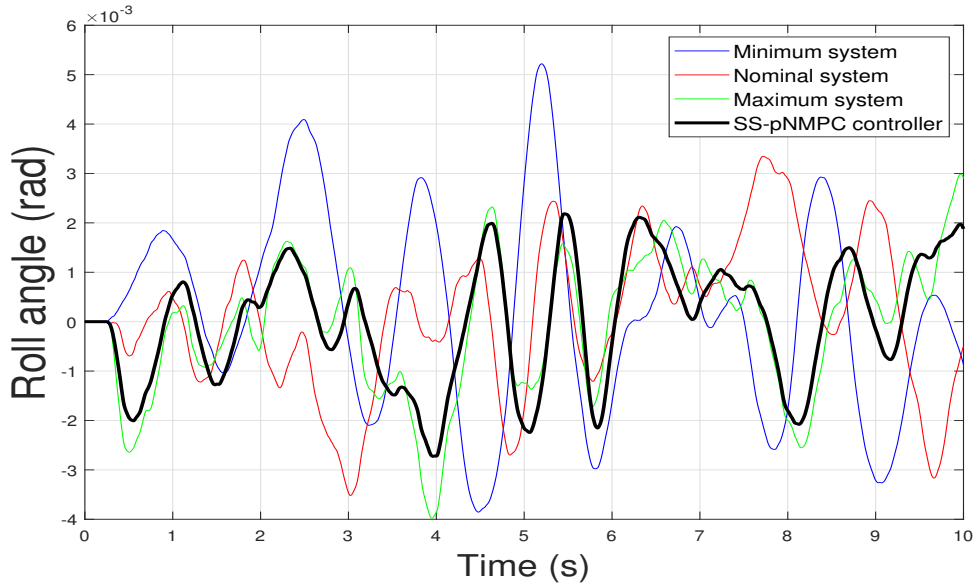

Figure 4.11: Roll angle (θ) plot for different controllers for ISO-E road profile

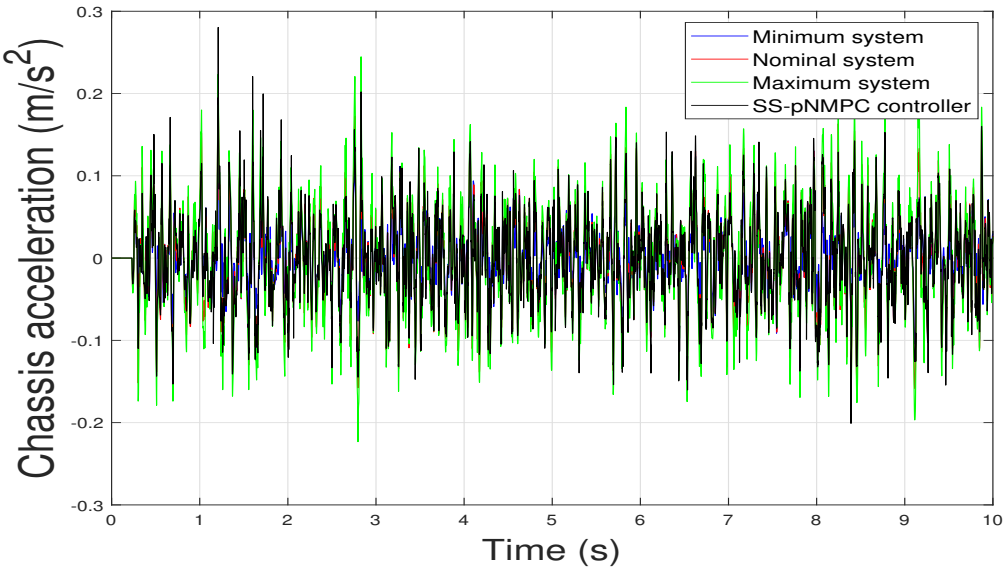

Figure 4.12: Chassis acceleration ($\ddot{z}_s$) plot for different controllers for ISO-C road profile

of a massively multi-core processor for implementing the proposed parallelized version of the pNMPC controller. In this chapter, two methods has been proposed for control of vertical dynamics of a half-car vehicle through ER semi-active suspension system. The first method proposes the parallelized pNMPC method where the road model is not accounted in the pNMPC problem formulation. In the second method, the stochastic road model in included into the pNMPC problem formulation and this method is termed the SS-pNMPC scheme. The methods were both tested in simulation and as well as on NVIDIA embedded boards

Table 4.5: RMS value of chassis acceleration

ISO	SS-pNMPC(ms^{-2})	$u_n(ms^{-2})$	$\underline{u}(ms^{-2})$	$\overline{u}(ms^{-2})$
A	0.012	0.014	0.011	0.017
B	0.023	0.029	0.021	0.034
C	0.049	0.058	0.042	0.076
D	0.093	0.118	0.087	0.138
E	0.194	0.234	0.178	0.277

to validate and verify the feasibilty of the proposed approach. For future implementation of the parallelized pNMPC scheme under the purview of the EMPHYSIS project, the eFMU container could be potentially designed to be amenable on GPUs or any multi-core processors by the likes of GPUs. Thus, by this feature, several simulations of the eFMU models could be conducted in parallel and the parallelized pNMPC method could be seamlessly implemented for RT control.

Part III
pNMPC - A code generation software tool for implementation of derivative free pNMPC scheme for embedded control systems

Chapter 5

pNMPC code generation tool

Contents

5.1 Introduction

5.1.1 Prelude

Over the last decade, there has been a tremendous amount of effort in the nonlinear MPC (NMPC) and optimal control community to build real-time (RT) operable software (S/W) for embedded control of engineering systems. There have been several contributions on code generation based NMPC/Optimal control toolboxes, however, by and large most of the existing toolboxes shares a common thread which is the need for derivatives of objective, constraints and dynamical model for solving the underlying optimization problem for the NMPC controller. In cases where the derivatives are not available and automatic differentiation may not be feasible, the only recourse is to compute the derivatives using numerical methods such as finite difference methods. However, numerically computing the derivatives is highly prone to error and as well as computationally taxing which could encumber RT feasibility. This brings the need for implementation of a complete derivative free NMPC package to address situations as mentioned before. As of today, the only derivative free NMPC toolbox available on market is the PDF-MPC package [Alamir 2017] which uses MATLAB as the front end and MATLAB coder on top to deploy the code into embedded devices. The pNMPC code generation software tool presented in this chapter is highly influenced from the aforementioned tool with improved support and features to cater the open source community and embedded control programmers. The pNMPC S/W is available in GitHub repository [Rathai 2020].

5.1.2 Motivation

Optimization solver plays a pivtol role in solving the underlying OCP for the MPC controller. The entire edifice of MPC controller reposes on the underpinning optimization problem. Typically, any state of the art optimization problem can be broadly classified into three types i) Zero order methods, ii) First order methods and iii) Second order methods. The order determines the type of information that the optimization solver can query to determine the solution, i.e. for zero order methods, only the function value can be obtained, for first order methods, the function value and it's first order derivatives (Jacobian) can be obtained and for second order methods, the function value, the first (Jacobian) and second order derivatives (Hessian) can be obtained or approximated. In optimization parlance, this is demarcated by classifying zero-order methods as derivative free methods or black box optimization (BBO) problem and the rest as derivative based methods. The focus of this chapter circles around implementation of MPC controller using derivative free methods. Some of the compelling reasons for a need of a BBO based MPC are

- It is not uncommon in the real world industrial application, that the plant model and parameters are secured due to intellectual property (IP) rights and perhaps this sensitive information is never divulged even to the internal engineers working with the system. Thus, the knowledge of the system is completely obscured from the control engineer and

renders nearly impossible to design a MPC controller that predominantly relies upon the derivative information.

- In cases where the model of system exists only as computer codes or in binary format (executable files), it becomes very cumbersome to obtain accurate derivatives by numerical methods. To exacerbate, if the code involves several break, if-else or goto statements, it is highly impractical to obtain the derivatives using automatic differentiation methods. Case in point, this is common in the functional mock-up interface (FMI) standard [Blockwitz et al. 2012], which is used for model exchange and co-simulation purposes. Also, this is the main framework of the EMPHYSIS project [EMPHYSIS 2017].
- Even when the functional form is available to the control engineer, at times, computation of derivatives can be computationally too expensive or noisy and this could preclude from practical implementation of MPC controller for fast sampled systems. Also, in cases where the function is discontinuous and not differentiable, the derivative based methods can lead to undefined behavior.
- In today's world, with availability of deluge of data and increased computation power, data-driven modeling has challenged the pre-existing notions of first principles modeling and perhaps it wouldn't be too much of a stretch to consider the former superseding the latter in a few decades now. Machine learning models such as neural networks, Gaussian processes etc. provide excellent empirical approximations of the underlying dynamical systems and typically one can right away utilize these models for control design. It is also important to note that many of these models are not differantiable by definition.

As per the cited reasons, it is certainly not an unreasonable requirement but, also, a matter of paramount importance to address the control problem by designing a derivative free MPC scheme. In this chapter, a BBO based parameterized NMPC (pNMPC) S/W tool is proposed to circumvent the aforementioned issues. The term parameterized refers to the parameterization of the control input [Alamir 2006]. The parameterization serves two purposes a) reduces the computational burden for the optimization solver and b) design of a parsimonious control input profile. It is also important to note that the potential of the pNMPC method as well as the toolbox relies upon the astuteness of the control engineer to model the input profile with efficient, effective and economical parameterization.

5.1.3 Related works

There has been several contributions on development of code generation based MPC toolbox for embedded systems. The first automatic NMPC code generation toolbox traces back to AutoGen [Ohtsuka and Kodama 2002] which is based on indirect methods and the toolbox generates C code of a RT algorithm to update the initial co-state of the OCP problem. A newer version of the toolbox AutoGenU [Ohtsuka 2015] provides extension with continuation/GMRES method and a RT optimization (RTO) algorithm with interface to Maple. Ever since AutoGen, there has been significant development in building code generation based MPC toolbox for RT applications.

The μAO-MPC [Zometa, Kögel, and Findeisen 2013] toolbox provides code generation feature for embedded RT linear MPC. The underlying optimization solver for the tool involves a first order method based on Nesterov's gradient method coupled with augmented Lagrangian method. The toolbox provides front end interface to Python and the generated C code is implemented on embedded systems.

The CVXGEN [Mattingley and Boyd 2012] toolbox provides code generation feature for an embedded convex optimization problem. The front end involves a quadratic programming (QP) modeling framework, where the user enters the optimization variables, dimensions, parameters, objective, inequality/equality constraints etc. and the output of the program is an optimized C-code of the optimization problem. The toolbox exploits the structure of the QP problem such as sparsity, optimal Karush–Kuhn–Tucker (KKT) matrix factorization etc. to speed up the computation. Despite the toolbox is agnostic to any specific application, the method has been mostly used for MPC controller design. The only limitation is that the method applies only for a convex QP problem.

FiOrdOs [Ullmann 2011b] is an automated C-code generation MATLAB toolbox for first order methods for parameteric convex programs. The only limitation of the toolbox is that it can only handle only convex QPs with simple convex sets on which projections can be evaluated at low cost.

The ACADO toolbox [Houska, Ferreau, and Diehl 2011] provides an automatic C-code generation feature for implementation of NMPC controller based on real-time iteration (RTI) using Gauss-Newton method. The toolbox is implemented in C++ and also provides interface with MATLAB. The front end involves a modeling framework to model the states, inputs, parameters, differential equations, equality/inequality constraints, integrator parameters, solver parameters etc. and the back end involves the optimization solver, which is outsourced to other third party S/W such as qpOASES [Ferreau et al. 2014b], QPdunes [Kouzoupis et al. 2015], FORCES [Zanelli et al. 2020], IPOPT [Wächter and Biegler 2006b]. The final output is the respective C codes of the model, constraints, integrator sensitivities etc.

The VIATOC toolbox [Kalmari, Backman, and Visala 2015] which shares similar syntax as ACADO toolbox, however the NMPC problem is specifically tailored to an optimization solver based on projected gradient method. The toolbox also provides automatic code generation feature similar to ACADO. The toolbox was programmed in C++ environment.

The Multi-Parameteric toolbox (MPT) [Kvasnica, Rauová, and Fikar 2010] provides code generation feature for implementation of explicit MPC. The method uses QP parameteric programming method and precomputes the optimal control input over the partitioned polyhedral regions of the constraint space. The states are considered as the parameters and the optimal feedback law given current state of the system is finally computed by performing a table traversal algorithm using binary tree search method and the optimal control input is recovered. The method is suitable for linear MPC problems with low state, input dimensions and horizon length. The toolbox is programmed in MATLAB and uses MATLAB coder to generate C-code for embedded systems.

The SPLIT toolbox [Shukla et al. 2017], a C code generation tool for linear MPC problems uses operator splitting methods and also, the toolbox is capable of generating both software as well as hardware specific code such as FPGA boards. The toolbox provides a MATLAB front-end interface.

The Optimization Engine toolbox (OpEn) [Sopasakis, Fresk, and Patrinos 2020] is a code generation tool for RT embedded non-convex optimization problems. The OpEn tool combines proximal averaged Newton-type method with penalty and augmented Lagrangian methods. The method was implemented in Rust programming language and provides front end interfaces with MATLAB, Python, C/C++ or via a TCP socket.

The ParNMPC [Deng and Ohtsuka 2018] is a parallel code generation toolbox to generate C/C++ code for NMPC controller. ParNMPC utilizes the OpenMP programming framework and implements Newton-type methods across the multi-core processors.

5.1.4 Chapter contributions

The main contribution of this chapter is to present a derivative free pNMPC toolbox for embedded control of engineering systems. The salient features of the pNMPC S/W are

- The pNMPC S/W was completely programmed in C++ environment. The S/W by it's inbuilt design provides an OCP modeling framework, where the user can provide the objectives, constraints, differential equations, Black box modules, solver parameters, OCP parameters in a hassle-free way.
- The S/W is completely derivative free and as well as free of any online matrix operations such as matrix-matrix addition, multiplication, inverse computation, linear algebra solve etc.
- A one-stop control solution is provided to the end user for generation of portable, efficient, optimized and embeddable C code of the pNMPC controller. Extensions to MATLAB/Simulink environment is also provided. The inbuilt code generation module is scrupulously engineered, thus leading to more optimized code in terms of memory footprint and computation time.
- The toolbox is independent of any external libraries such as Basic Linear Algebra Subprograms (BLAS), Linear Algebra Package (LAPACK) etc. and consumes less memory footprint.
- The S/W also includes Graphic Processing Unit (GPU) based parallel implementation of the optimization solver as well as Compute Unified Device Architecture (CUDA) code generation module suited for problems with large optimization variables.
- Additional to the inbuilt modeling framework, the S/W also provides the flexibility to include and call external functions (black box functions) which can be in the form of either source code or static/dynamic libraries.

- The S/W is open sourced under GNU-LGPL v3 license.

The proposed pNMPC S/W tool was tested in Hardware in the Loop (HiL) simulation for a quarter car vertical dynamics model on dSPACE MicroAutoBoX-II (MABXII, See Section 5.7).

5.2 pNMPC theoretical background

5.2.1 pNMPC problem formulation

The goal of the pNMPC toolbox is to solve the following Optimal Control Problem (OCP) problem at every sampling period.

$$\begin{aligned} \min_{\mathbf{x}(.),\mathbf{p}(.)} \quad & \int_0^T l(\mathbf{x},\mathbf{u}(\mathbf{x},\mathbf{p},\boldsymbol{\kappa}),\boldsymbol{\kappa})\, dt + \psi(\mathbf{x}(T),\boldsymbol{\kappa}(T)) \\ \text{subject to} \quad & \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u}(\mathbf{x},\mathbf{p},\boldsymbol{\kappa}),\boldsymbol{\kappa},t),\ \forall t \in [0,T] \\ & \mathbf{u}(\mathbf{x},\mathbf{p},\boldsymbol{\kappa}) \in \mathcal{U},\ \mathbf{x} \in \mathcal{X},\ \mathbf{p} \in \mathcal{P}\ ,\forall t \in [0,T] \\ & \mathbf{x}(0) = \mathbf{x}_0,\ \mathbf{x}(T) \in \mathcal{X}_T \end{aligned} \tag{5.1}$$

where, $\mathbf{x} \in \mathbb{R}^{n_x}$, $\mathbf{u} \in \mathbb{R}^{n_u}$, $\mathbf{p} \in \mathbb{R}^{n_p}$ and $\boldsymbol{\kappa} \in \mathbb{R}^{n_\kappa}$ represents the state vector, input vector, input parameterization vector and external parameters (i.e. model parameters, set point for tracking or measured disturbances) vector respectively. The input map $\mathbf{u} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_\kappa} \rightarrow \mathbb{R}^{n_u}$ maps the states, input parameterization vector and external parameters to actual input for the system. The sets $\mathcal{X}$, $\mathcal{U}$, $\mathcal{P}$ and $\mathcal{X}_T$ denote the state constraint, input constraint, input parameterization constraint and the terminal state constraint respectively. The OCP is subjected to a set of differential equations denoted with $\mathbf{f} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_\kappa} \times \mathbb{R}_+ \rightarrow \mathbb{R}^{n_x}$ and the Lagrangian cost (stage cost) and Mayer's cost (terminal cost) are denoted with $l : \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_\kappa} \rightarrow \mathbb{R}$ and $\psi : \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_\kappa} \rightarrow \mathbb{R}$ respectively. It is important to note that the cost terms can be economic and no stipulations are enforced such as non-negativity or convexity. It is also important to note that handling the impact of the objectives or constraints design or choise of sampling period (Δt) on the closed loop behavior is left to the user.

The pNMPC S/W transcribes the above OCP formulation (5.1) into a generic constrained optimization problem. The OCP transcription is implemented by discretizing the problem in time with a finite time step of Δt. Thus, the states of the system are eliminated from the problem by simulating the differential equation over time (direct single shooting method, see Chapter 2), the integral objective is numerically approximated (Riemann sum) and the constraints are quantified at every discretized time step. Let J and h_i, $\forall i \in \{1 \dots n_c\}$ denote the transcribed objective and inequality constraints (n_c inequality constraints). The transcribed constrained optimization problem is described by the following form

$$\begin{aligned} &\min_{\mathbf{p}\in\mathbb{R}^{n_p}} \; J(\mathbf{p}) \\ &\text{s.t. } \; h_i(\mathbf{p}) \leq 0, \; \forall i \in \{1,\ldots,n_c\} \end{aligned} \tag{5.2}$$

where $\mathbf{p} \in \mathbb{R}^{n_p}$ is the optimization vector or decision variables which is also the input parameterization vector. Let the solution of the above problem (5.2) be denoted with $\mathbf{p}^*$. Utilizing the optimized input parameterization vector, the optimal input $\mathbf{u}(\mathbf{x}^*(\tau), \mathbf{p}^*(\tau), \boldsymbol{\kappa}(\tau))$ is injected into the system over the time period $\tau \in [0, \Delta t]$. Henceforth, by marching forward in time and with receipt of new state vector, this process is repeated in receding horizon manner.

5.2.2 Visualization of control parameterization

To understand the representative and expressive power of the input parameterization technique, consider a sinusoidal input parameterization over a prediction horizon $T = 2s$ as described below.

$$u(\mathbf{p}(t), t) = p_1(t) sin(2\pi p_2(t) t + p_3(t)) \tag{5.3}$$

where the time dependent input parameterization vector is $\mathbf{p}(t) = [p_1(t), p_2(t), p_3(t)]$ and

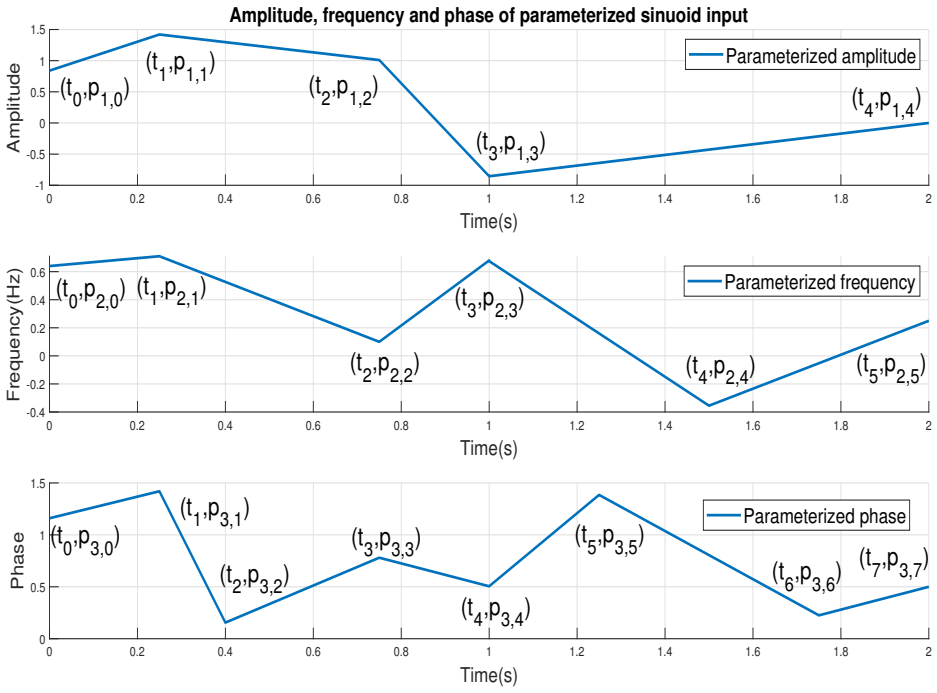

Figure 5.1: Parameterized amplitude ($p_1(t)$), frequency ($p_2(t)$) and phase ($p_3(t)$) for sinusoidal control input

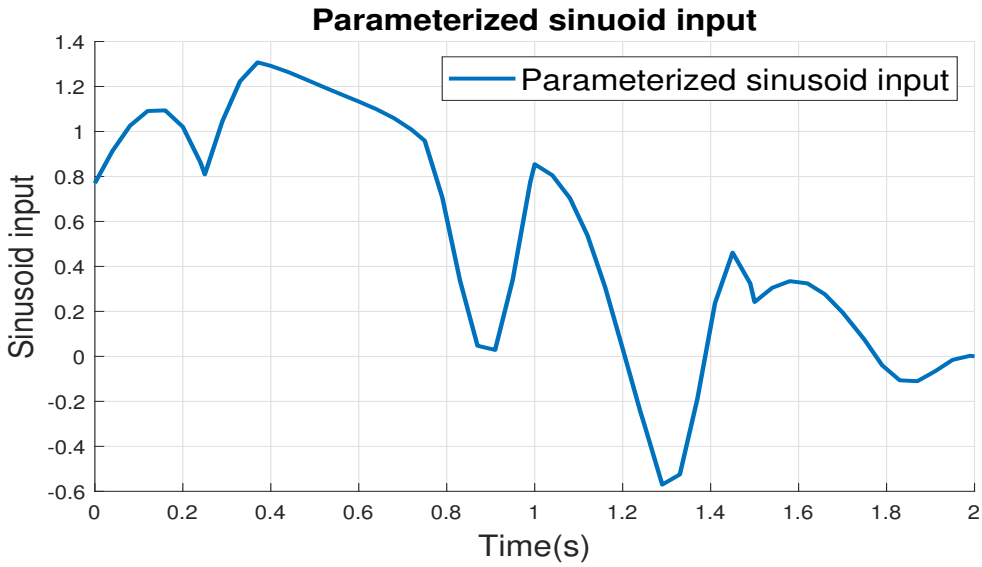

Figure 5.2: Parameterized sinusoidal control input $(u(\mathbf{p}(t), t))$

$\mathbf{p}(t) \in \mathcal{P}$. The physical interpretation of the input parmeterization vector corresponds to the amplitude, frequency and phase of the sinusoid input respectively. In order to discretize the continuous time input parameterization vector $\mathbf{p}(t)$, consider $n_{p_1}, n_{p_2}, n_{p_3}$ finite control points over the prediction horizon, then each of parameterized input at a specific time instant can be described with the pairs $\{t_i, p_{1,i}\}_{i=0}^{i=n_{p_1}-1}$, $\{t_i, p_{2,i}\}_{i=0}^{i=n_{p_2}-1}$, $\{t_i, p_{3,i}\}_{i=0}^{i=n_{p_3}-1}$. For a simplistic case, let the control points be connected with a linear interpolation curve and let $n_{p_1} = 5$, $n_{p_2} = 6$, $n_{p_3} = 8$, then the respective input parameterization profiles and the sinusoid control input is illustrated in Fig. 5.1 and Fig. 5.2.

By virtue of these parameterization points, a flexible control input profile can be designed and in the context of pNMPC toolbox, the parameterization control points are the optimization variables which ought to be ascertained from the solver. In this sinusoid example, in total there are $n_{p_1} + n_{p_2} + n_{p_3} = 19$ control points, which in this example is certainly an over parameterization of the control input. However, in practice, with a careful and deliberate placements of control points one can obviate the perils of over-parameterization of control input.

5.3 Derivative free optimization module

In this section, the underlying BBO module for the proposed pNMPC controller is discussed in detail. The content and core results summarized in this section is based on the Sequential Quadratic Programming based Black Box Optimization (SQP-BBO) method proposed in [Alamir 2017], [Alamir 2013], [Alamir 2012]. The method falls under the category of interpolation based trust region methods for derivative free optimization. The method is based on the technique of sequentially approximating the cost and constraint functions by means

of quadratic functions and at successive iterations the optimal solution is computed based on multiple trust region switch conditions. In the Subsection 5.3.2, uni-variate case of the optimization problem is discussed in detail and in Subsection 5.3.3, the method is extended for the multi-variate case.

5.3.1 Constraint reformulation (Scalarization)

Consider an optimization problem as defined below

$$\begin{aligned} &\min_{\mathbf{p}\in\mathbb{R}^{n_p}} J(\mathbf{p}) \\ &\text{s.t. } g(\mathbf{p}) \leq 0 \end{aligned} \tag{5.4}$$

where $\mathbf{p} \in \mathbb{R}^{n_p}$ is the vector of optimization variables and J and g represents the scalar cost and scalar constraint of the optimization problem respectively. In cases where there exists several inequality constraints, then all the constraints are scalarized by either of the two following forms.

Consider there exists n_c constraints acting upon the optimization problem, i.e.

$$h_i(\mathbf{p}) \leq 0,\ \forall i = \{1, \dots n_c\} \tag{5.5}$$

The two forms of constraint scalarization are

- **Form 1** - The scalar function g can be expressed as a sum over all the maximum of inequality violating constraints (if any) i.e.

$$g(\mathbf{p}) := \sum_{i=1}^{n_c} \max\{h_i(\mathbf{p}), 0\} \tag{5.6}$$

- **Form 2** - The scalar function g can be expressed as maximum over all the inequality constraints, i.e.

$$g(\mathbf{p}) := \max_{i\in\{1,2,\dots n_c\}} \{h_i(\mathbf{p}), 0\} \tag{5.7}$$

5.3.2 SQP based BBO (Uni-variate case)

Consider the optimization problem defined in (5.4) for an uni-variate case, then the optimization problem is defined with

$$\min_{p\in[p^{\min},\, p^{\max}]} J(p) \text{ s.t. } g(p) \leq 0 \tag{5.8}$$

where the optimization variable $p \in \mathbb{R}$ belongs to a bounded interval $[p^{\min}, p^{\max}]$ with $p^{\max} \geq p^{\min}$. In order to define a local quadratic approximation of a function f (f is a generic

representation of a function which can be either J or g) over an interval I, consider a variable $\alpha > 0$ with respect to a point p such that the interval I is defined with

$$I := [p - \alpha, p + \alpha] \cap [p^{\min}, p^{\max}] \tag{5.9}$$

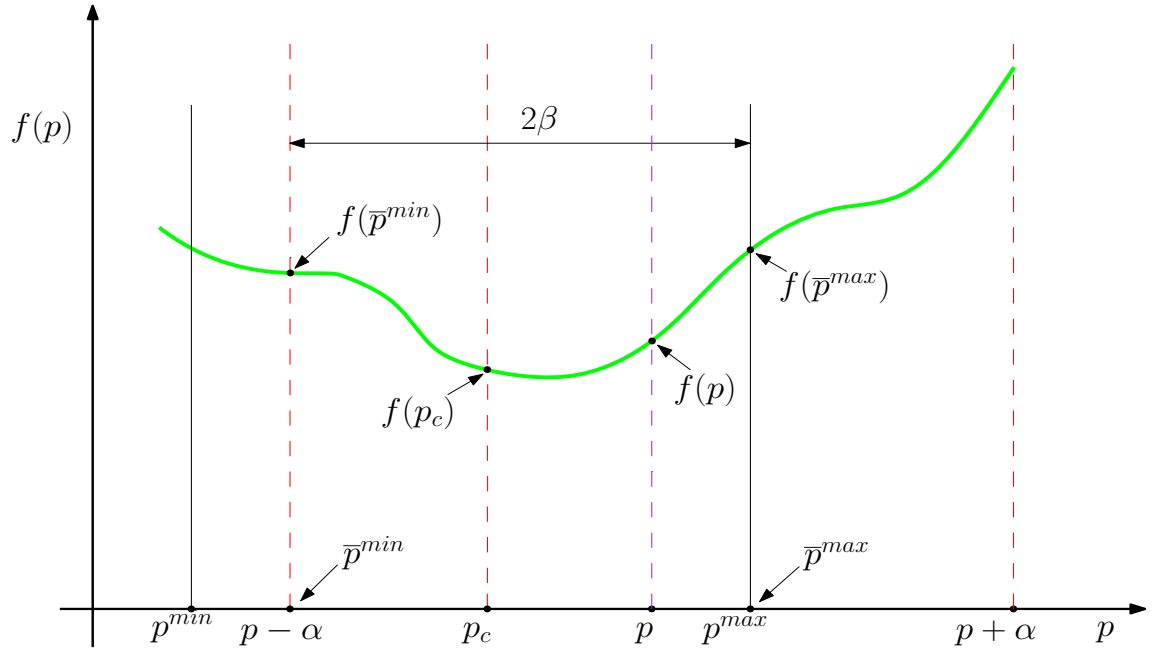

Figure 5.3: Graphical interpretation of the terms used in SQP BBO method

Reformulating this interval with respect to the center of interval p_c between the extreme bounds $\overline{p}^{\min}$ and $\overline{p}^{\max}$ yields

$$I := [p_c - \beta, p_c + \beta] \tag{5.10}$$

where p_c, $\overline{p}^{\min}$, $\overline{p}^{\max}$ and β (semi-length of the interval I) are defined as follows

$$\begin{aligned} \overline{p}^{\min} &= \max\{p^{\min}, p - \alpha\} \\ \overline{p}^{\max} &= \min\{p^{\max}, p + \alpha\} \\ p_c &= \frac{1}{2}[\overline{p}^{\min} + \overline{p}^{\max}] \\ \beta &= \frac{1}{2}[\overline{p}^{\max} + \overline{p}^{\min}] \end{aligned} \tag{5.11}$$

Illustration Fig. 5.3 lucidly presents the defined reformulation. In order to fit a local quadratic function $q_f(p)$, three function points are considered $\{f(\overline{p}^{\min}), f(p_c), f(\overline{p}^{\max})\}$. The above points are equivalently represented with $\{f^-, f^0, f^+\}$ respectively. The locally approximated quadratic function $q_f(p)$ can be expressed using a parabolic parameteric form with

$$q_f(p) = a_f \left(\frac{p - p_c}{\beta} \right)^2 + b_f \left(\frac{p - p_c}{\beta} \right) + c_f \tag{5.12}$$

where a_f, b_f, c_f define the coefficients of the approximated quadratic function. As there exists three unknown coefficients and three sets of equations defined for the functions values $\{f^-, f^0, f^+\}$, the coefficients can be computed by solving the following linear algebra problem.

$$\begin{pmatrix} 1 & -1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_f \\ b_f \\ c_f \end{pmatrix} = \begin{pmatrix} f^- \\ f^0 \\ f^+ \end{pmatrix} \tag{5.13}$$

The solution for the above problem (5.13) yields the results

$$\begin{aligned} a_f &= \frac{1}{2}[f^- + f^+] - f^0 \\ b_f &= \frac{1}{2}[f^+ - f^-] \\ c_f &= f^0 \end{aligned} \tag{5.14}$$

In order to compute the local minimizer for the approximated quadratic function (5.12), it is of high importance to consider the parameter a_f to determine the existence of minimizer, i.e. when $a_f \neq 0$, finite value for the minimizer exists and when $a_f = 0$, the solution can be either at $-\infty$ (when $\text{sign}(b_f) > 0$) or $+\infty$ (when $\text{sign}(b_f) < 0$). Thus, the solution $p_s^{(f)}$ is expressed with

$$p_s^{(f)} = \begin{cases} p_c - \dfrac{\beta b_f}{2a_f}, & \text{if } a_f \neq 0 \\ +\infty, & \text{if } a_f = 0 \text{ and } b_f \leq 0 \\ -\infty, & \text{if } a_f = 0 \text{ and } b_f > 0 \end{cases} \tag{5.15}$$

In order to confine the solution (5.15) obtained from the local quadratic approximation within the interval I defined in (5.10), the solution is projected onto the interval I and the projected solution is defined with

$$p_s^{(f,*)} := \min\left\{\overline{p}^{\max}, \max\left\{\overline{p}^{\min}, p_s^{(f)}\right\}\right\} \tag{5.16}$$

The optimal function value of the locally approximated quadratic function at the projected solution (5.16) is defined with

$$q_f^* = a_f\left(\frac{p_s^{(f,*)} - p_c}{\beta}\right)^2 + b_f\left(\frac{p_s^{(f,*)} - p_c}{\beta}\right) + c_f \tag{5.17}$$

Using the computed q_f^* value from (5.17) the extreme values of parabola $q_f(.)$ over the interval of interest I are obtained. These extreme values are defined with

$$\begin{aligned} q_f^{\min} &:= \min\Big\{f^-, f^+, q_f^*\Big\} \\ q_f^{\max} &:= \max\Big\{f^-, f^+, q_f^*\Big\} \end{aligned} \tag{5.18}$$

The values of p at these extreme values are denoted with $p_f^{\min}$ and $p_f^{\max}$ and are defined with

$$\begin{aligned} p_f^{\min} &:= \begin{cases} \overline{p}^{\min}, & \text{if } q_f^{\min} = f^- \\ \overline{p}^{\max}, & \text{if } q_f^{\min} = f^+ \\ p_s^{(f,*)}, & \text{if } q_f^{\min} = q_f^* \end{cases} \\ p_f^{\max} &:= \begin{cases} \overline{p}^{\min}, & \text{if } q_f^{\max} = f^- \\ \overline{p}^{\max}, & \text{if } q_f^{\max} = f^+ \\ p_s^{(f,*)}, & \text{if } q_f^{\max} = q_f^* \end{cases} \end{aligned} \tag{5.19}$$

When the generic function (f) is the inequality constraint function i.e. $f = g$, special cases arises for the local quadratic function ($q_g(p)$) approximation over the interval I. The possible cases are

1. $q_g(p)$ is non-negative for all $p \in I$ and this applies when $q_g^{\min} > 0$. In this case, g is non-negative in the interval I This means that the inequality constraint is violated over the entire interval I.

2. $q_g(p)$ is negative for all $p \in I$ and this applies when $q_g^{\max} \leq 0$. This means that the inequality constraint is admissible over the entire interval I.

3. $q_g(p)$ is negative on a strict subset of the interval I and this occurs when $q_g^{\min} \times q_g^{\max} < 0$.

In the last case, there could be either one or two values of p that belong to the interval I and this occurs only when $q_g(p) = 0$ and these value can be obtained by analyzing the discriminant $\Delta := b_g^2 - 4a_g c_g$ and when $\Delta \geq 0$ (non-negative) there is at least one real solution. Thus, the possible candidate solutions are

$$\begin{aligned} p_g^{(0,+)} &:= p_c + \beta \max\Big\{\frac{-b_g \pm \sqrt{\Delta}}{2a_g}\Big\} \\ p_g^{(0,-)} &:= p_c + \beta \min\Big\{\frac{-b_g \pm \sqrt{\Delta}}{2a_g}\Big\} \end{aligned} \tag{5.20}$$

Let $\mathcal{Z}_g^- \subset I$ denote the subset of values of p that belong to I where $q_g(.)$ is negative. The set $\mathcal{Z}_g$ is computed with

$$\mathcal{Z}_g^- := \begin{cases} [\overline{p}^{\min}, p_g^{(0,-)}], & \text{if } p_g^{(0,-)} < \overline{p}^{\min} \ \& \ g^- \leq 0 \\ [p_g^{(0,+)}, \overline{p}^{\max}], & \text{if } p_g^{(0,-)} < \overline{p}^{\min} \ \& \ g^+ \leq 0 \\ [\overline{p}^{\min}, p_g^{(0,-)}], & \text{if } p_g^{(0,+)} > \overline{p}^{\max} \ \& \ g^- \leq 0 \\ [p_g^{(0,-)}, \overline{p}^{\max}], & \text{if } p_g^{(0,+)} > \overline{p}^{\max} \ \& \ g^+ \leq 0 \\ [p_g^{(0,-)}, p_g^{(0,+)}], & \text{if } [p_g^{(0,-)}, p_g^{(0,+)}] \subset I \ \& \ a_g > 0 \\ \mathrm{A} \cup \mathrm{B}, & \text{if } [p_g^{(0,-)}, p_g^{(0,+)}] \subset I \ \& \ a_g < 0 \end{cases} \tag{5.21}$$

where, $\mathrm{A} = [\overline{p}^{\min}, p_g^{(0,-)}]$ and $\mathrm{B} = [p_g^{(0,+)}, \overline{p}^{\max}]$. It is important to note that the set $\mathcal{Z}_g$ is either an interval or a union of two intervals and this difference is demarcated with $n_g^z \in \{1, 2\}$ which corresponds to an interval $I_g^{(1)}$ or union of intervals $I_g^{(2)}$. All the terminologies used are summarized and tabulated in Table 5.1.

Table 5.1: Notation and meaning

Notation	Meaning
p_c	Center $I = [p - \alpha, p + \alpha] \cap [p^{\min}, p^{\max}]$
β	Semi-length of I
$q_f(.)$	Local quadratic approximation of f over I
a_f, b_f, c_f	Coefficients of parabola $q_f(.)$, $f \in \{J, g\}$
$p_s^{(f)}$	Position of singular point $q_f(.)$
$p_s^{(f,*)}$	Projection of $p_s^{(f)}$ over I
q_f^*	The value of parabola $q_f(.)$ at $p_s^{(f,*)}$
$q_f^{\min}$	Minimum value of $q_f(.)$ on I
$q_f^{\max}$	Maximum value of $q_f(.)$ on I
$p_f^{\min}$	Location of minimum value of $q_f(.)$ on I
$p_f^{\max}$	Location of maximum value of $q_f(.)$ on I
$p_g^{(0,+)}, p_g^{(0,-)}$	Solution of $q_g(p) = 0$
$\mathcal{Z}_g^-$	Subset of I where $q_g(.) \leq 0$
n_g^z	Number of intervals in $\mathcal{Z}_g^-$
$I_g^{(1)}, I_g^{(2)}$	Interval defining $\mathcal{Z}_g^-$

The BBO algorithm is completely premised upon the previously defined three cases. The algorithm for the next iteration $p^{(i+1)}$ and trust region update size $\alpha^{(i+1)}$ are given by the following steps

1. When $q_g^{\max} \leq 0$, then the whole interval I is the search space and the function J is minimized over the whole interval I. The candidate value of the update $p^{(i+1)}$ is given by

$$p^{\text{cand}} \leftarrow p_J^{\min} \tag{5.22}$$

The minimizer in the above equation (5.22) is obtained from equation (5.19) where the function f is replaced with J. This computation is based on the assumption that the

quadratic approximation is appropriate. The logical condition to verify this assumption and also, to update the trust region size is given by $\mathcal{C} \leftarrow \mathcal{C}_1 \bigvee \mathcal{C}_2$ where,

$$\begin{aligned} \mathcal{C}_1 &\leftarrow \Big(J(p^{\text{cand}}) < J(p^{(i)})\Big) \bigwedge \Big(g(p^{\text{cand}}) \leq 0\Big) \\ \mathcal{C}_2 &\leftarrow \Big(J(p^{\text{cand}}) \leq J(p^{(i)})\Big) \bigwedge \Big(g(p^{\text{cand}}) < 0\Big) \end{aligned} \tag{5.23}$$

2. When $q_g^{\max} > 0$, which means that the constraints are strictly non-negative which tantamount to constraint violation and the priority ought to be given to minimization of the inequality constraint g. In this case, the candidate value of the update $p^{(i+1)}$ is given by

$$p^{\text{cand}} \leftarrow p_g^{\min} \tag{5.24}$$

and the trust region update condition is given with

$$\mathcal{C} \leftarrow \Big(g(p^{\text{cand}}) < g(p^{(i)})\Big) \tag{5.25}$$

3. When $q_g^{\max} \geq 0$ and $q_g^{\min} \leq 0$, which means that there exists a subset in I where there exists a solution. In this case, the integer n_g^z and the corresponding intervals $I_g^{(1)}$ and $I_g^{(2)}$ are computed. Utilizing these intervals, the potential candidate for $p^{(i+1)}$ update is computed by

$$p^{\text{cand}} \leftarrow \arg \min_{p \in \{p_J^{(\min,l)}\}_{l=1}^{n_g^z}} J(p) \tag{5.26}$$

where $p_J^{(\min,1)}$ and $p_J^{(\min,2)}$ are optimal solutions that minimize J over the intervals $I_g^{(1)}$ and $I_g^{(2)}$ respectively. The trust region update condition is given by

$$\mathcal{C} \leftarrow \begin{cases} \Big(g(p^{\text{cand}}) < g(p^{(i)})\Big), & \text{if } g(p^{(i)}) > 0 \\ \Big(J(p^{\text{cand}}) < J(p^{(i)})\Big) \bigwedge \Big(g(p^{\text{cand}})\Big) \leq 0, & \text{else} \end{cases} \tag{5.27}$$

The update of next iterate $p^{(i+1)}$ and $\alpha^{(i+1)}$ is implemented according to following rules.

- If $\mathcal{C}$ is true, then
  - $p^{(i+1)}$ is assigned to the computed candidate value p^{cand}.
  - The trust region parameter $\alpha^{(i)}$ is increased according to

$$\alpha^{(i+1)} \leftarrow \beta^+ \cdot \alpha^{(i)};\ \beta^+ > 1 \tag{5.28}$$

- Otherwise, the current value $p^{(i+1)} \leftarrow p^{(i)}$ is used and the trust region size is decreased according to

$$\alpha^{(i+1)} \leftarrow \max\Big\{\alpha^{\min}, \beta^- \cdot \alpha^{(i)}\Big\};\ 0 < \beta^- < 1 \tag{5.29}$$

Let N_{iter} represent the number of iterations the above algorithm is repeated, then the total number of function evaluations of the objective and constraint functions (J, g) is

$$N_{eval} = 4N_{iter} + 1 \tag{5.30}$$

5.3.3 SQP based BBO (Multi-variate case)

The multi-variate case is an extension to the uni-variate case, where an uni-variate optimization problem is solved over each component of the decision variables while the rest of the values are maintained constant. Consider the decision variables to be $\mathbf{p} \in \mathbb{R}^{n_p}$ and let the list of decision variables be indexed with $l \in \{1, 2, \ldots, n_p\}$. Let $\eta \in \mathbb{R}$ denote the scalar variable over which the uni-variate optimization is performed. In notational form, this is expressed with $\mathbf{p}^{(\eta,l)}$ and an element in $\mathbb{R}^{n_p}$ is defined as

$$\mathbf{p}_j^{(\eta,l)} := \begin{cases} p_j & \text{if } j \neq l \\ \eta & \text{if } j = l \end{cases} \tag{5.31}$$

The formulation (5.31) is extended for $\forall j \in \{1, 2, \ldots, n_p\}$. The total number of loop count to visit all the n_p components for N_{iter} iterations and N_{eval} is given by

$$N_{loop} = \left\lceil \frac{N_{eval} - 1}{4n_p \times N_{iter}} \right\rceil \tag{5.32}$$

It is important to note that a feasible choice of pair (N_{eval}, N_{iter}) and must satisfy the inequality

$$N_{eval} \geq 4n_p N_{iter} + 1 \tag{5.33}$$

For convergence results for the SQP-BBO algorithm, refer [Alamir 2013][Alamir 2012].

5.4 pNMPC S/W structure

In this section, the pNMPC S/W structure, syntax and features are discussed in a bird's eye view level. It is important to note that a complete coverage of all the features of the S/W is not feasible within the scope of this chapter. The goal of this section is to point out the crucial elements of the S/W along with it's use case. The core components of the S/W can be broadly categorized into five parts which are a) Symbolic classes, b) OCP design classes, c) Real/Symbolic classes, d) Control parameterization classes and e) Code generation classes.

5.4.1 Symbolic classes

The symbolic classes of the pNMPC S/W are

- `HyperStates` - This serves as the base class for other derived symbolic classes. `HyperStates` can also be used as a substitution or an intermediate variable.
  Usage: `HyperStates H = 2*x1*k1+u1*x2+F;`
- `States` - This symbolic class is used to define the states for the system.
  Usage: `States x1,x2;`

- `Inputs` - This symbolic class is used to define the inputs for the system.
  Usage: `Inputs u1;`

- `Params` - This symbolic class is used to define the parameters for the system.
  Usage: `Params k1;`

- `External` - This symbolic class is used to define external variables which invokes function calls from source files or libraries. The below example links the symbolic object/variable to a function named `NeuralNet`
  Usage: `External F = "NeuralNet";`

5.4.2 OCP design classes

The OCP design classes of the pNMPC S/W are

- `ParameterizationMap` - This OCP class maps the input parameterization vector to the actual input for the system. The parameterization map must be defined within the begin and end guards. **Usage**:

```
BEGIN_PARAMETERIZATION_MAP
 ParameterizationMap u1 = p1*x1+p2;
 ParameterizationMap u2 = p3*x2+p4;
END_PARAMETERIZATION_MAP
```

  where `p1,p2,p3,p4` denote the input parameterized variables, `x1,x2` denote the states of the system and `u1,u2` denote the actual input to the system.

- `DiffEquation` - This OCP object is used to define the underlying differential equation of the system. The differential equations must be defined within the begin and end guards. **Usage**:

```
BEGIN_DIFFERENTIAL
 DiffEquation x1d = -x1*x2+u1;
 DiffEquation x2d = -x2+u2;
END_DIFFERENTIAL
```

- `ScalarConstraint` - This OCP class is used to define the scalar constraints for the system. The scalar constraint class can be classified into two types which are a) Regular constraints and b) Terminal constraints and these are defined within the respective begin and end guards. **Usage**:

```
BEGIN_CONSTRAINTS
 BEGIN_REGULAR_CONSTRAINTS
  ScalarConstraint GR1 = {-5<=x1<=5};
  ScalarConstraint GR2 = {-5<=x2<=5};
  ScalarConstraint GR3 = {-1<=u1<=1};
  ScalarConstraint GR4 = {-1<=u2<=1};
 END_REGULAR_CONSTRAINTS
 BEGIN_TERMINAL_CONSTRAINTS
  ScalarConstraint GT1 = {-1<=x1<=1};
  ScalarConstraint GT2 = {-1<=x2<=1};
  ScalarConstraint GT3 = {-1<=u1<=1};
  ScalarConstraint GT4 = {-1<=u2<=1};
 END_TERMINAL_CONSTRAINTS
END_CONSTRAINTS
```

- `ScalarObjective` - This OCP class is used to define the scalar objectives for the system. The scalar objective class can be classified into two types which are a) Lagrangian (stage cost) and b) Mayer (terminal cost) and these are defined within the respective begin and end guards. Multiple definition of objectives within each guard will be scalarized by addition. **Usage**:

```
BEGIN_OBJECTIVES
 BEGIN_LAGRANGIAN
  ScalarObjective L1 = x1*x1+x2*x2;
  ScalarObjective L2 = u1*u1+u2*u2;
 END_LAGRANGIAN
 BEGIN_MAYER
  ScalarObjective M1 = 5*x1*x1+5*x2*x2;
  ScalarObjective M2 = 2*u1*u1+2*u2*u2;
 END_MAYER
END_OBJECTIVES
```

5.4.3 Control parameterization classes

The pNMPC S/W has two control parameterization features namely a) Piecewise parameterization and b) Linear parameterization. The definitions for the classes are given below

- `ControlParamZ<Piecewise>` - This class parameterizes the input in a piecewise fashion over the prediction horizon. There are multiple constructor overloads for this class, however in the interest of brevity, only the simplest case is presented here. **Usage**:

```
ControlParamZ<Piecewise> {5,p1,-1,1};
```

  where, the first argument defines the number of piecewise parameterization placed equidistantly over the prediction horizon, the second argument `p1` represents the input object from the `Inputs` class, so the specified parameterization is latched to this

input and the last two arguments represents the minimum and maximum bounds over the input `p1` respectively.

- `ControlParamZ<Linear>` - This class parameterizes the input in a linear fashion over the prediction horizon. The other specifications follow suit as the piecewise parameterization. **Usage**:

```
ControlParamZ<Linear> {2,p2,-1,1};
```

5.4.4 Real/Symbolic classes

The pNMPC software supports several functions and operation between real numbers and as well as symbolic objects. The list of supported functions are {`sin`, `cos`, `tan`, `sinh`, `cosh`, `tanh`, `exp`, `log`, `abs`, `asin`, `acos`, `atan`, `asinh`, `acosh`, `atanh`, `minimum`, `maximum`, `sign`}. The list of supported numerical operations are $\{+, -, *, \backslash, \wedge, \geq, \leq$, &&, $||\}$, where the last two are logical AND and OR operations, which is used to couple multiple constraints to one (Example - blocking constraints in state space region). The scalar real numbers are declared with `Real` keyword and real valued matrices are declared with `MATReal` keyword. Symbolic matrices are declared with `MATHyperStates` keyword. **Usage**:

```
MATHyperStates A(2,2); MATReal B(2,2);
// Populate symbolic matrix
A[0][0] = x1; A[0][1] = x1+sin(x2);
A[1][0] = u1; A[1][1] = u1*x2;
// Populate real matrix
B[0][0] = 2; B[0][1] = -1;
B[1][0] = 5; B[1][1] = 3;
// Symbolic Matrix - Real Matrix multiplication
MATHyperStates C = A*B;
```

There are several in-build matrix operations such as matrix-vector operations, matrix-matrix operations etc. packaged along with the S/W.

5.4.5 Code generation classes

The code generation classes are formed by a composition of three classes which are a) `INTEGRATOR`, b) `CONST_FORM` and c) `PNMPCGEN`. The definitions of the classes are given below.

- `INTEGRATOR` - This enum class specifies the integrator to be used to simulate the underlying differential equations. It is important to note that the S/W for now has support only for explicit solvers. **Usage**:

```
INTEGRATOR iODE = INTEGRATOR::RK45;
```

- `CONST_FORM` - This enum class specifies the method for constraint scalarization either using Form 1 or Form 2 technique as mentioned in Section 5.3.1. **Usage**:

```
CONST_FORM cF = CONST_FORM::FORM_1;
```

- `PNMPCGEN` - This class is used to create the instance of the OCP with all the aforementioned classes and functions. The class follows singleton design pattern, thereby only one instance of the controller can be generated. The setting function calls are initial and final time of the OCP, step-size for the embedded integrator, SQP solver parameters, constraint form, integrator object and a Boolean flag to toggle between parameter data at current time step or over the prediction horizon. Finally, the method `genCCode()` is invoked to generate the respective C files. **Usage**:

```
// Get Singleton instance of code generation class
PNMPCGEN* pNMPC = PNMPCGEN::getSton();
pNMPC->setInitialTime(0);
pNMPC->setFinalTime(2);
pNMPC->setStepSize(0.1);
pNMPC->getSolver()->setNiter(4);
pNMPC->setConstForm(cF);
pNMPC->setIntegrator(iODE);
// Generate C codes
pNMPC->genCCode();
```

5.5 pNMPC code generation module

The pNMPC code generation module is not dissimilar to any compiler design paradigm [Aho, Sethi, and Ullman 1986] at the same time the steps involved are not as extensive as for compilation process of any programming language. The motivation for adopting a scheme as such are two folds which are:

1. The ingredients of the OCP problem (objectives, constraints, dynamics etc.) fed by the user are broken down into fundamental elements and then modeled in an appropriate way to suit the optimization module. Case in point, the inequality constraints ought to be aligned in a non-positive formulation as described in equation (5.4).

2. By breaking down the OCP into it's fundamental elements, code optimization can be performed efficiently which in turn benefits in reducing the memory footprint (space complexity) and burning less computer clock cycles (time complexity). This feature has high practical importance, especially for low-end embedded devices.

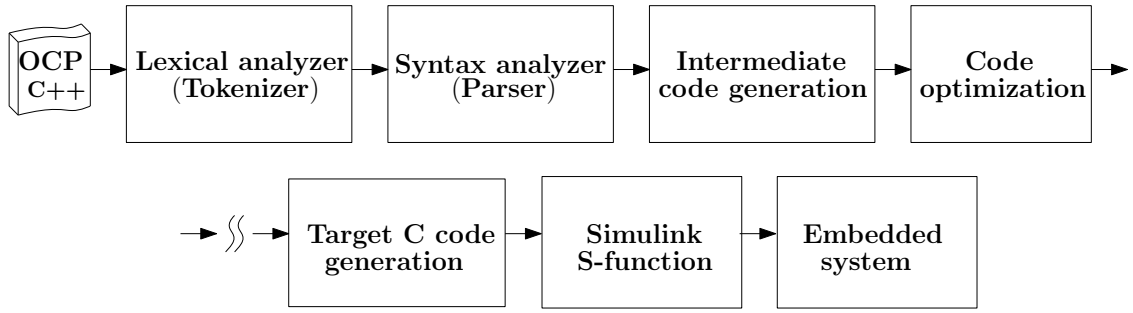

Figure 5.4: pNMPC code generation process

The pNMPC code generation process is illustrated as a process flow diagram in Fig. 5.4. The key stages involved in the process are briefed below:

- **OCP design and specification**: The OCP's design and specification are programmed by the user using C++ as a front-end modeling language.
- **Lexical analyzer**: Lexical analyzer or tokenizer takes the user's OCP design and specification and breaks down the entries into separate characters or special tokens such as the states, inputs, parameters, math operations etc.
- **Syntax analyzer**: Syntax analyzer or parser scans through these tokens and contrives a relational tree or parse tree. Consider an example $y = x_1x_2 + sin(x_1x_2)$, then the computed parse tree for this relation is illustrated in Fig. 5.5.

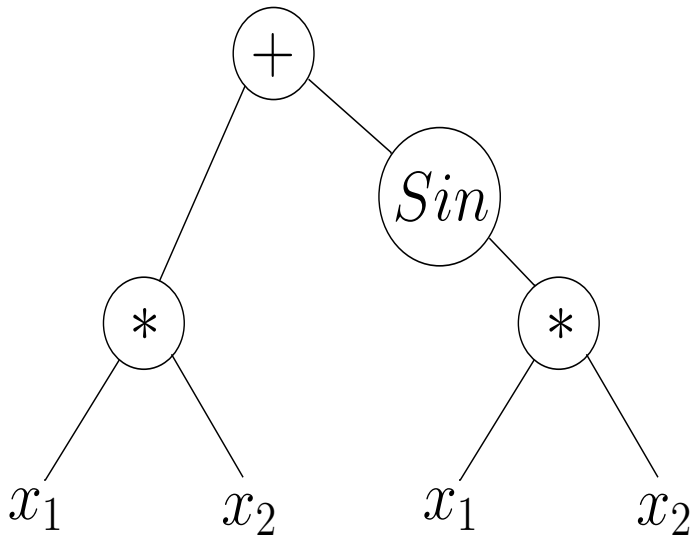

Figure 5.5: Parse tree structure

- **Intermediate code generation**: In this stage, the parse tree is stored as Three-address code (3AC) and stored in a stack data structure. Additional variables x_3, x_4, x_5 and x_6

are induced in the due process. It is important to note that the code is unoptimized at this stage.

- **Code optimization**: The code optimization stage is a crucial stage of the code generation module. Typically, the code optimization module obliterates redundant relations, self negation operations, identity and inverse relations for addition, subtraction, multiplication and division operations such as (to list a few) $0 * x$, $x + 0$ etc. In the above example, it is clearly evident that the term $x1 * x2$ is computed twice. After code optimization, this redundancy is removed and the stack is updated.

- **Target C code generation**: In this stage, the embedded C files are exported by using the file stream operations. Under circumstances of securing the generated source code, either a static or dynamic library can be created. However, this has to be done manually by the user.

- **Simulink S-function**: This stage is optional, however in today's world, embedded control has virtually become MATLAB/Simulink's fiefdom and it plays a dominant role in design, development and deployment of production code to embedded systems both in industry and academia. The pNMPC S/W provides the flexibility to provide Simulink compliant C codes and this can be included into Simulink by availing MEX wrappers in MATLAB or C-MEX S-functions features from Simulink.

- **Embedded system**: Finally, the generated code is deployed into the embedded system either through Simulink or manually by the user.

5.6 Application of pNMPC toolbox

The pNMPC S/W was tested for several examples and the simulation results looks promising and viable for RT implementation. In this Chapter, in the interest of space, the results and simulation of two examples are described in detail which are

1. Cart-pole swing up problem.

2. PVTOL stabilization problem.

The outputs were compared against ACADO toolkit [Houska, Ferreau, and Diehl 2011] as a benchmark S/W to study the performance and computation time of the two examples. The former example was implemented with the assumption of a white box model and the latter example was implemented with the assumption of a black box model. The examples were simulated in MATLAB/Simulink on a Intel Core i7, 16GB RAM PC. The pNMPC C++ codes for the respective examples are listed in Appendix A.

5.6.1 Cart-pole swing up problem

The task of the cart-pole swing up problem [Mills, Wills, and Ninness 2009] is to stabilize a pole in an upright direction (typically starting from downward position) which is attached to a movable cart by means of a revolute joint. The control input to this system is the horizontal force applied to the cart and system is bounded by physical constraints which are the length of cart travel and the input force. The nonlinear state space equations of the system are given below

$$\begin{aligned}
\dot{x} &= v \\
\dot{v} &= \frac{-m_2 l sin(\theta)\dot{\theta}^2 + u + m_2 g cos(\theta) sin(\theta)}{m_1 + m_2(1 - cos^2(\theta))} \\
\dot{\theta} &= \omega \\
\dot{\omega} &= \frac{-m_2 l cos(\theta) sin(\theta)\dot{\theta}^2 + u cos(\theta) + (m_1 + m_2) g sin(\theta)}{l(m_1 + m_2(1 - cos^2(\theta)))}
\end{aligned} \tag{5.34}$$

where m_1, m_2, l, g represents the mass of the cart, mass of the pole, length of the pole and acceleration due to gravity respectively. The values for the parameters are listed in Appendix A. The state vector of the system are $\mathbf{x} = [x, v, \theta, \omega]$, which are the cart position, cart velocity, pole angle and pole angular velocity respectively and input to the system is u, which represents the force acting on the cart. The constraints acting on the system are

$$\begin{aligned}
-x_{max} \leq x \leq x_{max} \\
-u_{max} \leq u \leq u_{max}
\end{aligned} \tag{5.35}$$

where x_{max} and u_{max} represents the maximum bounds of the cart position and cart force respectively. The OCP for the system is given as

$$\begin{aligned}
\min_{\mathbf{x}(.),\mathbf{p}(.)} \quad & \mathbf{x}(t_f)^T Q_f \mathbf{x}(t_f) + \int_0^{t_f} (\mathbf{x}^T Q \mathbf{x} + u^T R u)\, dt \\
\text{subject to} \quad & (5.34),\ (5.35),\ \mathbf{x}(0) = \{[0, 0, \pi, 0], [0, 0, \frac{\pi}{2}, 0]\} \\
& u(\mathbf{p}, \mathbf{x}) = p_1 x + p_2 v + p_3 \theta + p_4 \omega + p_5
\end{aligned} \tag{5.36}$$

where t_f, Q_f, Q, R and T_s represents the look ahead period, quadratic terminal state cost, quadratic stage state cost, quadratic input cost and sampling period respectively. The input parameterization is an affine state feedback policy where the parameterization vector is $\mathbf{p} = [p_1, p_2, p_3, p_4, p_5]$ which is modeled a constant over the horizon. Once the OCP is solved for $\mathbf{p}^*$, the input $\mathbf{u}(\mathbf{p}^*, \mathbf{x}(0))$ is injected into the system over the period T_s and this process is repeated in a receding horizon fashion. To compare against the ACADO controller, an un-parameterized version of the NMPC problem (5.36) was implemented with the following settings - **Integrator** - 4th order Implicit Runge Kutta integrator, **QP solver** - qpOASES, **Hessian approximation** - Gauss-Newton, **Discretization** - Multiple shooting, **Discretization intervals** - 30 and for the rest, default parameters were utilized. The study was conducted in two parts with two initial conditions which are $\mathbf{x}(0) = [0, 0, \frac{\pi}{2}, 0]$ and $\mathbf{x}(0) = [0, 0, \pi, 0]$.

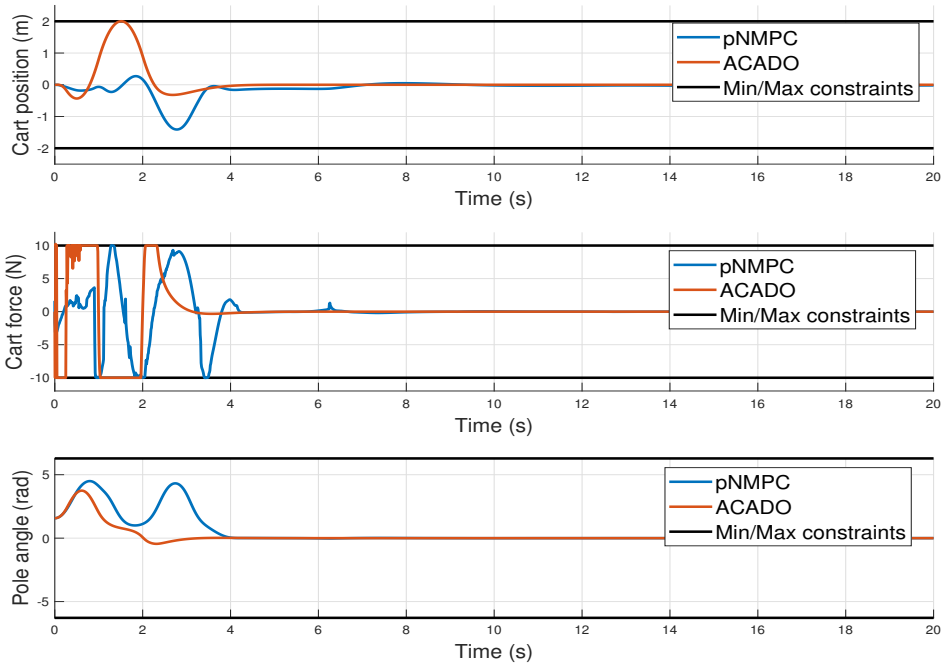

Figure 5.6: Cart position, cart force and the pole angle of the system for initial condition $\mathbf{x}(0) = [0, 0, \frac{\pi}{2}, 0]$ (Case 1)

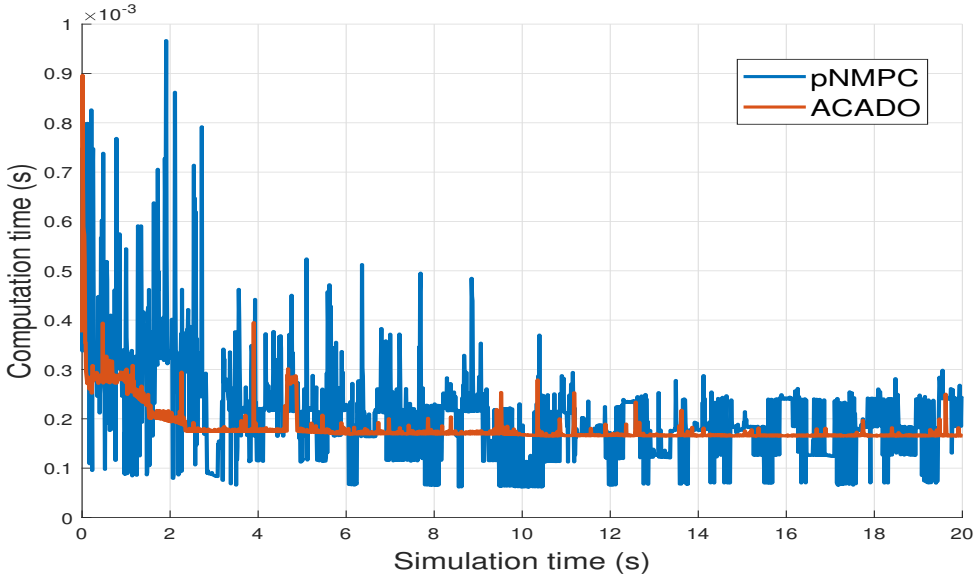

Figure 5.7: Cart-pole system computation time (Case 1)

Fig. 5.6 illustrates the cart force, cart position and the pole angle of the system for pNMPC and ACADO controller for the first case respectively. The computation time of the pNMPC

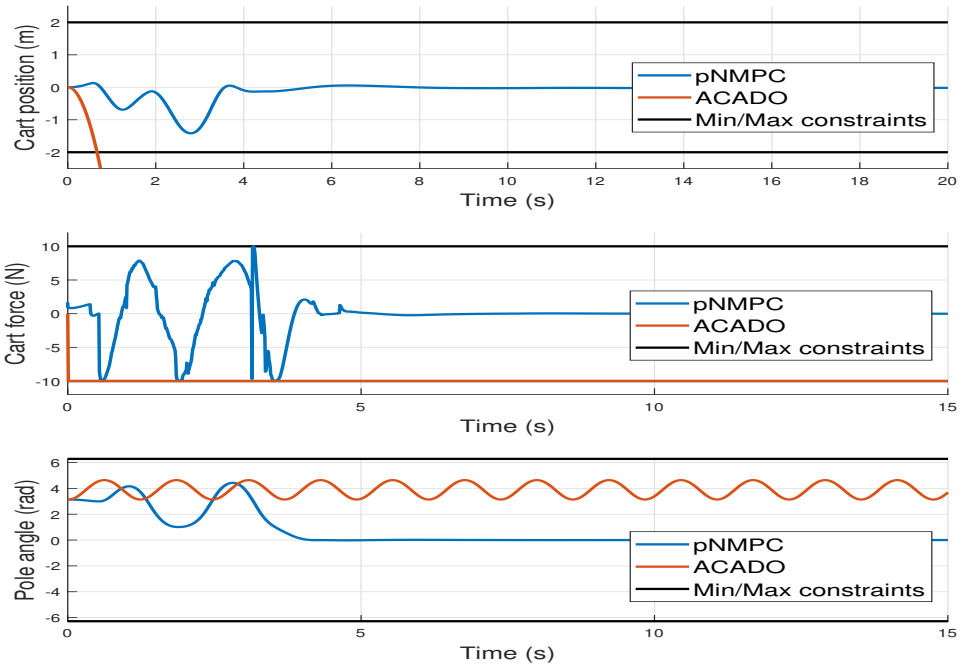

Figure 5.8: Cart position, cart force and the pole angle of the system for initial condition $\mathbf{x}(0) = [0, 0, \pi, 0]$ (Case 2)

and ACADO controller for the first case is plotted in Fig. 5.7. The mean computation time of ACADO hovers around 185.57 ms and 205.05 μs for the pNMPC controller. The maximum computation time of the ACADO hovers around 897.5 μs and 966.7 μs for the pNMPC controller.

Fig. 5.8 illustrates the cart force, cart position and the pole angle of the system for pNMPC and ACADO controller for the second case respectively. From the plots, it is evident that the ACADO controller crashes, however, despite the numerical ill-conditioning of model, the pNMPC controller fares well with state feedback parameterization and at the same time, the system is stabilized. Fig. 5.9 illustrates the parameterization values for the second case.

5.6.2 PVTOL stabilization problem with Black-box models

The task of the PVTOL (planar vertical takeoff and landing aircraft) [Martin, Devasia, and Paden 1996] stabilization problem is to regulate the states of the system to the origin given

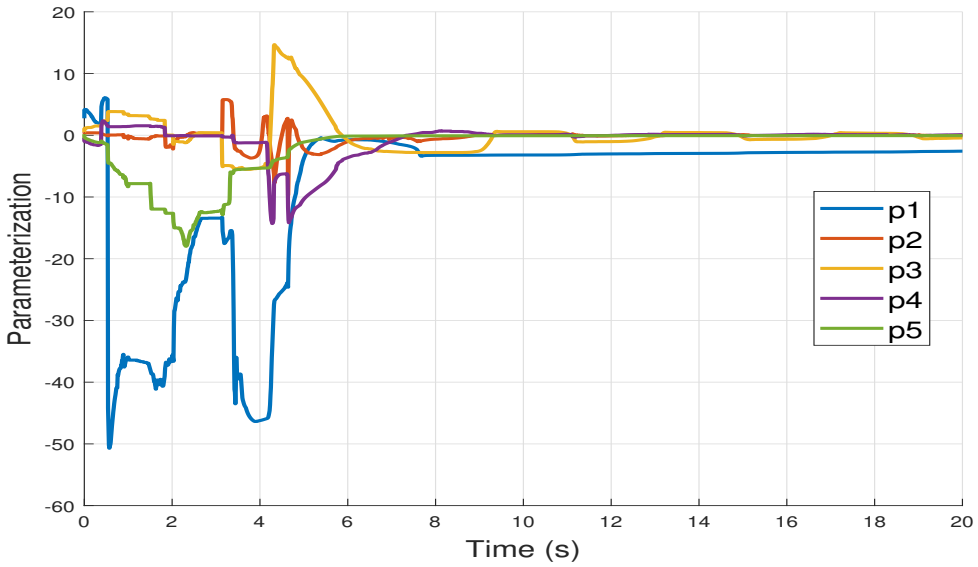

Figure 5.9: Parameterization p_1, p_2, p_3, p_4, p_5 (Case 2)

an initial perturbation. The nonlinear state space equations of the system are given below

$$\begin{aligned} \dot{y} &= v_y \\ \dot{v}_y &= \psi_y^{BB}(\mathbf{x}, \mathbf{u}, \boldsymbol{\kappa}) \\ \dot{z} &= v_z \\ \dot{v}_z &= \psi_z^{BB}(\mathbf{x}, \mathbf{u}, \boldsymbol{\kappa}) \\ \dot{\theta} &= \omega \\ \dot{\omega} &= u_2 \end{aligned} \tag{5.37}$$

where, the state vector is $\mathbf{x} = [y, v_y, z, v_z, \theta, \dot{\theta}]$ which represents the vertical position, vertical velocity, horizontal position, horizontal velocity, roll angle and roll rate respectively, the input vector is $\mathbf{u} = [u_1, u_2]$ which represents the lift acceleration and angular acceleration respectively and the parameter vector is $\boldsymbol{\kappa} = [\sigma]$ which represents the coupling between roll and lift effects. The variables $\psi_y^{BB}(\mathbf{x}, \mathbf{u}, \boldsymbol{\kappa}) = -u_1 sin(\theta) + \sigma u_2 cos(\theta)$ and $\psi_z^{BB}(\mathbf{x}, \mathbf{u}, \boldsymbol{\kappa}) = u_1 cos(\theta) + \sigma u_2 sin(\theta) - 1$ represents the dynamics of the system which are deliberately modeled as a black-box model and invoked using the function calls `"ModelY"` and `"ModelZ"` respectively. The input arguments for these functions are the state vector, input vector and parameter vector and the output is the respective dynamics of system. The source codes for these functions were compiled to a dynamic library file and linked during the compilation process of the pNMPC controller. This example serves as an use case of the pNMPC S/W, where external black box models can be linked with the S/W's inbuilt symbolic variables.

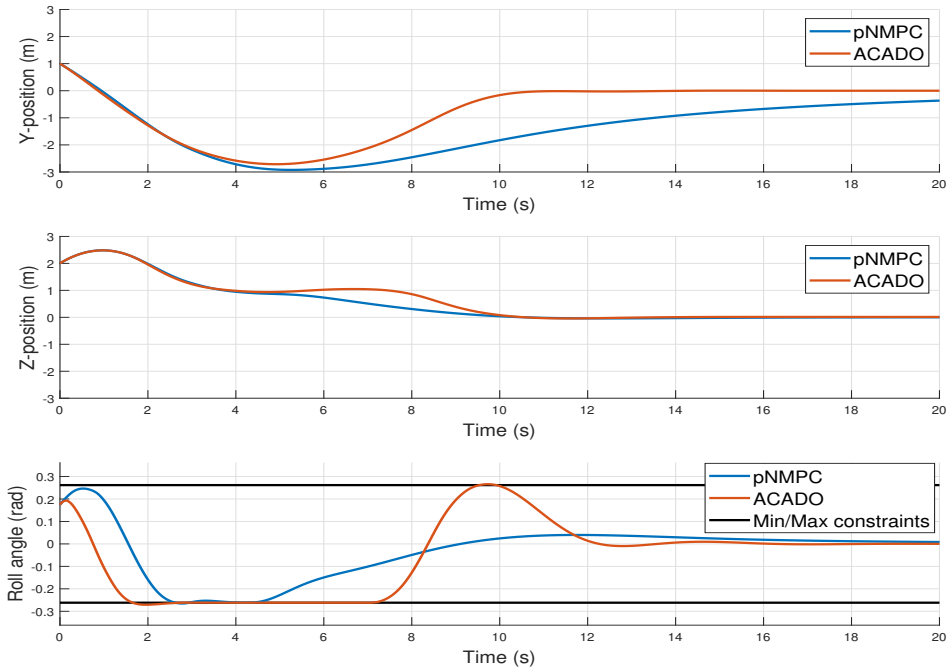

Figure 5.10: PVTOL Y-position, Z-position, Roll angle (θ)

The constraints acting on the system are

$$\begin{aligned} 0 \leq u_1 &\leq u_1^{max} \\ -u_2^{max} \leq u_2 &\leq u_2^{max} \\ -\theta^{max} \leq \theta &\leq \theta^{max} \end{aligned} \tag{5.38}$$

where $u_1^{max}, u_2^{max}, \theta^{max}$ represents the maximum bounds over the inputs u_1 and u_2 and the roll angle state θ. The OCP for the stabilization problem is defined as

$$\begin{aligned} \min_{\mathbf{x}(.),\mathbf{p}(.)} \quad & \mathbf{x}(t_f)^T Q_f \mathbf{x}(t_f) + \int_0^{t_f} (\mathbf{x}^T Q \mathbf{x} + \mathbf{u}^T R \mathbf{u})\, dt \\ \text{subject to} \quad & (5.37),\ (5.38),\ \mathbf{x}(0) = [1, -1, 2, 1, \tfrac{-14\pi}{10}, -0.1] \\ & \mathbf{u}(\mathbf{p}) = \mathbf{p} \end{aligned} \tag{5.39}$$

where t_f, Q, R and T_s represents the look ahead period, quadratic stage state cost, input cost and sampling period respectively. The input parameterization is $\mathbf{p} = [p_1, p_2]$ where each parameter has two control points placed equidistantly over the prediction horizon and follows a linear profile. Once the OCP is solved for $\mathbf{p}^*$, the input $u(\mathbf{p}^*)$ is injected into the system over the period T_s and this process is repeated in a receding horizon fashion. The following setting was chosen for the ACADO controller, **Integrator** - 4th order Runge Kutta integrator, **QP**

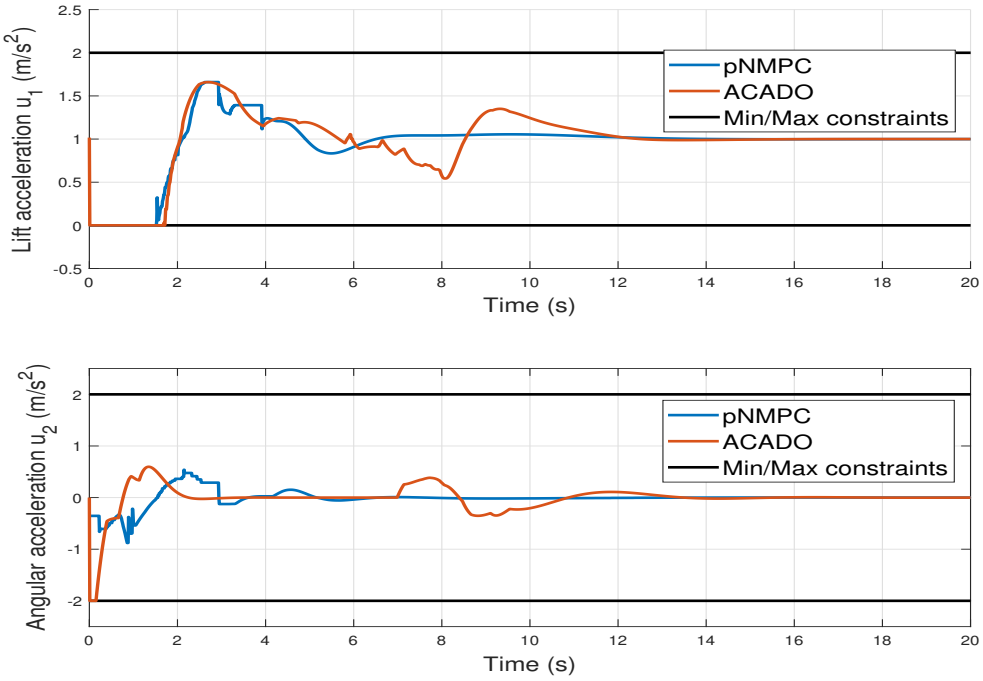

Figure 5.11: PVTOL inputs (u_1, u_2)

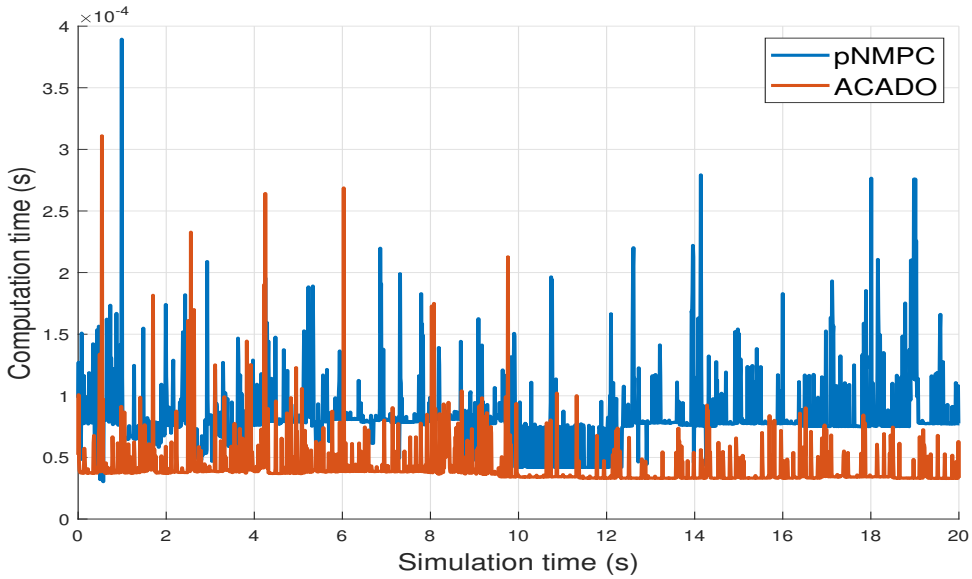

Figure 5.12: PVTOL computation time

solver - qpOASES, **Hessian approximation** - Gauss-Newton, **Discretization** - Multiple shooting, **Discretization intervals** - 5 and for the rest, default parameters were utilized. Also, for simulation of ACADO controller, the whole dynamics of the system was presumed

to known already as ACADO toolkit has no feature to incorporate external function calls into it's symbolic framework.

Fig. 5.10 and Fig. 5.11 illustrates the Y-position, Z-position and roll angle (θ) and the inputs u_1 and u_2 of the system for the pNMPC and ACADO NMPC controller respectively. The computation time of the pNMPC and ACADO controller is plotted in Fig. 5.12. The mean computation time of ACADO hovers around 41.50 μs and 44.7 μs for the pNMPC controller. The maximum computation time of ACADO hovers aroud 311.0 μs and 389.4 μs for the pNMPC controller. The comparison highlights the fact that in-spite of incorporating black box models, the performance and computation time is nearly on par with the ACADO toolkit.

5.7 HiL tests on dSPACE MABXII for control of semi-active suspension system for quarter car vertical dynamics model

The proposed pNMPC S/W was tested on dSPACE MicroAutoBox II (MABXII) embedded hardware via Hardware In the Loop (HiL) simulations to verify and validate the feasibility of generated code under RT conditions. The example involved a quarter car model of the INOVE test platform (refer Chapter 1, Section 1.1) with the following experimental settings

- The sampling frequency was set to 200 Hz i.e. a sampling period of 5 ms.
- The road profile involved a chirp signal with amplitude of 5mm and frequency sweep from 0.1 Hz to 20 Hz for duration of 20s.
- The pNMPC S/W settings were **Integrator** - 4th order explicit Runge Kutta integrator, **Step size** - 5 ms, **Number of SQP iterations** - 5, **Constraint form** - `CONST_FORM::FORM_1` and for the rest, default parameters were utilized.
- The constraints imposed on the system were the min/max damper force constraints, the min/max stroke deflection constraints and the min/max PWM-DC (Pulse Width Modulation - Duty Cycle) signal which operates the damper characteristics.
- The objective was considered to maximize the comfort, which in turn is reflected in minimizing the chassis acceleration of the vehicle.

Fig. 5.13 illustrates the computation time on dSPACE MABXII. The mean and maximum computation time hovers around 586.04 μs and 745.36 μs respectively and the generated code was doable under RT conditions. Fig. 5.15 illustrates the nonlinear frequency response (refer [Savaresi et al. 2010]) from road profile to chassis position for minimum, nominal, maximum and pNMPC controller. It is clearly evident that the pNMPC controller attentuates the road disturbance and performs better than other passive damper settings. Fig. 5.14 illustrates the PWM-DC signal, damper force and stroke deflection respectively. The PWM-DC signal is chosen judiciously to satisfy both minimization of objective as well as constraint satisfaction.

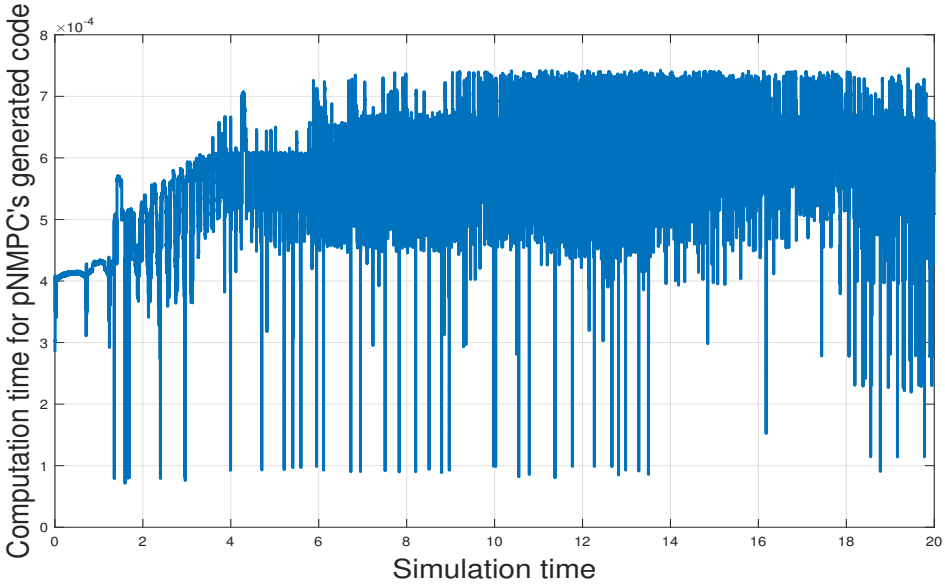

Figure 5.13: Computation time of pNMPC's generated code on dSPACE MABXII

5.8 Parallelized pNMPC patch

5.8.1 Parallelization of the optimization module on CUDA GPUs and CUDA code generation module

The parallelized pNMPC S/W augments the proposed multi-variate SQP-BBO module to simultaneously run on the multi-core CUDA based GPUs. The parallelized SQP-BBO module spawns multiple threads for each component in the optimization vector and for each component, a uni-variate SQP-BBO is solved in each CUDA core of the GPU. The GPU version of the pNMPC controller codes can be generated by toggling the conditional compilation headers at compilation time.

The method deserves some explanation to elucidate it's working principle. Consider an example as illustrated in Fig. 5.16. The optimization vector is four and black dots represents the unoptimized values of the variables in its respective places at the current iteration. In the first iteration, the optimization vector is disseminated to four CUDA processors and an uni-variate SQP-BBO routine is executed in parallel for each component of the optimization vector with rest of the components held constant with previous values. The cross mark represents the component of interest for optimization and as well as the optimized component. Once the optimization is carried out in the CUDA core, the optimized components are collected and the optimization vector is updated and this is passed on for the second iteration. This procedure is repeated till the end of iteration count.

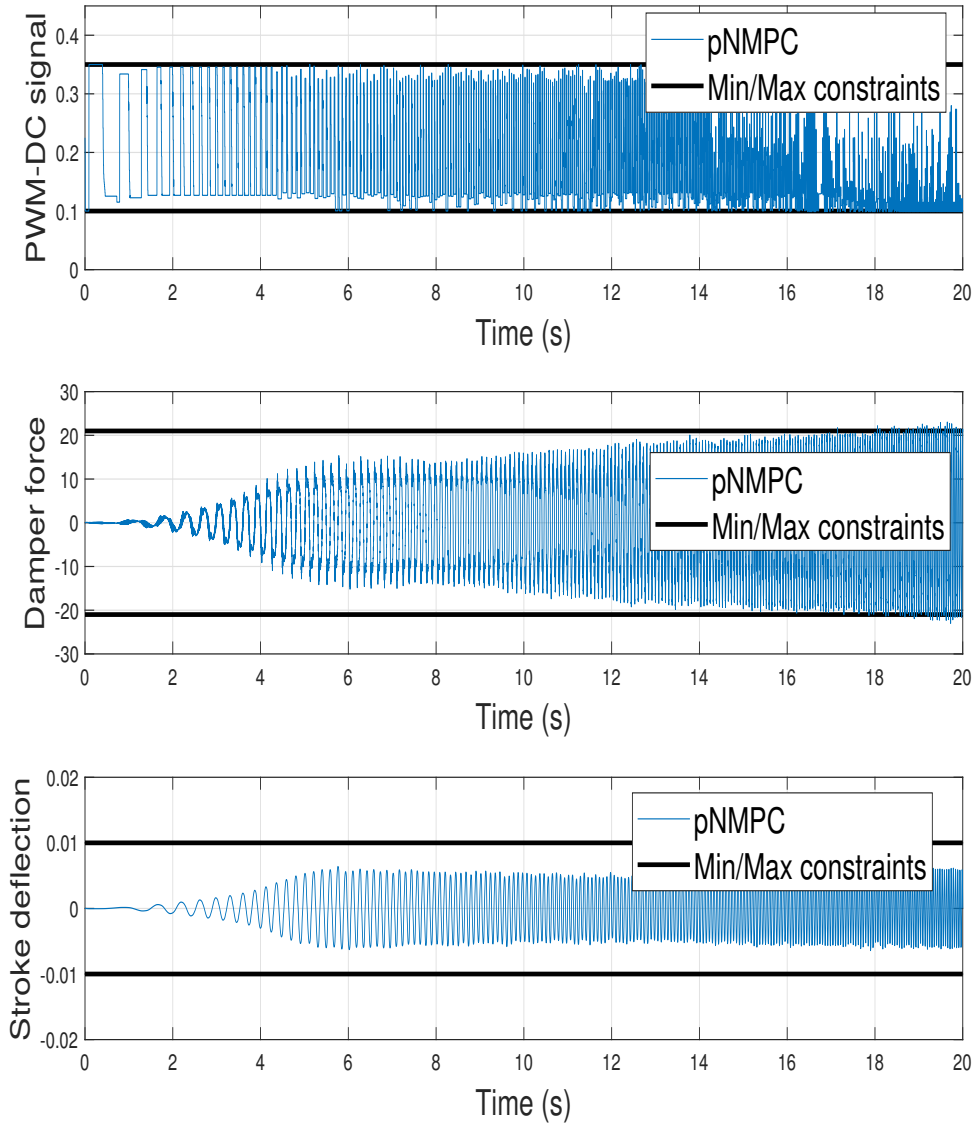

Figure 5.14: PWM-DC signal, Damper force and stroke deflection plots

5.8.2 Application of Parallelized pNMPC S/W for 2D crane control

The task of 2D crane or gantry crane control [Alamir 2013] to track a reference signal, typically the position of the crane under minimum oscillations of the crane, i.e. the swing angle. The nonlinear state space equations of the system are given below

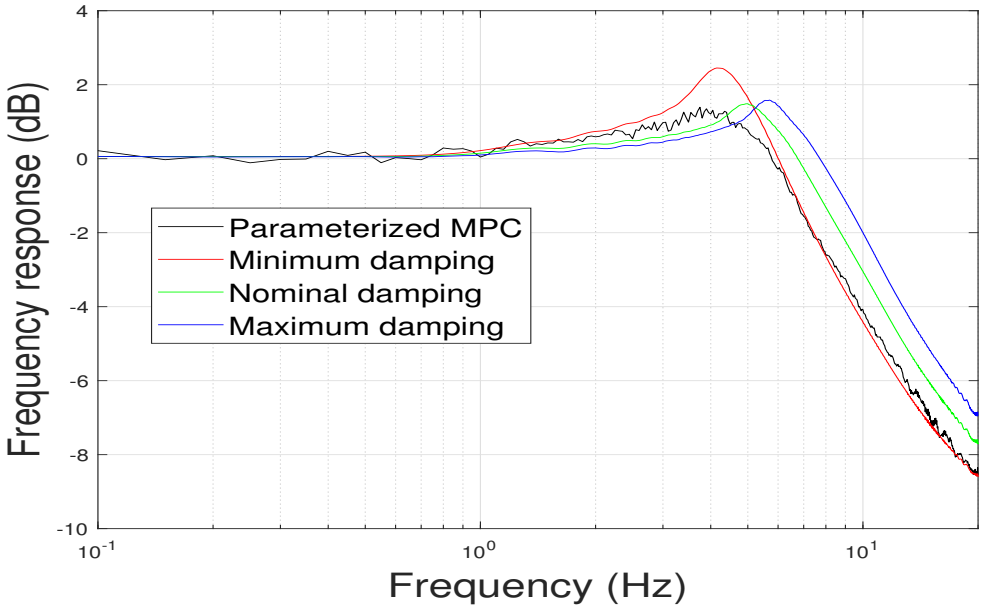

Figure 5.15: Nonlinear frequency response from road profile (z_r) to chassis position (z_s) for pNMPC controller, minimum, nominal and maximum damping setup

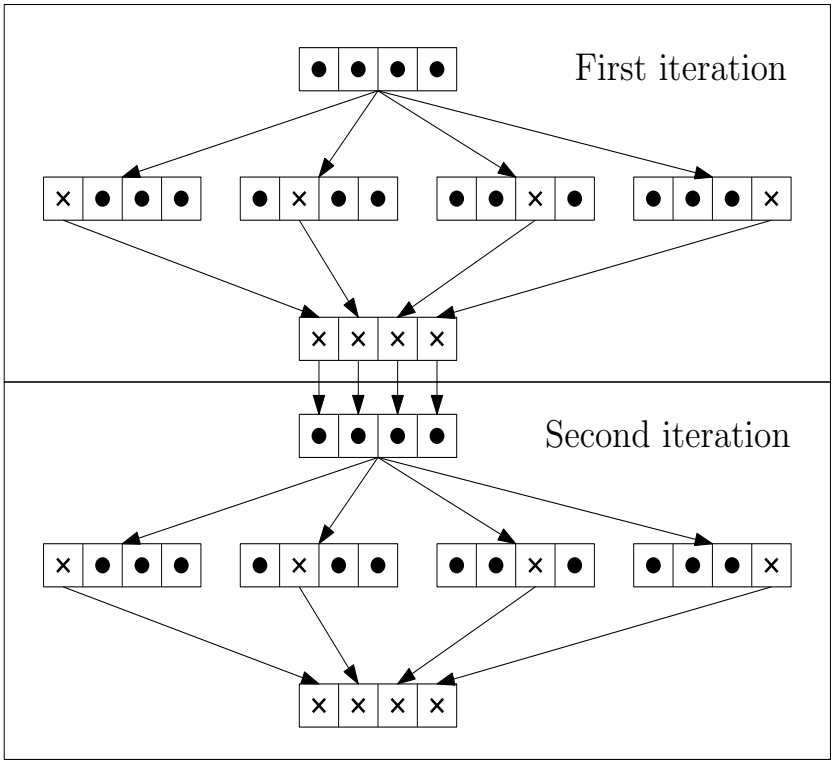

Figure 5.16: First and second iteration of parallelized SQP-BBO module

$$\begin{aligned}
\dot{r} &= r_d \\
\dot{r}_d &= \frac{u + mg\cos\theta\sin\theta + ml\omega^2\sin\theta}{M + m(1-\cos^2\theta)} \\
\dot{\theta} &= \omega \\
\dot{\omega} &= \frac{-u\cos\theta - ml\omega^2\cos\theta\sin\theta - (M+m)g\sin\theta - f_r\omega}{(M + m\sin^2\theta)l}
\end{aligned} \tag{5.40}$$

where, the state vector is $\mathbf{x} = [r, r_d, \theta, \omega]$ which represents the crane position, velocity, crane angle of swing and angular rate of swing respectively. The input is u which represents the applied input force. M, m, l, f_r and g represents the mass of the moving body, mass of the suspended mass, length of the crane, drag resistance and acceleration due to gravity respectively. The constraints acting on the system are

$$\begin{aligned}
-u^{max} \leq u \leq u^{max} \\
-\theta^{max} \leq \theta \leq \theta^{max} \\
-\omega^{max} \leq \omega \leq \omega^{max}
\end{aligned} \tag{5.41}$$

where, u^{max}, θ^{max} and ω^{max} represents the maximum bounds over the input, swing angle and swing angular rate respectively. The OCP for the tracking problem is given as

$$\begin{aligned}
\min_{\mathbf{x}(.),\mathbf{p}(.)} \quad & (\mathbf{x}(t_f) - \mathbf{x}_d(t_f))^T Q_f (\mathbf{x}(t_f) - \mathbf{x}_d(t_f)) + \int_0^{t_f} (\mathbf{x} - \mathbf{x}_d)^T Q (\mathbf{x} - \mathbf{x}_d) + \mathbf{u}^T R \mathbf{u})\, dt \\
\text{subject to} \quad & (5.40),\ (5.41),\ \mathbf{x}(0) = [0,0,0,0] \\
& \mathbf{u}(\mathbf{p}) = \mathbf{p}
\end{aligned} \tag{5.42}$$

where t_f, Q, Q_f R and T_s represents the look ahead period, quadratic stage state cost, terminal state cost, input cost and sampling period respectively. The reference tracking vector $\mathbf{x}_d = [\kappa, 0, 0, 0]$, where κ is the reference signal for position tracking. In order to validate, verify and assess the performance, efficiency and computation time of the GPU version of the pNMPC controller, the input parameterization was considered to be of length 128 and subsequently compared against the CPU version of the pNMPC controller. The input parameterization vector is $\mathbf{p} = [p_1, p_2, \ldots p_{128}]$, where the control points are placed equidistantly over the prediction horizon and follows a linear profile. Once the OCP is solved for $\mathbf{p}^*$, the input $u(\mathbf{p}^*)$ is injected into the system over the period T_s and this process is repeated in a receding horizon fashion. The pNMPC S/W settings for both the CPU and GPU version were **Integrator** - 4th order explicit Runge Kutta integrator, **Step size** - $0.2s$, **Prediction horizon** - 10s, **Number of SQP iterations** - 8, **Constraint form** - `CONST_FORM::FORM_1` and for the rest, default parameters were utilized.

Fig. 5.17 illustrates the crane position, swing angle and the angular velocity for pNMPC-GPU and pNMPC-CPU controllers respectively. As from the plot, the difference in perfor-

mance of the controllers isn't very significant. Both nearly performs well, which validates the performance of pNMPC-GPU version.

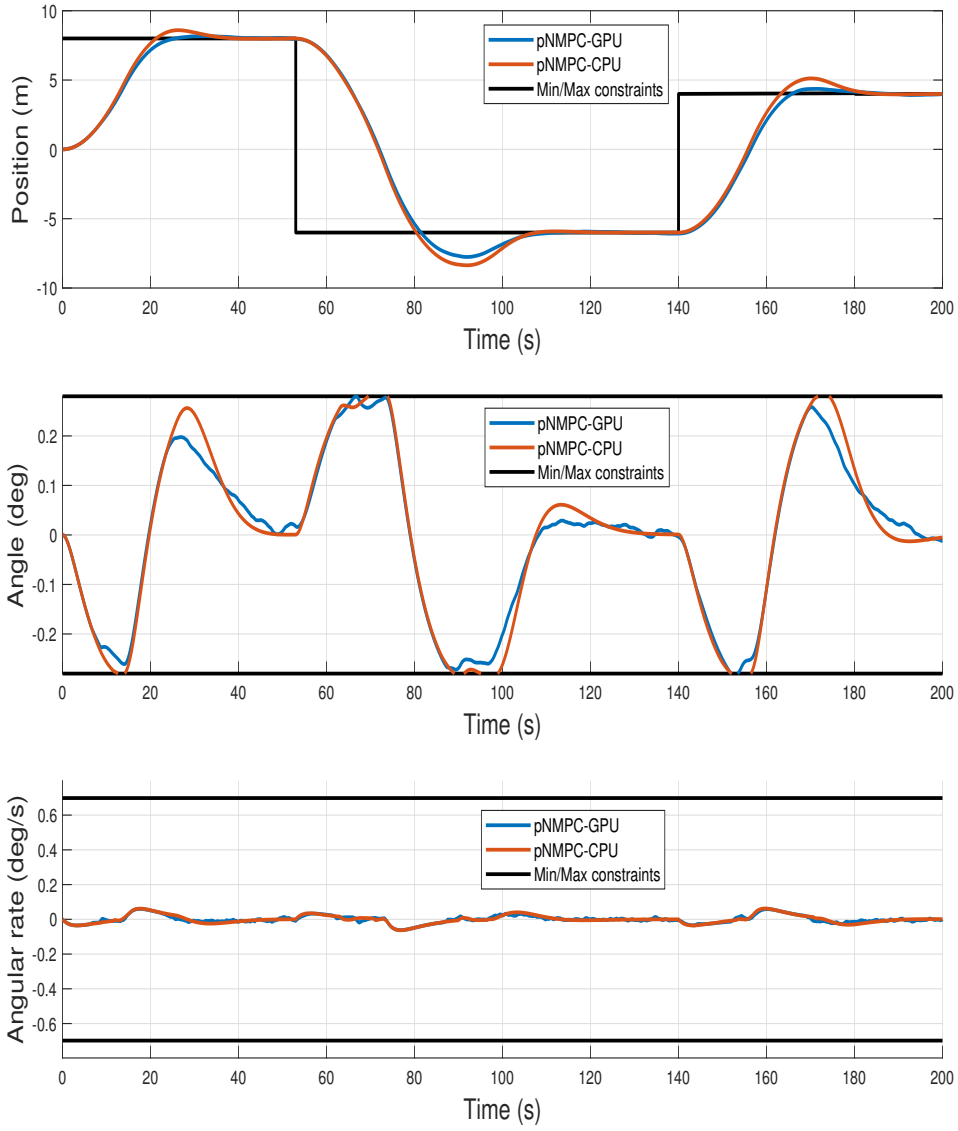

Figure 5.17: Crane position, swing angle and angular velocity comparison b/w pNMPC-GPU and pNMPC-CPU controller

Fig. 5.18 illustrates the input force to the crane for pNMPC-GPU and pNMPC-CPU controllers respectively.

Remark: The NVIDIA CUDA based GPUs intrinsically uses floating (single) precision for parallel computing. The fluctuations in the input as seen in the plot for the pNMPC-GPU

controller stems due to single precision operations. However, one can observe that on an average the pNMPC-GPU controller follows the pNMPC-CPU controller's input profile. In later versions of the S/W, support for double precision computation would be implemented.

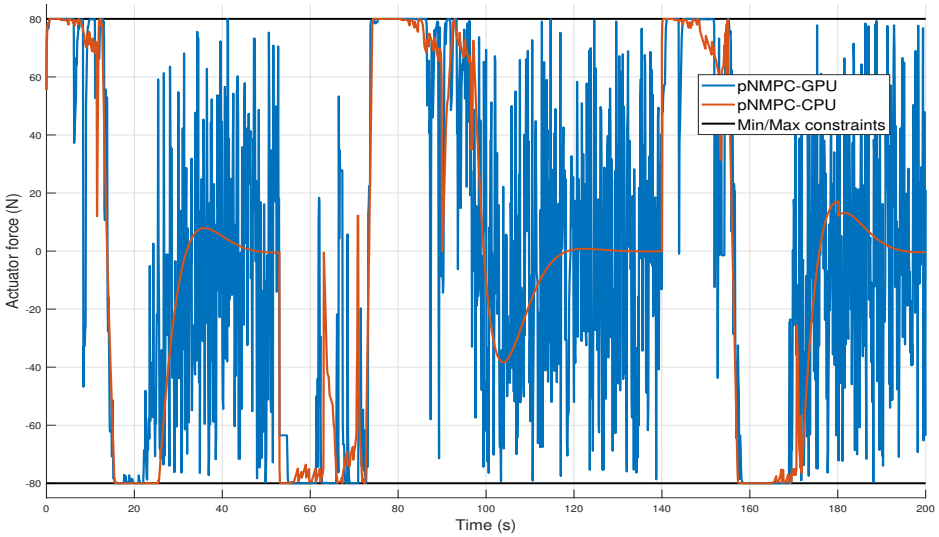

Figure 5.18: Crane input force comparison b/w pNMPC-GPU and pNMPC-CPU controller

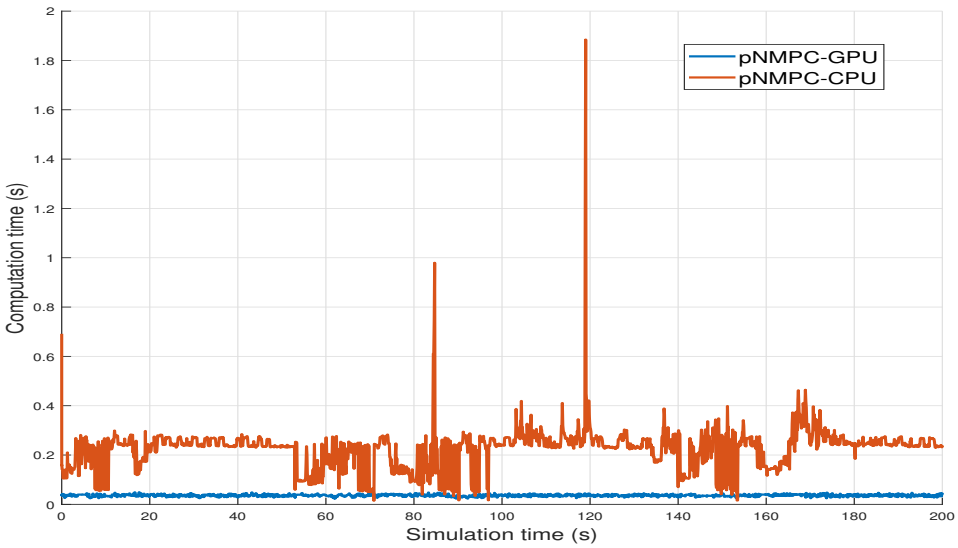

Figure 5.19: Computation time pNMPC-GPU vs pNMPC-CPU controller

Fig. 5.19 illustrates the computation time for pNMPC-GPU and pNMPC-CPU controllers respectively. It is clearly evident that computation time for pNMPC-GPU version is nearly ten folds lower than the pNMPC-CPU version and this difference would be more pronounced when large parameterization vectors (greater than 128) are used, as the throughput of CUDA GPU

devices are much larger. It is also important to note that the sampling period T_s considered for this example is $0.1s$ and the mean computation time of pNMPC-CPU controller hovers around $0.2232s$ and renders impossible for RT implementation. However, the mean computation time of pNMPC-GPU controller hovers around $0.036s$, which is feasible for RT implementation.

5.9 Future works and conclusions

The crux of this chapter is to present a derivative free pNMPC code generation S/W and to validate its performance by means of simulation on multiple examples. From the simulation study as well as from HiL simlations conducted, it is certainly evident that the proposed S/W has applications for several engineering systems, where the model exists either as computer codes or as a data driven model. Despite the S/W is suffice for several real world applications, there are certain directions for improvement. The future line of work can be stratified into two parts a) Technical challenges and b) Software challenges. A detailed examination of the aforementioned division of work is expounded below

- **Smart parameterization** - As mentioned in the Section 5.1.2 and to re-emphasize the fact again that the potential of the method solely depends upon the parameterization technique adopted by the control engineer. However, in cases when this becomes intricate in nature, the onus of determining an optimal parameterization is substantially increased. In such a situation, it would be remiss of not utilizing tools from the machine learning community. Methods developed in reinforcement learning (RL) [Sutton and Barto 2018] zones in parallel to the proposed approach and by availing tools and methods from RL community a smart parameterization technique for pNMPC controller can be developed.

- **Implicit solvers** - As of now, the proposed S/W provides only explicit ODE solvers for the pNMPC controller. However, in many applications where the system is intrinsically stiff in nature, one is obliged to use implicit solvers for numerical stability. The future version of the S/W would encompass support for several implicit solvers.

- **Equality constraints** - In the future, the S/W would include support for equality constraints, which is important for control of periodic systems or to enforce time point constraints on the system.

- **PDEs, DAEs and Hybrid systems** - In the future, the S/W would include support for control of partial differential equations (PDEs), differential algebraic equations (DAEs) and hybrid systems. The overarching goal is to widen the scope of the S/W and provide a one-stop shop pNMPC solution.

- **S/W challenges** - The current S/W provides a primitive interface to MATLAB/Simulink. In the future, it is planned to provide better interfaces to MATLAB/Simulink, Python and Julia to benefit all the embedded programmers across the board.

To recall, the GitHub repository for the pNMPC S/W tool is available in this link - The pNMPC S/W is available in GitHub repository in `https://github.com/Kartz4code/pNMPC_CODEGEN`.

Appendix A

pNMPC C++ code examples

Listing A.1: pNMPC OCP C++ code for cart-pole swing up problem

```
#include "pNMPC_headers.hpp"
#define PI 3.1416
using namespace pNMPC;
int main()
{
// States
States p, theta, pd, thetad;
// Inputs
Inputs p1, p2, p3, p4, p5;
// Constant parameters
Real m1 = 1, m2 = 0.1, g = 9.81, l = 0.5;
// Bounds
Real pmax = 2, umax = 10, thetamax = 2*PI;
// OCP data
MATHyperStates Xs(4,1); Xs = {p,theta,pd,thetad};
MATReal Q = diag({5,10,1,1}); MATReal Qf = diag({10,20,1,1});
Real R = 0.1;
// Input parameterization
BEGIN_PARAMETERIZATION_MAP
    ParameterizationMap u = p1*p + p2*theta + p3*pd + p4*thetad + p5;
END_PARAMETERIZATION_MAP
// Differential equations
BEGIN_DIFFERENTIAL
    DiffEquation d1 = pd;
    DiffEquation d2 = thetad;
    DiffEquation d3 = (-l*m2*sin(theta)*((thetad)^2)+u+
    m2*g*cos(theta)*sin(theta))/(m1+m2*(1-(cos(theta))^2));
    DiffEquation d4 = (-l*m2*cos(theta)*sin(theta)*((thetad)^2)+
    u*cos(theta)+(m1+m2)*g*sin(theta))/(l*m1+l*m2*(1-(cos(theta))^2);
END_DIFFERENTIAL
// Constraints
BEGIN_CONSTRAINTS
    BEGIN_REGULAR_CONSTRAINTS
        ScalarConstraint G1 = {-umax <= u <= umax};
```

```
35          ScalarConstraint G2 = {-pmax <= p <= pmax};
36          ScalarConstraint G3 = {-thetamax <= theta <= thetamax};
37      END_REGULAR_CONSTRAINTS
38  END_CONSTRAINTS
39  // Lagrangian and Mayer Cost
40  BEGIN_OBJECTIVES
41  // Lagrangian Cost
42      BEGIN_LAGRANGIAN
43          ScalarObjective LC1 = transpose(Xs)*Q*(Xs);
44          ScalarObjective LC2 = R*((u)^2);
45      END_LAGRANGIAN
46  // Mayer Cost
47      BEGIN_MAYER
48          ScalarObjective MC1 = transpose(Xs)*Qf*(Xs);
49      END_MAYER
50  END_OBJECTIVES
51  // Input parameterization
52  ControlParamZ<Linear>{1,p1};
53  ControlParamZ<Linear>{1,p2};
54  ControlParamZ<Linear>{1,p3};
55  ControlParamZ<Linear>{1,p4};
56  ControlParamZ<Linear>{1,p5};
57  // PNMPCGEN singleton object
58  PNMPCGEN* pNMPC = PNMPCGEN::getSton();
59  pNMPC->setInitialTime(0);
60  pNMPC->setFinalTime(1);
61  pNMPC->setStepSize(0.05);
62  pNMPC->getSolver()->setNiter(4);
63  pNMPC->setConstForm(CONST_FORM::FORM_1);
64  pNMPC->setIntegrator(INTEGRATOR::RK45);
65  // Generate C code
66  pNMPC->genCCode();
67  // Free allocated memory
68  pNMPC_free();
69  return 0;
70  }
```

Listing A.2: pNMPC OCP C++ code for PVTOL stabilization problem

```
1  #include "pNMPC_headers.hpp"
2  using namespace pNMPC;
3  int main()
4  {
5  // States
6  States x1, x2, x3, x4, x5, x6;
```

```
7  // Inputs
8  Inputs p1, p2;
9  // External variables
10 External psi_y = "ModelY";
11 External psi_z = "ModelZ";
12 // Constant parameters
13 Real a_l = 0, a_u = 2, alp_l = -2, alp_u = 2;
14 // OCP data
15 MATHyperStates Xs(4,1), Us(2,1);
16 Xs = {x1,x2,x3,x4,x5,x6};
17 MATReal Q = 5*eye(6);
18 MATReal Qf = 10*eye(6);
19 Real R = 0.01*eye(2);
20 // Input parameterization
21 BEGIN_PARAMETERIZATION_MAP
22     ParameterizationMap a = p1;
23     ParameterizationMap alp = p2;
24 END_PARAMETERIZATION_MAP
25 Us = {a,alp};
26 // Differential equations
27 BEGIN_DIFFERENTIAL
28     DiffEquation d1 = x2;
29     DiffEquation d2 = psi_y;
30     DiffEquation d3 = x4;
31     DiffEquation d4 = psi_z;
32     DiffEquation d5 = x6;
33     DiffEquation d6 = alp;
34 END_DIFFERENTIAL
35 // Constraints
36 BEGIN_CONSTRAINTS
37     BEGIN_REGULAR_CONSTRAINTS
38         ScalarConstraint G1 = {a_l <= a <= a_u};
39         ScalarConstraint G2 = {alp_l <= alp <= alp_u};
40         ScalarConstraint G3 = {-15*PI/180 <= x5 <= 15*PI/180};
41     END_REGULAR_CONSTRAINTS
42 END_CONSTRAINTS
43 // Lagrangian and Mayer Cost
44 BEGIN_OBJECTIVES
45 // Lagrangian Cost
46     BEGIN_LAGRANGIAN
47         ScalarObjective LC1 = transpose(Xs)*Q*(Xs);
48         ScalarObjective LC2 = transpose(Us)*R*(Us);
49     END_LAGRANGIAN
50 // Mayer Cost
```

```
51      BEGIN_MAYER
52           ScalarObjective MC1 = transpose(Xs)*Qf*(Xs);
53      END_MAYER
54 END_OBJECTIVES
55 // Input parameterization
56 ControlParamZ<Linear>{2,p1,a_l,a_u};
57 ControlParamZ<Linear>{2,p2,alp_l,alp_u};
58 // PNMPCGEN singleton object
59 PNMPCGEN* pNMPC = PNMPCGEN::getSton();
60 pNMPC->setInitialTime(0);
61 pNMPC->setFinalTime(2);
62 pNMPC->setStepSize(0.1);
63 pNMPC->getSolver()->setNiter(4);
64 pNMPC->setConstForm(CONST_FORM::FORM_1);
65 pNMPC->setIntegrator(INTEGRATOR::RK45);
66 // Generate C code
67 pNMPC->genCCode();
68 // Free allocated memory
69 pNMPC_free();
70 return 0;
71 }
```

Listing A.3: pNMPC OCP C++ code for 2D Crane control problem

```
1  #include "pNMPC_headers.hpp"
2  using namespace pNMPC;
3  int main()
4  {
5  // States
6  States r, rd, theta, thetad;
7  // Inputs
8  Inputs p1;
9  // Parameter
10 Params kappa;
11 Real m = 200, M = 1500, l = 100, fr = pow(10,5), umax = 80;
12 Real thetamax = 0.005, thetadmax = (2*PI)/9, g = 9.81;
13 // State and input buffer
14 MATHyperStates Xs(4,1), Xd(4,1);
15 // Set point for tracking
16 Xd(0,0) = kappa;
17 // States buffer
18 Xs(0,0)= r; Xs(1,0) = rd;  Xs(2,0) = theta; Xs(3,0) = thetad;
19 // Input parameterization
20 BEGIN_PARAMETERIZATION_MAP
21      ParameterizationMap u = p1;
```

```
22 END_PARAMETERIZATION_MAP
23 // Differential equations
24 BEGIN_DIFFERENTIAL
25     DiffEquation d1 = rd;
26     DiffEquation d2 = (u + m*g*cos(theta)*sin(theta)
27                        + m*l*thetad*thetad*sin(theta))/
28                        (M + m*(1-(cos(theta)*cos(theta))));
29     DiffEquation d3 = thetad;
30     DiffEquation d4 = (-u*cos(theta) -
31                        m*l*thetad*thetad*cos(theta)*sin(theta) -
32                        (M+m)*g*sin(theta) - fr*thetad)/
33                        ((M + m*sin(theta)*sin(theta))*l);
34 END_DIFFERENTIAL
35 // Constraints
36 BEGIN_CONSTRAINTS
37     BEGIN_REGULAR_CONSTRAINTS
38         ScalarConstraint G1 = {-umax <= u <= umax };
39         ScalarConstraint G2 = {-thetamax <= theta <= thetamax };
40         ScalarConstraint G3 ={-thetadmax <= thetad <= thetadmax};
41     END_REGULAR_CONSTRAINTS
42 END_CONSTRAINTS
43 // Cost matrices
44 MATReal Q = diag({100,1,1,1});
45 MATReal Qf = 100*eye(4);
46 Real R = 0.01;
47 // Lagrangian and Mayer Cost
48 BEGIN_OBJECTIVES
49     // Lagrangian Cost
50     BEGIN_LAGRANGIAN
51         ScalarObjective LC1 =  0.5*transpose(Xs-Xd)*Q*(Xs-Xd);
52         ScalarObjective LC2 =  0.5*R*u*u;
53     END_LAGRANGIAN
54     //  Mayer Cost
55     BEGIN_MAYER
56         ScalarObjective MC = 0.5*transpose(Xs-Xd)*Qf*(Xs-Xd);
57     END_MAYER
58 END_OBJECTIVES
59 // Input parameterization
60 ControlParamZ<Linear>{1024, p1, -umax, umax };
61 // pNMPC codegen object
62 PNMPCGEN* pNMPC = PNMPCGEN::getSton();
63 pNMPC->setInitialTime(0);
64 pNMPC->setFinalTime(10);
65 pNMPC->setStepSize(0.2);
```

```
66 pNMPC->getSolver()->setNiter(8);
67 pNMPC->setConstForm(CONST_FORM::FORM_1);
68 pNMPC->setIntegrator(INTEGRATOR::RK45);
69 pNMPC->setParameterPredict(0);
70 // Generate C code
71 pNMPC->genCCode();
72 // Free memory
73 pNMPC::pNMPC_free();
74 }
```

Listing A.4: pNMPC OCP C++ code for control of semi-active suspension system for quarter car model

```
1  #include "pNMPC_headers.hpp"
2  using namespace pNMPC;
3  int main()
4  {
5  // States
6  States x1, x2, x3, x4, x5, x6;
7  // Inputs
8  Inputs p1, p2;
9  // External variables
10 External psi_y = "ModelY";
11 External psi_z = "ModelZ";
12 // Constant parameters
13 Real a_l = 0, a_u = 2, alp_l = -2, alp_u = 2;
14 // OCP data
15 MATHyperStates Xs(4,1), Us(2,1);
16 Xs = {x1,x2,x3,x4,x5,x6};
17 MATReal Q = 5*eye(6);
18 MATReal Qf = 10*eye(6);
19 Real R = 0.01*eye(2);
20 // Input parameterization
21 BEGIN_PARAMETERIZATION_MAP
22     ParameterizationMap a = p1;
23     ParameterizationMap alp = p2;
24 END_PARAMETERIZATION_MAP
25 Us = {a,alp};
26 // Differential equations
27 BEGIN_DIFFERENTIAL
28     DiffEquation d1 = x2;
29     DiffEquation d2 = psi_y;
30     DiffEquation d3 = x4;
31     DiffEquation d4 = psi_z;
32     DiffEquation d5 = x6;
```

```
33      DiffEquation d6 = alp;
34  END_DIFFERENTIAL
35  // Constraints
36  BEGIN_CONSTRAINTS
37      BEGIN_REGULAR_CONSTRAINTS
38          ScalarConstraint G1 = {a_l <= a <= a_u};
39          ScalarConstraint G2 = {alp_l <= alp <= alp_u};
40          ScalarConstraint G3 = {-15*PI/180 <= x5 <= 15*PI/180};
41      END_REGULAR_CONSTRAINTS
42  END_CONSTRAINTS
43  // Lagrangian and Mayer Cost
44  BEGIN_OBJECTIVES
45      // Lagrangian Cost
46      BEGIN_LAGRANGIAN
47          ScalarObjective LC1 = transpose(Xs)*Q*(Xs);
48          ScalarObjective LC2 = transpose(Us)*R*(Us);
49      END_LAGRANGIAN
50  // Mayer Cost
51      BEGIN_MAYER
52          ScalarObjective MC1 = transpose(Xs)*Qf*(Xs);
53      END_MAYER
54  END_OBJECTIVES
55  // Input parameterization
56  ControlParamZ<Linear>{2,p1,a_l,a_u};
57  ControlParamZ<Linear>{2,p2,alp_l,alp_u};
58  // PNMPCGEN singleton object
59  PNMPCGEN* pNMPC = PNMPCGEN::getSton();
60  pNMPC->setInitialTime(0);
61  pNMPC->setFinalTime(2);
62  pNMPC->setStepSize(0.1);
63  pNMPC->getSolver()->setNiter(4);
64  pNMPC->setConstForm(CONST_FORM::FORM_1);
65  pNMPC->setIntegrator(INTEGRATOR::RK45);
66  // Generate C code
67  pNMPC->genCCode();
68  // Free allocated memory
69  pNMPC_free();
70  return 0;
71  }
```

Bibliography

Abughalieh, Karam M and Shadi G Alawneh (2019). "A Survey of Parallel Implementations for Model Predictive Control." In: *IEEE Access* (cit. on p. 67).

Adams, Brian M et al. (2009). "DAKOTA, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: version 5.0 user's manual." In: *Sandia National Laboratories, Tech. Rep. SAND2010-2183* (cit. on p. 31).

Aho, Alfred V, Ravi Sethi, and Jeffrey D Ullman (1986). "Compilers, principles, techniques." In: *Addison wesley* 7.8, p. 9 (cit. on p. 111).

Alamir, Mazen (2006). *Stabilization of nonlinear systems using receding-horizon control schemes: a parametrized approach for fast systems*. Vol. 339. Springer (cit. on pp. 34, 95).

— (2012). "A framework for real-time implementation of low-dimensional parameterized NMPC." In: *Automatica* 48.1, pp. 198–204 (cit. on pp. 100, 107).

— (2013). *A pragmatic story of model predictive control: self-contained algorithms and case-studies*. CreateSpace Independent Publishing Platform (cit. on pp. 100, 107, 122).

— (2017). "The PDF-MPC package: A free-MATLAB-coder package for real-time nonlinear model predictive control." In: *arXiv preprint arXiv:1703.08255* (cit. on pp. 31, 94, 100).

Ascher, Uri M and Linda R Petzold (1998). *Computer methods for ordinary differential equations and differential-algebraic equations*. Vol. 61. Siam (cit. on pp. 34, 52).

Astrom, Karl J and Bjorn Wittenmark (1982). *Computer controlled systems: theory and design*. Vol. 3. Prentice-Hall Englewood Cliffs, New Jersey (cit. on p. 47).

Aswani, Anil et al. (2013). "Provably safe and robust learning-based model predictive control." In: *Automatica* 49.5, pp. 1216–1226 (cit. on p. 2).

Aubouet, Sébastien (2010). "Modélisation et commande de suspensions semi-actives SOBEN." PhD thesis. Institut National Polytechnique de Grenoble-INPG (cit. on p. 13).

Baumal, AE, JJ McPhee, and PH Calamai (1998). "Application of genetic algorithms to the design optimization of an active vehicle suspension system." In: *Computer methods in applied mechanics and engineering* 163.1-4, pp. 87–94 (cit. on pp. 71, 80).

Bazaraa, Mokhtar S, Hanif D Sherali, and Chitharanjan M Shetty (2013). *Nonlinear programming: theory and algorithms*. John Wiley & Sons (cit. on p. 29).

Beck, Amir (2014). *Introduction to nonlinear optimization: Theory, algorithms, and applications with MATLAB*. SIAM (cit. on p. 31).

— (2017). *First-order methods in optimization*. SIAM (cit. on p. 29).

Becker, Stephen, E Candes, and M Grant (2011). "TFOCS: flexible first-order methods for rank minimization." In: *Low-rank Matrix Optimization Symposium, SIAM Conference on Optimization* (cit. on p. 29).

Bemporad, Alberto et al. (2002). "The explicit linear quadratic regulator for constrained systems." In: *Automatica* 38.1, pp. 3–20 (cit. on p. 2).

Bertsekas, Dimitri P (2019). *Reinforcement learning and optimal control*. Athena Scientific Belmont, MA (cit. on p. 33).

Bertsekas, Dimitri P et al. (1995). *Dynamic programming and optimal control*. Vol. 1. 2. Athena scientific Belmont, MA (cit. on p. 33).

Blochwitz, Torsten (2014). "Functional mock-up interface for model exchange and co-simulation." In: *July [Online] https://www.fmi-standard.org/Downloads (Accessed January 2016)* (cit. on p. 20).

Blochwitz, Torsten et al. (2011). "The functional mockup interface for tool independent exchange of simulation models." In: *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*. 063. Linköping University Electronic Press, pp. 105–114 (cit. on p. 19).

Blockwitz, Torsten et al. (2012). "Functional mockup interface 2.0: The standard for tool independent exchange of simulation models." In: *Proceedings* (cit. on pp. 19, 66, 95).

Bojarski, Mariusz et al. (2016). "End to end learning for self-driving cars." In: *arXiv preprint arXiv:1604.07316* (cit. on p. 2).

Borrelli, Francesco, Alberto Bemporad, and Manfred Morari (2017). *Predictive control for linear and hybrid systems*. Cambridge University Press (cit. on p. 26).

Boyd, Stephen, Stephen P Boyd, and Lieven Vandenberghe (2004). *Convex optimization*. Cambridge university press (cit. on p. 27).

Boyd, Stephen, Neal Parikh, and Eric Chu (2011). *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc (cit. on p. 28).

Bujarbaruah, Monimoy et al. (2018). "Adaptive MPC for iterative tasks." In: *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, pp. 6322–6327 (cit. on p. 2).

Butz, T and O Von Stryk (2002). "Modelling and Simulation of Electro-and Magnetorheological Fluid Dampers." In: *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik: Applied Mathematics and Mechanics* 82.1, pp. 3–20 (cit. on p. 44).

Byrd, Richard H, Jorge Nocedal, and Richard A Waltz (2006). "KNITRO: An integrated package for nonlinear optimization." In: *Large-scale nonlinear optimization*. Springer, pp. 35–59 (cit. on p. 30).

Canale, M, M Milanese, and C Novara (2006). "Semi-active suspension control using "fast" model-predictive techniques." In: *IEEE Transactions on Control Systems Technology* 14.6, pp. 1034–1046 (cit. on p. 43).

Conn, Andrew R, Katya Scheinberg, and Luis N Vicente (2009). *Introduction to derivative-free optimization*. SIAM (cit. on p. 31).

Conn, Andy, Katya Scheinberg, and Ph Toint (1998). "A derivative free optimization algorithm in practice." In: *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, p. 4718 (cit. on p. 31).

Cseko, L.H., M. Kvasnica, and B. Lantos (2010). "Analysis of the explicit model predictive control for semi-active suspension." In: *Periodica Polytechnica, Electrical Engineering* 54 (cit. on p. 43).

Custódio, AL and JF Aguilar Madeira (2015). "GLODS: global and local optimization using direct search." In: *Journal of global optimization* 62.1, pp. 1–28 (cit. on p. 31).

Custódio, Ana Luísa, Katya Scheinberg, and Luís Nunes Vicente (2017). "Methodologies and software for derivative-free optimization." In: *Advances and Trends in Optimization with Engineering Applications*, pp. 495–506 (cit. on p. 31).

Custodio, Ana Luisa and Luis N Vicente (2008). "SID-PSM: A pattern search method guided by simplex derivatives for use in derivative-free optimization." In: *Departamento de Matemática, Universidade de Coimbra, Coimbra* (cit. on p. 31).

Cutler, Charles R and Brian L Ramaker (1980). "Dynamic matrix control?? A computer control algorithm." In: *joint automatic control conference*. 17, p. 72 (cit. on p. 2).

Deng, Haoyang and Toshiyuki Ohtsuka (2018). "A parallel code generation toolkit for nonlinear model predictive control." In: *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, pp. 4920–4926 (cit. on p. 97).

Dennis Jr, John E and Jorge J Moré (1977). "Quasi-Newton methods, motivation and theory." In: *SIAM review* 19.1, pp. 46–89 (cit. on p. 29).

Do, Anh Lam (2011). "LPV approach for the vehicles dynamics robust control: joint comfort and safety improvement." PhD thesis. Université de Grenoble (cit. on p. 12).

Do, Anh-Lam, Olivier Sename, and Luc Dugard (2010). "An LPV control approach for semi-active suspension control with actuator constraints." In: *American Control Conference*, pp. 4653–4658 (cit. on p. 43).

Domahidi, Alexander and Juan Jerez (2014). "FORCES Professional. embotech GmbH (http://embotech. com/FORCES-Pro)." In: *J uly* (cit. on p. 30).

Doumiati, Moustapha et al. (2017). "Road profile estimation using an adaptive Youla–Kučera parametric observer: Comparison to real profilers." In: *Control Engineering Practice* 61, pp. 270–278 (cit. on pp. 71, 81).

Drud, Arne Stolbjerg (1994). "CONOPT—a large-scale GRG code." In: *ORSA Journal on computing* 6.2, pp. 207–216 (cit. on p. 30).

Ellis, Matthew, Helen Durand, and Panagiotis D Christofides (2014). "A tutorial review of economic model predictive control methods." In: *Journal of Process Control* 24.8, pp. 1156–1178 (cit. on p. 2).

EMPHYSIS (2017). *EMPHYSIS: Embedded systems with physical models in the production code software*. `https://itea3.org/project/emphysis.html`. "[Online; accessed 22-June-2020]" (cit. on pp. 1, 20–22, 42, 95).

Ferreau, Hans Joachim et al. (2014a). "qpOASES: A parametric active-set algorithm for quadratic programming." In: *Mathematical Programming Computation* 6.4, pp. 327–363 (cit. on p. 30).

— (2014b). "qpOASES: A parametric active-set algorithm for quadratic programming." In: *Mathematical Programming Computation* 6.4, pp. 327–363 (cit. on p. 96).

Fischer, Daniel and Rolf Isermann (2004). "Mechatronic semi-active and active vehicle suspensions." In: *Control Engineering Practice* 12.11, pp. 1353–1367 (cit. on p. 12).

Fletcher, Roger and Sven Leyffer (1998). "User manual for filterSQP." In: *Numerical Analysis Report NA/181, Department of Mathematics, University of Dundee, Dundee, Scotland* 35 (cit. on p. 30).

Fougner, Christopher and Stephen Boyd (2018). "Parameter selection and preconditioning for a graph form solver." In: *Emerging Applications of Control and Systems Theory*. Springer, pp. 41–61 (cit. on p. 29).

Fürst, Simon et al. (2009). "AUTOSAR–A Worldwide Standard is on the Road." In: *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*. Vol. 62, p. 5 (cit. on p. 22).

Gade-Nielsen, Nicolai Fog, Bernd Dammann, and John Bagterp Jørgensen (2014). "Interior point methods on GPU with application to model predictive control." In: (cit. on p. 67).

Gilbert, J Charles (2009). *SQPlab–A Matlab software for solving nonlinear optimization problems and optimal control problems* (cit. on p. 30).

Gill, Philip E, Walter Murray, and Michael A Saunders (2005). "SNOPT: An SQP algorithm for large-scale constrained optimization." In: *SIAM review* 47.1, pp. 99–131 (cit. on p. 30).

Gill, Philip E et al. (1986). *User's guide for NPSOL (version 4.0): A Fortran package for nonlinear programming*. Tech. rep. Stanford Univ CA Systems Optimization Lab (cit. on p. 30).

Gillespie, Thomas D (1992). *Fundamentals of vehicle dynamics* (cit. on p. 10).

Giorgetti, N. et al. (2006). "Hybrid model predictive control application towards optimal semi-active suspension." In: *International Journal of Control* 79, pp. 521–533 (cit. on p. 43).

Giselsson, Pontus (2018). "QPgen." In: *Dosegljivo: http://www. control. lth. se/QPgen/index. html* (cit. on p. 29).

Gohrle, Christoph et al. (2012). "Active suspension controller using MPC based on a full-car model with preview information." In: *American Control Conference (ACC), 2012*. IEEE, pp. 497–502 (cit. on p. 43).

Goodarzi, Avesta and Amir Khajepour (2017). "Vehicle suspension system technology and design." In: *Synthesis Lectures on Advances in Automotive Technology* 1.1, pp. i–77 (cit. on pp. 10, 16).

Goodfellow, Ian et al. (2016). *Deep learning*. Vol. 1. MIT press Cambridge (cit. on p. 28).

Griewank, Andreas and Andrea Walther (2008). *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM (cit. on p. 35).

Griewank, Andreas, Andrea Walther, and Maciek Korzec (2007). "Maintaining factorized KKT systems subject to rank-one updates of Hessians and Jacobians." In: *Optimisation Methods and Software* 22.2, pp. 279–295 (cit. on p. 30).

Gros, Sébastien et al. (2020). "From linear to nonlinear MPC: bridging the gap via the real-time iteration." In: *International Journal of Control* 93.1, pp. 62–80 (cit. on pp. 36–38).

Guanetti, Jacopo and Francesco Borrelli (2017). "Stochastic MPC for cloud-aided suspension control." In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, pp. 238–243 (cit. on pp. 66, 67, 69).

Guo, Shuqi, Shaopu Yang, and Cunzhi Pan (2006). "Dynamic modeling of magnetorheological damper behaviors." In: *Journal of Intelligent material systems and structures* 17.1, pp. 3–14 (cit. on pp. 44, 45).

Hansen, Nikolaus, Sibylle D Müller, and Petros Koumoutsakos (2003). "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)." In: *Evolutionary computation* 11.1, pp. 1–18 (cit. on p. 31).

Holmström, Kenneth, Anders O Göran, and Marcus M Edvall (2010). "User's Guide for TOMLAB 7." In: *Tomlab Optimization Inc* (cit. on p. 31).

Houska, Boris, Hans Joachim Ferreau, and Moritz Diehl (2011). "An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range." In: *Automatica* 47.10, pp. 2279–2285 (cit. on pp. 96, 113).

Houska, Boris et al. (2009). "ACADO toolkit user's manual." In: (cit. on p. 75).

Huyer, Waltraud and Arnold Neumaier (2008). "SNOBFIT–stable noisy optimization by branch and fit." In: *ACM Transactions on Mathematical Software (TOMS)* 35.2, pp. 1–25 (cit. on p. 31).

Hyatt, Phillip and Marc D Killpack (2020). "Real-time nonlinear model predictive control of robots using a graphics processing unit." In: *IEEE Robotics and Automation Letters* 5.2, pp. 1468–1475 (cit. on p. 67).

Ingber, Lester (1989). "Very fast simulated re-annealing." In: *Mathematical and computer modelling* 12.8, pp. 967–973 (cit. on p. 31).

ISO, I (1995). *Mechanical vibration—Road surface profiles—Reporting of measured data* (cit. on p. 69).

J Lozoya-Santos, Jorge de et al. (2012). "Magnetorheological damper—an experimental study." In: *Journal of Intelligent Material Systems and Structures* 23.11, pp. 1213–1232 (cit. on p. 13).

Kalmari, Jouko, Juha Backman, and Arto Visala (2015). "A toolkit for nonlinear model predictive control using gradient projection and code generation." In: *Control Engineering Practice* 39, pp. 56–66 (cit. on p. 96).

Kamath, Gopalakrishna M and Norman M Wereley (1997). "A nonlinear viscoelastic-plastic model for electrorheological fluids." In: *Smart Materials and Structures* 6.3, p. 351 (cit. on p. 44).

Karnopp, Dean, Michael J Crosby, and RA Harwood (1974). "Vibration control using semi-active force generators." In: *Journal of engineering for industry* 96.2, pp. 619–626 (cit. on pp. 42, 60).

Kelley, CT (2011). "Users Guide for IMFIL version 1.0." In: *Available at www4. ncsu. edu/˜ ctk/imfil. html* (cit. on p. 31).

Koegel, MJ and R Findeisen (2012). "Parallel architectures for model predictive control." In: *4th IFAC Nonlinear Model Predictive Control Conference*. Vol. 4 (cit. on p. 67).

Kouzoupis, D et al. (2015). "Block condensing for fast nonlinear MPC with the dual Newton strategy." In: *IFAC-PapersOnLine* 48.23, pp. 26–31 (cit. on p. 96).

Kvasnica, Michal, Ivana Rauová, and Miroslav Fikar (2010). "Automatic code generation for real-time implementation of model predictive control." In: *2010 IEEE International Symposium on Computer-Aided Control System Design*. IEEE, pp. 993–998 (cit. on p. 96).

Larson, Jeffrey, Matt Menickelly, and Stefan M Wild (2019). "Derivative-free optimization methods." In: *arXiv preprint arXiv:1904.11585* (cit. on p. 31).

Le Digabel, Sébastien (2011). "Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm." In: *ACM Transactions on Mathematical Software (TOMS)* 37.4, pp. 1–15 (cit. on p. 31).

Leyffer, Sven and Ashutosh Mahajan (2010). "Nonlinear constrained optimization: methods and software." In: *Argonee National Laboratory, Argonne, Illinois* 60439 (cit. on p. 30).

Liberzon, Daniel (2011). *Calculus of variations and optimal control theory: a concise introduction*. Princeton University Press (cit. on p. 33).

Lin, Youdong and Linus Schrage (2009). "The global solver in the LINDO API." In: *Optimization Methods & Software* 24.4-5, pp. 657–668 (cit. on p. 30).

Ljung, Lennart (1987). *System identification: theory for the user*. Prentice-hall (cit. on p. 48).

Maciejowski, Jan Marian (2002). *Predictive control: with constraints*. Pearson education (cit. on p. 32).

Mahala, K Manoj, Prasanna Gadkari, and Anindya Deb (2009). "Mathematical models for designing vehicles for ride comfort." In: *ICORD 09: Proceedings of the 2nd International Conference on Research into Design, Bangalore, India 07.-09.01. 2009* (cit. on p. 14).

Martin, Philippe, Santosh Devasia, and Brad Paden (1996). "A different look at output tracking: control of a VTOL aircraft." In: *Automatica* 32.1, pp. 101–107 (cit. on p. 116).

Mattingley, Jacob and Stephen Boyd (2012). "CVXGEN: A code generator for embedded convex optimization." In: *Optimization and Engineering* 13.1, pp. 1–27 (cit. on pp. 44, 56, 96).

Mayne, David Q, María M Seron, and SV Raković (2005). "Robust model predictive control of constrained linear systems with bounded disturbances." In: *Automatica* 41.2, pp. 219–224 (cit. on p. 2).

Mesbah, Ali (2016). "Stochastic model predictive control: An overview and perspectives for future research." In: *IEEE Control Systems Magazine* 36.6, pp. 30–44 (cit. on p. 2).

Miller, Lane R (1988). "Tuning passive, semi-active, and fully active suspension systems." In: *Proceedings of the 27th IEEE Conference on Decision and Control*. IEEE, pp. 2047–2053 (cit. on p. 12).

Mills, Adam, Adrian Wills, and Brett Ninness (2009). "Nonlinear model predictive control of an inverted pendulum." In: *2009 American control conference*. IEEE, pp. 2335–2340 (cit. on p. 114).

Morato, Marcelo M, Julio E Normey-Rico, and Olivier Sename (2020). "Model predictive control design for linear parameter varying systems: A survey." In: *Annual Reviews in Control* (cit. on p. 2).

Morato, Marcelo Menezes et al. (2019). "Design of a fast real-time LPV model predictive control system for semi-active suspension control of a full vehicle." In: *Journal of the Franklin Institute* 356.3, pp. 1196–1224 (cit. on p. 43).

Moré, Jorge J and Stefan M Wild (2009). "Benchmarking derivative-free optimization algorithms." In: *SIAM Journal on Optimization* 20.1, pp. 172–191 (cit. on p. 31).

Naidu, D Subbaram (2002). *Optimal control systems*. CRC press (cit. on p. 33).

Nguyen, Manh Quan (2016a). "LPV approaches for modelling and control of vehicle dynamics: application toa small car pilot plant with ER dampers." PhD thesis. Université Grenoble Alpes (cit. on p. 12).

— (2016b). "LPV approaches for modelling and control of vehicle dynamics: application toa small car pilot plant with ER dampers." PhD thesis. Université Grenoble Alpes (cit. on p. 13).

Nguyen, Minh Tri et al. (2014). "Fault Tolerant Predictive Control for Multi-Agent dynamical: formation reconfiguration using set-theoretic approach." In: (cit. on p. 2).

Nguyen, M.Q. et al. (2016a). "A Model Predictive Control approach for semi-active suspension control problem of a full car." In: *2016 IEEE 55th Conference on Decision and Control, CDC 2016*. Las Vegas, NV, USA (cit. on p. 43).

Nguyen, M.Q. et al. (2016b). "Semi-active suspension control problem: Some new results using an LPV/H∞ state feedback input constrained control." In: *Proceedings of the IEEE Conference on Decision and Control*. Osaka, Japan (cit. on p. 43).

Nocedal, Jorge and Stephen Wright (2006). *Numerical optimization*. Springer Science & Business Media (cit. on pp. 29–31).

O'donoghue, Brendan et al. (2016). "Conic optimization via operator splitting and homogeneous self-dual embedding." In: *Journal of Optimization Theory and Applications* 169.3, pp. 1042–1068 (cit. on p. 29).

Ohtsuka, Toshiyuki (2015). "A tutorial on C/GMRES and automatic code generation for nonlinear model predictive control." In: *2015 European Control Conference (ECC)*. IEEE, pp. 73–86 (cit. on p. 95).

Ohtsuka, Toshiyuki and Akira Kodama (2002). "Automatic code generation system for nonlinear receding horizon control." In: *Transactions of the Society of Instrument and Control Engineers* 38.7, pp. 617–623 (cit. on p. 95).

Ohyama, Shimpei and Hisashi Date (2017). "Parallelized nonlinear model predictive control on GPU." In: *2017 11th Asian Control Conference (ASCC)*. IEEE, pp. 1620–1625 (cit. on p. 67).

Parikh, Neal and Stephen Boyd (2014). "Proximal algorithms." In: *Foundations and Trends in optimization* 1.3, pp. 127–239 (cit. on p. 28).

Patrinos, Panagiotis and Alberto Bemporad (2012). "An accelerated dual gradient-projection algorithm for linear model predictive control." In: *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE, pp. 662–667 (cit. on p. 29).

Pham, Nam, Aleksander Malinowski, and Tomasz Bartczak (2011). "Comparative study of derivative free optimization algorithms." In: *IEEE Transactions on Industrial Informatics* 7.4, pp. 592–600 (cit. on p. 31).

Plantenga, Todd D (2009). "Hopspack 2.0 user manual." In: *Sandia National Laboratories Technical Report Sandia National Laboratories Technical Report SAND2009-6265* (cit. on p. 31).

Poussot-Vassal, Charles (2008). "Commande robuste LPV multivariable de chassis automobile." PhD thesis. Institut National Polytechnique de Grenoble-INPG (cit. on p. 12).

Poussot-Vassal, Charles et al. (2011). "Survey on some automotive semi-active suspension control methods: A comparative study on a single-corner model." In: *18th IFAC World Congress (IFAC WC 2011)*, pp. 1802–1807 (cit. on p. 43).

Powell, Michael JD (1994). "A direct search optimization method that models the objective and constraint functions by linear interpolation." In: *Advances in optimization and numerical analysis*. Springer, pp. 51–67 (cit. on p. 31).

— (2006). "The NEWUOA software for unconstrained optimization without derivatives." In: *Large-scale nonlinear optimization*. Springer, pp. 255–297 (cit. on p. 31).

— (2007). "A view of algorithms for optimization without derivatives." In: *Mathematics Today-Bulletin of the Institute of Mathematics and its Applications* 43.5, pp. 170–174 (cit. on p. 31).

— (2009). "The BOBYQA algorithm for bound constrained optimization without derivatives." In: *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, pp. 26–46 (cit. on p. 31).

Rabinow, Jacob (1948). "The magnetic fluid clutch." In: *Electrical Engineering* 67.12, pp. 1167–1167 (cit. on p. 13).

Rajamani, Rajesh (2011). *Vehicle dynamics and control*. Springer Science & Business Media (cit. on p. 3).

Raković, Saša V and William S Levine (2018). *Handbook of model predictive control*. Springer (cit. on p. 2).

Rathai, Karthik Murali Madhavan (2020). *pNMPC: A Code Generation Tool For Implementation of pNMPC Controller For Embedded Control Systems*. `https://github.com/Kartz4code/pNMPC_CODEGEN`. [Online; accessed 22-June-2020] (cit. on pp. 5, 94).

Rathai, Karthik Murali Madhavan, Mazen Alamir, and Olivier Sename (2019). "Experimental implementation of model predictive control scheme for control of semi-active suspension system." In: *IFAC-PapersOnLine* 52.5, pp. 261–266 (cit. on p. 5).

Rathai, Karthik Murali Madhavan, Olivier Sename, and Mazen Alamir (2018). "A comparative study of different NMPC schemes for control of semi-active suspension system." In: *14th International Conference on Vehicle System Dynamics, Identification and Anomalies (VSDIA 2018)* (cit. on pp. 33, 35, 36, 38).

— (2019). "GPU-Based Parameterized NMPC Scheme for Control of Half Car Vehicle With Semi-Active Suspension System." In: *IEEE Control Systems Letters* 3.3, pp. 631–636 (cit. on pp. 5, 82).

Rathai, Karthik Murali Madhavan et al. (2018). "A Parameterized NMPC scheme for embedded control of semi-active suspension system." In: *IFAC-PapersOnLine* 51.20, pp. 301–306 (cit. on pp. 5, 52, 72, 82).

Rawlings, James B, Edward S Meadows, and Kenneth R Muske (1994). "Nonlinear model predictive control: A tutorial and survey." In: *IFAC Proceedings Volumes* 27.2, pp. 185–197 (cit. on p. 2).

Rawlings, James Blake, David Q Mayne, and Moritz Diehl (2017). *Model predictive control: theory, computation, and design*. Vol. 2. Nob Hill Publishing Madison, WI (cit. on p. 34).

Rios, Luis Miguel and Nikolaos V Sahinidis (2013). "Derivative-free optimization: a review of algorithms and comparison of software implementations." In: *Journal of Global Optimization* 56.3, pp. 1247–1293 (cit. on p. 31).

Rogers, Jonathan (2013). "GPU-enabled projectile guidance for impact area constraints." In: *Modeling and Simulation for Defense Systems and Applications VIII*. Vol. 8752. International Society for Optics and Photonics, p. 87520I (cit. on p. 67).

Ruder, Sebastian (2016). "An overview of gradient descent optimization algorithms." In: *arXiv preprint arXiv:1609.04747* (cit. on pp. 28, 29).

Sampathirao, Ajay Kumar et al. (2017). "GPU-accelerated stochastic predictive control of drinking water networks." In: *IEEE Transactions on Control Systems Technology* 26.2, pp. 551–562 (cit. on p. 67).

Sanders, Jason and Edward Kandrot (2010). *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional (cit. on pp. 72, 82).

Savaresi, Sergio M., Sergio Bittanti, and Mauro Montiglio (2005). "Identification of semi-physical and black-box non-linear models: The case of MR-dampers for vehicles control." In: *Automatica* 41.1, pp. 113–127 (cit. on p. 42).

Savaresi, Sergio M. and Cristiano Spelta (2007). "Mixed Sky-Hook and ADD: Approaching the Filtering Limits of a Semi-Active Suspension." In: *Journal of Dynamic Systems, Measurement, and Control* 129.4, p. 382 (cit. on p. 43).

Savaresi, Sergio M et al. (2010). *Semi-active suspension control design for vehicles*. Elsevier (cit. on pp. 3, 10, 12–14, 16, 50, 67, 70, 80, 82, 120).

Schittkowski, Klaus (2006). "NLPQLP: A fortran implementation of a sequential quadratic programming algorithm with distributed and non-monotone line search–user's guide." In: *Report, Department of Computer Science, University of Bayreuth* (cit. on p. 30).

Sename, Olivier et al. (2012). "Some LPV Approaches for Semi-active Suspension Control." In: *American Control Conference 2012* (cit. on p. 43).

Shukla, Harsh A et al. (2017). "Software and hardware code generation for predictive control using splitting methods." In: *IFAC-PapersOnLine* 50.1, pp. 14386–14391 (cit. on p. 97).

Sopasakis, Pantelis, Emil Fresk, and Panagiotis Patrinos (2020). "OpEn: Code Generation for Embedded Nonconvex Optimization." In: *arXiv preprint arXiv:2003.00292* (cit. on p. 97).

Spelta, Cristiano (2008). "Design and applications of semi-active suspension control systems." PhD thesis. Phd thesis, Politecnico di Milano, dipartimento di Elettronica e Informazione (cit. on p. 13).

Spencer Jr, BF et al. (1997). "Phenomenological model for magnetorheological dampers." In: *Journal of engineering mechanics* 123.3, pp. 230–238 (cit. on p. 44).

Stanway, R, JL Sproston, and AK El-Wahed (1996). "Applications of electro-rheological fluids in vibration control: a survey." In: *Smart Materials and Structures* 5.4, p. 464 (cit. on p. 44).

Stanway, RSJL, JL Sproston, and NG Stevens (1987). "Non-linear modelling of an electro-rheological vibration damper." In: *Journal of Electrostatics* 20.2, pp. 167–184 (cit. on p. 44).

Stathopoulos, Georgios et al. (2016). "Operator splitting methods in control." In: *Foundations and Trends in Systems and Control* 3.ARTICLE, pp. 249–362 (cit. on p. 28).

Stellato, Bartolomeo et al. (2020). "OSQP: An operator splitting solver for quadratic programs." In: *Mathematical Programming Computation*, pp. 1–36 (cit. on p. 29).

Sun, Weichao, Huijun Gao, and Peng Shi (2020). *Advanced control for vehicle active suspension systems*. Springer (cit. on p. 12).

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press (cit. on p. 127).

Tangirala, Arun K, Siddhartha Mukhopadhyay, and Akhilanand P Tiwari (2013). "Wavelets applications in modeling and control." In: *Advances in chemical engineering*. Vol. 43. Elsevier, pp. 107–204 (cit. on p. 46).

Tora, Grzegorz (2012). "The active suspension of a cab in a heavy machine." In: *Advances on Analysis and Control of Vibrations: Theory and Applications*, p. 105 (cit. on p. 12).

Tseng, H. Eric and Davor Hrovat (2015). "State of the art survey: Active and semi-active suspension control." In: *Vehicle System Dynamics* 53, pp. 1034–1062 (cit. on p. 43).

Tyan, Feng et al. (2009). "Generation of random road profiles." In: *Journal of Advanced Engineering* 4.2, pp. 1373–1378 (cit. on p. 69).

Ullmann, Fabian (2011a). "FiOrdOs: A Matlab toolbox for C-code generation for first order methods." In: *MS thesis* (cit. on p. 29).

Ullmann, Fabian (2011b). "FiOrdOs: A Matlab toolbox for C-code generation for first order methods." In: *MS thesis* (cit. on p. 96).

Vandenberghe, Lieven (2010). "The CVXOPT linear and quadratic cone program solvers." In: *Online: http://cvxopt. org/documentation/coneprog. pdf* (cit. on p. 30).

Vanderbei, Robert J (1999). "LOQO: An interior point code for quadratic programming." In: *Optimization methods and software* 11.1-4, pp. 451–484 (cit. on p. 30).

Vaz, A Ismael F and Luís Nunes Vicente (2009). "PSwarm: a hybrid solver for linearly constrained global derivative-free optimization." In: *Optimization Methods & Software* 24.4-5, pp. 669–685 (cit. on p. 31).

Vidyasagar, Mathukumalli (2001). "Randomized algorithms for robust controller synthesis using statistical learning theory." In: *Automatica* 37.10, pp. 1515–1528 (cit. on pp. 84, 85).

Vivas-Lopez, Carlos et al. (2014a). "INOVE: a testbench for the analysis and control of automotive vertical dynamics." In: *14th International Conference on Vehicle System Dynamics, Identification and Anomalies (VSDIA 2014)* (cit. on pp. 3, 16, 17).

Vivas-Lopez, Carlos A et al. (2014b). "Modeling Method for Electro-Rheological Dampers." In: *IFAC Proceedings Volumes (IFAC-PapersOnline)* (cit. on p. 13).

Wächter, Andreas and Lorenz T Biegler (2006a). "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming." In: *Mathematical programming* 106.1, pp. 25–57 (cit. on p. 30).

— (2006b). "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming." In: *Mathematical programming* 106.1, pp. 25–57 (cit. on p. 96).

Williams, Grady et al. (2016). "Aggressive driving with model predictive path integral control." In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1433–1440 (cit. on p. 67).

Winslow, Willis M (1947). *Method and means for translating electrical impulses into mechanical force*. US Patent 2,417,850 (cit. on p. 13).

Zanelli, Andrea et al. (2020). "FORCES NLP: an efficient implementation of interior-point methods for multistage nonlinear nonconvex programs." In: *International Journal of Control* 93.1, pp. 13–29 (cit. on pp. 29, 96).

Zhang, Lixian et al. (2017). "Cloud-Aided State Estimation of A Full-Car Semi-Active Suspension System." In: *arXiv preprint arXiv:1701.03343* (cit. on p. 81).

Zometa, Pablo, Markus Kögel, and Rolf Findeisen (2013). "μAO-MPC: a free code generation tool for embedded real-time linear model predictive control." In: *2013 American Control Conference*. IEEE, pp. 5320–5325 (cit. on p. 96).