SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF CHEMICAL AND FOOD TECHNOLOGY

Reference number: FCHPT-10881-77258

EMBEDDED IMPLEMENTATION OF EXPLICIT MODEL PREDICTIVE CONTROL

DISSERTATION THESIS

Bratislava, 2017 **Deepak Ingole**

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF CHEMICAL AND FOOD TECHNOLOGY

Reference number: FCHPT-10881-77258

EMBEDDED IMPLEMENTATION OF EXPLICIT MODEL PREDICTIVE CONTROL

DISSERTATION THESIS

Study programme:	Process Control
Study field number:	2621
Study field:	5.2.14 Automation
Workplace:	Department of Information Engineering and Process Control
Thesis supervisor:	doc. Ing. Michal Kvasnica, PhD.

Bratislava, 2017 **Deepak Ingole**

Slovak University of Technology in Bratislava
Faculty of Chemical and Food Technology
Institute of Information Engineering, Automation and Mathematics

DISSERTATION THESIS TOPIC

Author of the thesis:	Deepak Ingole
Study programme:	Process Control
Study field:	5.2.14. automation
Registration number:	FCHPT-10881-77258
Student's ID:	77258
Thesis supervisor:	doc. Ing. Michal Kvasnica, PhD.
Title of the thesis:	Embedded Implementation of Explicit Model Predictive Control
Date of entry:	03.09.2014
Date of submission:	05.09.2017

Deepak Ingole
Solver

prof. Ing. Miroslav Fikar, DrSc.
Head of Department

prof. Ing. Miroslav Fikar, DrSc.
Study Programme Supervisor

To my dear wife Shrutika

Acknowledgments

I am deeply indebted to my supervisor doc. Ing. Michal Kvasnica, PhD. for his support, motivation, and inspiring ideas. I thank him for giving me total freedom to choose my research direction that allowed me to collaborate with several people and for allowing me to travel around the Europe several times. His energy and enthusiasm, particularly during the scientific seminars have been a constant source of motivation for me. My sincere gratitude goes to prof. Ing. Miroslav Fikar, DrSc. for his support and motivation. He introduced me to the theory of optimal control which was a nice learning experience.

Several people outside of the Slovak University of Technology have had a substantial impact on this thesis. I would like to thank Dr. Paul Goulart for hosting me at the Control Group at the University of Oxford during winter of 2017. He has willingly shared his knowledge and software tools with me. The discussions we had during my visits were of great importance for my understanding about the software development and my subsequent work. I am grateful to Dr. Eric Kerrigan for hosting me Imperial College London during the wonderful summer of 2017. Throughout my stay, he has taken an active interest in my work, and our discussions have made this collaboration particularly fruitful. He has taught me that results can always be presented better. I would like to take an opportunity to thank Prof. John Gustafson from the National University of Singapore for introducing the idea of universal numbers which have significant impact on this thesis. I would like to thank Dr. Simon Byrne from University College, London and Himeshi De Silva from the National University of Singapore. They always made time to answer my questions. It was a great support during the development of mnum toolbox. I am also thankful to Bulat Khusainov from Imperial College London for always having time to answer my numerous questions about PROTOIP and FPGA implementation. My thanks go to Goran Banjac and Bartolomeo Stellato from the University of Oxford for their support during my stay in Oxford. My thanks go to Dr. Dayaram Sonawane and Dr. Divyesh Ginoya from College of Engineering Pune and Vihangkumar Naik from IMT Institute of Advance Studies Lucca for their support and encouragement.

The research was made possible by the financial support by the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement number 607957 (Training in Embedded Predictive control and Optimization (TEMPO) - A Marie Curie Initial Training Network (ITN)). I consider myself very fortunate to have been the part of TEMPO network.

Within STU I would especially like to thank Prof. Monika Bakošová, Prof. Ján Mikleš and Prof. Alajos Mészáros for guidance during my coursework. I would like to thank my friends and colleagues Martin Klaučo, Ján Drgoňa, Martin Kalúz, Juraj Holaza, Juraj Oravec, Luboš Čirka, Radoslav Paulen, Filip Janeček, Anna Vasičkaninová, and Stanislav Vagač for their support and fruitful discussions. My special thanks go to Ayush Sharma and Richard Valo for their lovely friendship, always ready to lend a helping hand, boosting my mood, and arranging nice trips to explore Slovakia. I also thank former colleagues Martin Jelemenský, Bálint Takács, Juraj Števek, and Daniela Pakšiová for their help. Also, thanks to secretaries, Katarína Macušková and Andrea Kalmárová, who helped me with the administration, forms and financial matters.

My thanks must be to my family. I am grateful to my in-law's family, who has always encouraged and supported me and my decisions. Last but not the least, I am grateful for the endearing and patience support from my wife Shrutika, for her love and kindness.

Deepak Ingole
Bratislava, 2017

Abstract

The Model Predictive Control (MPC) feedback law is given by the solution to a multi-parametric Quadratic Programming (mp-QP) problem that can be pre-computed off-line and stored in the form of Look-Up Table (LUT) to be used in on-line synthesis. The on-line computation reduces to simple evaluations of a Piecewise Affine (PWA) function, allowing implementations on simple hardware and with fast sampling rates. One of the main bottlenecks in the embedded implementation of explicit MPC is the memory required to store optimal solutions; this often limits its applicability to systems with a few states and controls, simple constraints, and short prediction horizons.

Therefore, the focus of this thesis lies on the embedded implementation of low-memory explicit MPC feedback laws for the real-time control of constrained linear systems. In detail, a novel memory reduction technique for low-memory explicit MPC laws is proposed. The technique is based on encoding all data (i.e., the critical regions and the feedback laws) as universal numbers (unums), which can be viewed as a more memory efficient extension of IEEE floating-point standard for representing real numbers. Specifically, we show that the controller data (in the form of a floating-point standard) to be stored on the hardware memory can be replaced by the unum format which takes fewer bits to store same value and get more accurate answers than floating-point arithmetic. As unum needs fewer bits to represent a number, it saves memory and bandwidth. Unlike floating-point numbers, unums make no rounding errors, and cannot overflow or underflow.

To show the feasibility of unums in explicit MPC, a MATLAB toolbox called `munum` is developed which can be used to export unum-based explicit MPC algorithm in low-level C language code. For the implementation of explicit MPC in C application, we developed a unum toolbox called `cunum`. The hardware specific routines of unum arithmetic are developed in C and implemented on Field Programmable Gate Array (FPGA). The closed-loop simulation results of software and real-time FPGA implementation are presented with anesthesia control problem. The memory comparisons (floating-point and unum) indicates that the total memory footprint can be reduced by 80% without sacrificing the control performance. Another advantage of the proposed approach is; that it can be applied on top of other existing complexity reduction techniques.

Abstrakt

Spätnoväzbový prediktívny regulátor (MPC) je daný riešením parametrického kvadratického programovania, ktorý môže byť dopredu vypočítaný a uložený vo forme vyhľadávacej tabuľky. Výpočtové nároky získania optimálnych akčných zásahov, prostredníctvom zostrojenej vyhľadávacej tabuľky, sú následne znížené na jednoduché vyhodnocovanie po častiach affinej funkcie. Dôsledkom takéhoto zníženia výpočtových nárokov je možné implementovať MPC aj na jednoduchý riadiaci hardvér, pri súbežnom dosahovaní rýchlej vzorkovacej frekvencie. Avšak jednou z hlavných prekážok implementácie MPC, v jeho zmienenej explicitnej podobe, je pamäťová náročnosť predpočítaného riešenia, ktoré je potrebné uložiť do riadiaceho hardvéru. Z tohoto dôvodu je použiteľnosť explicitného MPC obmedzená na nízko-rozmerové systémy, jednoduché ohraničenia a krátke predikčné horizonty.

Zameranie tejto práce preto spočíva v integrovanej implementácii pamäťovo nízko náročných explicitných MPC regulátorov pre riadenie obmedzených lineárnych systémov v reálnom čase. Je navrhnutá nová metóda na redukciu potrebnej pamäte pre explicitné MPC regulátori. Technika je založená na kódovaní všetkých údajov (t.j. kritických oblastí a zákonov o spätnej väzbe) pomocou univerzálnych čísiel (unums), ktoré možno považovať za pamäťovo efektívnejšie rozšírenie normy IEEE s pohyblivou desatinou čiarkou pre reprezentáciu reálnych čísel. Konkrétne ukážeme, že dáta regulátora (vo forme štandardnej veličiny s pohyblivou desatinou čiarkou), ktoré sa majú uložiť do hardvérovej pamäte, môžu byť nahradené formátom unum, ktorý potrebuje menej bitov na ukladanie rovnakej hodnoty a získanie presnejších odpovedí ako aritmetika s pohyblivou desatinou čiarkou. Keďže unum potrebuje menej bitov na reprezentáciu čísla, šetrí pamäť a šírku pásma. Na rozdiel od čísel s pohyblivou desatinou čiarou, kódovanie unum neumožňuje žiadne chyby pri zaokrúhľovaní a nemôže pretekať alebo spadnúť.

Pre aplikáciu aritmetiky unums, v kontexte explicitného prediktívneho riadenia, bol v prostredí MATLABzostrojený balík s názvom `munum`, ktorý slúži na export explicitného MPC regulátora v tvare nízko-úrovňového jazyka C. Na vyhodnocovanie aritmetiky unums v jazyku C bola vytvorená knižnica `cunum`. Hardvérové špecifické rutiny unum aritmetiky sú vyvíjané v C a implementované na programovateľnom hradlovom poli (FPGA). Výsledky simulácie uzavretého regulačného obvodu, softvéru a implementácie FPGA v reálnom čase, sú prezentované prostredníctvom riadenia systému anestézie. Porovnávanie pamäte (s pohyblivým a nekonečným bodom) ukázalo, že celkové použitie pamäte môže byť znížené o 80% bez toho, aby sme ovplyvnili výkonnosť riadenia. Ďalšou výhodou navrhovaného prístupu je, že môže byť použitý nad ostatnými existujúcimi technikami znižovania implementačnej zložitosti prediktívnych regulátorov MPC.

Contents

List of Figures

List of Tables

List of Acronyms

ALU	Arithmetic Logic Unit
BIS	Bispectral Index
BRAM	Block RAM
BW	Body Weight
CFTOC	Constrained Finite-Time Optimal Control
CLB	Configurable Logic Block
cunum	C/C++ -based Unum Toolbox
DoA	Depth of Anesthesia
DSP	Digital Signal Processor
EEG	Electroencephalography
EMPC	Explicit MPC
FF	Flip-Flop
FP	Floating-Point
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
HIL	Hardware-In-the-Loop
HR	Heart Rate

IEEE	Institute of Electrical and Electronics Engineers
ISE	Integrated Square Error
KKT	Karush–Kuhn–Tucker
LP	Linear Programming
LTI	Linear Time-Invariant
LUT	Look-Up Table
MAP	Mean Atrial Pressure
MIMO	Multi-Input Multi-Output
mp-QP	multi-paramtric Quadratic Programming
MPC	Model Predictive Control
MPT	Multi-Paramteric Toolbox
munum	MATLAB-based Unum Toolbox
OSQP	Operator Splitting Quadratic Programming
PD	Pharmacodynamic
PID	Proportional–Integral–Derivative
PK	Pharmacokinetic
PLA	Point Location Algorithm
PLC	Programmable Logic Controller
POP	Paramteric Optimization
PWA	Piecewise Affine
QP	Quadratic Programming
RHC	Receding Horizon Control
SIL	Software-In-the-Loop

SISO	Single-Input Single-Output
SoC	System on-Chip
unum	Universal Number

Chapter 1

Introduction

"Prediction is very difficult, especially if it's about the future."
Niels Bohr (1885 - 1962)

The aim of this thesis is to develop and implement low-memory explicit Model Predictive Controller (MPC) using Universal Numbers (unums). The thesis is accompanied with open-source toolboxes for unum-based arithmetic and explicit MPC. The toolboxes are MATLAB-based unum toolbox called `munum`, C/C++ -based unum toolbox called `cunum` and the FPGA specific unum implementation. The main focus of this thesis is to reduce memory complexity of explicit MPC problems which are excluded due to the high complexity of resulting control laws.

1.1 Motivation

Model predictive control is a modern control technology that enjoys great success in process industries and in academia, due to its ability to control Multi-Input Multi-Output (MIMO) systems with constraints. However, limitations on computational efficiency (to solve an on-line optimization problem at each time step) has restricted the application range. This has lead to a substantial efforts to develop hardware specific optimization solvers that have more attractive on-line computational properties than nominal solvers typically implemented on computers, laptops, and servers.

Generally, real-time control applications are controlled by well-know Proportional–Integral–Derivative (PID) controller implemented on embedded hardware like Programmable Logic Controller (PLC), Field Programmable Gate Array (FPGA), Digital Signal Processor (DSP), and microcontroller. Embedded hardware could also be preferred because of power supply limitations, limited space, reliability, crucial real-time, and safety requirements. MPC has capabilities beyond PID controller to increase profit and reduce overall operating cost. However, the computational efficiency becomes even more significant when dealing with embedded platforms that have limited resources (memory, speed, and power) and one have no flexibility to replace or extend the available hardware. The first motivation is therefore to implement explicit MPC algorithm on hardware. This would create base for further improvements in implementation.

In explicit MPC, the optimal control law is pre-computed off-line as a function of all possible initial states and stored in the form of Look-Up Table (LUT) for on-line evaluation to obtain optimal control action. There are two main limitations of explicit MPC. First, the storage requirements of pre-computed off-line data grows exponentially with respect to the number of constraints in MPC problems. Second, as the controller complexity grows, the worst case computation time may rise above a practical value, thereby eliminating it as a viable choice in a real-time system.

To tackle with these limitations, an effort has been made to reduce the complexity of explicit MPC which is mainly focused on two distinct directions: first, how to make the feedback law simple; second, how to reduce the amount of bits required to store LUT data with the prescribed accuracy; this direction is less addressed in literature. To make simpler feedback laws an approximation techniques have been proposed by many researchers. However, those techniques leads to a simpler approximate suboptimal solution which is unacceptable in some practical applications. Moreover, the common thing among all the existing techniques is that the controller data is stored in the form IEEE-754 floating-point numbers (single and double precision). The bit size of numbers is thus constant regardless of the values they store. Here comes the second motivation that to reduce number of bits required to store each number in the LUT while maintaining the original properties of the controller. This allows one to use explicit MPC for a number of systems that would otherwise be excluded due to the high complexity of the resulting explicit controllers due to the lack of available memory or powerful computing devices to make point location algorithms faster.

The final motivation is that bit reduction technique can be used on the top of existing optimization solvers that are used for general purposes and embedded MPC. First order optimization methods would get benefit of high accuracy and hybrid explicit MPC would reduce memory.

1.2 Goals and Contributions of the Thesis

This section describes the goals of thesis and our contributions towards fulfilling set goals as follows:

1.2.1 Reduction of Memory Footprints of Explicit MPC Controllers

In explicit MPC one needs to store all the floating-point numbers associated with the critical regions and control laws in the form of LUTs. The size of LUTs decide the amount of memory required to store given controller. Embedded hardware often comes with kilobyte (kB) to megabyte (MB) of on-chip memory which is insufficient to store large LUTs and due to this issue the applicability of explicit MPC has been restricted to the small systems. One of the main goal of this thesis is to develop low-memory explicit MPC controller which will reduce the amount of memory required on the target hardware.

One of the main contribution of this thesis is to present a novel way of memory reduction in explicit MPC. The procedure is based on encoding all data (i.e., the critical regions and the feedback laws) as *Universal Numbers* (Unums) (see, Chapter 5), which can be viewed as a memory-efficient extension of IEEE floating-point standard (see, Chapter 4). Since there is a one-to-one correspondence between floating-point numbers and their unum representation, the unum-based control law exhibits the same properties (e.g., control performance, closed-loop stability, and constraint satisfaction) as the floating-point-based controller. Main results of the proposed approach are presented in the Chapter 7 and reported in the following article:

- **Ingole D.**, Kvasnica M., De Silva H., Gustafson J., "Reducing Memory Footprints in Explicit Model Predictive Control using Universal Numbers", *In Preprints of the 20th IFAC World Congress*, IFAC, Toulouse, France, vol. 20, pp. 12100-12105, 2017.

1.2.2 Development and Implementation of Open-Source Toolboxes for Low-Memory Controller

There are several state-of-the-art software tools available for the constriction and export of explicit MPC control laws. Existing software tools for explicit MPC construction are Multi-Parametric Toolbox (MPT), Parametric Optimization (POP) toolbox and hybrid toolbox. All these toolboxes export critical regions and control laws in the form of LUTs comprised of IEEE floating-point numbers which needs more memory. Therefore, the goal of this thesis is to develop software tools for the export of unum-based explicit MPC and implement exported controller on embedded hardware. Specifically, our goal is to develop open-source MATLAB and C/C++ toolboxes for unum arithmetic, explicit MPC, and its implementation on Xilinxs FPGA.

Next contribution of this thesis is the development of `munum` (MATLAB toolbox) and `cunum` (C/C++ toolbox) for universal number format and its arithmetic for general purpose use which can be used to develop any algorithm incorporating unum idea. The `munum` toolbox provides a functionality to export unum-based explicit MPC algorithm in low-level C language which can be deployed on PLC, FPGA, and microcontroller with the help of developed `cunum` toolbox for unum arithmetic. To this end, these toolboxes are applied and tested on linear MPC and hybrid MPC using explicit solutions. The functionality of `munum` and `cunum` and FPGA implementation of unum-based explicit MPC presented in Chapter 6.

1.2.3 Testing of Developed Approaches on Anesthesia Control Problem

The final goal of this thesis is to test and validate developed low-memory unum-based explicit MPC approach on anesthesia control problem where the objective is to test closed-loop performance, run-time, and memory complexity. Developed unum toolboxes in MATLAB and C/C++ will be tested via Software-In-the-Loop (SIL) approach where controller and plant/model will be in MATLAB or C/C++ application. The implemented unum-based explicit MPC on FPGA will be tested via Hardware-In-the-Loop (HIL) co-simulation approach where controller will be on FPGA and plant/model will be in MATLAB.

The results of unum-based explicit MPC (implemented using developed toolboxes (`munum, cunum`)) for anesthesia and double integrator control problem are reported in the following article:

- **Ingole D.**, Kvasnica M., De Silva H., Gustafson J., "Low-Memory Explicit Model Predictive Controller using Universal Numbers", *Draft is ready for submission to IEEE Transactions on Control Systems Technology*, IEEE, 2017.

Unum arithmetic and explicit MPC algorithm is implemented on FPGA and tested on anesthesia control problem (see, Chapter 7). The proposed explicit MPC algorithm stores LUT data in less bits and gives better accuracy and precision as compared to that of floating-point format-based algorithm. The results of this work will be appeared in the following article:

- **Ingole D.**, Kvasnica M., Kerrigan E, Khusainov B., De Silva H., Gustafson J., "FPGA Implementation of Memory Efficient Explicit Model Predictive Controller using Universal Numbers", *Draft is under preparation for the submission to IEEE Transactions on Control Systems Technology*, IEEE, 2017.

Apart from the explicit MPC we employed developed C/C++ toolbox in the implementation of hybrid MPC for the anesthesia and hybrid vehicle control problem, the results of which will be appear in the following article:

- Naik V., **Ingole D.**, Kvasnica M., Bemporad A., De Silva H., Gustafson J., "Embedded Mixed-Integer Quadratic Programming using Universal Numbers", *Draft is under preparation for the submission to IEEE Transactions on Control Systems Technology*, IEEE, 2017.

Other contributions are the embedded implementation of unum-based general purpose Operator Splitting Quadratic Programming (OSQP) solver using C/C++ language and floating-point number-based explicit MPC for anesthesia control problem which is published in the following articles:

- **Ingole D.** and Kvasnica M., "FPGA Implementation of Explicit Model Predictive Control for Closed Loop Control of Depth of Anesthesia", *In Preprints of the 5th Conference on Nonlinear Model Predictive Control*, IFAC, Seville, Spain, pp. 484–489, 2015.

- **Ingole D.**, Holaza J., Takács B., and Kvasnica M., "FPGA-Based Explicit Model Predictive Control for Closed-Loop Control of Intravenous Anesthesia", *In Proceedings of the 20th International Conference on Process Control*, IEEE, Štrbské Pleso, Slovakia, pp. 42-47, 2015.

The author has also participated in research covering other areas of control system, however, the results of that work are not included in this thesis. Specifically, the implementation of model predictive control schemes has been explored and the results of that are published/submitted in:

- **Ingole D.**, Drgoňa J., Kalúz, M., Klaučo, M., Bakošová, M., Kvasnica M., "Model Predictive Control of a Combined Electrolyzer-Fuel Cell Educational Pilot Plant", *In Proceedings of the 21th International Conference on Process Control*, IEEE, Štrbské Pleso, Slovakia, pp. 142-154, 2017.

- **Ingole D.**, Drgoňa J., and Kvasnica M., "Offset-Free Hybrid Model Predictive Control of Bispectral Index in Anesthesia", *In Proceedings of the 21th International Conference on Process Control*, IEEE, Štrbské Pleso, Slovakia, pp. 422-427, 2017.

- Dani S., Sonawane D., **Ingole D.**, and Patil S., "Performance Evaluation of PID, LQR and MPC for DC Motor Speed Control", *In Proceedings of International Conference for Convergence in Technology (I2CT)*, IEEE, Pune, India, pp. 1-7, 2017.

- **Ingole D.**, Drgoňa J., Kalúz, M., Klaučo, M., Bakošová, M., Kvasnica M., "Explicit Model Predictive Control of a Fuel Cell", *In The European Conference on Computational Optimization*, Leuven, Belgium, vol. 4, 2016.

- Sonawane D., **Ingole D.**, and Naik V., "FPGA implementation of linear model predictive controller for real-time position control of DC motor", *International Journal of Circuits and Architecture Design*, Inderscience, vol. 1, issue 4, pp. 281-294, 2015.

Full publication list of the author can be found in the Appendix A, which also includes publications related to this thesis as well as other research work.

1.3 Collaborations

During the journey of this thesis, we collaborated with following universities.

1. National University of Singapore: With the department of Computer Science we have been working on developing unum arithmetic for control algorithms.

2. University of Oxford, United Kingdom: In collaboration with Control group we implemented universal number format-based Operator Splitting Quadratic Programming (OSQP) solver in C/C++ language. Also, the C++ wrapper were developed to used unums on the top of existing algorithms. This work is described in detailed in Section 6.2.

3. Imperial College London, United Kingdom: With the Control and Power group we worked on FPGA implementation of unum-based explicit MPC algorithm. The work of FPGA implementation is presented in Section 6.3.

4. IMT School for Advanced Studies Lucca, Italy: With Dynamical Systems, Control, and Optimization research unit we worked on C implementation of universal numbers for hybrid explicit MPC.

1.4 Organization of the Thesis

The thesis consists of eight chapters including this introductory chapter. Introduction to the model predictive control concept and its different formulations used in control application and short discussion about the on-line optimization methods are given in Chapter 2. The next Chapter 3 discusses concept of explicit MPC, point location algorithms, overview of the complexity reduction techniques, and problem statement. The idea of universal numbers lies on the floating-point standard. So, understanding the floating-point format is important to understand unum idea. In Chapter 4, the introduction of number systems, IEEE floating-point standard and its arithmetic, and interval arithmetic is given. The idea of universal numbers and its arithmetic is discussed in Chapter 5. Chapter 6 is devoted to the software and FPGA implementation of unum toolboxes developed in Matlab, C/C++ , and FPGA platforms. Applicability of unums is demonstrated with anesthesia control problem in Chapter 7. Conclusions are drawn in Chapter 8 with some notes on future research directions.

Chapter 2

Model Predictive Control

2.1 Introduction

Model Predictive Control (MPC) is a well-known family of control algorithms which has made a significant impact on process industries where it proved to be highly successful in comparison with alternative methods of multi-variable control for its capability to take into account the operating constraints on input and output variables. Its remarkable success in the process industries is mainly due to its ability to handle Multi-Input Multi-Output (MIMO) systems with slow dynamics and constraints on process variables. The MPC research literature is large, but review papers have appeared at regular intervals. Introduction to the concept of MPC can be found in Muske and Rawlings (1993), Rawlings (2000). The properties of MPC are described in Mayne et al. (2000), de Oliveira and Biegler (1994). Theoretical and practical issues associated with MPC technology are summarized in Mayne et al. (2000), Maasoumy et al. (2014), Zong et al. (2017), Forbes et al. (2015). The great success of MPC in the process industries is well described in Qin and Badgwell (2003), Hrovat et al. (2012). Several papers describe the variant of MPC algorithms such as linear MPC (Muske and Rawlings, 1993), non-Linear MPC (Findeisen and Allgöwer, 2002), explicit (Bemporad et al., 2000), (Bemporad et al., 2002) MPC, and hybrid MPC (Borrelli et al., 2015), (Lazar, 2006).

MPC is commonly applied to large systems with slow dynamics, but recently with the increase of computational power and the development of new algorithms that is more efficient, systems with faster dynamics are being targeted to be controlled by predictive methods. In the last few years, research has been directed to develop fast MPC algorithms intended for embedded implementation see, e.g., Wang and Boyd (2010), Johansen (2014), Jones and Kerrigan (2015).

A more recent overview of MPC theory development can be found in Mayne (2014). Moreover, several excellent books (Maciejowski, 2002), (Rossiter, 2003), (Kwon and Han, 2006), (Wang, 2009), (UNE and Pannek, 2011), (Camacho and Alba, 2013) and review papers (Bemporad, 2006), (Lee, 2011), (Christofides et al., 2013), (Yu-Geng et al., 2013) have appeared recently.

Model predictive control is a control strategy that offers attractive solutions for the control of constrained linear or non-linear systems and, more recently, also for the control of hybrid systems. MPC is an optimal control method, where the control action is obtained by solving a Constrained Finite-Time Optimal Control (CFTOC) problem for the current state of the plant at each sampling time. The sequence of optimal control inputs is computed for a predicted evolution of the system model over a finite-time horizon. However, only the first element of the control sequence is applied, and the state of the system is then measured again at the next sampling time. This so-called Receding Horizon Controller (RHC) introduces feedback to the system, thereby allowing for compensation of potential modeling errors or disturbances acting on the system (Borrelli et al., 2015, Chapter 13). While the basic idea of MPC is well-established, there exist many variants for guaranteeing closed-loop feasibility, stability, robustness or reference tracking.

Despite the early implementation of MPC, its computational complexity has restricted it to process industry, where slow dynamics is dominant. The successive improvements in electronic systems which led to higher computation capabilities opened the door for application of MPC to systems with faster dynamics to the degree that it could be used for embedded systems that require advanced control strategies (Johansen, 2014).

The success of MPC technology as a process control strategy can be attributed to three significant factors (Qin and Badgwell, 2000).

- The incorporation of an explicit process model into the control optimization. This allows the controller, in principle, to deal directly with all significant features of the process dynamics.

- The MPC algorithm considers plant behavior over a future horizon in time. Effects of measured and unmeasured disturbances can be predicted and eliminated.

- MPC considers process input, state, and output constraints directly in the optimization problem. It means, that constraint violations are far less likely, resulting in tighter control at the optimal constrained steady-state for the process. It is the inclusion of constraints that most clearly distinguishes MPC from other process control strategies.

2.1.1 Concept of MPC

Nevertheless, all MPC algorithms have in common the same control structure. The current control signal is obtained by solving a finite horizon open-loop optimal control problem in receding horizon fashion as stated above. Fig. 2.1 shows the basic principle of MPC

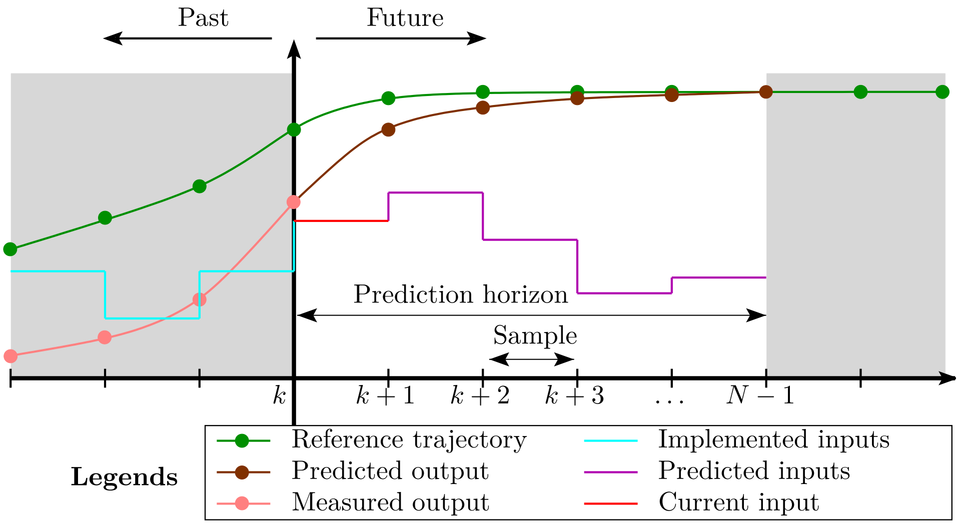

Figure 2.1: Characteristic behavior of a receding horizon control policy.

Step 1: The predicted future outputs by $\hat{y}_k$; $k = 1, \dots, N$, for the prediction horizon are calculated at each instant k using the process model. These depend upon the known values up to instance k (past inputs and outputs), including the current output (initial condition) $y(t)$ and on the future control signals u_k; $k = 0, \dots, N-1$, to be calculated.

Step 2: The sequence of future control signals is computed to optimize a performance criterion, often to minimize the error between a reference trajectory r_k and the predicted process output. Usually, the control effort is included in the performance criterion.

Step 3: Only the current control signal u_k is transmitted to the process. At the next sampling instant $k := k + 1$, y_{k+1} is measured and step 1 is repeated and all sequences brought up to date. Thus u_{k+1} is then calculated using the receding horizon concept, since the prediction horizon remains of the length as before, but slides along by one sampling interval at each step.

The resulting controller is referred to a receding horizon controller. A receding horizon controller where the finite-time optimal control law is computed by solving an optimization problem on-line is usually referred to as MPC.

2.1.2 Components of MPC

The structure of MPC is illustrated in Fig. 2.2 and below its main components are described.

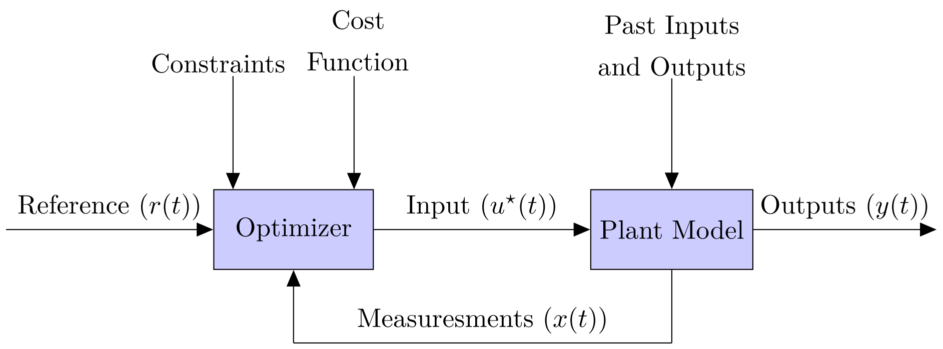

Figure 2.2: Basic structure of model predictive control.

- System model and predictions

  The modeling stage in MPC design is one of the most important activities. The plant model can be used to predict the future trajectories of the plant outputs for a given sequence of future control signals. The simplest model that gives accurate enough predictions is usually best. Accurate enough is ill-defined, but practice predictions can often be $10 - 20\%$ out is steady-state and still be highly effective as long as they also capture major dynamic changes in transit. Also, small modeling errors can be corrected by feedback control.

- Control problem and MPC formulation

  This is an important part which usually requires practical experiences. Sometimes, it is difficult to even identify what should be controlled and optimized. We have to know all the basic properties and limitations of MPC at this stage.

- Optimization problem

  The last step is a translation of the MPC control problem to a numerical optimization problem. It determines the future control signal such that the objective function is minimized. The optimization may also account for constraints on the process inputs and the process outputs.

2.2 MPC Formulations

When formulating the optimization problem in MPC, it is important to ensure that it can be solved in the one sampling instant. For that reason, the optimization problem is typically formulated into one of two standard forms:

1. Linear programming: In this formulation, both the objective function and the constraints are linear. One can formulate the LP in a way that minimizes the maximal deviation from the desired trajectory; this formulation is usually called a robust MPC.

2. Quadratic programming: In QP formulation, the objective function is quadratic, whereas the constraints have to be linear.

We will focus on the QP formulation since the quadratic penalization can be more expressive than the linear one. A linear formulation penalizes large errors more and additionally provides a deadband in the controller. In the following, we will describe how to formulate MPC problem as a QP optimization problem considering different objective functions.

2.2.1 State Regulation MPC

We start by designing a controller to take the state of a deterministic, linear system to the origin. If the reference is not the origin, or we wish to track a time-varying reference trajectory, we will subsequently make modifications of the zero reference problem to account for that. Considering the system state dynamics, most basic control problem formulation using squared Euclidean norm ($\|\cdot\|_2$) can be written as follows

$$\begin{aligned}
\min_{U} \quad & x_N^T P x_N + \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k && \text{(2.1a)}\\
\text{s.t.} \quad & x_{k+1} = A x_k + B u_k, \quad k = 0, \dots, N-1, && \text{(2.1b)}\\
& x_k \in \mathcal{X}, \quad k = 0, \dots, N-1, && \text{(2.1c)}\\
& u_k \in \mathcal{U}, \quad k = 0, \dots, N-1, && \text{(2.1d)}\\
& x_N \in \mathcal{X}_f, && \text{(2.1e)}\\
& x_0 = x(t), && \text{(2.1f)}
\end{aligned}$$

where P, Q and R are the weighting matrices with conditions $Q = Q^T \succeq 0$ and $P = P^T \succeq 0$ to be positive semi-definite and $R = R^T \succ 0$ to be positive definite, N is the prediction horizon, x_{k+1} is the vector of predicted states based on prediction model (2.1b), $U = \{u_0, \dots, u_{N-1}\} \in \mathbb{R}^{n_u N \times n_u N}$ is the sequence of control actions, and $\mathcal{X}$,$\mathcal{U}$, and $\mathcal{X}_f$ are the polyhedral constraint sets.

2.2.2 Reference Tracking MPC

Formulation of MPC control problem in (2.1) is one of the standard formulation which handles an only regulation problem. Using such controller, we can regulate the system toward its origin, i.e., to a zero state. When tracking of non-zero references need to be achieved (which is needed in a vast majority of a control application in industry), a modified cost function must be considered. For the reference tracking MPC problem it is necessary to extend system model (2.2b) to full state-space model by adding output dynamics (2.2c). The control problem formulation for output tracking MPC is of as follows

$$
\begin{aligned}
\min_{U} \quad & \sum_{k=0}^{N-1} (y_k - r_k)^T Q (y_k - r_k) + \sum_{k=0}^{N-1} u_k^T R u_k && (2.2a)\\
\text{s.t.} \quad & x_{k+1} = Ax_k + Bu_k, \quad k = 0, \ldots, N-1, && (2.2b)\\
& y_k = Cx_k + Du_k, \quad k = 0, \ldots, N-1, && (2.2c)\\
& x_k \in \mathcal{X}, \quad k = 0, \ldots, N-1, && (2.2d)\\
& y_k \in \mathcal{Y}, \quad k = 0, \ldots, N-1, && (2.2e)\\
& u_k \in \mathcal{U}, \quad k = 0, \ldots, N-1, && (2.2f)\\
& x_0 = x(t), && (2.2g)
\end{aligned}
$$

where r_k is the time-varying reference and $\mathcal{Y}$ is the output constraint set.

2.2.3 Integral Action in Reference Tracking MPC

The majority of industrial control systems have integral action (e.g., PID controllers). This integral functionality has also been embedded in the predictive control systems in conjunction with system model to achieve offset-free tracking. The integral action uses the deviation of current and past measured output values relative to the desired state, to bias the discrete-time model predictions until they converge upon their corresponding measured values. This method has been demonstrated to be effective in providing offset-free tracking in the presence of plant model mismatch (Maeder et al., 2009). In the following control problem formulation, we are penalizing tracking error $y_k - r_k$, which we require to converge to zero, along with increments of the control action, denoted by Δu. This Δu is defined as a difference between current value of control action and previous one, formally, $\Delta u_k = u_k - u_{k-1}$. When the reference is reached, thus the system is in steady state, also the control action is not changing, thus $\Delta u_k = 0$. As a consequence, the objective function is equal to zero. The mathematical formulation is as follows

$$
\begin{aligned}
\min_{U} \quad & \sum_{k=0}^{N-1} (y_k - r_k)^T Q (y_k - r_k) + \sum_{k=0}^{N-1} \Delta u_k^T R \Delta u_k && \text{(2.3a)}\\
\text{s.t.} \quad & x_{k+1} = Ax_k + Bu_k, \quad k = 0, \dots, N-1, && \text{(2.3b)}\\
& y_k = Cx_k + Du_k, \quad k = 0, \dots, N-1, && \text{(2.3c)}\\
& \Delta u_k = u_k - u_{k-1}, \quad k = 0, \dots, N-1, && \text{(2.3d)}\\
& x_k \in \mathcal{X}, \quad k = 0, \dots, N-1, && \text{(2.3e)}\\
& y_k \in \mathcal{Y}, \quad k = 0, \dots, N-1, && \text{(2.3f)}\\
& u_k \in \mathcal{U}, \quad k = 0, \dots, N-1, && \text{(2.3g)}\\
& \Delta u_k \in \Delta\mathcal{U}, \quad k = 0, \dots, N-1, && \text{(2.3h)}\\
& u_{-1} = u(t-1), && \text{(2.3i)}\\
& x_0 = x(t). && \text{(2.3j)}
\end{aligned}
$$

The term integral control action originates from the definition of Δu_k as a discrete-time integrator. Let us rewrite (2.3h) into following form

$$u_k = u_{k-1} + \Delta u_k \implies u_{k+1} = u_k + \Delta u_{k+1}, \tag{2.4}$$

such form resembles dynamical system with integrating behavior. It is to be notated that with this formulation we can not ensure offset-free controller if there is model mismatch. To deal with the model mismatch case we can design observer by augmenting plant model (2.3b)-(2.3c) with a disturbance model in order to capture the mismatch between actual plant and it's model in steady state. The detailed procedure to design observer can be found in Pannocchia and Rawlings (2003), Mohammadkhani et al. (2014), Borrelli et al. (2015)[Chapter 13].

To formulate above MPC problems we will need system model and its prediction over finite-time horizon. In the next sections, we will describe common components used in all formulations such as state-space model and prediction.

2.2.4 State-Space Model

As described in the Section 2.1.2, a mathematical model of the plant is one of the main components of MPC which allows MPC algorithm to predict future movements and react accordingly to the future measurements or estimations. We consider the discrete-time Linear Time-Invariant (LTI) state space model of the type

$$x_{k+1} = Ax_k + Bu_k, \tag{2.5a}$$
$$y_k = Cx_k + Du_k, \tag{2.5b}$$

where $x(t) \in \mathbb{R}^{n_x}$ is the system state vector, $u(t) \in \mathbb{R}^{n_u}$ is the system input vector and $y(t) \in \mathbb{R}^{n_y}$ is the system output vector, moreover, $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_x \times n_u}$, $C \in \mathbb{R}^{n_y \times n_x}$ and $D \in \mathbb{R}^{n_y \times n_u}$ are system matrices with the assumption that pair (A, B) is stabilizable and (C, A) is detectable. The $k \in \mathbb{N}^0$ denotes absolute discrete time. Full state measurement and no disturbances or model uncertainty are assumed, unless explicitly specified.

2.2.5 Prediction

The future response of the controlled plant is predicted using a dynamic model (2.5). Let us consider the state dynamics in the system given by (2.5a). Assume that the whole state vector is measured, so that $\hat{x}_k = x_k$. Also, assume that we know nothing about any disturbances or measurement noise. Then the predicted state sequence (over a time-interval, from 1 to k) generated by the linear state-space model (2.5a) with input sequence u_k can be written as

$$\begin{aligned}
x_1 &= Ax_0 + Bu_0, && (2.6a)\\
x_2 &= Ax_1 + Bu_1, && (2.6b)\\
&= A\left(Ax_0 + Bu_0\right) + Bu_1, \\
&= A^2x_0 + ABu_0 + Bu_1, \\
x_3 &= Ax_2 + Bu_2, && (2.6c)\\
&= A\left(A^2x_0 + ABu_0 + Bu_1\right) + Bu_2, \\
&= A^3x_0 + A^2Bu_0 + ABu_1 + Bu_2, \\
&\vdots \\
x_k &= A^k x_0 + \sum_{j=0}^{k-1}\left(A^{k-1-j}B\right)u_j. && (2.6d)
\end{aligned}$$

Predictive control uses prediction of system evolution over some finite horizon for a decision of optimal control strategy. The prediction is parametrized by currently measured or estimated state x_k. The state predictions x_{k+1} along the prediction horizon N can be formulated as

$$X = \Psi x_0 + \Upsilon U, \tag{2.7}$$

with $\Psi \in \mathbb{R}^{n_x(N+1)\times n_x}$ and $\Upsilon \in \mathbb{R}^{n_x(N+1)\times n_u}$, (2.7) can be summarized as

$$\underbrace{\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix}}_{X} = \underbrace{\begin{bmatrix} I \\ A \\ A^2 \\ A^3 \\ \vdots \\ A^N \end{bmatrix}}_{\Psi} x_0 + \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & \dots & 0 \\ B & 0 & 0 & 0 & \dots & 0 \\ AB & B & 0 & 0 & \dots & 0 \\ A^2B & AB & B & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ A^{N-1}B & A^{N-2}B & A^{N-3}B & A^{N-4}B & \dots & B \end{bmatrix}}_{\Upsilon} \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-1} \end{bmatrix}}_{U}. \tag{2.8}$$

In the following sections we will use state predictions (2.7) in different types of MPC formulations.

2.2.6 QP Problem Formulation for State Regulation MPC

In this type of problem formulation, the goal is to regulate system states to the origin without violating the imposed constraints. Let us consider the system model described in Section 2.2.4 and state predictions described in Section 2.2.5.

Cost Function

The set of control increments is calculated by minimizing an objective function for a prediction horizon. Consider the objective function described in (2.1a) which can be simplified as follows

$$\min_U J(U, x_0) = x_N^T P x_N + X^T Q X + U^T R U, \tag{2.9}$$

where the weighting matrices Q and R are diagonal matrices with dimensions $\mathbb{R}^{n_x N \times n_x N}$ and $\mathbb{R}^{(n_u N)\times(n_u N)}$, respectively. The terminal cost matrix $P \in \mathbb{R}^{n_x \times n_x}$ is solution of the associated algebraic Riccati equation (Borrelli et al., 2015, Chapter 9). Taking the 2-norm of (2.1a) we can get

$$\min_U J(U, x_0) = X^T Q X + U^T R U. \tag{2.10}$$

Substitute (2.7) in to (2.10), we get

$$\begin{aligned} X^TQX + U^TRU &= (\Psi x_0 + \Upsilon U)^T Q (\Psi x_0 + \Upsilon U) + U^TRU, && (2.11a)\\ &= (\Psi x_0)^T Q\Psi x_0 + (\Psi x_0)^T Q\Upsilon U + (\Upsilon U)^T Q\Psi x_0 + && (2.11b)\\ &\quad (\Upsilon U)^T Q\Upsilon U + U^TRU, \\ &= U^T\Upsilon^TQ\Upsilon U + U^TRU + 2x_0^T\Psi^TQ\Upsilon U + x_0^T\Psi^TQ\Psi x_0, && (2.11c)\\ &= U^THU + 2x_0^TFU + x_0^TVx_0, && (2.11d) \end{aligned}$$

where the matrix H, vector F, and constant V are given by

$$\begin{aligned} H &= \Upsilon^TQ\Upsilon + R, && (2.12a)\\ F &= \Psi^TQ\Upsilon, && (2.12b)\\ V &= \Psi^TQ\Psi. && (2.12c) \end{aligned}$$

Unconstrained QP formulation of state regulation MPC (2.10) can be written as

$$J^\star(x_0) \quad = \quad \min_U \{U^THU + 2x_0^TFU\} + x_0^TVx_0. \qquad (2.13)$$

The optimization problem in (2.13) can be written in to the following standard QP form

$$J^\star(x_0) \quad = \quad \min_U \left\{ \frac{1}{2}U^THU + x_0^TFU \right\} + \frac{1}{2}x_0^TVx_0. \qquad (2.14)$$

Solution of Unconstrained Optimization QP Problem

In the absence of constraints, the optimization $U^\star(x_0) = J^\star(x_0)$ has a closed-form solution which can be derived by considering the gradient of J with respect to U

$$\nabla_U J = 2Hu + 2Fx_0. \qquad (2.15)$$

Clearly, $\nabla_U J = 0$ must be satisfied at a minimum point of J, and since H is positive definite, any U such that $\nabla_U J = 0$ is necessarily a minimum point. Therefore, the optimal $U^\star$ is unique only if H is non-singular and is then given by

$$U^\star = -H^{-1}Fx_0. \qquad (2.16)$$

If H is singular (i.e. positive semi-definite rather than positive definite), then the optimal $U^\star$ is non-unique, and a particular solution of $\delta_U J = 0$ has to be defined as $U = -H^\dagger Fx_0$ where $H^\dagger$ is a left inverse of H (so that $H^\dagger H = I$).

Constraints

The purpose of MPC is clearly not to emulate the unconstrained optimal controller like Linear Quadratic Regulator (LQR) (Maciejowski, 2002), which after all is simply a linear feedback law that can be computed off-line using knowledge of the plant model. The real advantage of MPC lies in its ability to determine nonlinear feedback laws which are optimal for constrained systems through numerical calculations that are performed on-line. In the following, we will show how to construct constraints for MPC problem stated in (2.1).

Consider now polyhedral state and input constraint sets of the form

$$\mathcal{X} = \{x \mid D_x x \leq d_x\}, \tag{2.17a}$$

$$\mathcal{U} = \{u \mid D_u u \leq d_u\}, \tag{2.17b}$$

where $\mathcal{X} \in \mathbb{R}^{n_x}$, $\mathcal{U} \in \mathbb{R}^{n_u}$, $D_x \in \mathbb{R}^{n_{xg} \times n_x}$, $D_u \in \mathbb{R}^{n_{ug} \times n_u}$, $d_x \in \mathbb{R}^{n_{xg}}$, and $d_u \in \mathbb{R}^{n_{ug}}$. Here, n_{xg} and n_{ug} are the number of inequality constraints associated with state and input, respectively.

Input Constraints

Input polyhedral constraints allow the controller to find a solution (control actions) that satisfies the input limitations of system actuators. We can find an analogy in adding a saturation on control outputs of PID controller, but in MPC they are part of the optimization problem. They can be modeled by set of inequalities taking the following form

$$u_{\min} \leq u_k \leq u_{\max}, \tag{2.18}$$

this can also be written in vector form as

$$\underbrace{\begin{bmatrix} u_{\min} \\ u_{\min} \\ \vdots \\ u_{\min} \end{bmatrix}}_{U_{\min}} \leq \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}}_{U} \leq \underbrace{\begin{bmatrix} u_{\max} \\ u_{\max} \\ \vdots \\ u_{\max} \end{bmatrix}}_{U_{\max}}, \tag{2.19}$$

which is equivalent to $U \leq U_{\max}$ and $-U \leq -U_{\min}$

$$\underbrace{\begin{bmatrix} -I \\ I \end{bmatrix}}_{G_1} U \leq \underbrace{\begin{bmatrix} -U_{\min} \\ U_{\max} \end{bmatrix}}_{w_1} + \underbrace{\begin{bmatrix} 0_{(Nl \times n)} \\ 0_{(Nl \times n)} \end{bmatrix}}_{E_1} x_0. \tag{2.20}$$

State Constraints

The ability to constrain particular state values is one of the main features of the MPC. It can be used to drive the system within some safety region of state variables and to find a control action that won't push the system into unwanted states (sometimes irreversibly, e.g., in chemical processes). By inducing state constraints to the optimization task, it suddenly becomes more difficult to solve. Similar to the input constraints, state constraints can be modeled as

$$x_{\min} \leq x_k \leq x_{\max}, \tag{2.21}$$

and can be re-written in stacked vector form for $k = 0$ to $k = N - 1$ as

$$\underbrace{\begin{bmatrix} x_{\min} \\ x_{\min} \\ \vdots \\ x_{\min} \end{bmatrix}}_{X_{\min}} \leq \underbrace{\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_N \end{bmatrix}}_{X} \leq \underbrace{\begin{bmatrix} x_{\max} \\ x_{\max} \\ \vdots \\ x_{\max} \end{bmatrix}}_{X_{\max}}. \tag{2.22}$$

Predicted state X in (2.7) can be re-written in terms of U as $\Upsilon U = X - \Psi x_0$. Subsequently, (2.22) turns into the following form

$$X_{\min} \leq X \leq X_{\max}, \tag{2.23a}$$

$$X_{\min} \leq \Psi x_0 + \Upsilon U \leq X_{\max}, \tag{2.23b}$$

$$X_{\min} - \Psi x_0 \leq \Upsilon U \leq X_{\max} - \Psi x_0, \tag{2.23c}$$

and (2.23c) can be simplified as $\Upsilon U \leq X_{\max} - \Psi x_0$ and $-\Upsilon U \leq -X_{\min} + \Psi x_0$. The state constraints in terms of U can be written as

$$\underbrace{\begin{bmatrix} -\Upsilon \\ \Upsilon \end{bmatrix}}_{G_2} U \leq \underbrace{\begin{bmatrix} -X_{\min} \\ X_{\max} \end{bmatrix}}_{w_2} + \underbrace{\begin{bmatrix} \Psi \\ -\Psi \end{bmatrix}}_{E_2} x_0. \tag{2.24}$$

By combining these two inequalities from (2.20) and (2.24), we obtain the constraints in the compact form

$$GU \leq w + Ex_0, \tag{2.25}$$

with

$$G = \begin{bmatrix} G_1 \\ G_2 \end{bmatrix}, \ w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}, \ E = \begin{bmatrix} E_1 \\ E_2 \end{bmatrix}. \tag{2.26}$$

Finally, combining (2.14) with (2.25) gives the QP formulation of the MPC problem in (2.1) as

$$J^*(x_0) = \min_U \left\{ \frac{1}{2} U^T H U + x_0^T F U \right\} + \frac{1}{2} x_0^T V x_0 \tag{2.27a}$$

$$\text{s.t. } GU \leq w + Ex_0, \tag{2.27b}$$

where $x_0 = x(t)$ is the initial condition, $H \in \mathbb{R}^{n_u N \times n_u N}$, $F \in \mathbb{R}^{n_x \times n_u N}$, $V \in \mathbb{R}^{n_x N \times n_x N}$, $G \in \mathbb{R}^{n_{ug} \times n_u N}$, $w \in \mathbb{R}^{n_{ug}}$ and $E \in \mathbb{R}^{n_{ug} \times n_x N}$. The QP problem is strictly convex since, R in (2.10) is assumed to be positive definite. It is worth to mention that it is always possible to reformulate QP problem (2.27) in to standard QP form.

2.2.7 QP Problem Formulation for Reference Tracking with Integral Action

In the reference tracking problem, we will need system model with state and output dynamics as described in (2.5) with full state measurements and same assumptions. Next, consider the predicted state sequence (over a time-interval, from 1 to k) generated by the linear state-space model (2.5a) with input sequence u_k as given in (2.7). Here, the goal is to minimize the error between predicted output and output reference for that we will need to predict output which can be done by iterating output along the prediction horizon N as we did for states in Section 2.2.5, i.e.,

$$\begin{aligned}
y_0 &= Cx_0 + Du_0, && (2.28a)\\
y_1 &= Cx_1 + Du_1, && (2.28b)\\
&= C(Ax_0 + Bu_0) + Du_1, &&\\
&= CAx_0 + CBu_0 + Du_1, &&\\
y_2 &= Cx_2 + Du_2, && (2.28c)\\
&= C(Ax_1 + Bu_1) + Du_2, &&\\
&= CA^2x_0 + CABu_0 + CBu_1 + Du_2, &&\\
&\vdots && (2.28d)\\
y_k &= CA^kx_0 + CA^{k-1}Bu_0 + CA^{k-2}Bu_1 + \cdots + Du_k. && (2.28e)
\end{aligned}$$

With the help of state prediction expression (2.7), output predictions (2.28e) can be expressed as follows

$$Y = \bar{C}\Psi x_0 + \bar{C}\Upsilon U, \tag{2.29}$$

where $Y \in \mathbb{R}^{n_y N}$ is the vector of predicted values of the output and $\bar{C} \in \mathbb{R}^{Nn_x \times Nn_y}$ is the diagonal matrix of system output matrix C. Y and $\bar{C}$ is given as

$$Y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{bmatrix}, \quad \bar{C} = \begin{bmatrix} C & 0 & 0 & 0 & \dots & 0 \\ 0 & C & 0 & 0 & \dots & 0 \\ 0 & 0 & C & 0 & \dots & 0 \\ 0 & 0 & 0 & C & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & C \end{bmatrix}. \tag{2.30}$$

Cost Function

The goal of the controller is, to make the difference between the output, y_k and the reference r_k as small as possible without offset. This can be done by using a least squares problem. The cost function for the reference tracking problem with weighted 2-norm is given as

$$\min_U J(U, x_0) = \sum_{k=0}^{N-1} \parallel y_k - r_k \parallel_Q^2 + \parallel \Delta u_k \parallel_R^2. \tag{2.31}$$

In above matrices, Q and R are assumed to be symmetric and positive definite. In the above cost function, the first term provides a mechanism to allow different weightings on different outputs and second term allows different penalties for different input moves. Furthermore, as described in Section 2.2.3 for offset-free reference tracking we need to incorporate integral action in the controller i.e., defining $\Delta u = u_k - u_{k-1}$ and applying u_k to the system. By introducing $Y_r \in \mathbb{R}^{n_y N}$ as a vector containing the output reference

$$Y_r = \begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_N \end{bmatrix}, \tag{2.32}$$

and using output predictions (2.29) the objective function can be written as

$$\min_{U} J(U, x_0) = \sum_{k=0}^{N-1} \| Y - Y_r \|_Q^2 + \| u_k - u_{k-1} \|_R^2. \tag{2.33}$$

To make this problem easier to solve, it is convenient to express it as a QP problem. Next, we will show translation of (2.33) in to QP problem. Due to the lengthy expressions we will first translate output term (J_y) and then input term (J_u).

By substituting (2.29) for Y into (2.33) we get

$$\begin{aligned} J_y =& \| \bar{C}\Psi x_0 + \bar{C}\Upsilon U - Y_r \|_Q^2, && \text{(2.34a)} \\ =& \| \bar{C}\Upsilon U - (Y_r - \bar{C}\Psi x_0) \|_Q^2, && \text{(2.34b)} \\ =& \| \bar{C}\Upsilon U - v \|_Q^2, \quad v = (Y_r - \bar{C}\Psi x_0). && \text{(2.34c)} \end{aligned}$$

By taking the 2−norm of above expression we can further simplify it as follows

$$\begin{aligned} J_y =& (\bar{C}\Upsilon U - v)^T Q (\bar{C}\Upsilon U - v), && \text{(2.35a)} \\ =& U^T \Upsilon^T \bar{C}^T Q \bar{C} \Upsilon U - 2(\Upsilon^T \bar{C}^T Q v)^T U + v^T Q v, && \text{(2.35b)} \\ =& U^T H_y U + F_y^T U + \rho_y, && \text{(2.35c)} \end{aligned}$$

where H_y, F_y^T, ρ_y are given by

$$\begin{aligned} H_y =& \Upsilon^T \bar{C}^T Q \bar{C} \Upsilon, \\ F_y =& -2\Upsilon^T \bar{C}^T Q v, \\ =& 2\Upsilon^T \bar{C}^T Q \Phi x_0 - 2\Upsilon^T \bar{C}^T Q Y_r, \\ =& 2M_{x_0} x_0 + 2M_{Y_r} Y_r, \quad M_{x_0} = \Upsilon^T \bar{C}^T Q \Phi, M_{Y_r} = -\Upsilon^T \bar{C}^T Q, \\ \rho_y =& v^T Q v. \end{aligned}$$

Since ρ_y does not influence the solution to the optimization problem, it can be discarded. Also, note that the gradient F_y is dynamic and needs to be updated for every time step, as opposed to the Hessian H_y, which is static. The unconstrained QP problem for the objective function in (2.31) is

$$J_y = \frac{1}{2} U^T H_y U + F_y^T U, \tag{2.37}$$

where $H_y = \Upsilon^T \bar{C}^T Q \bar{C} \Upsilon$ and $F_y = M_{x_0} x_0 + M_{Y_r} Y_r$.

Now, we will formulate QP problem for input regularization. Similar to J_y we can formulate J_u as a QP problem,

$$J_u = \sum_{k=0}^{N-1} \parallel u_k - u_{k-1} \parallel_R^2, \tag{2.38a}$$

$$= \sum_{k=0}^{N-1} \left(u_k - u_{k-1}\right)^T R \left(u_k - u_{k-1}\right), \tag{2.38b}$$

$$= \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{bmatrix} \underbrace{\begin{bmatrix} 2R & -R & 0 & \dots & 0 \\ -R & 2R & -R & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & -R & 2R & -R \\ 0 & 0 & 0 & -R & R \end{bmatrix}}_{H_u} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{bmatrix} \tag{2.38c}$$

$$+ 2 \underbrace{\begin{bmatrix} -R \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{M_{u-1}} u_{-1}^T \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{bmatrix} + u_{-1} R u_{-1},$$

$$= U^T H_u U + 2(M_{u-1} u_{-1})^T U + u_{-1} R u_{-1}. \tag{2.38d}$$

This shows, that introducing Ju extends the QP problem by following term

$$F_u \quad = \quad M_{u-1} u_{-1}. \tag{2.39}$$

Like with ρ_y, the term $u_{-1} R u_{-1}$ is discarded, because the lack of influence on the solution to the problem. The unconstrained QP problem the objective function in (2.38a) is

$$J_u = \frac{1}{2} U^T H_u U + F_u^T U. \tag{2.40}$$

Now, its time to combine QP problems for output and input objective functions (2.37) and (2.40), we will get

$$J^\star(x_0) = \min_U \left\{ \frac{1}{2} U^T H U + F^T U \right\}, \tag{2.41}$$

where U is the vector of control inputs, $H = H_y + H_u$ is the Hessian and $F = F_y + F_u$ is the gradient which is updating at every time instant.

Constraints

Similar to the constraints in regulation problem, we can apply constraints for the tracking problem stated in (2.2). Having determined quadratic cost as a function of input predictions in Section (2.2.5), the input, output, state, and slew rate constraints are formulated in next sections.

Input Constraints

These are the most commonly encountered constraints among all constraint types. These are the hard constraints on the system. Simply, we demand that

$$u_{\min} \leq u_k \leq u_{\max}, \tag{2.42}$$

this can also be written in vector form as

$$\underbrace{\begin{bmatrix} u_{\min} \\ u_{\min} \\ \vdots \\ u_{\min} \end{bmatrix}}_{U_{\min}} \leq \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}}_{U} \leq \underbrace{\begin{bmatrix} u_{\max} \\ u_{\max} \\ \vdots \\ u_{\max} \end{bmatrix}}_{U_{\max}}, \tag{2.43}$$

which is equivalent to $U \leq U_{\max}$ and $-U \leq -U_{\min}$

$$\underbrace{\begin{bmatrix} -I \\ I \end{bmatrix}}_{G_1} U \leq \underbrace{\begin{bmatrix} -U_{\min} \\ U_{\max} \end{bmatrix}}_{w_1}. \tag{2.44}$$

Input Slew Rate Constraints

In the industrial system, it is important to take care of wear-and-tear of actuators due to a sudden increase or decrease of input, e.g., control valve or motor. To prevent some stress on actuators of the system, we can impose constraints on a change in input rate so that actuator will operate smoothly. The input rate of movement is the change from k to $k+1$ and therefore it is called Δu. These constraints can be modeled as

$$\Delta u_{\min} \leq \Delta u_k \leq \Delta u_{\max}, \tag{2.45}$$

in vector form it can be written as

$$\underbrace{\begin{bmatrix} \Delta u_{\min} \\ \Delta u_{\min} \\ \vdots \\ \Delta u_{\min} \end{bmatrix}}_{\Delta_{\min}} \leq \underbrace{\begin{bmatrix} \Delta u_0 \\ \Delta u_1 \\ \vdots \\ \Delta u_{N-1} \end{bmatrix}}_{\Delta U} \leq \underbrace{\begin{bmatrix} \Delta u_{\max} \\ \Delta u_{\max} \\ \vdots \\ \Delta u_{\max} \end{bmatrix}}_{\Delta_{\max}}, \tag{2.46}$$

as we know $\Delta u_k = u_k - u_{k-1}$, we can replace Δu_k with $u_k - u_{k-1}$ in (2.46)

$$\begin{bmatrix} \Delta u_{\min} \\ \Delta u_{\min} \\ \vdots \\ \Delta u_{\min} \end{bmatrix} \leq \begin{bmatrix} u_0 - u_{-1} \\ u_1 - u_0 \\ \vdots \\ u_{N-1} - u_{N-2} \end{bmatrix} \leq \begin{bmatrix} \Delta u_{\max} \\ \Delta u_{\max} \\ \vdots \\ \Delta u_{\max} \end{bmatrix}, \tag{2.47}$$

$$\begin{bmatrix} \Delta u_{\min} \\ \Delta u_{\min} \\ \vdots \\ \Delta u_{\min} \end{bmatrix} + \underbrace{\begin{bmatrix} -I \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{\mathcal{I}} u_{-1} \leq \underbrace{\begin{bmatrix} I & 0 & 0 & 0 \\ -I & I & 0 & 0 \\ 0 & 0 & \ddots & \ddots \\ 0 & 0 & -I & I \end{bmatrix}}_{\Lambda} \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}}_{U} \leq \begin{bmatrix} \Delta u_{\max} \\ \Delta u_{\max} \\ \vdots \\ \Delta u_{\max} \end{bmatrix} + \underbrace{\begin{bmatrix} -I \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{\mathcal{I}} u_{-1}, \tag{2.48}$$

subsequently, we obtain

$$\Delta U_{\min} + \mathcal{I} u_{-1} \leq \Lambda U \leq \Delta U_{\max} + \mathcal{I} u_{-1}. \tag{2.49}$$

The relation (2.49) is equivalent to $\Lambda U \leq \Delta U_{\max} + \mathcal{I} u_{-1}$ and $-\Lambda U \leq -\Delta U_{\min} + \mathcal{I} u_{-1}$. Finally, it can be re-written in the vector form as

$$\underbrace{\begin{bmatrix} -\Lambda \\ \Lambda \end{bmatrix}}_{G_2} U \leq \underbrace{\begin{bmatrix} -\Delta U_{\min} + \mathcal{I} u_{-1} \\ \Delta U_{\max} - \mathcal{I} u_{-1} \end{bmatrix}}_{w_2}. \tag{2.50}$$

Output Constraints

Output constraints are analogous to the input constraints, i.e., limitations to the maximum and minimum output. The output at $k = 0$ cannot be affected, so the constraint here is disregarded. The output constraints are expressed as

$$y_{\min} \leq y_k \leq y_{\max}, \tag{2.51}$$

and in vector from as

$$\underbrace{\begin{bmatrix} y_{\min} \\ y_{\min} \\ \vdots \\ y_{\min} \end{bmatrix}}_{Y_{\min}} \leq \underbrace{\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_N \end{bmatrix}}_{Y} \leq \underbrace{\begin{bmatrix} y_{\max} \\ y_{\max} \\ \vdots \\ y_{\max} \end{bmatrix}}_{Y_{\max}}. \tag{2.52}$$

From (2.29) insert $Y = \bar{C}\Psi x_0 + \bar{C}\Upsilon U$ in (2.52) we get

$$Y_{\min} \leq Y \leq Y_{\max}, \tag{2.53a}$$

$$Y_{\min} \leq \bar{C}\Psi x_0 + \bar{C}\Upsilon U \leq Y_{\max}, \tag{2.53b}$$

$$Y_{\min} - \bar{C}\Psi x_0 \leq \bar{C}\Upsilon U \leq Y_{\max} - \bar{C}\Psi x_0. \tag{2.53c}$$

The expression in (2.53) can be re-written as $\bar{C}\Upsilon U \leq Y_{\max} - \bar{C}\Psi x_0$ and $-\bar{C}\Upsilon U \leq -Y_{\min} + \bar{C}\Psi x_0$. Output constraints in terms of U can be written as

$$\underbrace{\begin{bmatrix} -\bar{C}\Upsilon \\ \bar{C}\Upsilon \end{bmatrix}}_{G_3} U \leq \underbrace{\begin{bmatrix} -Y_{\min} + \bar{C}\Psi x_0 \\ Y_{\max} - \bar{C}\Psi x_0 \end{bmatrix}}_{w_3}. \tag{2.54}$$

State Constraints

Inequality state constraints can be represented as

$$x_{\min} \leq x_k \leq x_{\max}, \tag{2.55}$$

and can be re-written in stacked vector form for $k = 0$ to $k = N - 1$ as

$$\underbrace{\begin{bmatrix} x_{\min} \\ x_{\min} \\ \vdots \\ x_{\min} \end{bmatrix}}_{X_{\min}} \leq \underbrace{\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_N \end{bmatrix}}_{X} \leq \underbrace{\begin{bmatrix} x_{\max} \\ x_{\max} \\ \vdots \\ x_{\max} \end{bmatrix}}_{X_{\max}}. \tag{2.56}$$

Predicted state $X = \Psi x_0 + \Upsilon U$ in (2.7) can be re-written in terms of U as $\Upsilon U = X - \Psi x_0$. Subsequently, (2.56) turns into the following form

$$X_{\min} \leq X \leq X_{\max}, \tag{2.57a}$$

$$X_{\min} \leq \Psi x_0 + \Upsilon U \leq X_{\max}, \tag{2.57b}$$

$$X_{\min} - \Psi x_0 \leq \Upsilon U \leq X_{\max} - \Psi x_0. \tag{2.57c}$$

The expression (2.57c) can be simplified as $\Upsilon U \leq X_{\max} - \Psi x_0$ and $-\Upsilon U \leq -X_{\min} + \Psi x_0$. The state constraints in terms of U can be written as

$$\underbrace{\begin{bmatrix} -\Upsilon \\ \Upsilon \end{bmatrix}}_{G_4} U \leq \underbrace{\begin{bmatrix} -X_{\min} + \Psi x_0 \\ X_{\max} - \Psi x_0 \end{bmatrix}}_{w_4}. \tag{2.58}$$

Combining all the constraints from (2.44), (2.50), (2.54), and (2.58), we get

$$\underbrace{\begin{bmatrix} G_1 \\ G_2 \\ G \\ G_4 \end{bmatrix}}_{G} U \leq \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}}_{w}. \tag{2.59}$$

Th unconstrained QP problem stated in (2.41) can be extended for constrained QP problem as

$$J^*(x_0) = \min_U \left\{ \frac{1}{2} U^T H U + c^T U \right\}, \tag{2.60a}$$

$$\text{s.t. } GU \leq w. \tag{2.60b}$$

2.3 Solving QP Problems in MPC

Once the MPC problem is formulated as a constrained quadratic programming problem, the next task is to obtain optimal control actions based on current measurements, which can be carried out using different optimization methods. The reliable general purpose solvers are available for the solution of QP problems and nominal MPC problems are solved by directly applying one of the solver on-line, which has, however, restricted the applicability of MPC to slow dynamic processes. In recent years, various methods have been developed with the goal of enabling MPC to be used for fast sampled systems. These approaches can generally be classified into two main paradigms: on-line MPC and explicit/off-line MPC methods.

The most general approach used to solve QP problems is to use the Active Set (Fletcher, 2013), Interior Point (Nocedal and Wright, 2006, Chapter 16), and gradient (Snyman, 2005) methods, which have shown good convergence and stability properties. In particular, the active set and interior point methods provide an excellent framework for the solution of very large-scale optimization problems arising in the process industries. However, the main challenge that arises for applications in fast-sampled dynamic systems is the requirement of solving the optimal control problem in real-time that to on embedded devices. To accomplish this requirement, it is necessary to implement an efficient solver that exploits the characteristics of the problem and the available hardware resources in order to reduce the computational time, memory requirements and power consumption.

2.4 Summary

This chapter has presented the concept of model predictive control and the features which make this control strategy one of the most employed for controlling complex systems. In MPC, at each sample time, an open-loop optimal control problem is solved over a finite horizon considering current state of the plant. The computed optimal input signal is applied to the plant, in the next sample time and corresponding measurements are sent back to the controller. This procedure is repeated iteratively, which makes MPC a kind of feedback controller. MPC offers an elegant framework to solve a wide range of control problems such as state regulation, output tracking, supervision, etc. and have ability to handle constraints on input and state. This chapter gives detailed formulations of state regulation MPC (see Section 2.2.6) and reference tracking MPC with integral action (see Section 2.2.7). The formulated QP problem can be solved using active set or interior point methods (see Section 2.3). MPC has been widely employed in oil and gas refineries, automotive, aerospace, biomedical, process industries, and many more. The main bottleneck in the success of MPC lies in solving one optimization problem at each sample time. This restricts MPC in many fast sampled applications or needs powerful computational unit. The issues of computational complexity can be overcome by moving the on-line burden of optimization to off-line. This concept of off-line MPC or explicit MPC is presented in next chapter.

Chapter 3

Explicit Model Predictive Control

3.1 Introduction

In Chapter 2, we have seen that the model predictive control problem intended for regulation or tracking can be formulated into an unconstrained or constrained QP problem. The formulated QP problem can be solved using on-line optimization methods presented in Section 2.3. On-line methods have shown good convergence and stability properties. However, the main challenge appears when MPC is used for controlling real-time dynamic systems with high sampling rates and running on resource-constrained embedded platforms, such as PLCs, FPGAs, and micro-controllers. In this type of set-ups, the computational time, limited resource, and the requirement of control accuracy becomes a crucial factor (Johansen, 2014). To extend the use of MPC on embedded hardware, it is necessary to implement an efficient solver that exploits the characteristics of the problem and the available hardware resources in order to reduce the computational time, memory requirements, and power consumption.

To overcome the limitations of on-line implementation of MPC, a multi-parametric Quadratic Programming (mp-QP) (Pistikopoulos et al., 2007a) (Borrelli et al., 2015) based approach called *Explicit* Model Predictive Control (EMPC) was proposed by Bemporad et al. (2000), Bemporad et al. (2002) where the on-line burden of optimization is moved off-line. In a multi-parametric programming, the optimal solution of an optimization problem is determined as an explicit function of certain varying parameters. Therefore, multi-parametric programming avoids the need to solve a new optimization problem when the parameter changes since the optimal solution can readily be updated using the pre-computed function. In relation to MPC, the multi-parametric programming can be used to obtain the optimal control inputs as an explicit function of the state measurements, considering these as the parameters of the optimization problem. This allows the on-line computational burden to be reduced to a sequence of function evaluations, eliminating the need of a real-time optimization solver which is the main bottleneck of the embedded implementation of MPC controllers. Explicit MPC has found its applications in many areas of science, engineering and technology. Table 3.1 summarizes some application reported till the date.

3.1.1 Explicit MPC Concept

In explicit MPC, the optimal control law is pre-computed off-line and once as a function of all possible initial states. For a large class of MPC problems, such a control law can be shown to take a form of the Piecewise Affine (PWA) function defined over a polyhedral partition in the state-space, which maps state measurements onto the optimal control inputs. Having a pre-computed PWA function at hand, explicit MPC needs to evaluate the PWA function on-line at each sample time to compute the optimal control actions based on the current state measurement. Fig. 3.1 shows the explicit MPC scheme where the task is divided into two phases, i.e., off-line and on-line. In off-line phase, the PWA control law is constructed, and in on-line phase, it is evaluated at each sample time using point location algorithm.

3.2 MPC Problem as a Multi-Parametric QP Problem

Recall the discrete-time LTI system model presented in Section 2.2.4,

$$x_{k+1} = Ax_k + Bu_k, \tag{3.1a}$$

$$y_k = Cx_k + Du_k, \tag{3.1b}$$

Table 3.1: Applications of EMPC.

Area of Application	**Contributors**
Electric	Beccuti et al. (2007), Beccuti et al. (2009), Mariethoz et al. (2009), Ameen et al. (2012), Mariethoz et al. (2012), Shen et al. (2013), Dirscherl et al. (2015)
Automotive	Lee and Line (2008), Naus et al. (2010), Alamir et al. (2010), Oliveri et al. (2011), El Hadef et al. (2013), Montague et al. (2013), Honek et al. (2015), Csekő et al. (2015)
Chemical	Grancharova et al. (2003), Grancharova et al. (2004), Ławryńczuk (2009), Zanini et al. (2009), Sanchez-Cossio et al. (2015), Pu and Yu-hong (2015), Drgoňa et al. (2017)
Aerospace	Krogstad et al. (2005), Liu et al. (2011), Liu et al. (2012), Zhao et al. (2014), Liu et al. (2015), Pu and Yu-hong (2015), Zhang et al. (2016)
Bio-medical	Pistikopoulos (2009), Kirubakaran et al. (2013), Naşcu et al. (2016)
Power Generation	Puig et al. (2007), Hredzak et al. (2015), Jiang et al. (2016)
Industrial Systems	de la Peña et al. (2005), Stephens et al. (2011), Kirubakaran et al. (2016)
Mechatronic	Ulbig et al. (2008), Almurib et al. (2010), Gerkšič and de Tommasi (2013), Takács et al. (2016b), Klaučo et al. (2017)
Smart Buildings	Drgona et al. (2013), Koehler and Borrelli (2013), Parisio et al. (2014), Sahu et al. (2015)

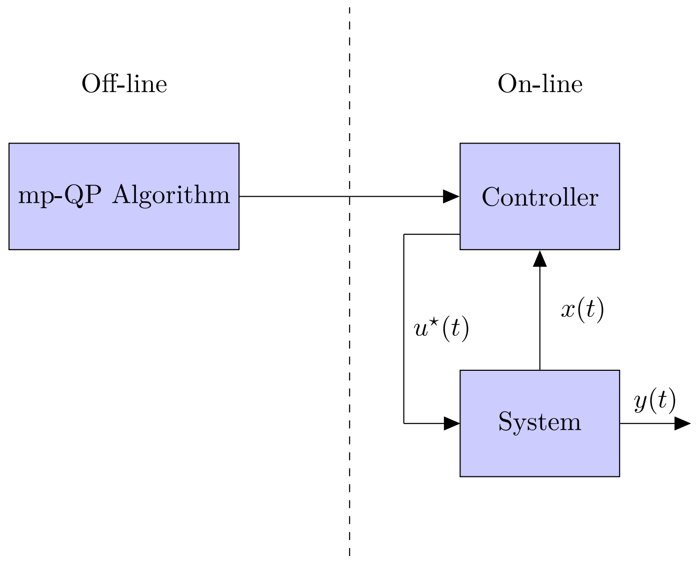

Figure 3.1: Explicit model predictive control scheme.

where $x(t) \in \mathbb{R}^{n_x}$ is the system state vector, $u(t) \in \mathbb{R}^{n_u}$ is the system input vector and $y(t) \in \mathbb{R}^{n_y}$ is the system output vector, moreover, $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_x \times n_u}$, $C \in \mathbb{R}^{n_y \times n_x}$ and $D \in \mathbb{R}^{n_y \times n_u}$ are system matrices.

In Section 2.2, we have seen that using system model (3.1) the MPC problem can be designed for state regulation (see, Section 2.2.6) or for reference tracking (see, Section 2.2.7) purpose. For the convenience recall, the CFTOC problem designed for state regulation purpose

$$\min_{U} \; x_N^T P x_N + \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k \tag{3.2a}$$

$$\text{s.t.} \; x_{k+1} = A x_k + B u_k, \quad k = 0, \ldots, N-1, \tag{3.2b}$$

$$x_k \in \mathcal{X}, \quad k = 0, \ldots, N-1, \tag{3.2c}$$

$$u_k \in \mathcal{U}, \quad k = 0, \ldots, N-1, \tag{3.2d}$$

$$x_N \in \mathcal{X}_f, \tag{3.2e}$$

$$x_0 = x(t), \tag{3.2f}$$

where $Q \in \mathbb{R}^{n_x \times n_x}$, $P \in \mathbb{R}^{n_x \times n_x}$, and $R \in \mathbb{R}^{n_u \times n_u}$ are the weighting matrices with conditions $Q \succeq 0$ and $P \succeq 0$ to be positive semi-definite and $R \succ 0$ to be positive definite. Moreover, N is the prediction horizon, x_{k+1} is the vector of predicted states based on prediction model (3.1a), $U = \{u_0, \ldots, u_{N-1}\}$ is the sequence of control actions, and $\mathcal{X}$, $\mathcal{U}$, and $\mathcal{X}_f$ are the polyhedral constraint sets.

In Section 2.2.6, it is shown that the MPC problem can be formulated as a QP problem,

$$J^*(x_0) = \min_U \left\{ \frac{1}{2} U^T H U + x_0^T F U \right\} + \frac{1}{2} x_0^T V x_0 \tag{3.3a}$$

$$\text{s.t. } GU \leq w + E x_0, \tag{3.3b}$$

where $H \in \mathbb{R}^{n_u N \times n_u N}$, $F \in \mathbb{R}^{n_x \times n_g}$, $V \in \mathbb{R}^{n_x \times n_x}$, $G \in \mathbb{R}^{ng \times n_u N}$, $w \in \mathbb{R}^{n_g}$, $E \in \mathbb{R}^{n_g \times n_x}$, and x_0 is the current state of the system.

The on-line solution of this QP problem is computationally demanding which restricts the use of MPC controller in real-time and resource constrained systems. To overcome this issue, an idea of reformulating optimization problem (3.3) as a multi-parametric quadratic programming problem was proposed by Bemporad et al. (2000). It has been shown that by doing some algebraic manipulation the CFTOC problem (3.2) can be reformulated as a mp-QP problem, i.e.,

$$J_z^\star(x_0) = \min_z \tfrac{1}{2} z^T H z \tag{3.4a}$$

$$\text{s.t. } Gz \leq w + S x_0, \tag{3.4b}$$

where $z \in \mathbb{R}^{n_z} := U + H^{-1} F^T x_0$ and x_0 is treated as vector of parameters. $S = E + G H^{-1} F^T$ and $J_z^\star(x_0) = J^\star(x_0) - \frac{1}{2} x_0^T (V - F H^{-1} F^T) x_0$. The number of inequalities are denoted n_g and the number of free variables is $n_z = n_u N$. The goal of explicit MPC/mp-QP problem is to find the solution of the optimization problems(3.4) in an explicit form $z^\star = z^\star(x_0)$. In the next, we will present the solution mp-QP problem which is a continuous PWA function of the current state x_0.

3.2.1 Solution of mp-QP Problem

The solution of mp-QP problem (3.4) can be approached by employing the principles of parametric non-linear programming, and in particular the first-order Karush-Kuhn-Tucker (KKT) optimality conditions (Karush, 1939) to derive the $\mathcal{H}$-polyhedral representation of the critical regions and to compute the optimizer function $z^\star(x_0)$

and the value function $J^\star(x_0)$ inside each critical region. The first-order (KKT) optimality conditions mp-QP problem are given as

$$\begin{aligned} Hz^\star + G^T\mu &= 0, \ \ \mu \in \mathbb{R}^{n_g}, && (3.5a)\\ \mu_j^\star(G_j z^\star - w_j - S_j x_0) &= 0, \ \ j = 1, 2, \ldots, n_g, && (3.5b)\\ \mu^\star &\geq 0, && (3.5c)\\ Gz^\star - w - Sx_0 &\leq 0. && (3.5d) \end{aligned}$$

We solve (3.5a) for $z^\star$

$$z^\star = -H^{-1}G^T\mu^\star, \tag{3.6}$$

and complementary slackness conditions can be obtained by inserting (3.6) in (3.5b).

$$\mu_j^\star(-G_j H^{-1} G_j^T \mu_j^\star - w_j - S_j x_0) = 0 \ \ j = 1, 2, \ldots, n_g. \tag{3.7}$$

Let $\mu_{\mathcal{A}}$ denote the Lagrange multipliers corresponding to the active constraint. For active constraint

$$(-G_{\mathcal{A}} H^{-1} G_{\mathcal{A}}^T)\mu_{\mathcal{A}}^\star - w_{\mathcal{A}} - S_{\mathcal{A}} x_0 = 0, \tag{3.8}$$

where $G_{\mathcal{A}}, w_{\mathcal{A}}$, and $S_{\mathcal{A}}$ are the sub-matrices of G, w, and S respectively, consisting of the rows indexed by active constraint $\mathcal{A}$.

If the set of active constraint $\mathcal{A}$ is empty, then $\mu^\star = 0$ and therefore $z^\star = 0$ which implies that the critical region $CR_{\mathcal{A}}$ is

$$CR_{\mathcal{A}} = \{x_0 \colon w + Sx_0 > 0\}. \tag{3.9}$$

If the rows of $G_{\mathcal{A}}$ are linearly independent then it implies that $(G_{\mathcal{A}} H^{-1} G_{\mathcal{A}}^T)$ is a square full rank matrix and therefore

$$\mu_{\mathcal{A}}^\star = -(G_{\mathcal{A}} H^{-1} G_{\mathcal{A}}^T)^{-1}(w_{\mathcal{A}} + S_{\mathcal{A}} x_0). \tag{3.10}$$

Combining (3.6) with (3.10) we get the expression for $z^\star$ as

$$z^\star = H^{-1} G_{\mathcal{A}}^T (G_{\mathcal{A}} H^{-1} G_{\mathcal{A}}^T)^{-1}(w_{\mathcal{A}} + S_{\mathcal{A}} x_0). \tag{3.11}$$

Note that $z^\star$ is also an affine function of x and objective function $J^\star(x_0)$ is quadratic as $J^\star(x_0) = z^\star Hz$. In the back-transformation, the minimizer $U^\star(x_0)$ of the problem (3.3) is given as

$$U^\star(x_0) = T_i x_0 + v_i, \tag{3.12}$$

which is an affine function of x_0 and T_i and v_i are given as

$$T_i = H^{-1}G_{\mathcal{A}}^T(G_{\mathcal{A}}H^{-1}G_{\mathcal{A}}^T)^{-1}S_{\mathcal{A}} - H^{-1}F^T,\ v_i = H^{-1}G_{\mathcal{A}}^T(G_{\mathcal{A}}H^{-1}G_{\mathcal{A}}^T)^{-1}w_{\mathcal{A}}.$$

Moreover, as the active constraints are defined over the set $\mathcal{A}(x_0)$, conditions (3.5c) and (3.5d) must be satisfied. Therefore by plugging in (3.10) in (3.5c) and (3.11) in (3.5d) one obtains a polyhedral description in the parameter space where $U^\star(x_0)$ is valid

$$\mathcal{R}_i = \{x_0 \in \mathbb{R}^{n_x} \mid L_i x_0 \leq h_i\}, \tag{3.13}$$

where,

$$L_i = \begin{bmatrix} G(T_i + H^{-1}F^T) - S \\ (G_{\mathcal{A}}H^{-1}G_{\mathcal{A}}^T)S_{\mathcal{A}} \end{bmatrix}, h_i = \begin{bmatrix} w - Gv_i \\ -(G_{\mathcal{A}}H^{-1}G_{\mathcal{A}}^T)w_{\mathcal{A}} \end{bmatrix}.$$

Note that the optimizer (3.12) is actually associated with the region of active constraints (3.13). To cover the whole feasible area, the algorithm traverses through the parameter space and iteratively detects the active sets $\mathcal{A}(x_0)$. By this way a sequence of affine functions (3.12) is generated, each corresponding to the given region (3.13). The final result of the mp-PQ problem forms $i = 1, \ldots, n_{\mathcal{R}}$ partitions of PWA function defined over $\mathcal{K}^\star = \cup \mathcal{R}_i$ partitions, where $\mathcal{K}^\star$ is the set of feasible parameters.

In the above, we have seen that if Linear Independence Constraint Qualification (LICQ) holds we get optimizer (3.13). But, if there can be a situation where LICQ does not hold meaning the rows of $G_{\mathcal{A}}$ are not linearly independent. For instance, this happens when more than n_z constraints are active at the optimizer, i.e., in a case of primal degeneracy. In this case, the vector of Lagrange multipliers $\lambda^\star$ might not be uniquely defined, as the dual problem of (3.4) is not strictly convex. Note that dual degeneracy and non-uniqueness of optimizer cannot occur, as $H \succ 0$. The optimizer can be obtained for this case using procedure as given in (Borrelli et al., 2015, Chapter 7).

3.2.2 Properties of the mp-QP Problem

It has been shown by several authors, see, e.g., Bemporad et al. (2002), Dua et al. (2002), (Borrelli et al., 2015, Chapter 7) that the multi-parametric programs have the following properties

1. the closure of critical regions $CR_{\mathcal{A}}$ is a polyhedron,

2. the optimizer $U^\star(x_0)$ is a linear PWA function of the state inside $CR_{\mathcal{A}}$, i.e., $U^\star(x_0) = T_i x_0 + v_i$ as shown in (3.13),

3. the objective function $J^\star(x_0)$ is a quadratic function of the parameter inside $CR_{\mathcal{A}}$, i.e., $J^\star(x_0) = z^\star Hz$.

First of all, this gives us the ability to finitely represent and compute the explicit solution. Second, a PWA function is well suited for real-time applications, which can be stored in the form of a LUT.

3.2.3 Multi-Parametric QP Algorithm

The goal of a mp-QP algorithm is to determine the partition of $\mathcal{K}^\star$ into critical regions $CR_{\mathcal{A}}$, and find the expression of the functions $J^\star(x_0)$ and $z^\star(x_0)$ for each critical region. The mp-QP algorithm has two components:

- active set generator: it computes the set of active constraints $\mathcal{A}(x_0)$
- KKT solver: it computes $CR_{\mathcal{A}}$ and the expression of $J^\star(x_0)$ and $z^\star(x_0)$ required for $CR_{\mathcal{A}}$ as descried in(3.11)

The active set generator is the critical part in the mp-QP algorithm. In principle, one could simply generate all the possible combinations of active sets. However, in many problems only a few active constraints sets generate full-dimensional critical regions inside the region of interest. Therefore, the goal is to design an active set generator algorithm which computes only the active sets $\mathcal{A}$ with the associated full-dimensional critical regions covering only $\mathcal{K}^\star$.

There are several approaches to solve mp-QP algorithm. The methods of solving mp-QP problem were discussed in detailed in Bank et al. (1982). A geometric method for solving mp-QPs was presented in Bemporad et al. (2002). The method constructs a critical region in a neighborhood of a given parameter, by using the KKT conditions for optimality, and then recursively explores the parameter space outside such region. However, as this method introduces many artificial cuts, several researchers have proposed modifications, such as a variable step-size approach (Baotić, 2002), exploiting the active set of the neighboring critical regions from the irredundant constraints (TøNdel et al., 2003). These algorithms cannot guarantee that the entire parameter space will be explored. So, to overcome this issue, a facet-to-facet property of problem is explored in Spjøtvold et al. (2006).

Each region of optimality, i.e. critical region, is uniquely identified by its active set. An enumeration based approach was suggested in Seron et al. (2002) which needs the exhaustive enumeration of all active sets in order to solve mp-QP problems.

Until the beginning of this decade, mp-QP solvers were computational intractable but, it was improved when a branch-and-bound algorithm was suggested in Gupta et al. (2011) which is based on the fact that if a candidate active set is infeasible, so it is its superset. The efficiency of this algorithm was improved in Feller et al. (2013) by considering symmetry elements. Recently, a graph-based approach is presented in Oberdieck et al. (2017). In graph-based method the solution of mp-QP problem is given by connected graphs. Each node is thereby an active set and a connection between two nodes is generated based on geometrical arguments which indicate adjacency. This approach limits the number of candidate active sets to be considered. In the case of primal and dual degeneracy, it can be proven that there exists a single graph which solves the problem, resulting in a set of disjoint critical regions. For the overview of theoretical and algorithmic advances in multi-parametric programming see, e.g., Pistikopoulos et al. (2007a), Pistikopoulos et al. (2007b), Pistikopoulos et al. (2012).

3.3 Point Location Algorithm

Once the state-space is divided off-line into critical regions, the on-line evaluation of explicit MPC controllers is reduced to find the critical region in which the current state variables belong to, and then to evaluate the control action as the PWA function associated with that region. Since the critical regions tend to be irregular, the problem of determining the region is known as the point location problem. In Point Location Algorithm (PLA), current state, $x(t)$ of the system is measured/estimated and based on that state index, i of the region containing the current state is identified. Then, based on the index, the corresponding control law $(u^{\star}(t))$ is applied to the system. Fig. 3.2 shows the working of the Sequential search algorithm used in the on-line evaluation of EMPC. Sequential search algorithm is the most direct way of determining the index, i of the region, $\mathcal{R}$ in which the current state, x_0 belong to. Once we get an index of the region from (3.13), then the optimal control action is obtained by evaluation the corresponding PWA function (3.12). For the on-line synthesis of explicit MPC, one has to store all the data related to regions and PWA function (i.e., T_i, v_i, L_i and h_i) in the form of LUTs. Algorithm 1 repre-

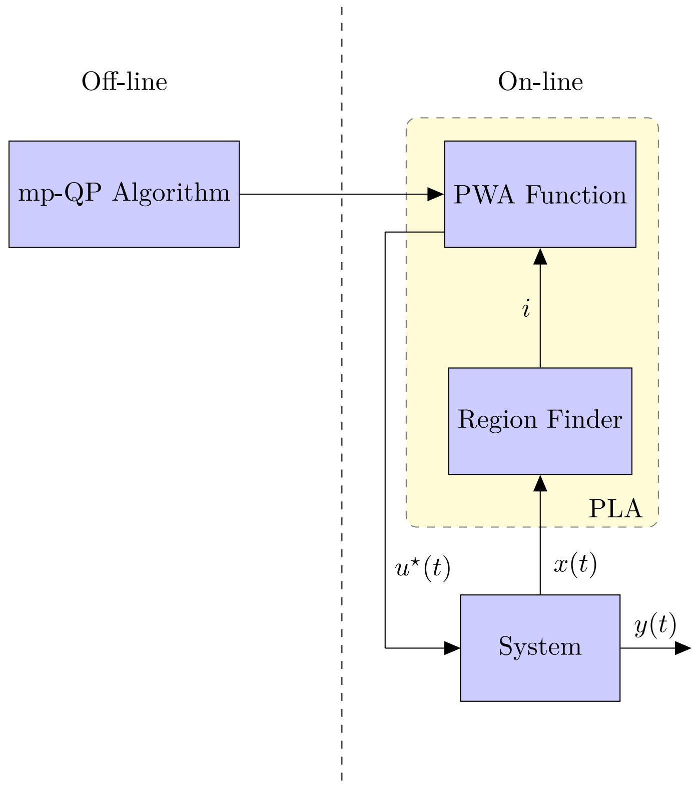

Figure 3.2: The idea of point location algorithm in EMPC.

sents the idea of sequential search. The algorithm traverses sequentially through all

Algorithm 1 Sequential search.

```
INPUT: Regions $\mathcal{R}_i$, feedback laws $T_i$, $v_i$, number of regions $n_{\mathcal{R}}$, initial condition $x_0$.
OUTPUT: $U^\star(x_0)$ solving (3.12) for a given $x_0$.
1: for $i = 1, \dots, n_{\mathcal{R}}$ do
2:     if $L_i x_0 \leq h_i$ then
3:         return $U^\star(x_0) \leftarrow T_i x_0 + v_i$
4:     end if
5: end for
```

the regions of (3.12). At Step (2) the algorithm verifies whether x_0 belongs to the i-th region by checking the inequality representation in (3.3). If $x_0 \in \mathcal{R}_i$, the optimal solution to (3.3) is given by (3.12) upon which the algorithm terminates. The worst-case run-time of the sequential search algorithm is $\mathcal{O}(M)$, i.e., linear in the number of regions. Some other point location algorithms used in the on-line evaluation of EMPC are, extended sequential search (Takács et al., 2016a), Truncated Binary Search Tree (TBST) (Bayat et al., 2011), multi-way trees (Mönnigmann and Kastsian, 2011), etc.

3.4 Advantages and Disadvantages of Explicit MPC

Advantages

- The on-line evaluation of the PWA function is usually faster and simpler compared to solving the on-line QP problem. This is particularly useful in high-bandwidth applications when high control update rates are required.
- On-line synthesis of PWA function needs only addition and multiplication whereas on-line QP needs the inverse of KKT matrix.
- Once the explicit optimal solution has been computed off-line, it can be implemented into simple hardware such as a microchip and can be replicated cheaply for mass production.
- In contrast to the implicit nature of standard MPC implementations, explicit MPC solutions provide a more accurate and deep intuitive understanding of

the control behavior and properties, allowing analysis of performance such as safety verification needed in safety-critical application.

Disadvantages

- Both the computation time and the complexity of the regions grow in the worst case exponentially with the problem size (length of the prediction horizon (N), dimension of the states (n_x) and inputs (n_u), number of constraints due to the combinatorial nature of the problem.
- If the explicit solution can be computed, it can still be highly complex and needs to store all pre-computed data in the memory of the target control hardware, which may prohibit its application due to restricted storage space and on-line computation time.

3.5 Complexity

In the above sections, we have seen the properties of explicit MPC which makes it a suitable controller for real-time applications demanding high control updates rates in milliseconds to microseconds. However, the applicability of this controller is limited to fairly small problems, since the complexity of controller increase more or less exponentially with problem size. Generally, with the complexity of explicit MPC one refers to the number of regions required to construct the corresponding PWA controller (3.12). The number of regions increases exponentially when the problem size grows, and this results in increased off-line computations. The required memory and on-line computations also increase and this, in turn, makes the explicit solutions inefficient for large-scale problems. For the detailed analysis of explicit MPC complexity see, e.g. (Pistikopoulos et al., 2007b, Chapter 1), Borrelli et al. (2009). The complexity of EMPC can be categorized into two factors:

1. Off-line complexity: This complexity is associated with the constructions of critical regions. In other words, as problem size increases the computational burden on mp-QP algorithms increases. Generally, mp-QP algorithm runs on Personal Computer (PC) which typically perform numerical calculations in 64-bit floating-point arithmetic, which is too expensive and power demanding. This restricts the use of EMPC for the system with large number of states,

inputs and large prediction horizon. To handle the issue, Gupta et al. (2011) presented a novel approach which uses an implicit enumeration of active sets. Using this approach, it is possible to extend EMPC for large systems, e.g., distillation column with 82 states (Feller and Johansen, 2013).

2. On-line complexity: By the on-line complexity we mean storage memory and computational time. As it is mentioned above, in the on-line phase of EMPC one has to synthesis PWA function defined over polytopic regions. It is reasonable to say that the on-line complexity of controller depends on a number of regions required to store PWA functions. In other words, we can say that it is number of floating-point numbers which decides memory required to store all the data related to the PWA function, i.e., (T_i, v_i, L_i and h_i). In particular, an explicit PWA function often consumes hundreds of kilobytes (kB) to several megabytes (MB) of memory. Such an amount can easily exceed capabilities of a given hardware implementation platform, especially when the hardware has to accommodate multiple feedback loops. In order to implement an EMPC algorithm on resource constrained embedded hardware, it is therefore of imminent importance to reduce the memory footprint to an acceptable level.

In the literature, various complexity reduction techniques have been presented to make low-memory EMPC controllers that easily fits on embedded devices with several kB of memory. In this thesis, we are mainly focusing on the on-line complexity reduction technique to reduce memory footprints of the EMPC controller for the embedded implementation.

3.5.1 An Overview of Complexity Reduction Techniques

In the last few years, an effort has been made to reduce the complexity of explicit MPC which is mainly focused on two distinct directions: first, how to make the feedback law simple; second, how to reduce the amount of bits required to store LUT data with the prescribed accuracy. Following are the some techniques suggested for low-complexity controller.

Suboptimal Techniques

- Short prediction horizon: The number $n_{\mathcal{R}}$ mainly depends on the number of constraints (n_g), and only mildly on the number of states (n_x). It also depends on the number of optimization variables (n_z), but mainly because n_g depends on n_z. At the price of a possible loss of closed-loop performance, one way of reducing complexity is to shorten the prediction horizons (and/or blocking input moves (Tøndel and Johansen, 2002), (Oldewurtel et al., 2009)) in order to decrease n_z and n_g.

- KKT relaxation: To reduce the number of region, authors Bemporad and Filippi (2003) suggests solution which consists of finding an approximate solution to mpQP by relaxing the KKT conditions for optimality, except primal feasibility.

- Interpolation: With negligible deterioration of performance authors in Rossiter and Grieder (2005) shown that one can reduce on-line effort by a factor of 10 by using interpolation.

- Orthogonal partition: In Johansen and Grancharova (2003), Cychowski and O'Mahony (2005) it is shown that the approximate solution to MPC problem can be pre-computed off-line in an explicit form as a PWA state feedback law defined on an orthogonal partition of the state space. It has shown this approach leads to a real-time computational complexity that is logarithmic in the number of regions in the partition, and the algorithm yields guarantees on the suboptimality, asymptotic stability and constraint fulfillment.

- Model reduction: It has been proposed in Hovland et al. (2008) which reduces number of states and subsequently the number of regions required to store feedback law. But, if the performance obtained by model order reduction approach is low then it one need longer control horizons as compared to horizons needed for the original model to obtain good performance.

- Bilevel optimization: An approximation approach that generates a low-complexity piecewise affine function directly from the optimal MPC formulation (i.e. without computing the optimal solution first) was proposed in Jones and Morari (2009).

- Delaunay tessellation: The concept from computational geometry theory called Delaunay tessellation was used by Scibilia et al. (2009) to approximate PWA controller which allows fast online implementation without the need for any additional support structure.

All the above techniques lead to a simpler PWA controller with the remarkable reduction of controller complexity. However, these techniques lead to suboptimal solutions which are unacceptable in the mission critical application. Therefore, the another direction of research is devoted to simplifying the PWA feedback law while preserving optimality. Following are some performance-lossless complexity reduction techniques suggested for the complexity reduction in EMPC.

Optimal Techniques

- Optimal Region Merging (ORM): For a given PWA function (3.12), the authors in Geyer et al. (2008) provide an approach to reduce the number of partitions by optimally merging regions where the affine gain is the same, so that the original solution is maintained but equivalently expressed with a minimal number of partitions.

- Lattice representation: A general lattice representation for continuous EMPC solutions obtained by the mpQP were proposed in Wen et al. (2009). The main advantage of a lattice expression is, it is a global and compact representation, which automatically removes the redundant parameters in an EMPC solution.

- Saturated region clipping: An idea of eliminating regions of the PWA function over which the function attains a saturated value and replace it by extensions of unsaturated regions was proposed in Kvasnica and Fikar (2012). In this procedure, a new control law is constructed which is simpler and faster than original control law.

- Region separation: The separating functions were employed to define PWA functions in Kvasnica et al. (2011), Kvasnica et al. (2013). If a current state resides in a region where the optimal control action attains a saturated value, the optimal control move is determined from the sign of the separator. Thus, in this approach instead of storing all regions, only the unconstrained regions and the separator was stored, and the complexity of explicit MPC feedback laws is reduced considerably without sacrificing optimality.

- Partial selection: Authors in Kvasnica et al. (2012) shown that a simpler equivalent explicit feedback can be obtained by selecting a particular subset of the controller regions which avoids pre-processing.
- Inner and outer approximation of EMPC solutions: A polygonic representation of regions of the explicit PWA feedback law was presented in Oravec et al. (2013) to reduce the memory footprints of controller. In this approach one needs to store only the outer boundaries of such polygons, a significant amount of memory can be saved. But, this comes at the price of increased computational resources required to perform the point location task.
- Low rank: A method for exploiting low-rank structure in the parametric solution of a multi-parametric programming problem is introduced in Nielsen and Axehill (2016), to reduce the memory footprints.

In all the techniques one can obtain less complex and performance-lossless explicit MPC solution, but the downside is that these techniques are limited to small systems due to the significant pre-computing efforts required to solve non-trivial optimization problems.

A common drawback of all aforementioned approaches (optimal and suboptimal) is, the data of the simplified EMPC feedback law needs to be stored as a floating-point numbers in the IEEE-754 format, 32-bits for single precision and 64-bits for double precision numbers. The bit size of numbers is thus constant regardless of the values they store. Thus, another direction of research is focused on reducing the bit size of underlying controller data (T_i, v_i, L_i and h_i) by using different data representation techniques.

Complexity Reduction Via Data Representation Techniques

- Data de-duplication and Huffman encoding: One way of reducing memory footprints of explicit MPC by exploiting geometric properties is shown in Szücs et al. (2011). They used three-layer procedure, first identifies similarities between polytopic regions in the form of an affine transformation. If such a mapping exists, certain regions can be represented using less data.

The second layer then applies data de-duplication to identify and remove repeating sequences of data (where identical half-spaces are not duplicated in the description of the critical regions). Regions are then described by integer pointers to such a unique set. Finally, Huffman encoding (Knuth, 1985) is applied to compress such integer pointers using prefix-free variable-length bit encoding. The memory reduction comes at the price of having to perform additional computation on-the-fly, amount of which was quantified for each level. Since, there is a one-to-one correspondence between the original data and their compressed counterparts, the compressed feedback law exhibits the same properties (e.g., control performance, closed-loop stability and constraint satisfaction) as the original one.

- Low-precision data representation: An alternative was presented by Suardi et al. (2014) and Suardi et al. (2016) where the authors have proposed the use of low precision arithmetic. However, the use of low precision requires extra effort to guarantee the constraint satisfaction in explicit MPC Suardi et al. (2016). Using this approach one can gain speed and save hardware resources but, at the price of losing controller performance.

Limitations of the Complexity Reduction Techniques

Using above mentioned complexity reduction techniques, one can achieve a drastic reduction in the complexity of controller or speed-up the on-line computations. However, this is achieved at the price of suboptimality, performance deterioration, and in some cases constraint violation. Following are some limitations of complexity reduction techniques for the use in real-time applications running on embedded hardware.

- Approximation techniques leads to the suboptimal solutions which are unacceptable in safety critical applications such as in bio-medical, aerospace, automotive, etc.
- With some techniques performance of the systems reduces with the reduced complexity which is generally not allowed in aerospace applications for example in spacecraft where it is expected to use less fuel with more performance.
- Performance-lossless techniques can overcome above two limitations but downside is that these techniques are limited to small problems and generally in-

dustrial problems are of medium to large. For such problem size the controller needs more memory which is not the case with embedded hardware like PLC, where memory is in kB.

- Generally, all techniques use floating-point representation for data storage and arithmetic operations. It is the fact that floating-point format gives different output on different hardware with round-off, overflow and underflow errors which are not considered in the design of the controller.

- Although significant amount of memory can be saved by performance-lossless techniques, it suffers from the increase of on-line computational complexity needed to evaluate the PWA function.

3.6 Software Tools

There are several software frameworks available for design of explicit MPC control laws for linear and hybrid systems. Following are some popular and well-known toolboxes used in EMPC design and export of on-line algorithms in different low level languages such as C.

- Multi-Parametric Toolbox (MPT): It is a freely available MATLAB based toolbox which implements state of the art numerical solvers for solving EMPC problems. MPT contains easy to use interfaces for modeling, control design, computation, analysis, and post-processing of optimal controllers in an explicit form. The obtained solution can be exported as a stand-alone lookup table to the C programming language and compiled on a target application. It can be downloaded from the link:
  `http://people.ee.ethz.ch/~mpt/3/`.
  Some complexity reduction techniques are available in MPT and demonstrated in Kvasnica et al. (2015). More details can be found in Kvasnica et al. (2004). Herceg et al. (2013b)

- Hybrid Toolbox: It is a MATLAB based toolbox for modeling, simulating, and verifying hybrid dynamical systems, for designing and simulating model predictive controllers for hybrid systems subject to constraints, and for generating linear and hybrid MPC control laws in a piecewise affine form that can

be directly embedded as C code in real-time applications. It can be downloaded from following link
`http://cse.lab.imtlucca.it/~bemporad/hybrid/toolbox/`.

- Parametric Optimization Toolbox (POP): It is a MATLAB based state-of-the-art multi-parametric programming solver for continuous and mixed-integer problems. It supports a comprehensive problem library featuring an ever-increasing number of example problems. It can be downloaded from the following link
`http://parametric.tamu.edu/POP/`.

- MOBY-DIC: It is a MATLAB toolbox for the integrated design of MPC state-feedback control laws and the digital circuits implementing them on FPGAs. Explicit MPC laws can be designed using optimal and suboptimal formulations, directly taking into account the specifications of the digital circuit implementing the control law, together with the usual control specifications. Tools for a-posteriori stability analysis of the closed-loop system, and for the simulation of the circuit in Simulink, is also included in the toolbox. It can be downloaded from the following link
`http://ncas.dibe.unige.it/software/MOBY-DIC_Toolbox/`

- Model Predictive Control Toolbox: Explicit MPC can be designed using MATLAB MPC Toolbox is available at `https://www.mathworks.com/`.

3.7 Problem Statement

Explicit MPC is the attractive solution for the real-time applications demanding high update rates. The main bottleneck in implementation of EMPC for real-time applications are the controller complexity expressed in the form of a number of regions. Storing controller data requires larger and larger memory blocks in the hardware implementation as the number of regions grows. Needless to say, unless all pre-computed data can be fit into memory, the controller cannot be implemented in practice. Therefore, it is of imminent importance to keep the memory footprint of the controller on an acceptable level defined by any hardware. Therefore, in order to deploy EMPC controller on embedded hardware without losing optimality and closed-loop performance accuracy, we intended to represent controller data in a more efficient data format which requires less number of bits to store the same information as floating-point format stores. Before going into the solution, we will see how many floating-point numbers do we need to store for a controller.

3.7.1 Memory Calculations

The total memory footprint of the explicit solution in (3.12) expressed in floating-point numbers can be compactly given by

$$\mathcal{S}(\kappa) = n_{\mathcal{R}} n_u N(n_x + 1) + \sum_{i=1}^{n_{\mathcal{R}}} c_i(n_x + 1), \tag{3.14}$$

where the first part represents the size of all T_i, v_i pairs[1] (which have constant dimensions for all regions), and the second part represents the memory footprints of all polyhedral regions (with c_i being the number of half-spaces[2] of the i-th region). The bit size of κ is then $\mathcal{B}(\kappa) = b\mathcal{S}(\kappa)$ where either $b = 32$ or $b = 64$, depending on whether single precision or double precision floating-point numbers are used.

3.7.2 Solution

Our objective is, to reduce the bit size of a given explicit optimizer in (3.12) by devising a more memory efficient representation of floating-point numbers contained in the real-valued vectors/matrices L_i, hi, T_i, v_i. This will be achieved by representing each floating-point number as a *universal number (unum)* with a variable size of its bit code. In other words, instead of using a constant value of b as the bit size of each floating-point number, each such number is represented by b_j bits with $b_j \leq b$. This needs to be done in such a way that the variable-sized bit codes encode the same information as fixed-size floating-point numbers, i.e., the encoding/decoding is done in a *performance-lossless* fashion.

The key idea of unums is to store a real number with a variable bit length format using six sub-fields: the sign bit, exponent, fraction, uncertainty bit, exponent size, and fraction size. Basically, unum is a superset of IEEE-754 floating-point formats (Muller et al., 2009) that tracks whether a number is an exact float or lies

[1]In fact, if the MPC controller is implemented in a receding horizon fashion, then only the first component of $U^\star$, i.e., $u_0^\star$, is required. In such case, the matrices T_i, v_i can be truncated to just the first n_u rows and the first part in (3.14) becomes $n_{\mathcal{R}} n_u(n_x + 1)$.

[2]At this point we assume that only the non-redundant half-spaces are stored, i.e., $c_i \leq n_g$ where n_g is the number of constraints in (3.4b).

in an open interval between two exact floats. Compared to the standard floating-point formats; the variable size in unum offers an ability to change its representative range and precision, and the uncertainty bit indicates the exactness of the value represented. Thus, unums use fewer bits, obey algebraic laws, and do not require rounding, overflow, and underflow for proper operations Gustafson (2015). Flexible dynamic range and precision allow one to avoid fixed bit size data format. Generally, control engineers choose either single (32-bit) or double (64-bit) precision data formats to get higher accuracy and precision. However, this typically wastes storage and bandwidth by a factor of two or more since the maximum precision is only needed for some fraction of the calculations. The base of the unum format is floating-point formats, therefore the next chapter is devoted to the floating-point numbers.

3.8 Summary

This chapter has presented the concept of explicit model predictive control which is an attractive approach to overcome the limitations of on-line optimization problems. The idea of explicit MPC is to pre-compute a constrained optimization problem considering the current state of the plant as a parameter. This is so-called explicit solution yields a look-up table of optimal feedback gain matrices. Then on-line synthesis of controller boils down to a point location problem (see Section 3.3) which can be solved very quickly on low-cost and low-end embedded hardware. The main advantage of this approach is to simplify the on-line computational burden to off-line. However, this approach suffers from the curse of dimensionality. The number of region, computed by the mp-QP algorithm, increases exponentially with the number of constraints, length of prediction horizon and is highly sensitive to the dimension of parameter space. Several efforts has been made to reduce the computational complexity of explicit MPC, the brief summary of many of memory reduction techniques is given in Section 3.5. The computational complexity of explicit MPC can be reduced by approximating the controller or by reducing the number of bits to store controller data. Existing approximation techniques turns to reduce complexity but it gives suboptimal controller. Generally, controller data is stored in the form of IEEE floating-point number standard which takes the same number of bits to store all the numbers irrespective of the value. So, this thesis focuses on reducing the number of bits for store controller data. We propose to use universal number format to design low-memory explicit MPC (see Section 3.7.2). The back bone of universal number format is IEEE floating-point standard therefore next chapter presents number systems and floating-point format.

Chapter 4

Number System

The focus of this chapter is, representation of data in an embedded devices such as FPGAs, PLCs, micro-controllers, and DSPs. We begin with a brief introduction to fixed-point and floating-point number systems and a discussion of floating-point arithmetic.

Representing and manipulating real numbers efficiently is required in many fields of science, engineering, finance, and more. Nowadays, in all these fields computers are used to make calculations. Computers were originally built as fast, reliable, and accurate computing machines. It does not matter how large computers get, one of their main tasks will be to always perform the computation. The history of computer arithmetic is always intertwined with that of digital computers. In all microprocessor based computers, the Arithmetic Logic Unit (ALU) is one of the units that takes most of the design effort to introduce more accurate and faster techniques. Most of these computations need real numbers as an essential category in any real-world calculations. Real numbers are not finite; therefore no finite representation method is capable of representing all real numbers, even within a small range. Thus, most real values will have to be represented in an approximate manner. To be able to finish calculations in a reasonably short time, a computer must limit the numbers it may represent for all number types. For an integer, this limits the maximum value it can take and for a real number it limits both the maximum value and the precision of the numbers it may represent. While this technically makes a real number to a rational number, which may be represented by two integers such a scheme is not efficient, and there are better ways to represent these rounded reals. Following are some number systems used to represent real value.

4.1 Fixed-Point Number System

The finite-word representation of fractional numbers is known as fixed-point. A fixed-point representation of a number consists of an integer and fractional components. In the system of fixed-point, a real number is represented as follows

$$x = x_{n_t - n_i - 1}, \ldots, x_1, x_0 \ . \ x_{-1}, x_{-2}, \ldots, x_{-n_f}, \tag{4.1}$$

where n_i is the number of digits to the right of the decimal point or integers, n_f is the number of digits to the left or fraction and n_t is the total number of digits used to represent a real number. In the actual practical implementation, the decimal point is not stored physically. Instead, a particular choice of base, $\bar{b}$, e.g., (2 or 10) is adopted where the real number corresponding to the above representation (without the point) is given by $\frac{x_i}{\bar{b}^{n_i}}$, where x_i is the integer represented by the same sequence of digits without a point.

Representing sign numbers in binary fixed-point is normally done using 2's complement notation for easy addition and subtraction. The value, x, of a fixed-point number is given by

$$x = -s\bar{b}^{n_i} + \sum_{j=0}^{n_t - 1} x_i \bar{b}^{j - n_f}, \tag{4.2}$$

where s is the sign of the number. In this thesis, we use different colors to specify various bits in a number presentation. There are two parameters go with every computer-based system used in manipulating real numbers. The first parameter is the possible range of the values that can be represented. In the fixed-point system, the range of real numbers that can be represented is given as follows

$$\left(\frac{x_{\max}}{\bar{b}^{n_i}}, \frac{x_{\min}}{\bar{b}^{n_i}}\right), \tag{4.3}$$

where $x_{\min}$ and $x_{\max}$ are integers whose values depend on the representation scheme. If a sign-magnitude is being adopted to represent the integer x, the range would be given respectively by

$$\left(\frac{1 - \bar{b}^{n_t - 1}}{\bar{b}^{n_i}}, \frac{\bar{b}^{n_t - 1} - 1}{\bar{b}^{n_i}}\right). \tag{4.4}$$

If on the other hand, a $\bar{b}$'s complement representation is used, the range of values is given by

$$\Big(\frac{\bar{b}^{n_t}/2}{\bar{b}^{n_i}}, \frac{\bar{b}^{n_t}/2-1}{\bar{b}^{n_i}}\Big). \tag{4.5}$$

The other parameter necessary to characterize a number representation system is related to the accuracy of the system. This parameter refers to the Unit of Least Precision (ULP).

Definition 4.1.1. (ULP). ULP is a difference between exact values represented by bit strings that differ by one unit in the last significant position, the last bit of the fraction.

The maximum error that can be incurred as a result of representing a real number with infinite precision in a finite-precision machine is $ULP/2$. For the system of fixed-point, ULP is given by $1/\bar{b}^{n_t}$, thus making the maximum error ($e_{\max}$) associated with that system equal to

$$e_{\max} = \frac{1/\bar{b}^{n_t}}{2}. \tag{4.6}$$

Another related parameter is the maximum relative error ($re_{\max}$) which is scaled by the absolute value of x_i, as shown next

$$re_{\max} = \frac{1}{(2 \times |x_i| \times \bar{b}^{n_t})} \leq 0.5. \tag{4.7}$$

The fixed-point number is defined by its format n_t, n_i, n_f or its properties range, resolution, and bias. For detailed description and fixed-point arithmetic see, e.g., Padgett and Anderson (2009), Parhami (1999). In this thesis, we are dealing with base 2 so, in all the examples we will consider $\bar{b} = 2$.

4.1.1 Examples

Example 1: Represent a real number 2.75 in to 8-bit fixed-point format with 3 fraction bits.

Solution: To store value 2.75 in given bits we will need 8-bits out-off that 3-bits will be reserved for a fraction, the steps to represent a number in the fixed-point format are:

Step 1: Represent number in binary format, i.e., 10.11_2

Step 2: Insert above bits into 8-bit format with 3-bits for fraction, i.e. 00010.110_2.

Example 2: Represent a real number -2.75 in to 10-bit fixed-point format with 5 fraction bits.
Solution: To store value -2.75 in given bits we will need 10-bits out-off that 5-bits will be reserved for fraction, as this is a negative number we will need to store the steps to represent number in the fixed-point format are
Step 1: Represent number in binary format, i.e. -10.11_2,
Step 2: Insert above bits into 10-bit format with 5-bits for a fraction, i.e., -00010.11000_2,
Step 3: Represent number in 2's complement form, i.e., 11101.00111_2.

4.1.2 Advantages

- Arithmetic and logical operations may be performed on fixed-point numbers using integer arithmetic.
- A fixed-point number representation can use less memory to store values.

4.1.3 Disadvantages

- It is easy for an arithmetic operation to produce an "overflow" or "underflow".
- A fixed-point number has a limited integer range. It is not possible to represent very large and very small numbers with the same representation.
- A fixed-point number has limited accuracy. You must choose the representation which will best suit your needs.

Definition 4.1.2. (Overflow). Overflow occurs when the result of an arithmetic operation is too large to fit into the given representation of a real number.

Definition 4.1.3. (Underflow). Underflow occurs when the result of an arithmetic operation is too small to fit into the given representation of a real number.

4.2 Floating-Point Number System

Floating-Point (FP) arithmetic is a popular method of implementing real numbers on computers and embedded devices. The binary point is not fixed and can be placed anywhere concerning the significand digits of the number. The main idea behind the floating-point representation is, only a fixed number of bits are used and the binary point “floats” to wherever it is needed in those bits. As a matter of fact, the computer only holds bit patterns. Floating-point expressions can represent a wide span of numbers. When a floating-point calculation is performed, the binary point floats to the correct position in the result. But the floating-point format requires slightly larger storage to represent the binary point. The speed of floating-point operations is one of the important figures of merit for computers in many application domains. It is usually measured in Floating-Point Operations Per Second (FLOPS).In general, real numbers are represented approximately by a fixed number of significand digits and scaled using an exponent. The base for scaling is normally 2, 10 or 16. The typical number that can be represented exactly is of the form:

$$m \times \bar{b}^{e}, \tag{4.8}$$

where m is the mantissa or significand which is a non-negative number represented by fixed-point form and e is the exponent. A typical floating-point number representation is composed of four main components: the sign (s), the mantissa (m), the exponent base ($\bar{b}$), the exponent (e). The exponent base ($\bar{b}$) is usually implied (not explicitly represented) and is usually a power of 2, except of course in the case of decimal arithmetic. A floating-point number x has the value

$$x = (-1)^{s} \times (2)^{e} \times (1.m). \tag{4.9}$$

The mantissa (m) is made up of the leading “1” and the fraction, where the leading “1” is implied in hardware. This means that for computations that produce a leading “0”, the fraction must be shifted. The only exception for a leading one is for gradual underflow. The exponent is usually kept in a biased format, where the value of e is

$$e = e_{\text{true}} + bias, \tag{4.10}$$

where e_{true} is the true exponent and e is the biased exponent with following bias

$$bias = 2^{es-1} - 1, \tag{4.11}$$

where es is the size of the exponent (in bits). This is done to make comparisons of floating-point numbers easier. The main point to observe is that there are two signs involved in a floating-point number

1. The number sign indicates a positive or negative floating-point number and is usually represented by a separate sign bit (signed-magnitude convention).

2. The exponent sign is embedded in the exponent and it indicates mainly a large or small number. If the bias is a power of 2, the exponent is the complement of its Most Significant Bit (MSB).

The use of biased exponent format has virtually no effect on the speed or cost of exponent arithmetic (addition/subtraction), given the small number of bits involved. It does, however, facilitate zero detection (zero will be represented with the smallest biased exponent of 0 and all-zero significant) and magnitude comparison (we compare normalized floating-point numbers as if they were integers).

4.3 The Floating-point Standard

4.3.1 History

The current standard on floating-point numbers is called IEEE Std 754−1985 IEEE Standard for binary floating-point arithmetic and is probably the most popular standard that the IEEE ever published. As far as most young programmers and hardware designers are concerned, IEEE 754 numbers are the only floating-point numbers that ever existed. But floating-point arithmetic has existed on commercial computers since the 1950's and has a long history of inconsistent behavior as well as producing different results on different computer types.

In 1976 Intel, long before it becomes a household name, decides that they want to create the best floating-point arithmetic on a single chip based on the popular VAX minicomputer by Digital Equipment Corporation (DEC). Their competitors around the Silicon Valley begin to hear rumors about this chip and form the IEEE 754 working group in 1977. Intel joins the group and discloses a part of their specification mainly the representation formats and the basic arithmetic operations along with some of the reasoning behind it. The standard process takes years mainly due to a battle over gradual underflow which specifies how numbers between the smallest representable number and zero should be handled. DEC believes that gradual underflow is infeasible with current technology, if not impossible, but a Berkeley student proves them wrong by designing a floating-point unit for the VAX that handles gradual underflow. IEEE 754 is released in 1985 and today only old Macs with Motorola 68 K processors lack IEEE 754 support (Kahan, 1996).

Every IEEE Standard has a lifetime of five years, whereafter it must reaffirm. IEEE 754 has been reaffirmed twice (Overton, 2001), (Muller et al., 2009), but in 2001 it was determined to be so out-of-date that a revision was due. One of the major changes in the revision is the incorporation of the lesser known IEEE Std 854 – 1987 IEEE Standard for radix-independent floating-point arithmetic (IEEE 854) which is the decimal version of IEEE 754. Other changes include the Fused Multiplication and Addition (FMA) operation which is common in modern floating-point units, representations of floating-point numbers in 16-bit, 128-bit and higher in addition to the 32-bit and 64-bit, a new list of recommended operations, as well as resolution of some ambiguities. Finally, the revision emphasizes that an implementation of the standard may be designed hardware, written in software or be a combination of both. As in the initial standardization process which took eight years, progress is slow and despite seven years of work, the standard has not been published at the time of writing. Fortunately, the public drafts have been publicly available and quite stable, so there has been considerable research on implementing arithmetic operations with the new decimal formats. In 2008, the revised format was published by IEEESA standards board (IEEE Std 2008, 2008). In the literature, IEEE 754 – 1985 and its new revision are frequently called IEEE 754 and IEEE 754−R, respectively.

For the more detailed description about standard see, e.g., ANSI/IEEE Std 1985 (1985), Overton (2001), IEEE Std 2008 (2008), Muller et al. (2009).

4.3.2 Formats Defined In Standard

The IEEE standard defines, in fact, four floating-point formats in binary form, half-precision (16-bits), single precision (32-bits), double precision (64-bits), and quadruple precision (128-bits). As mentioned above, every format is composed of three fields: sign(s), exponent (e), and mantissa (m). Fig. 4.1 shows the representation of half precision format. Table 4.1 shows the parameters each format like number of total number of bits, bits in each field, and values of true and biased exponents.

Half Precision

This is relatively new as a standard format with a width of 16-bits. It was promoted starting around 2002 by graphics and motion picture companies as a format that could store visual quantities like light intensity, with a reasonable coverage of the accuracy and dynamic range of the human eye. Only recently chip designers have started to support the format as a way to store numbers, and existing processors do not perform native arithmetic in half precision; they promote to single precision internally and then demote the final result. The real value of a number x assumed by a given 16-bit data with a given biased sign, exponent, and mantissa is

$$x = (-1)^{s} \times \left(1 + \sum_{j=1}^{10} x_{10-j} 2^{-j}\right) \times 2^{(e-15)}. \tag{4.12}$$

Single Precision

Single precision binary floating-point is used due to its wider range over fixed-point (of the same bit-width), even if at the cost of precision. The real value of a number x assumed by a given 32-bit data with a given biased sign, exponent (the 8-bit unsigned integer), and a 23-bit mantissa is

$$x = (-1)^{s} \times \left(1 + \sum_{j=1}^{23} x_{23-j} 2^{-j}\right) \times 2^{(e-127)}. \tag{4.13}$$

Single precision floats are often used in imaging and audio applications (medical scans, seismic measurements, video games) where their accuracy and range is more than sufficient.

Double Precision

This gives 15˘17 significant decimal digits precision. If a decimal string with at most 15 significant digits is converted to IEEE 754 double precision representation and then converted back to a string with the same number of significant digits, then the final string should match the original. If an IEEE 754 double precision is converted to a decimal string with at least 17 significant digits and then converted back to double, then the final number must match the original. The real value assumed by a given 64-bit double precision data with a given biased exponent and a 52-bit mantissa is

$$x = (-1)^{s} \times \left(1 + \sum_{j=1}^{52} x_{52-j} 2^{-j}\right) \times 2^{(e-1023)}. \tag{4.14}$$

Double precision and single precision are the most common sizes built into processor chips as fast, hard-wired data types.

Quadruple Precision

This precision is mainly available through software libraries, with no mainstream commercial processor chips designed to do native arithmetic on such large bit strings. The most common demand for quad precision is when a programmer has discovered that the double precision result is very different from the single precision result, and wants assurance that the double precision result is adequate. Because quad precision arithmetic is typically executed with software instead of native hardware, it is about twenty times slower than double precision. Despite its impressive accuracy and dynamic range, quad precision is every bit as capable of making disastrous mathematical errors as its lower-precision kin. And, of course, it is even more wasteful of memory, bandwidth, energy, and power than double precision. In this format, the value of real number can be obtained by

$$x = (-1)^{s} \times \left(1 + \sum_{j=1}^{112} x_{112-j} 2^{-j}\right) \times 2^{(e-16383)}. \tag{4.15}$$

4.3.3 Ranges of FP Numbers

Let's consider single-precision FP for a second. Note that we're taking essentially a 64-bit number and reinterpreting the fields to cover a much broader range. Something has to give, and it's precision. For example, regular 32-bit integers, with all precision centered around zero, can precisely store integers with 32-bits of resolution. Single-precision floating-point, on the other hand, is unable to match this resolution with its 24-bits. It does, however, approximate this value by effectively truncating from the lower end and rounding up. Consider a binary number in 32 bit integer,

11110000 11001100 10101010 10101111

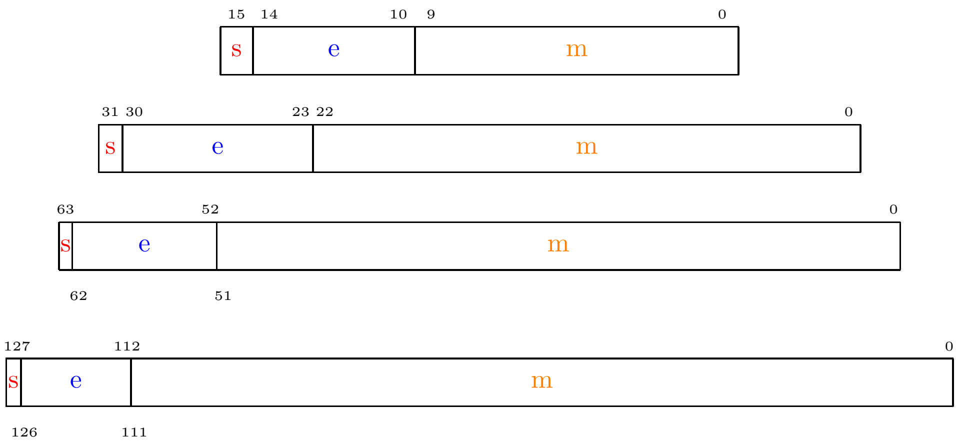

Figure 4.1: Representation of IEEE 754 half, single, double, and quad precision floating-point format with three sub-fields.

in single precision FP it is given as

$$1.11100001100110010101011 \times 2^{31}$$

which is exactly equal to

$$11110000\ 11001100\ 10101011\ 00000000$$

This approximates the 32-bit value, but doesn't yield an exact representation. On the other hand, besides the ability to represent fractional components (which integers lack completely), the floating-point value can represent numbers around 2^{127}, compared to 32-bit integers maximum value around 2^{32}.

The range of positive floating-point numbers can be split into normalized numbers (which preserve the full precision of the mantissa), and denormalized numbers which use only a portion of the mantissa's precision. Table 4.2 summarizes the ranges of all four FP formats for normalized numbers.

4.3.4 Special Values

There are several theorems and special cases which needed special attention for floating-point numbers Muller et al. (2009). Such cases are documented in great detail in the literature leaving no room for confusion. Even small changes in the

Table 4.1: Main parameters of the formats specified by the IEEE 754 standard.

Parameter/Format	**Half**	**Single**	**Double**	**Quad**
Width	16	32	64	128
sign	1	1	1	1
exponent	5	8	11	15
mantissa	8	23	52	112
exponent min	−14	−126	−1022	−16382
exponent max	15	127	1023	16383
bias	15	127	1023	16383
accuracy (decimals)	3	7	15	34

calculation of the final value can have disastrous effects with respect to the correct result. The floating-point representation of zero and other special numbers will be discussed below.

Normalized

As said before, they are represented by (4.9).

Denormalized

If all the bits of the exponent are 0 but the mantissa is non-zero (else it would be interpreted as a zero), then the number is a denormalized number, which does not have a hidden bit (1.) before binary point. It can be represented as follows:

$$x = (-1)^{s} \times (2)^{e_{\min}} \times (0.m). \tag{4.16}$$

From this it can be seen that 0 is a special case of denormalized number.

Zero

According to IEEE 754 standard floating-point format it provides signed zeros, that "+0" and "0". The exponent and mantissa bits are zero. Computation of

Table 4.2: Main parameters of the formats specified by the IEEE 754 standard.

Ranges/Format	**Normalized**	**Approximate Decimal**
Half	$\pm 2^{-14}$to$(2 - 2^{-10}) \times 2^{15}$	$\pm \approx 6.1 \times 10^{-5}$ to $\approx 6.5504 \times 10^{4}$
Single	$\pm 2^{-126}$ to $(2 - 2^{-23}) \times 2^{127}$	$\pm \approx 1.175 \times 10^{-38}$ to $\approx 3.403 \times 10^{38}$
Double	$\pm 2^{-1022}$ to $(2 - 2^{-52}) \times 2^{1023}$	$\pm \approx 2.225 \times 10^{-308}$ to $\approx 1.798 \times 10^{308}$
Quad	$\pm 2^{-16382}$ to $(2 - 2^{-63}) \times 2^{16383}$	$\pm \approx 3.362 \times 10^{-4932}$ to $\approx 1.190 \times 10^{4932}$

Table 4.3: IEEE 754 Standard Special Values.

Exponent	**Mantissa**	**Represents**
$e = e_{\min} - 1$	$m = 0$	± 0
$e = e_{\min} - 1$	$m \neq 0$	$0.m \times 2^{e_{\min}}$
$e_{\min} \leq e \leq e_{\max}$	-	$1.m \times 2^{e}$
$e = e_{\min} - 1$	$m = 0$	± 0
$e = e_{\max} + 1$	$m = 0$	$\pm\infty$
$e = e_{\max} + 1$	$m \neq 0$	NaN

division might require signed zeros. A division operation performed with positive zero results in positive infinity and the same operation with negative zero results in negative infinity. Divide by zero operation will be handled by the exception handler (discussed in the next section).

Not a Number (NaN)

The standard defines two types of NaNs;

1. signaling NaNs (sNaNs) do not appear, in default mode, as the result of arithmetic operations. They signal the invalid operation exception whenever they appear as operands. For instance, they can be used for uninitialized variables;

2. quiet NaNs (qNaNs) propagate through almost all operations without signaling exceptions. They can be used for debugging and diagnostic purposes. As stated above, a quiet NaN is returned whenever an invalid operation exception occurs with the corresponding trap disabled.

Infinity

The values $+\infty$ and $-\infty$ are denoted with all the exponent bits equal to 1 and mantissa bits equal to 0. The sign bit decides whether it is $+\infty$ or $-\infty$. Being able to denote ∞ as a specific value is useful because it allows operations to continue pass overflow situation. The IEEE standard specifies the following special values (see Table 4.3): ± 0, denormalized numbers, $\pm\infty$ and NaNs. These special values are all encoded with exponents of either $e_{\max} + 1$ or $e_{\min}$.

Table 4.4: Results of Special Operations.

Operation	Result
$x \div \pm\infty$	0
$\pm\infty \times \pm\infty$	$\pm\infty$
$\pm x(\neq 0) \times \pm 0$	$\pm\infty$
$\pm x(\text{finite}) \times \pm\infty$	$\pm\infty$
$\infty + \infty$	$+\infty$
$\infty - -\infty$	$+\infty$
$-\infty - \infty$	$-\infty$
$-\infty - +\infty$	$-\infty$
$\infty - \infty$	NaN
$-\infty + \infty$	NaN
$\pm 0 \div \pm 0$	NaN
$\pm\infty \div \pm\infty$	NaN
$\pm\infty \times 0$	NaN

4.3.5 Special Operations

Operations on special numbers are well-defined by IEEE. In the simplest case, any operation with a NaN yields a NaN result. Other operations are as shown in the Table 4.4:

4.3.6 Exceptions

The standard states five exceptions (invalid, division by zero, overflow, underflow, inexact) must be signaled when detected. This can be done by taking a trap (discussed below) or by setting a status flag. The default mode is not to use traps.

For each type of exception, a status flag must be provided: that status flag is set each time the corresponding exception occurs and no corresponding trap occurs. The status flags are "sticky", so that the user does not need to check them immediately, but after some sequence of operations, such as at the end of a function. A system that is compliant with the standard must provide the user with ways of resetting, testing, and altering the flags individually. The standard also recommends (yet does not require) that the user should be able to save and restore all the flags simultaneously.

Traps

The standard allows the user to choose what should be done, when one of the five exceptions occurs by specifying a trap handler for that exception. He can choose to disable, save, or restore an existing trap.

- When a trap is disabled, the corresponding exception is handled according to the default mode.
- When an exception is signaled and the corresponding trap handler is enabled, the trap handler is activated. In some cases, a result is delivered to the trap handler.

Now, we discuss the various cases that lead to an exception in the IEEE standard.

Divide by Zero

When computing $x/0$, if x is a nonzero finite number, the division by zero exception is signaled. If no trap occurs, the result is infinity, with the correct sign.

Overflow

Let us call an intermediate result what would have been the rounded result if the exponent range was unbounded. The overflow exception is signaled when the absolute value of the intermediate result is strictly larger than the largest finite number,

$$x_{\max} = (2 - 2^{e}) \times 2^{e_{\max}}, \tag{4.17}$$

or, equivalently, when the exponent of the intermediate result is strictly larger than $e_{\max}$. When there is an overflow and no trap occurs, the returned result depends on the rounding modes:

- it will be $\pm\infty$ with the round-to-nearest mode, with the sign of the intermediate result;
- it will be $\pm x_{\max}$ with the round-toward-zero mode, with the sign of the intermediate result;
- it will be $+x_{\max}$ for a positive intermediate result and $-\infty$ for a negative one with the round-toward $-\infty$ mode;

- it will be $+x_{\max}$ for a negative intermediate result and $+\infty$ for a positive one with the round-toward $+\infty$ mode.

Underflow

When a nonzero result of absolute value less than $2^{e_{\min}}$ is obtained (i.e., it is in the subnormal range), a significant loss of accuracy may occur. And yet, sometimes, such a result is exact. To warn the user when an inaccurate very small result is computed, the standard defines two events: tininess (a nonzero result of absolute value less than $2^{e_{\min}}$ is obtained), and loss of accuracy. Concerning the detection of tininess, there is some ambiguity in the standard, (see, Muller et al. (2009)):

- tininess can be signaled either before rounding, that is, when the absolute value of the exact result is nonzero and strictly less than $2^{e_{\min}}$;
- or it can be signaled after rounding, that is, when the absolute value of the nonzero result rounded as if the exponent range were unbounded is strictly less than $2^{e_{\min}}$.

Also, loss of accuracy may be detected either when the result differs from what would have been obtained were exponent range unbounded, or when it differs from what would have been obtained were exponent range and precision unbounded. If an underflow trap is not implemented or is not enabled (which is the default), the result is always correctly rounded and underflow is signaled only when both tininess and loss of accuracy have been detected. When a trap has been implemented and is enabled, underflow is signaled when tininess is detected.

Invalid operation

The invalid operation exception is signaled:

- when one of the operands is a signaling NaN;
- when performing one of the following additions/subtractions: $(-\infty)-(-\infty)$, $(+\infty)-(+\infty)$, $(-\infty)+(+\infty)$, $(+\infty)+(-\infty)$;
- when performing multiplications of the form $(\pm 0) \times (\pm\infty)$;
- when performing divisions of the form $(\pm 0)/(\pm 0)$ or $(\pm\infty)/(\pm\infty)$;

- when computing remainder (a, b), where $a = \pm 0$ or $b = \pm\infty$;
- when computing $\sqrt{a}$ with $a < 0$;
- when converting a floating-point number to an integer or a decimal format when there is no satisfactory way of representing it in the target format. This can happen in case of overflow, or for converting infinity or NaN if the target format does not have representations for such data;
- when performing comparisons of unordered operands using predicates that are listed as invalid if unordered.

If the exception occurs without a trap, the returned result will be a quiet NaN.

Inexact

If the result of an operation (after rounding) is not exact, or if it overflows without an overflow trap, then the inexact exception is signaled. The correctly rounded or overflowed result is returned (to the destination or to the trap handler, depending on whether an inexact trap in enabled or not).

4.3.7 Rounding

In the IEEE standard, rounding occurs whenever an operation has a result that is not exact, since (except of binary decimal conversion) each operation is computed exactly and then rounded. By default, rounding means round toward nearest. The standard requires that three other rounding modes be provided, namely round toward 0, round toward $+\infty$, and round toward $-\infty$. For the round-to-nearest mode, two special rules are worth mentioning: the way numbers larger than the largest finite floating-point number are handled, and the way numbers exactly halfway between two consecutive floating-point numbers are rounded. More precisely, in round-to-nearest mode:

- some absolute value larger than or equal to $(2 - 2^{-(e+1)}) \times 2^{e_{\max}}$ will be rounded to ∞ (with the appropriate sign).
- other numbers will be rounded to the nearest floating-point number of the format under consideration. In case of a tie, the floating-point value whose last significand bit is a zero will be returned. Because of this, that rounding mode is frequently called round to nearest even.

When used with the convert to integer operation, round toward $-\infty$ causes the convert to become the floor function, while round toward $+\infty$ is ceiling. The rounding mode affects overflow, because when rounding toward 0 or round toward $-\infty$ is in effect, an overflow of positive magnitude causes the default result to be the largest representable number, not $+\infty$. Similarly, overflows of negative magnitude will produce the largest negative number when rounding toward $+\infty$ or round toward 0 is in effect.

4.3.8 Flag

The IEEE standard has a number of flags and modes. As discussed above, there is one status flag for each of the five exceptions: underflow, overflow, division by zero, invalid operation and inexact. There are four rounding modes: round toward nearest, round toward $+\infty$, round toward 0, and round toward $-\infty$. It is strongly recommended that there be an enable mode bit for each of the five exceptions.

4.4 FP Arithmetic

Now that number representation has been discussed; we can focus on arithmetic. Floating-point arithmetic is considerably more complex than integer arithmetic. We will limit our discussion to the four most basic floating-point arithmetic operations: addition/subtraction, multiplication, and division. The objective is not to provide the most efficient algorithms or give an exhaustive overview of all floating-point arithmetic, but rather to show the complexity involved in computations with floating-point numbers. For more detailed description see, e.g., Sites (2008), Behrooz (2000), Swartzlander (2015).

4.4.1 Addition/Subtraction

In contrast to integer arithmetic, addition, and subtraction are more complicated than multiplication and division. Assuming that the operands are already in the IEEE 754 format, performing floating-point addition. Let $a = s_a, e_a, m_a$ and $b = s_b, e_b, m_b$ represent two floating-point numbers. The change from fixed to floating-point turns the simple addition into a 10 step process:

1. Convert to internal representation

2. Find exponent difference
3. Swap the mantissas
4. Align mantissas
5. Add/subtract mantissas
6. Detect leading one
7. Normalize result
8. Round
9. Adjust exponent
10. Convert to IEEE-754 format

First, the exponents are compared and the difference between them is found. Based on this the mantissas are swapped so that the first operand is larger one and the second is the smaller one. The mantissas are then aligned so that they have the same exponent. The preferred exponent is $\max(e_a, e_b)$ for binary. The operands are then added or subtracted depending on the Effective addition/subtraction Operation (EOP).

While the mantissas are unsigned, the floating-point numbers are not, thus an addition of the mantissas must handle an effective subtraction of the unsigned mantissas as well. As subtraction is included in the internals of floating-point addition, all floating-point adders also take an operation signal indicating addition or subtraction that together with the sign bits determines the EOP.

After the addition/subtraction the significand may not fit in the format anymore as it may be a digit too large, or it may be unnormalized. A Leading One Detector (LOD) finds the position of first non-zero digit and a shifter normalizes the result to a format obeying the input format. The exponent is updated with the shift. After normalization inexact results must be rounded according to the current rounding mode as explained in Section 4.3.7. As the rounding may overflow the significand has to be normalized again and the exponent must be updated accordingly. Finally, the internal format must be converted to an IEEE 754 interchange format if it is possible. Otherwise, as special value must be chosen.

4.4.2 Multiplication

Floating-point multiplication and division are much simpler than addition/subtraction. In total it is a 7 step process where step 2 to 4 conceptually differ from addition:

1. Convert to internal representation
2. Multiply mantissas and add exponents in parallel
3. Detect leading one
4. Normalize result
5. Round
6. Adjust exponent
7. Convert to IEEE-754 format

The multiplier is a normal fixed-point multiplier that emits a result with a width that is twice the format. The result must be normalized since it is twice as wide as the IEEE 754 interchange format permits. Binary multiplication only needs to examine a single digit. The remaining steps are the same as for addition.

Division

Like floating-point subtraction is similar to addition, division is similar to multiplication. Conceptually only the second step is different.

1. Convert to internal representation
2. Divide mantissas and subtract exponents in parallel
3. Detect leading one
4. Normalize result
5. Round
6. Adjust exponent
7. Convert to IEEE-754 format

But the algorithms for division of the mantissas is not similar to multiplication at all. Where a fixed-point multiplier consists of three simple often combinatorial steps (partial product generation, operand reduction using redundant adders and a final adder), the fixed-point division is a sequential operation with many possible implementations. There are three common algorithms which are called SRT, Newton-Raphson reciprocal approximation and multiplicative normalization (Monsson, 2008).

4.4.3 Limitations of FP

IEEE-754 floating-point standard have many advantages, however, it also have many limitations especially when one wants to use it on the hardware. Following are the some limitations of FP:

- One-size-fits-all: In floating-point standard, the word width of each format is fixed and one needs to use all the bits irrespective of the actual bits required to store some number. In other words FP format is kind of i.e., "one-size-fits-all" format. For example, the value of 0 in single precision format is

0	00000000	00000000000000000000000

  from the representation, one can immediately notice that, all 32 bits are used to store 0 which is not required.

- Wastage of bits: As a consequence of one-size-fits-all, FP format waste many bits to store information. Now, let's take a look at another example of representation value of constant π using double precision FP. The constant π is approximated to 11-decimal accuracy as 3.1415926535 (Finch, 2003, Chapter 1). In general, a floating-point approximation of real value can be expressed as

0	10000000000	10010010000111111011010101 00010000010001011101000100

  In this representation, we used 64-bits to store the approximated value of π. Here, we can see that even after using 64-bits we are not getting the exact value of π as it is approximated and wasted many bits. Also, FP wastes many bits on NaNs.

- Floats prevent the use of parallelism: for faster algorithm, embedded device like FPGA provides features to parallelism but it is worth to say that FP format gives different answers for same operation performed in parallel and serial, i.e. $(a + b) + (c + d)$ (parallel) $\neq ((a + b) + c) + d$ (serial).
- Disobeys algebraic laws like associativity, commutativity or distributivity on real numbers (Carver, 2012, Chapter 4).
- FP format gives different answers on different hardware: conversion of a floating-point number to and from arbitrary strings is not guaranteed across platforms.
- IEEE floats report rounding, overflow, underflow in processor register bits that no one ever sees. Rounding errors prevent use of parallel methods.
- Too much energy and power needed per calculation.

To overcome the above limitations in this thesis, we are proposing the use of universal numbers for explicit MPC data storage and point location algorithm. In the unum arithmetic we will need the background of interval arithmetic. So next, we will give a short overview of interval arithmetic.

4.5 Interval Arithmetic

Interval arithmetic, interval mathematics, interval analysis, or interval computation, is a method developed by mathematicians since the 1950s and 1960s, as an approach to putting bounds on rounding errors and measurement errors in mathematical computation and thus developing numerical methods that yield reliable results. Very simply put, it represents each value as a range of possibilities. For example, instead of estimating the height of someone using standard arithmetic as 2.0 meters, using interval arithmetic, we might be certain that the person is somewhere between 1.97 and 2.03 meters.

This concept is suitable for a variety of purposes. The most common use is to keep track of and handle rounding errors directly during the calculation and of uncertainties in the knowledge of the exact values of physical and technical parameters.

The latter often arise from measurement errors and tolerances for components or due to limits on computational accuracy. Interval arithmetic also helps find reliable and guaranteed solutions to equations and optimization problems.

Mathematically, instead of working with an uncertain real x, we work with the two ends of the interval $[a, b]$ that contains x. In interval arithmetic, any variable x lies between $[a, b]$, or could be one of them. A function F, when applied to x, is also uncertain. In interval arithmetic F produces an interval $[c, d]$ that is all the possible values for $F(x)$ for all $x \in [a, b]$.

The main focus of interval arithmetic is the simplest way to calculate upper and lower endpoints for the range of values of a function in one or more variables. These endpoints are not necessarily the supremum or infimum, since the precise calculation of those values can be difficult or impossible.

4.5.1 Basic Terms and Concepts

Recall that the closed interval denoted by $[a, b]$ is the set of real numbers given by

$$[a, b] = \{x \in \mathbb{R}\colon a \leq x \leq b\}. \tag{4.18}$$

In general interval can be closed, half-open, open.

4.5.2 Relation, Width, Absolute Value, Midpoint

- Order relations for intervals: we know that the real numbers are ordered by the relation $<$. This relation is said to be transitive: if $a < b$ and $b < c$, then $a < c$ for any a, b, and $c \in \mathbb{R}$.

- Width: the width of an interval x is defined and denoted by

$$\text{width}(x) = b - a. \tag{4.19}$$

- The absolute value of x, denoted $|x|$, is the maximum of the absolute values of its endpoints:

$$|x| = \max(\text{abs}(a) - \text{abs}(b)). \tag{4.20}$$

- The midpoint of x is given by

$$\text{mid}(x) = \frac{(b - a)}{2}. \tag{4.21}$$

4.5.3 Operations of Interval Arithmetic

We are about to define the basic arithmetic operations between intervals. The key point in these definitions is that computing with intervals is computing with sets. For example, when we add two intervals, the resulting interval is a set containing the sums of all pairs of numbers, one from each of the two initial sets.

Addition

By definition, the sum of two intervals $x = [a, b]$ and $y = [c, d]$ is give by

$$x + y = [a + c, b + d]. \tag{4.22}$$

Subtraction

The difference of two intervals $x = [a, b]$ and $y = [c, d]$ is give by

$$x - y = [a - d, b - c]. \tag{4.23}$$

Multiplication

The multiplication of two intervals $x = [a, b]$ and $y = [c, d]$ is give by

$$x \times y = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]. \tag{4.24}$$

Division

The division of two intervals $x = [a, b]$ and $y = [c, d]$ is give by

$$x/y = [a, c] \times \frac{1}{[b, d]} \tag{4.25}$$

where $1/[b, d] = [1/d, 1/b]$ if $0 \notin [b, d]$. For division by an interval including zero $1/[b, 0] = [-\infty, 1/b]$ and $1/[0, d] = [1/d, \infty]$. For the detailed description about interval arithmetic see, e.g Moore (1979), Hickey et al. (2001), Moore et al. (2009).

4.6 Summary

This chapter has introduced the basics of number system used in computer arithmetic. Number system and computer arithmetic are important part of optimization algorithm especially when one have limited hardware resources. The choice of number system has a crucial impact on the hardware resources required to store data and perform arithmetic. This chapter gives a basic introduction to IEEE standard floating-point number system with its arithmetic and features. Furthermore, the short introduction to interval arithmetic given which is the basic for unum arithmetic. In the next chapter, we discuss about the unum and its arithmetic.

Chapter 5

Universal Numbers

If we are willing to reduce the memory footprints of explicit MPC without comprising on optimality and performance, we can achieve this goal by using universal number or unum format. In idea of unum arithmetic was introduced two years ago by Dr. John Gustafson, by breaking completely from the IEEE float-type format, resulting in fixed bit size values, fixed execution time, no exception values or "gradual underflow" issues, no wasted bit patterns, and no redundant representations (like "negative zero"). In this chapter, we will revisit the idea of unum arithmetic which Dr. John has published in his book, "The End of Error: Unum Computing" (Gustafson, 2015).

The universal number, encompasses all standard floating-point formats, as well as fixed-point and exact integer arithmetic. Unums get more accurate answers than floating-point arithmetic, yet use fewer bits in many cases, which saves memory, bandwidth, energy, and power. Unlike floating-point numbers, unums make no rounding errors, and cannot overflow or underflow. Unums are the superset of floating-point format which is a superset of integers. Unum arithmetic has more rigor than interval arithmetic, but uses far fewer bits of storage. A unum computing environment dynamically and automatically adjusts precision and dynamic range so programmers need not choose which precision to use; therefore it's easier and safer to use than floating-point arithmetic. Unlike floating-point, unum arithmetic guarantees bitwise identical answers across different computers. Unums help with issues like memory bandwidth, power efficiency, and programmer productivity which are precious and limiting.

5.1 Unum Format

Suppose we want to build on the IEEE principles, but be able to vary the precision and dynamic range to the optimum number of bits, and also record whether the number is exact or lies within a range, instead of rounding the number. We can do that by attaching additional bit fields that make the number self-descriptive. Call this a "universal number".

Definition 5.1.1. (Universal Number). It is a bit string of variable length that has six sub-fields: sign bit, exponent, mantissa, uncertainty bit or ubit, exponent size, and mantissa size.

Fig. 5.1 shows the general representation of the unum format. The left three fields are like IEEE FP format, but with unums these fields have better rules for handling special numbers like Not-a-Number (NaN) and infinity.

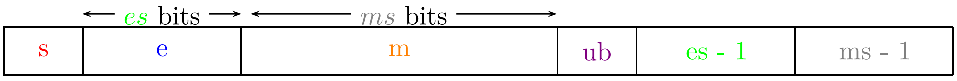
$\leftarrow es$ bits $\rightarrow$
$\leftarrow ms$ bits $\rightarrow$
s
e
m
ub
es - 1
ms - 1

Figure 5.1: General representation of the universal number format with six sub-fields.

The description of each field is given below:

1. sign (s): In unum it is the same as the sign bit in floating-point numbers. For positive numbers, the sign is 0, and for negative numbers, it is 1.
2. exponent (e): In unum exponent is like exponent in the floating-point number but its bit length is specified by exponent size denoted by es; see below.
3. mantissa (m): Like the exponent, the mantissa in unum is the same as in floating-point number, but its length depends on the number of bits specified by mantissa size denoted by ms; see below.
4. ubit (ub): This bit in the unum is used to indicate whether the number is exact or in an interval. It is 0 if unum is exact and 1 if unum is in the open interval between two exact unums. The ubit is exactly like the "…" in an expression like "$2/3 = 0.666\ldots$". It means there are more bits after

the last bit of the fraction, not all 0 and not all 1, but unspecified. Instead of rounding, the ubit stores the fact that a result lies between representable exact values.

The ubit also marks the case of value being between the largest representable real number and infinity or between the smallest magnitude representable number and zero. The ubit allows unums to handle cases that float would overflow to infinity or underflow to zero, instead treating them as precisely defined ranges that are never confused with infinity or zero. In short, an unum is honest about what it does and what it does not know about value. Unums manage uncertainty by making it explicit, and storing it in the number self-description.

5. exponent size (es): The fields exponent size and mantissa size are self-descriptive lengths offset by 1. The number of bits in this field depends on "how many bits we want to allocate to specify es", i.e., the size of exponent size (ess). This field allow the change of width with every calculation, much the way the exponent in a FP changes the binary point location with every calculation. In exponent sizefield with ess-bits, we can have es that range anywhere from 1-bit to 2^{ess} bits.

6. mantissa size (ms): The number of bits in this field depends on "how many bits we want to allocate to specify ms" i.e. the size of mantissa size (mss). With a mss-bit ms field, we can have ms that range anywhere from 1-bit to 2^{mss} bits. The es and ms values can thus be customized based on the needs of the application and the user, not set by any standard.

5.1.1 Environment

As in floating-point standard, there are different types of precision like half, single, double and quad, in unum we have environments. The environment is given by the pair $\{ess, mss\}$.

Definition 5.1.2. (Size of exponent size (ess)). The ess is the number of bits allocated to store the maximum number of bits in the exponent field of an unum.

For example, suppose we have exponent value $= 1111_2$; then $es = 4$, which is 100_2, and $ess = 3$ bits (number of bits needed to express es of 4 bits).

Definition 5.1.3. (Size of mantissa size (*mss*)). The *mss* is the number of bits allocated to store the maximum number of bits in the mantissa field of an unum.

For example, suppose we have mantissa value = 110101011_2; then $ms = 9$, which is 1001_2 and *mss*= 4 bits (number of bits needed to express ms of 9 bits).

The exponent sizes in IEEE FP format are 5 (101_2) for half precision, 8 (1000_2) for single precision, 11 (1011_2) for double precision, and 15 (1111_2) for quad precision. Hence, four bits suffice to cover all IEEE FP formats. There is always at least one exponent bit and one mantissa bit. Therefore, we keep an offset of one in the last two fields of unum, i.e., exponent size and mantissa size. Because of this offset in original exponent sizes, IEEE exponent sizes can be represented as (100_2), (111_2), (1010_2), and (1110_2). In the unum, four bits would be enough to specify any exponent size ranging from 2^0 bit (fixed-point format) to 2^4 bits (more than IEEE FP quad format).

As with the exponent sizes, mantissa sizes in IEEE FP format are 10 (1010_2), 23 (10111_2), 52 (101100_2), and 112 (1110000_2). So, seven bits are enough to cover all IEEE mantissa sizes (again, offset by one). This covers mantissa sizes from 2^0 bit to 2^7 bits. The number of bits needed to specify the size of exponent size (zero to four) and the size of mantissa size (zero to seven) in unum is called *ess* and *mss* respectively, and that pair is called the *Environment* or env$\{ess, mss\}$ and unum with specific *Environment* is denoted as unum$\{ess, mss\}$ e.g., unum$\{3, 2\}$. The user is free to define this pair based on the needs (accuracy, memory, etc.) of an application. The *Environment* can be as small as $\{0, 0\}$ or as large as computer memory and speed permits (Gustafson, 2015, Chapter 4).

5.1.2 Utag

Definition 5.1.4. (Utag). In the unum format, the set of the last three fields, ubit, exponent size, and mantissa size is called the *utag*.

The utag is the “tax” we pay for flexibility, compactness, and exactness information, just as having exponents embedded in a number is the “tax” that floating-point numbers pay for describing their individual scale factor. The additional self-descriptive information is why unums are to a floating-point numbers what floating-point numbers are to integers (Gustafson, 2015, Chapter 4).

The size of the “tax” is important enough that it deserves a name: *utagsize*.

Definition 5.1.5. (Utagsize). The number of bits in *utag* is called *utagsize.*

The number of bits in *utagsize* is equal to $1 + ess + mss$.

5.2 Type Conversion

This section describes the conversion of exact unum to a floating-point number and inexact unum to floating-point number.

5.2.1 Exact Unum To Floating-Point Number

We need a way to convert the floating-point part of an unum into its mathematical value. First, we use the self-descriptive bits in the utag to determine the exponent size and mantissa size. Then we can extract the sign, exponent, and mantissa bits using bit masks. From those values, we build a function that converts the part left part (s, e, and m) of unum number into a real number using IEEE binary float rules. When the ubit is 0, the formula for a unum value is exact and given as

$$x = (-1)^{s} \times \begin{cases} 2^{2-2^{es-1}} \times \left(\frac{m}{2^{ms}}\right) & \text{if } e = \text{all 0 bits,} \\ \infty & \text{if e, m, es, and ms have all bits}= 1, \\ 2^{1+e-2^{es-1}} \times \left(1 + \frac{m}{2^{ms}}\right) & \text{otherwise.} \end{cases} \tag{5.1}$$

This expression is the improved version of IEEE FP format which does not waste the huge number of bits on NaN.

It is easy to convert an exact unum to an IEEE standard FP. To do that, you find the smallest size IEEE float with fields large enough for the unum exponent and mantissa, and pad the unum with extra bits to match the worst-case size allotment in the float. If the exponent is all 1 bit and represents a finite value, you discard all the mantissa information and replace it with 0 bits, to represent ∞ in the style of the FP. There are no NaN values to convert, since an unum NaN is not exact; its ubit is set to 1. In same environments settings e.g., env$\{2, 2\}$ it is possible to list all the exact unums see in (Gustafson, 2015, Chapter 4) for more the list of exact unums.

5.2.2 Inexact Unum to Floating-Point Number

A unum that has its ubit set to 1 indicates a range of values strictly between two exact values, that is, an open bound. The formula for the value should be simple, but we do need to be careful of the values that express infinity and maximum real value, and if we go "beyond infinity" by setting the ubit, we get a NaN. Also, if the number is an inexact zero, then the direction "farther from zero" is determined by checking the sign bit. The expression for conversion of real number to inexact unum is given as follows:

$$x = \begin{cases} (5.1) & \text{if } ub = 0, \\ \text{NaN} & \text{if } u = \text{ sNaNu or } u \ = \ \text{qNaNu}, \\ (\text{big}, \infty) & \text{if } (5.1) \ = \ \text{bigu}, \\ (-\infty, -\text{big}) & \text{if } (5.1) \ = \ \text{bigu} + \text{signmask}, \\ ((5.1) \text{ for } u, (5.1) \text{ for } (u + \text{ulpu})) & \text{if } s = 0, \\ ((5.1) \text{ for } (u + \text{ulpu}), (5.1) \text{ for } u) & \text{if } s = 1, \text{which covers all other cases.} \end{cases} \tag{5.2}$$

In the above expression, u is denoted as unum; big denotes the biggest real value representable by an unum, bigu denotes the unum bit string that represents big value and signmask is consists of sign bit of unum and other bits are 0. Notation ulpu is the unum bit string with a 1 bit in the last bit of the mantissa and zero for its other bits. The detailed information about conversion can be found in (Gustafson, 2015, Chapter 4).

Example 5.2.1. Convert the constant π to an unum. Consider the constant π is approximated to 11-decimal accuracy as 3.1415926535 (Finch, 2003, Chapter 1)

To get the values of sign, exponent, and mantissa, we convert the real value in to the double precision FP representation using (4.14). The double precision representation of π is

0	10000000000	10010010000111111011010101 00010000010001011101000100

In this representation, we used 64-bits to store the approximated value of π. Here, we can see that even after using 64-bits we are not getting the exact value of π and wasted many bits.

Having the IEEE 64-bit representation of π at hand, one can proceed to convert it to unum. Assume that we want to have a 4-bit exponent and a 4-bit mantissa for the value of π; in unum it can be obtained by setting env$\{2, 2\}$. Using unum format the value of constant π can be represented in only 11-bits as given below

0	1	1001	1	00	11

It's interesting to notice that only 11 bits are used to store the value of π which saved 53-bits compared to the IEEE double precision format. Also, the *ub* bit is 1 which indicates that the value is in an interval. An inexact unum is not the same thing as a rounded floating-point number. In fact, it is contradictory, since IEEE float return inexact calculations as exact (incorrect) numbers. To obtain interval values, we used the Unit of the Least precision (*ULP*) which is equal to the difference between exact values in bit format that differs by one unit in the last place. If we insert the values of sub-fields in (5.2), we get the value of π as the open interval $(3.125, 3.25)$.

Now, one might ask "how to do the addressing of unum sub-fields?". With fixed-size floats, you tend to think of loading or storing all their bits at once. With unums, you have a two-step process like you would have reading character strings of variable length. Based on the environment settings, the computer loads the bits of the exponent size and mantissa size fields, which point to where the sign bit, exponent, and mantissa be to the left of the utag. It then loads the rest of the unum and points to the next unum. This is why exponent size and mantissa size are like processor control values; they tell the processor how to interpret bit strings. A collection of unums are packed together exactly the way the words in this paragraph are packed together. As long as a set of unums is of substantial length, there is little waste in using conventional power-of two sizes to move the unum data in blocks, such as cache lines or pages of memory. Only the last block is "ragged" in general, that is, only partly full. Blocking the set of unums can also make it easier to do random access into the set. For more details see (Gustafson, 2015, Chapter 4).

5.3 Special Values in Unum Environment

Similar to the special values of IEEE 754 FP numbers, unum also have special values for each environment. Table 5.1 and Table 5.2 shows the features of unum and approximate decimal versions of the maxreal and smallsubnormal real values computed when the environment is set to $\{3, 2\}$ and $\{3, 4\}$, $\{2, 2\}$, respectively. For the detailed meaning of each feature, see (Gustafson, 2015, Chapter 4).

Table 5.1: Features and values of the unum format for env$\{3, 2\}$

.

Feature	**Meaning**	**Value in env$\{3, 2\}$**
ess	Size of size of exponent	3
mss	Size of size of mantissa	3
utagsize	Number of bits in the *utag*	6
maxubits	Maximum bits in an unum	19
posinfu	The unum for $+\infty$	0 11111111 1111 0 111 11
neginfu	The unum for $-\infty$	1 11111111 1111 0 111 11
qNaNu	The unum for quiet NaN	0 11111111 1111 1 111 11
sNaNu	The unum for signaling NaN	1 11111111 1111 1 111 11
maxrealu	Finite unum closest to $+\infty$	0 11111111 1110 0 111 11
negbigu	The largest magnitude negative unum	1 11111111 1110 0 111 11
smallsubnormalu	unum for the smallest real	0 00000000 0001 0 111 11
maxreal	The largest representable real	$\approx 6.38 \times 10^{38}$
smallsubnormal	Smallest representable real > 0	$\approx 7.35 \times 10^{-40}$

Table 5.2: Features and values of the unum format for env$\{3,4\}$ and env$\{2,2\}$.

Feature	**Value in env**$\{3,4\}$	**Value in env**$\{2,2\}$
ess	3	2
mss	4	2
utagsize	8	3
maxubits	33	14
posinfu	0 11111111 1111111111111111 0 111 1111	0 1111 1111 0 11 11
neginfu	1 11111111 1111111111111111 0 111 1111	1 1111 1111 0 11 11
qNaNu	0 11111111 1111111111111111 1 111 1111	0 1111 1111 1 11 11
sNaNu	1 11111111 1111111111111111 1 111 1111	1 1111 1111 1 11 11
maxrealu	0 11111111 1111111111111110 0 111 1111	0 1111 1110 0 11 11
negbigu	1 11111111 1111111111111110 0 111 1111	1 1111 1110 0 11 11
smallsubnormalu	0 00000000 0000000000000001 0 111 1111	0 0000 0001 0 11 11
maxreal	$\approx 6.8 \times 10^{38}$	$\approx 4.8 \times 10^{2}$
smallsubnormal > 0	$\approx 1.79 \times 10^{-43}$	$\approx 9.76 \times 10^{-4}$

5.4 No Overflow, No Underflow, and No Rounding

If a number becomes too large to express in a particular floating-point precision (overflow), what should we do? The IEEE Standard says that when a calculation overflows, the value ∞ should be used for further calculations. However, setting the finite overflow result to exactly ∞ is infinitely wrong. Similarly, when a number becomes too small to express, the standard says to use 0 instead. Both substitutions are potentially catastrophic things to do to a calculation, depending on how the results are used. The Standard also says that a flag bit should be set in a processor register to indicate if an overflow occurred. There is a different flag for underflow, and even one for "rounded".

We do not need overflow because instead, we have "almost infinite" values $(maxreal, \infty)$ and $(-\infty, -maxreal)$. We do not need underflow because we have the "almost nothing" values $(-smallsubnormal, 0)$ and $(0, smallsubnormal)$. A computation need never erroneously tell you, say, that $10^{-100000}$ is "equal to zero" but with a hidden (or intentionally disabled) underflow error. Instead, the result is marked strictly greater than zero but strictly less than the smallest representable number. Similarly, if you try to compute something like a billion to the billionth power, there is no need to incorrectly substitute infinity, and no need to set off an overflow alarm.

5.5 How Unum Saves Number of Bits?

As described above, using env$\{4, 7\}$ we can achieve the superset of IEEE FP formats with the *utagsize* equal to $1 + 4 + 7 = 12$. Also, the minimum number of bits in an unum is $3 + utagsize$ and the maximum possible number of bits, i.e., maxubits with value is $2 + ess + mss + 2^{ess} + 2^{mss}$.

Definition 5.5.1. (Maxbits). It is the maximum number of bits an unum can have.

At this stage, one might wonder: *How can the unum approach help to reduce memory as compared to floating-point, if it adds more bits to the format?*.

The main reason is that it frequently allows us to use far fewer bits for the exponent and mantissa than a "one-size-fits-all" format choice. The mantissa size and the exponent size of an unum increases and decreases as needed, and the average size is so much less than the worst case size of an IEEE float that the savings are more than enough to pay for the *utag*. In a representation of π we can save 53-bits if env is set to $2, 2$. In that example, we paid 3-bits for utag but saved 53-bits and accuracy.

The exponent and mantissa in double precision IEEE FP format are 11 and 52 but if we use env$\{2, 2\}$, we can save at least 7 bits in exponent and 48 bits in mantissa as compared to exponent and mantissa in double FP. Also, we save bits for more common strings, as for π exponent is only 1-bit where env$\{2, 2\}$ supports up to 4, so we saved 3-bits there. In unums the *ess* and *mss* is automatically manged by computer. Automatic range and precision adjustment are inherent in the unum approach, the same way float arithmetic automatically adjusts the exponent. The key is the ubit, since it tells us what the level of certainty or uncertainty is in a value. The exponent size and fraction size change the meaning of an ULP, so the three fields together provide the computer with what it needs, at last, to automate the control of accuracy loss. If you add only one of the three fields in the utag, you will not get a very satisfactory number system.

5.6 The Vast Range of Unums

In unums, a computer can manage its own environment sizes by detecting unsatisfactory results and recalculating with a bigger utag, the programmer might want some control over this for improved efficiency. It is usually easy for a programmer to make a good estimate of the overall accuracy needs of an application, which still leaves the heavy lifting (dynamically figuring out the precision requirements of each operation) to the computer.

For example, the env$\{2, 2\}$ unum environment looks appropriate for seismic signal processing because of the absence of rounding, overflow, and underflow errors. It provides for an exponent and mantissa up-to 4 bits long. Therefore, its maximum dynamic range matches that of a half precision float (16-bit) and its mantissa has more than five decimals of accuracy, yet it cannot require more than 14 bits of total storage (and usually takes far less than that). Using an env$\{2, 2\}$ environment instead of a env$\{4, 7\}$ environment reduces the utag length to 3 instead of 12, a small but welcome savings of storage. It is of central interest to see how the unums hold up-to the previously introduced IEEE 754 floating-point numbers. To illustrate the behavior of the unums, different features of the unum and floats are laid out in Table 5.3. It can be seen from the table 5.3 that the unum takes fewer bits, for example in env$\{2, 2\}$ it takes 2 fewer bits and in env$\{4, 5\}$ unum takes 5 bits less than double precision FP. Even-though the maximum number of bits in env$\{3, 4\}$ is 33, the average number of bits will be less than 32 as a number of bits is varying in unums. So, env$\{3, 4\}$ can be preferred as compared to a single precision floating-point number. Range of values represented by env$\{2, 2\}$ and env$\{3, 4\}$ are almost equal to half and single precision FP, but env$\{4, 5\}$ gives a very high range as compared to double precision FP, that too in less number of bits.

5.7 Three Layers

Every computer has a hidden "scratchpad" for arithmetic. Perhaps the simplest example is the multiplier. One layer of calculation is where all numbers are stored in some standard format. There is also an internal layer, a hidden scratchpad with extra bits, where the computer performs math perfectly, or at least accurately enough to guarantee correct representation in the standard format. The bits shown stored in scratchpad are never shown to the user; they are part of the hidden scratchpad. There are similar structures for doing floating-point addition and subtraction (where the bits must usually be shifted first because their exponents put their binary point in different places), divides, and square roots. In unum, there are three layers, unum layer, math layer, and human layer.

5.7.1 The Unum Layer (u-layer)

The u-layer is the level of computer arithmetic where all the operands are unums (and data structures made from unums, like ubounds). With only unums one can not perform arithmetics as it does not create a representation that is closed under the four basic arithmetic operations.

ubound

Unums give us the vocabulary needed for precise control of sets of real numbers, not just point values. Now consider the case that we want to multiply unum $(0, 1)$ with the unum 2. The multiplication should be an unum $(0, 2)$, but there is no such unum. In unum representation even if ubit is active, it does not create a representation

Table 5.3: Unums features for env$\{2,2\}$, $\{3,4\}$, and $\{4,5\}$ selected to match IEEE FP formats (half, single and double precision).

Feature	**env**$\{2,2\}$	**Half**	**env**$\{3,4\}$	**Single**	**env**$\{4,5\}$	**Double**
maxbits	14	16	33	32	59	64
minbits	8	16	11	32	13	64
maxreal	4.8×10^{2}	6.5×10^{4}	6.8×10^{38}	3.4×10^{38}	1.0×10^{9864}	1.8×10^{308}
small	9.7×10^{-4}	6.1×10^{-5}	1.7×10^{-43}	1.2×10^{-38}	1.0×10^{-9873}	2.2×10^{-308}

that is closed under the four basic arithmetic operations $(+, -, \times, \div)$. We need to be able to define a range of real numbers where we can represent the endpoints carefully. The way to do this is the ubound.

Definition 5.7.1. (ubound). Ubound is a single unum or a pair of unums that represent a mathematical interval of the real line. Closed endpoints are represented by exact unums, and open endpoints are represented by inexact unums.

The set of ubound is closed under addition, subtraction, multiplication, and division. Also, square root, powers, logarithm, exponential, and many other elementary functions needed for technical computing. Ubounds are the superset of traditional "interval arithmetic" but are much more powerful. For more details see (Gustafson, 2015, Chapter 4).

Example 5.7.1. Suppose unum u_1 represents the exact value 4 and another unum u_2 represents the open interval $(5, 5.25)$. The pair $\{u_1, u_2\}$ is a ubound, one that represents the mathematical interval $[4, 5.25\}$. The left endpoint "(5" in "$(5, 5.25)$" is ignored. Any inexact unum that has a range that ends in "5.25)" will do, so we generally use the one that takes the fewest bits. The ubound is the outermost endpoints.

With ubounds, we are ready to define the unum equivalent of the layer the user and programmer see directly when working with real values.

5.7.2 The Math Layer

This layer is similar to the scratchpad on computers. To build a hardware system that uses unums, there must be a well-defined scratchpad layer. The scratchpad equivalent of a ubound is the *general bound* or *gbound*.

gbound

A gbound is the data structure used for temporary calculations at higher precision than in the unum environment, i.e., the scratchpad. The way to build a gbound data structure with integer values is given in (Gustafson, 2015, Chapter 5). The scratchpad in a unum environment is based on gbounds; it is called the *g-layer*.

g-layer

The g-layer is the scratchpad where results are computed such that they are always correct to the smallest representable uncertainty when they are returned to the u-layer. The g-layer knows all about the env$\{es, ms\}$, so when converting it does the minimum work needed to get to the right u-layer answer.

5.7.3 The Human Layer (h-layer)

Certainly, there is a scratchpad layer and a layer where numbers are represented as a bit strings with format rules. The third layer is the *human layer* or *h-layer*, where numbers exist in forms that humans can send to a computer and experience the results in a form understandable to humans.

5.8 Unum Arithmetic

Universal number arithmetic is a little bit similar to interval arithmetic (Hickey et al., 2001), but unum has additional complexity due to the open versus closed endpoints, dynamic exponent and mantissa sizes, and correct handling of math operation (addition, subtraction, multiplication, and division) that exceed the limits of unums in a particular *environment*. In the following, we will shortly explain some of the math operations that will be used in the explicit MPC algorithm.

5.8.1 Addition/Subtraction:

For the addition of two real numbers, x_1 and x_2 with using unum format the first task is to obtain their ubound like $[a, b]$ and $[c, d]$ and then the addition of these two ubounds are straightforward, $[a+c, b+d]$. But, enough care is needed to add open and closed intervals. If we want to add the open interval $(-\infty, 0)$ to ∞, the correct answer in unum is ∞ because the left endpoint "$(-\infty$" indicates some finite value. If we add ∞ to any finite value, it will always results in ∞. In contrast, conventional interval arithmetic would treat the computation as containing $-\infty + \infty$, which results in a NaN.

We need two addition tables for ubounds: one for the left endpoints and one for the right endpoints, where we write "$[x_1$" or "$(x_1$" to indicate a left endpoint and "$x_2]$" or "$x_2)$" to indicate a right endpoint. The tables tells us what to do in every possible situation. There are thought provoking cases that required for unums design on a computer and needs to be handle carefully, such as:

- $(-\infty + [\infty = [\infty$ and $-\infty] + \infty) = -\infty]$.Exact) $\pm\infty$ always "wins" over an open (inexact) $\pm\infty$.
- "$(\infty$" and "$-\infty)$" aren't included in the table because they cannot occur. Inexact ∞ can only be a right endpoint, and inexact $-\infty$ can only be a left endpoint.
- If $[x_1 + [x_2$ exceeds *maxreal*, the result is "(*maxreal*", not "[*maxreal*". If $x_1] + x_2]$ is bigger than *maxreal*, the result is "$\infty)$". Similarly for results that are less than -*maxreal*.

The rules for unum addition are laid in tables, covering the real number line and infinities, where x_1 and x_2 are exact floats expressible in the u-layer. If the table entry is simply the result of adding the values, it is shown in black, but exceptions are shown in RubineRed. Similarly, if the open-closed nature of the endpoint is simply the OR of the two inexact flags, the parenthesis or bracket is shown in black; exceptions are shown in RubineRed. The rules for adding the left endpoints in the g-layer are laid in Table 5.4:

Table 5.4: Rules for adding the left endpoints in the g-layer.

+Left	$[-\infty$	$(-\infty$	$[x_2$	$(x_2$	$[\infty$
$[-\infty$	$[-\infty$	$[-\infty$	$[-\infty$	$[-\infty$	(NaN
$(-\infty$	$[-\infty$	$(-\infty$	$(-\infty$	$(-\infty$	$[\infty$
$[x_1$	$[-\infty$	$(-\infty$	$[x_1 + x_2$	$(x_1 + x_2$	$[\infty$
$(x_1$	$[-\infty$	$(-\infty$	$[x_1 + x_2$	$(x_1 + x_2$	$[\infty$
$[\infty$	(NaN	$[\infty$	$[\infty$	$[\infty$	$[\infty$

Next, the rules for adding the right endpoints in the g-layer are laid in Table 5.5.

Returning to the u-layer is trivial except for the cases involving $[x_1 + x_2$ and $x_1 + x_2]$. For those, if $x_1 + x_2$ is not an exact unum, the inexact unum that contains the sum is used instead, which changes the closed endpoint to an open one. This also takes care of the case where $x_1 + x_2$ is less than $-$*maxreal* or greater than

Table 5.5: Rules for adding the right endpoints in the g-layer.

+Right	$-\infty]$	$x_2)$	$x_2]$	$\infty)$	$\infty]$
$-\infty]$	$-\infty]$	$-\infty]$	$-\infty]$	$-\infty]$	NaN)
$x_1)$	$-\infty]$	$x_1 + x_2)$	$x_1 + x_2)$	$\infty)$	$\infty]$
$x_1]$	$-\infty]$	$x_1 + x_2)$	$x_1 + x_2]$	$\infty)$	$\infty]$
$\infty)$	$-\infty]$	$\infty)$	$\infty)$	$\infty)$	$\infty]$
$\infty]$	NaN)	$\infty]$	$\infty]$	$\infty]$	$\infty]$

maxreal, since the open intervals $(-\infty, -maxreal)$ and $(maxreal, \infty)$ become the endpoint.

The ubound subtraction operation is computed by simply adding the first argument to the negative of the second argument. That is, $x_1 - x_2 = x_1 + (-x_2)$ is computed in the g-layer and converted back to the u-layer.

5.8.2 Multiplication

For real intervals, if both ubounds are closed intervals, multiplication is simple as described in the Section 4.5.3. But, for unbounded intervals that do not hold. In a multiplication of conventional interval the case $0 \times \pm\infty$ is undefined which results in NaN but, in unum we have more information about the endpoints and we get the following examples:

- The result of $\infty \times (a, b)$ is $[-\infty, \infty]$ and not NaN, where a and b are the nonzero endpoints of opposite sign.
- The result of $0 \times$ inexact ∞ is 0 and inexact $0 \times \infty$ is ∞.
- The result of inexact $0 \times a$ is a nonzero value, where a is any finite nonzero number.
- The result of inexact $0 \times$ inexact ∞ is the entire positive real number line.

For the multiplication of negative-negative and positive-negative numbers, see (Gustafson, 2015, Chapter 10).

To tackle special cases in multiplication of unbounded intervals we consider the multiplication table for left and right endpoints defined in (Gustafson, 2015, Chapter 10). Table 5.6 and Table 5.7 shows the rules for left and right endpoint multiplication with nonnegative inputs. As with the addition tables, we use RubineRed for cases that take a little thought and need to be handled carefully in the multiplication logic.

Table 5.6: Rules for multiplying the left endpoints in the g-layer.

$\times$Left	[0	(0	$[x_2$	$(x_2$	$[\infty$
[0	[0	[0	[0	[0	(NaN
(0	[0	(0	(0	(0	$[\infty$
$[x_1$	[0	(0	$[x_1 \times x_2$	$(x_1 \times x_2$	$[\infty$
$(x_1$	[0	(0	$[x_1 \times x_2$	$(x_1 \times x_2$	$[\infty$
$[\infty$	(NaN	$[\infty$	$[\infty$	$[\infty$	$[\infty$

Table 5.7: Rules for multiplying the right endpoints in the g-layer.

$\times$Right	0]	$x_2)$	$x_2]$	$\infty)$	$\infty]$
0]	0]	0]	0]	0]	NaN)
$x_1)$	0]	$x_1 \times x_2)$	$x_1 \times x_2)$	$\infty)$	$\infty]$
$x_1]$	0]	$x_1 \times x_2)$	$x_1 \times x_2]$	$\infty)$	$\infty]$
$\infty)$	0]	$\infty)$	$\infty)$	$\infty)$	$\infty]$
$\infty]$	NaN)	$\infty]$	$\infty]$	$\infty]$	$\infty]$

If x_1 and x_2 are both negative, the preceding tables work if we apply them to $-x_1$ and $-x_2$. If only one of x_1 or x_2 is negative, then we reverse the sign of the negative argument and also reverse the roles of left and right.

5.8.3 Compare Operator:

The easiest operations to write for unums are comparisons. They are different from comparisons of floats, since unums and ubounds can represent ranges of numbers and thus overlap completely, partially, or not at all.

The comparison $x_1 \leq x_2$ or $(a, b) \leq (c, d)$ is equivalent to checking $b < c$. Ubounds or unums are *equal* if they represent the same gbound. They must have the same endpoints and the same open-closed endpoint properties.

5.9 Summary

This chapter has described a new number system called universal numbers. A brief description of unum format with its six sub-fields is given at the beginning of a chapter. Unum is the superset of a floating-point numbers, and one can convert a floating-point number of any precision to the unum format with the specified unum environment. Section 5.2 describes conversion functions used to convert floating-point number to unum and vise-versa. Similar to the floating-point standard, unum format also has special values which are discussed in Section 5.3. Unum format takes fewer bits to store the same information as floating-point format and does not give flag for overflow, underflow, and rounding, see Section 5.4. At the end of this chapter, unum arithmetic is descried. This chapter is mainly dedicated to the idea of unum and its arithmetic. In the next chapter the implementation of unum format is discussed.

Chapter 6

Embedded Implementation

The solution of multi-parametric optimization problem enables MPC to be used on systems which need fast sampling rates since PWA function evaluation is usually a high speed operation (compared to solving an optimization problem). However, obtaining the explicit optimal MPC solution amounts to solve (offline) a parametric optimization problem, which is, in general, a difficult task. Although the problem is tractable and practically solvable for many interesting control applications, the offline computational effort grows fast as the problem size increases. This is the case for long prediction horizon, a large number of constraints and high dimensional systems. Moreover, as the optimization complexity grows, the explicit solution complexity also commonly grows in terms of the number of control laws forming the PWA function. This means the storage space needed for the explicit MPC implementation increases and the online function evaluation problem becomes more complex as described in Chapter 3.

There are several techniques to reduce computational complexity of explicit MPC solutions, some of them are suboptimal, and some are optimal. Another way of reducing controller complexity is by representing the number of bits required to store all the data (T_i, v_i, L_i, and h_i) by different number system as discussed in Chapter 3. We used universal numbers to represent the controller data to reduce the memory required to store that data. From the Chapter 5, it is well understood that unum can serve as a powerful tool to reduce the memory without losing closed-loop performance. The goal of this chapter is to show the feasibility of unum on CPU and embedded platforms like FPGA, PLC, and microcontroller. The reason for implementing unum on these platforms is that the embedded devices are most commonly used in the industrial environment, e.g., PLC is the popular and rugged hardware used in many process plants for automation and control, FPGA is mainly used in fast dynamic systems such as in automotive applications and microcontrollers are also used in industrial automation and automobile applications.

6.1 MATLAB Toolbox

As it is a well-known fact that MATLAB is very popular and widely used software both in the academics and industry, so obvious priority was to implement universal number arithmetics in MATLAB. There are several toolboxes available for model predictive control design and implementation in MATLAB which exports hardware-specific low-level codes and we can use unums on the top of exported codes to reduce memory footprints. To use the unums in MATLAB and export low-level code directly in the form of unums we have developed[1] a MATLAB-based unum toolbox called `munum` which is open-source and easy to use. This toolbox can be used for developing unum-based algorithms and generating unum numbers from floating-point numbers for embedded implementation, e.g., the data of explicit optimizer in (3.12) (i.e., the floating-point numbers contained in vectors/matrices T_i, v_i, L_i, and h_i.

6.1.1 Unum Arithmetic in MATLAB

The munum toolbox for MATLAB supports unum *environments* from $\{0, 0\}$ to $\{4, 7\}$ which covers all four IEEE-754 floating-point formats discussed in Chapter 4. In the following, a concise overview of the toolbox is given.

Real to Floating-point Conversion

To represent any real number in unum, the first step is to convert it to the standard IEEE floating-point number of appropriate precision. The conversion of a real number to double precision floating-point number is performed by calling the function

```
f = x2f(num).
```

[1]Available at `https://bitbucket.org/kvasnica/munum`

This will give the values of s, e, and m which can be directly used as an input to the next step.

Floating-point to unum Conversion

Before we start converting FP to unum, we need to set an *environment* as per the required accuracy. It can be set by calling

```
en = env(ess, mss),
```

and the maximum bits needed to store FP number is obtained by

```
mub = maxubits(en).
```

In `munum`, unum sub-fields are represented by two data types: boolean for the sign bit and uncertainty bit, and an `unsigned int` (UInt) of the appropriate bit size (depending on the `maxubits`) for all other sub-fields. Considering the above data types we can calculate the maximum number of bits needed to express each sub-field by a function

```
ufm = fieldmax(en, UInt).
```

The following function is called to obtain the unum representation of the FP number

```
ufv = fToU(f, ufm).
```

This will give the values of s, e, m, ub, es, and ms. If we add the bits used to store each field, it is $4 \times$ `UInt` $+ 2$-bits. Now, we want to reshape all the fields, so that it will fit in the `maxubits` bits. For that, we call

```
unum = ucall(en, UInt, ufv)
```

to obtain a formatted unum value. We can see the bit representation of the unum obtained from the real number by calling

```
printbits(en, UInt, unum).
```

The obtained `unum` can be unpacked to its normal form which was obtained from `f2u` by performing

```
ufv = unpackunum(en, UInt, unum)
```

and this can be converted back to a human-readable or real number by calling the function

```
num = u2x(en, UInt, ufv, unum).
```

In this function, we used a High Precision Floating Point (HPF) arithmetic class (D'Errico, 2012) developed for MATLAB. Depending on the ub bit in `unum`, the function `u2x` gives an exact value or an inexact *ubound*.

Basic Math Operations

Once we convert FP to unum or ubound it can be directly used to perform basic math operations such as addition, subtraction, multiplication, division, square root, and power of a number. Fig. 6.1 shows the steps and functions supported by `munum` toolbox for general purpose use. Let us consider; we have two ubound numbers *xub* and *yub* then their addition can be performed by calling function

```
zub = AddUb(xub, yub).
```

Similar to the addition, subtraction can be carried out using function

```
zub = SubUb(xub, yub).
```

For multiplication and division of two ubounds following functions are used

```
zub = MulUb(xub, yub),

zub = DivUb(xub, yub).
```

Square root and power of a number can also be performed by calling function

```
zub = SqrtUb(xub),

zub = PowUb(xub, yub).
```

Logical Operations

Logical operations are very important in optimization algorithms. Following is the list of functions available for logical operations using unum or ubound.
Less than

```
zub = isLessThanUb(xub, yub).
```

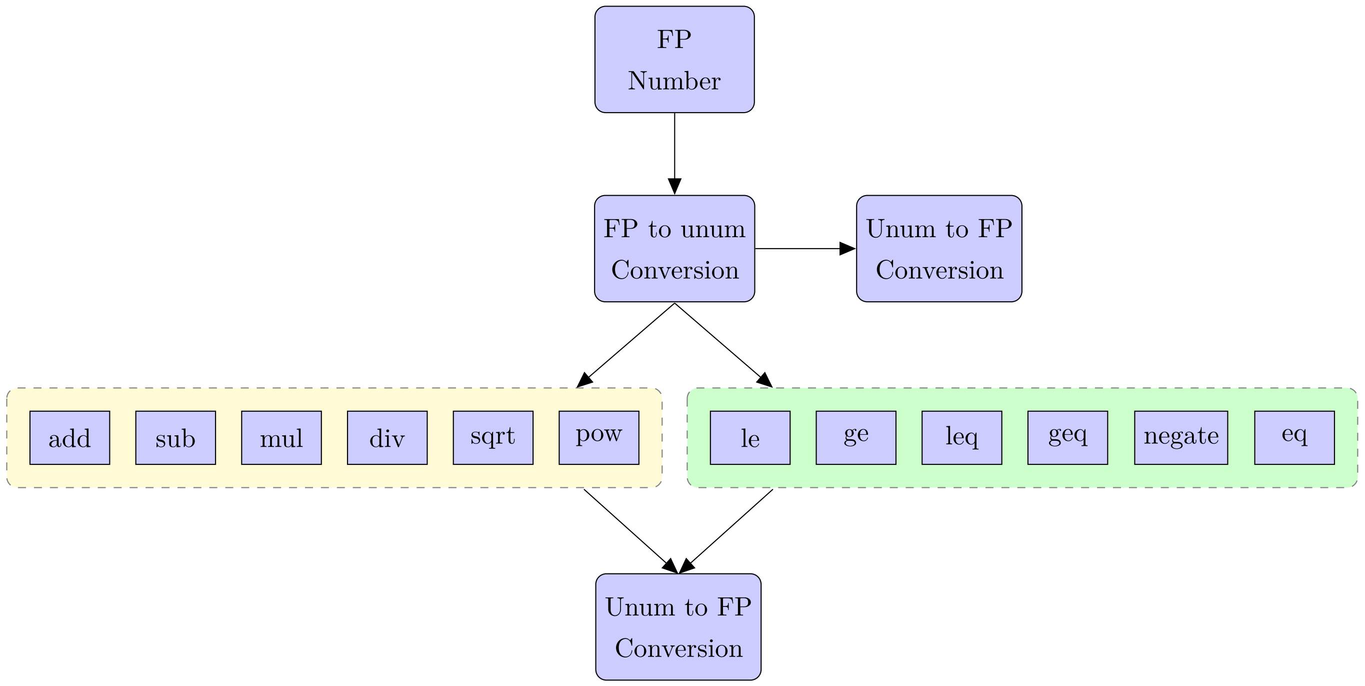

Figure 6.1: Basic arithmetic and logical functions available in `munum` toolbox.

Greater than

```
zub = isGreaterThanUb(xub, yub).
```

Less than equal to

```
zub = isLessThanOrEqualToUb(xub, yub).
```

Greater than equal to

```
zub = isGreaterThanOrEqualToUb(xub, yub).
```

Equal to the ubound

```
zub = isEqualToUb(xub, yub).
```

Negate ubound

```
zub = NegateUb(xub).
```

Similar to the ubound operations unum operations can be performed by calling functions names with U e.g., adition of two unums can be performed by function

```
zub = AddU(xu,yu).
```

It is to be notated that the output of unum math operations are always ubound.

6.1.2 Unum-based EMPC using `munum` Toolbox

The `munum` toolbox provides an automatic tool chain for the unun-based explicit MPC design, given a controller data, i.e., the floating-point numbers contained in vectors/matrices T_i, v_i, L_i, and h_i. MATLAB toolboxes are available that allow designing explicit MPC control laws, see, e.g., Multi-Parametric Toolbox (MPT) (Herceg et al., 2013a), the Hybrid toolbox (Bemporad, 2004), or the POP toolbox (Oberdieck et al., 2016). Apart from the basic arithmetic functions in `munum`, we have developed point locations algorithms discussed in Section 3.3 which uses functions from `munum`. Fig. 6.2 shows the steps involved in the designing of unum-based explicit MPC.

- MPC problem: At this step, one have to decide control problem and its components such as a Linear-Time Invariant (LTI) model of the physical process and the constraints on states and input variables.

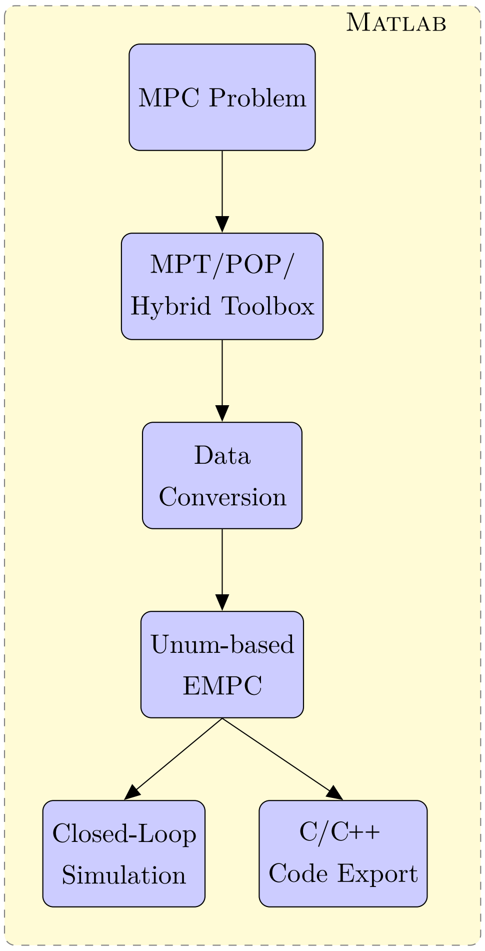

Figure 6.2: Design flow of unum-based explicit MPC using `munum` toolbox.

- Construct explicit MPC: In this step, explicit MPC problem is constructed using any of the available toolboxs which will give controller data in the form of a floating-point numbers. For this step, we need process model (discrete-time), constraints, reference values, and MPC tuning parameters like Q, R, and N.

- Data conversion: Once we have floating-point-based controller at hand, we need to convert data from float to unum using `munum` function for converting float vector/matrix to ubounds vector/matrix, i.e. `fToUbMV(f)` where `f` is vector/matrix of FP numbers. In this step it is important to select unum *environment* $\{ess, mss\}$ depending on the accuracy required for control action and close-loop performance. To fix best pair $\{ess, mss\}$ one have to perform some simulations with different values of $\{ess, mss\}$ so that best can be fixed for embedded implementation. At the end of this step, controller data is in the form of ubounds which is suitable for the point location algorithms.

- Unum-based EMPC: As mentioned above `munum` have sequential search and binary search tree point location algorithms developed based on unum arithmetic. The output of this step is optimal control action in the form of ubounds which can be converted back to floating-point form to apply control input to actual process or the process model.

- Closed-loop simulation: By using unum-based EMPC algorithm one can perform closed-loop simulations and do an analysis of performance. Once the closed-loop performance is analyzed, we can use unum-based explicit MPC for the control of actual process.

- Embedded code export: `munum` toolbox provides functionality to export embedded C/C++ code to deploy on hardware. To export embedded code one have to use the following function

  ```
  toUC(function,filename).
  ```

  Exported code need external library for unum arithmetic in C/C++ which is discussed in the next section.

6.2 C/C++ Toolbox

The `munum` toolbox provides automatic tool chain to export low-level hardware code (C/C++) for embedded implementation of explicit MPC which can be used for systems for which memory storage was the bottleneck in floating-point number based explicit MPC. Similar to the `munum` toolbox, we developed a C/C++ toolbox for unum arithmetic called `cunum`. The basic idea behind developing `cunum` is to verify the real-time implementation of explicit MPC on hardware like PLC or microcontroller. The `cunum` has same features as `mnum` prototype which is implemented using the GNU Multiple Precision Arithmetic Library (GMP) (Granlund, 2016) for math operations for unum with *environment* greater than $\{4, 5\}$.

6.2.1 Math and Logical Operations

The `cunum` toolbox provides the same functionality as `munum` to perform data conversions, addition, subtraction, multiplication, division, square root, the power of number, greater than, less than, etc. operations on unum and ubound numbers. With these function, one can develop unum-based algorithms to use wide features of unum format.

6.2.2 Unum-based EMPC using `cunum` Toolbox

The `cunum` toolbox provides the unum arithmetic/logical functions required for online synthesis of explicit MPC algorithm exported from `munum`. Fig. 6.3 shows the design flow of unum-based explicit MPC using `cunum` toolbox. The design steps are as follows

- MPC problem: In this step MPC problem is designed and constructed using explicit MPC toolbox.

- Controller data: After exporting the controller data (in floating-point) from MPC toolbox, data conversion function from `munum` can be used to generate controller data in ubound.

- `cunum`: The unum arithmetic and logical functions are developed in `cunum` toolbox which are required in the EMPC algorithm which can be exported from `munum` or developed in `cunum`.

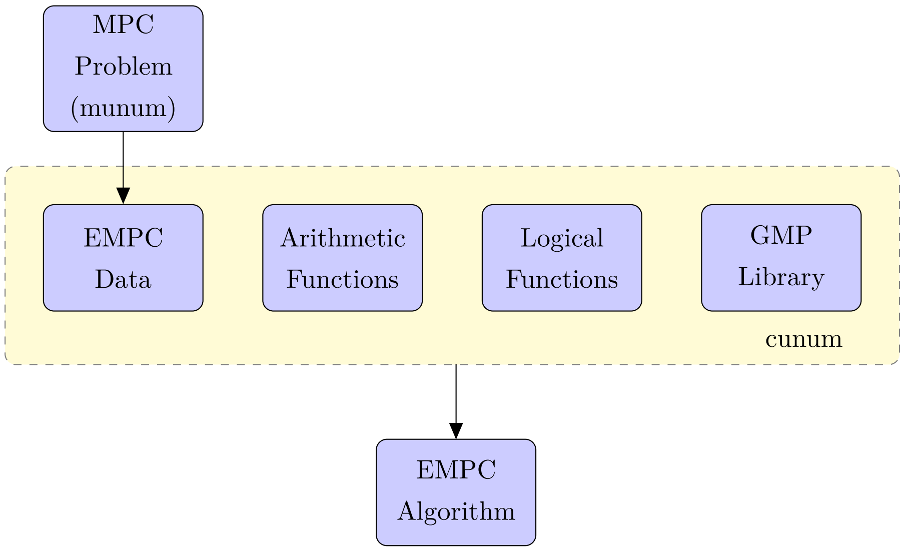

Figure 6.3: Design flow of unum-based explicit MPC using `cunum` toolbox.

In addition to the generated code implementing the unum-based EMPC solver, the tool can also export wrappers for C++ which can be directly used on the top of available algorithms without making any changes in algorithms. The wrapper is created with the specific aim to enable one to use the unum-based algorithms in real-world applications.

The main challenge in storing unums in C/C++ -based embedded platforms like microcontrollers and PLCs is that they do not provide flexibility to allocate variable bit memory other than standard/primitive data types like `int, unsigned int, short, etc.` and their size varies depending on the compiler used to compile the code. For example, consider that we want to store a value of the constant π in unum as shown in Example 5.2.1. It needs total 11-bits to store a value of π in env$\{2, 2\}$. These 11-bits are spread in six sub-fields, two fields (sign, ubit) needs 1-bit each for which we can use data type `bool`. But, for other four fields we need to use the data type of the closest possible number of bits, i.e., to store 2-bit exponent size we need to use 16-bit `unsigned int` which means the wastage of 14-bits. To store unums in an exact number of bits, we will need an embedded device like FPGA which provides flexibility customize architecture, where one can store data in an arbitrary number of bits and parallelize the operations. In the next section, implementation of unums on FPGA is discussed.

6.3 FPGA Toolbox

This section deals with the implementation of unum arithmetic and unum-based explicit MPC on FPGA which is the attractive solution for several real-time embedded systems where control update rate is fast.

6.3.1 Introduction to FPGA Devices

FPGA are chip-sets where connections between multiple logic blocks can be programmed by the user "on the field" to perform the desired computations. Input is presented to the FPGA device through input signals. The input signals propagate through the logic and internal connections of the device, and finally, the result is present on the outputs of the device. Main features of the FPGA devises are

- The FPGA technology compared to the processor is the inherited suitability for problems which are of parallel nature.
- The FPGA devices generally outperform a corresponding processor implementation in terms of speed. This is due to the fact that the actual design is made using the hardware directly and the overhead which is introduced in a processor to fetch and decode instructions etc. is avoided.
- Usually, developing and producing Application Specific Integrated Circuits (ASICs) requires the first phase of circuit synthesis on appropriate Computer Aided Design (CAD) software, and then a physical printing on silicon wafers in high-technology factories; the whole process may require an initial investment in the order of millions of dollars. Moreover, it is not possible to correct bugs and errors discovered after the design phase. On the contrary, FPGA devices can be implemented with a very short developing cycle. They can be reprogrammed as needed, and present almost no starting costs.

Fig. 6.4 shows the basic building blocks of a FPGA device which is a chip composed mainly of arrays of logic blocks and routing channels, with every single logic block generally constituted of a 4-input LUT, a Flip-Flop (FF), and one output; finally, a set of input/output blocks complete the schematic. These are the basic components of every integrated circuit; the only things missing are the interconnections between them. Here comes the key advantage of the FPGA technology:

those interconnections can be programmed as required by simply feeding a serial bit stream to the device after a reset.

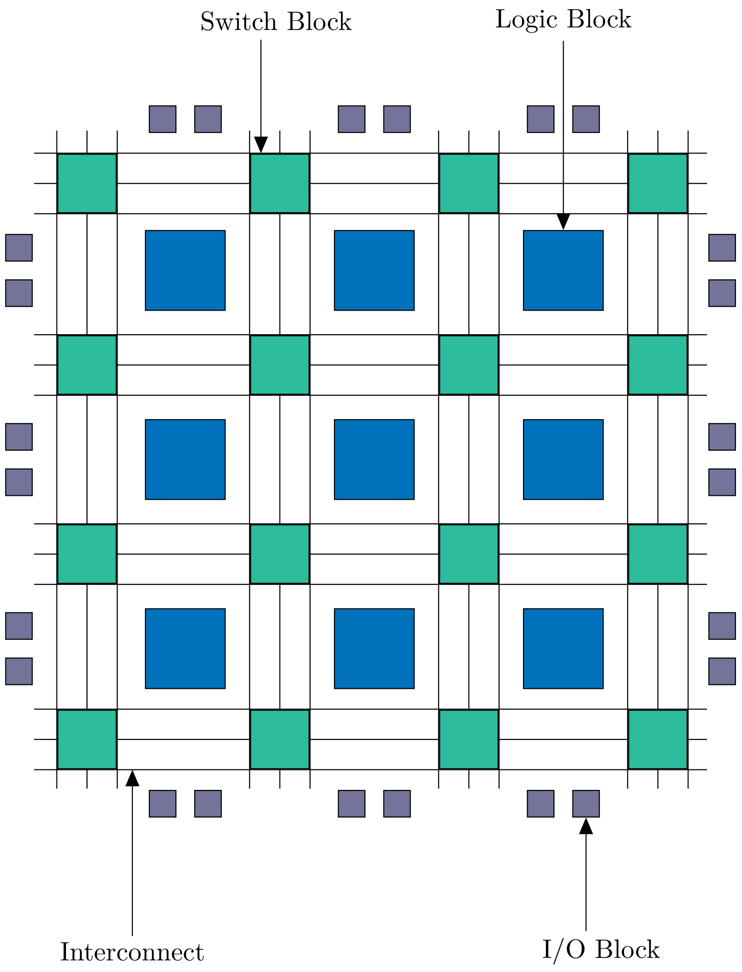

Figure 6.4: FPGA architectures, where arrays of logic blocks are surrounded by a ring of input/output blocks, connected together via interconnect.

- Configurable Logic Blocks (CLBs): These blocks contain the logic for FPGA. In large-grain architecture used by all FPGA vendors today, these CLBs contain enough logic to create a small state machine. The block contains RAM for creating arbitrary combinatorial logic functions, also known as LUTs. It also contains flip-flops for clocked storage elements, along with multiplexers in order to route the logic within the block and from external resources. The multiplexers also allow polarity selection, reset, and clear input selection.

- Configurable I/O blocks: A configurable input/output (I/O) block, as shown in Fig. 6.4, is used to bring signals onto the chip and send them back off again. It consists of an input buffer and an output buffer with three-state and open collector output controls. Typically, there are pull-up resistors on the outputs and sometimes pull-down resistors that can be used to terminate signals and buses without requiring discrete resistors external to the chip. The polarity of the output can usually be programmed for active high or active low output, and often the slew rate of the output can be programmed for the fast or slow rise and fall times. There are typical flip-flops on outputs so that clocked signals can be output directly to the pins without encountering a significant delay, more easily meeting the setup time requirement for external devices. Similarly, flip-flops on the inputs reduce delay on a signal before reaching a flip-flop, thus reducing the hold time requirement of the FPGA.

- Programmable interconnect: Programmable routing connects logic functions. There are long lines that can be used to connect critical CLBs that are physically far from each other on the chip without inducing much delay. Theses long lines can also be used as buses within the chip.

  There are also short lines that are used to connect individual CLBs that are located physically close to each other. Transistors are used to turn on or off connections between different lines. There are also several programmable `switch blocks` in the FPGA to connect these long and short lines in specific, flexible combinations.

  Three-state buffers are used to connect many CLBs to a long line, creating a bus. Special long lines, called global clock lines, are specially designed for low impedance and thus fast propagation times. These are connected to the clock buffers and to each clocked element in each CLB. This is how the clocks are distributed throughout the FPGA, ensuring minimal skew between clock signals arriving at different flip-flops within the chip. For the detailed description of FPGA architectures, see, e.g., Kuon et al. (2008), Farooq et al. (2012).

6.3.2 Hardware Design Flow

Hardware design requires the explicit handling of two concepts: cycle accurate design and structural design. Cycle-accurate design needs hardware designer to specify what happens at each clock cycle. Structural design requires hardware designers to specify exactly which resources to use and how they are connected. To ease the design flow for FPGA based hardware prototyping, specific CAD tools are used. These CAD tools typically accept the hardware description language which is a textual description of the circuit structure. To determine the precise logic implementation and routing in FPGA, CAD software performs the sophisticated optimization. The level of optimization depends on the quality of CAD tools used. FPGA engineering process usually involves the following stages (see Fig. 6.5) (Kilts, 2007):

1. Architecture design: This stage involves analysis of the project requirements, problem decomposition and functional simulation. The output of this stage is a document which describes the future device architecture, structural blocks, their functions and interfaces.

2. HDL design entry: There are different techniques for design entry. One way to perform FPGA design is to use Hardware Description Language (HDL). The most common HDLs are VHDL and Verilog. Hardware description languages are fairly complicated to work with since the designer has to describe how to connect digital logic to obtain a certain functionality. When writing a computer program, the programmer describes actual behavior of the program, rather than which digital logic to use to achieve the desired behavior.

   In recent years FPGA technology has evolved very rapidly while the development of the design tools has not evolved as much. The consequence is that it is getting increasingly difficult to utilize the full performance of digital circuits, one speaks of the design gap. To meet the design gap, new tools which enable circuit design on a higher level of abstraction (behavioral level) are emerging. The term for these tools is Electronic Design Automation (EDA). Typically, these tools use commonly known programming languages such as C or C++ to perform circuit design. For example, Xilinx Vivado Design Suite supports high-level synthesis, with a tool chain that converts C code into programmable logic.

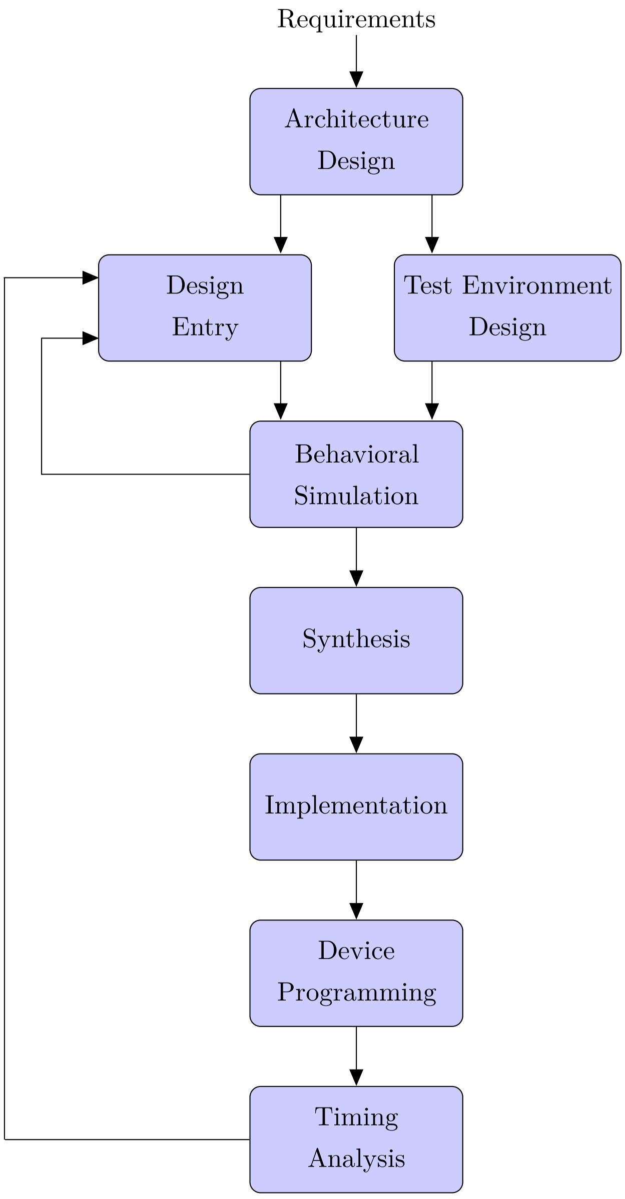

Figure 6.5: FPGA design flow.

3. Test environment design: This stage involves the writing of test conditions and behavioral models. They are later used to ensure that the HDL description of a device is correct.

4. Behavioral simulation: This is an important stage that checks HDL/C correctness by comparing outputs of the HDL model and the behavioral model.

5. Synthesis: This stage involves the conversion of an HDL/C description to a so-called netlist which is a formally written digital circuit schematic. Synthesis process will check code syntax and analyze the hierarchy of the design which ensures that the design is optimized for the design architecture, the designer has selected.

6. Implementation: A synthesizer-generated netlist is mapped onto particular device's internal structure. The main phase of implementation stage is, place and route or layout, which allocates FPGA resources (such as logic cells and connection wires) and generates a Native Circuit Description (NCD) file which physically represents the design mapped to the components of FPGA.

7. Device programming: Now the design must be loaded on FPGA. But the design must be converted to a format so that the FPGA can accept it. The routed NCD file is then given to the bit-stream generator to generate a bit stream (a .BIT file) which can be used to configure the target FPGA device.

8. Timing analysis: During the timing analysis special software checks whether the implemented design satisfies timing constraints (such as clock frequency) specified by the user.

6.3.3 Tools and Resources

In this work, we are focusing on *ZedBoard*^{TM} which is a low-cost development board for the Xilinx Zynq®-7000 All Programmable (AP) System on-Chip (SoC). The Zynq®-7000 is a device recently introduced by Xilinx. It has benefit of integrating Programmable Logic (PL) and a Processing System (PS) with a dual-core ARM Cortex-A9 Processor running at 667 MHz (Avnet, 2014). Having a powerful FPGA fabric along with a high performance pre-wired processor cores and a high speed 12 bits Analog to Digital Converter (ADC) makes this device suitable for the control of electrical systems. The board contains all the necessary interfaces and supporting functions to enable a wide range of applications as shown in In the following, the features of ZedBoard and overview of Softwares used for that is presented.

Features of ZedBoard

- Processor
  - Zynq®-7000 AP SoC XC7Z020-CLG484 – 1
  - 85000 logic cells
  - 1.3 million ASIC gates
  - 53200 LUTs
  - 106400 flip-flops
  - 560 kB of BRAM organized to 140 units, each containing 2048 by 18-bit storage
  - 220 DSP slices (Multiplier-Accumulator) organized to 18 $\times$ 25
  - 220 elementary DSP units.
- Memory
  - 512 MB DDR3
  - 256 Mb Quad-SPI Flash
- Clocking
  - 33.33 MHz clock source for processing system
  - 100 MHz oscillator for programmable logic
- On-board USB-JTAG Programming port
- 10/100/1000 Ethernet
- USB OTG 2.0 and USB-UART
- PS & PL I/O expansion
- Multiple displays (1080p HDMI, 8-bit VGA, 128 $\times$ 32 OLED)
- 12 V DC input @ 3.0 A (Max)
- Cost: 450 EUR

Softwares

The FPGA circuit design was performed in low-level C language routines in *Xilinx Vivado HLS* (Xilinx) and FPGA IP prototyping toolbox *PROTOIP* (Suardi et al., 2015). With these tools, the C/C++ -based algorithms can be prototyped into high-performance FPGA-based embedded systems with Xilinx Zynq devices.

6.3.4 Unum-based EMPC on FPGA

In this section, we report details on the implementation of the unum-based explicit MPC algorithm (sequential search and binary search tree) on a FPGA. FPGAs are the most attractive solution for the implementation of unum arithmetic, as in unums one have to store data in variable bit size which is difficult on fixed architecture hardware which already has their own circuitry and instruction set that the programmer must follow in order to write code for that hardware, e.g., microcontrollers, which restricts it to certain tasks. For the FPGA implementation of EMPC algorithm, we followed the development steps shown in Fig 6.6 `munum`, PROTOIP, and Xilinx Vivado HLS tools which are discussed next:

- EMPC design using `munum`: FPGA design flow starts with a problem statement, process model, and constraints. Considering the problem statement, explicit MPC problem is constructed in MPT toolbox for appropriate settings. The controller data (vectors/matrices T_i, v_i, L_i and h_i) in the form of the floating-point number is then converted to unum format. In the end, EMPC data in unum format is exported in C file to be stored in FPGA memory.

- Algorithmic description: The first step to build a prototype using PROTOIP consists of developing point location algorithm in C language code. We have developed library-free C codes of unum arithmetic and logical operations intended for optimizing hardware resources on FPGA. Developed unum-based ALU is then tested separately for several math operations by Hardware-In-the-Loop (HIL) co-simulation. This implementation of unum arithmetic supports `environments` from $\{0, 0\}$ to $\{4, 5\}$ as it covers up-to double precision floating-point stranded numbers which are sufficient for many control applications.

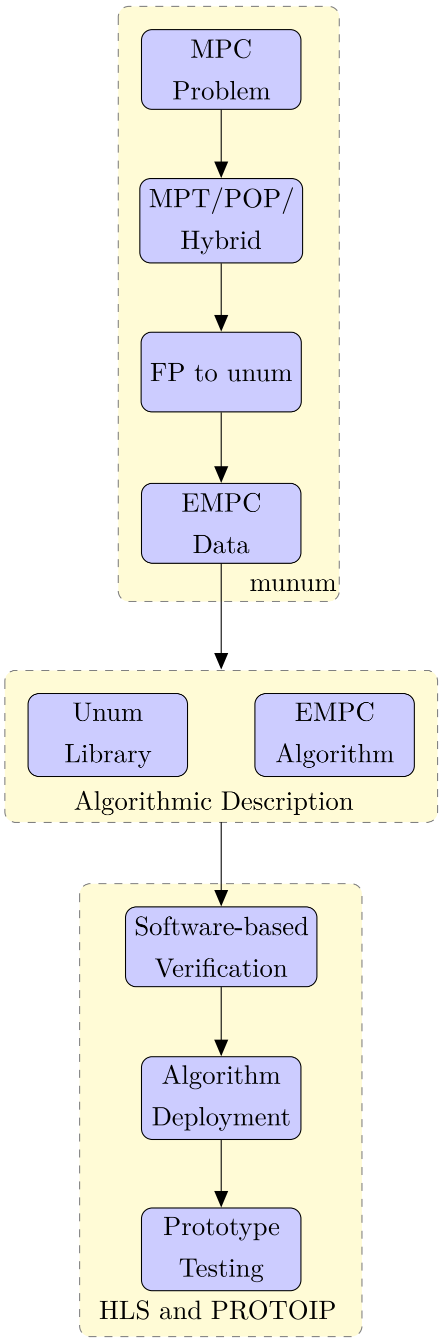

Figure 6.6: Development flow of unum-based EMPC on FPGA.

- Prototyping: In this step, the developed explicit MPC algorithm is first verified in software (Xilinx Vivado HLS) with the help of PROTOIP toolbox interfaced with MATLAB (see Fig. 6.7). For the algorithm verification in software, we kept process model in MATLAB and at each sample time initial conditions (current state and reference) were sent to Vivado HLS which computes next optimal control actions and send it back to MATLAB to get new initial conditions. As a result of software verification, we obtained the synthesis report of memory and resource utilization. After software verification step, next step is to deploy synthesized code on the FPGA board. FPGA device is programmed (.BIT file) through JATG cable with appropriate jumper settings. After the device is programmed, prototype testing is carried out by HIL co-simulation (see Fig. 6.8) where data is recorded in host PC via Ethernet interface by means of TCP/IP packets. A TCP/IP server bridges the communication between the physical Ethernet interface, the DDR memory that is used as a shared memory space and the EMPC algorithm. On the other side, the host PC runs a TCP/IP client accessible via a MATLAB function provided by PROTOIP. For HIL co-simulation the stimulus (model states) were generated through MATLAB and sent stimulus data to the FPGA and the optimal control input was read from FPGA.

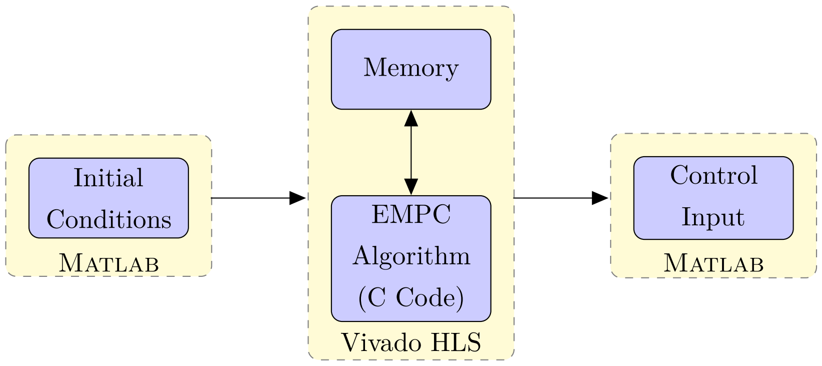

Figure 6.7: Design flow of unum-based EMPC on FPGA.

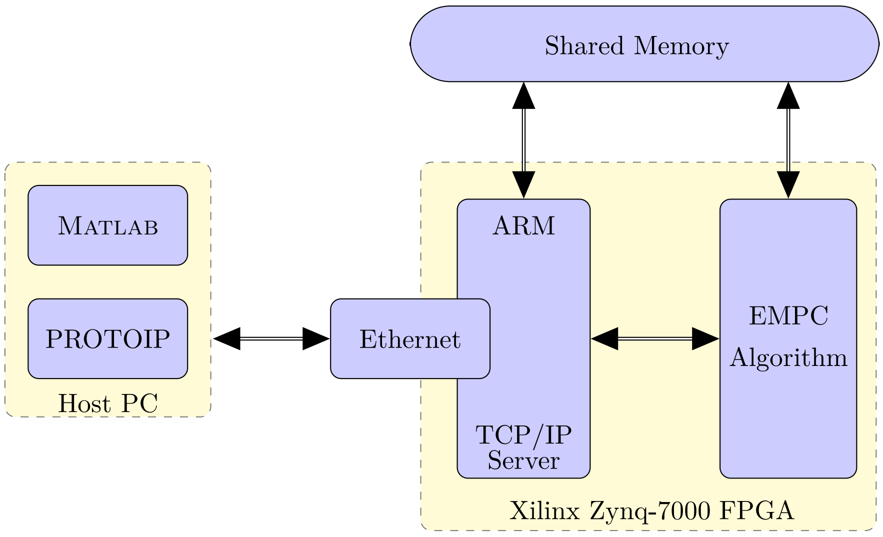

Figure 6.8: HIL co-simulation setup built by PROTOIP using the Xilinx Vivado FPGA tool chain.

6.4 Summary

This chapter contains main contributions of the thesis, i.e., embedded implementation of unums. This chapter describes toolboxes created for unum and its arithmetic. The first toolbox is developed is open-source MATLAB-based unum toolbox, i.e., `munum`. Section 6.1 describes details of toolbox features and implementation of unum-based explicit MPC in general. Taking into consideration that embedded devices mainly supports low level C/C++ languages, we developed an open-source unum toolbox based on C/C++ languages i.e. `cunum`. Section 6.2 describes features of `cunum` toolbox and implementation details of unum-based explicit MPC for generic control problems. The last section of this chapter describes FPGA implementation of unum arithmetics and explicit MPC. A short introduction to FPGA technology and its design flow is given. Overall this chapter describes the implementation of unum toolboxes and FPGA implementation. Next chapter is dedicated to the case study and results.

Chapter 7

Case Study and Results

Having introduced the theoretical background of the universal number-based explicit model predictive control strategy in Chapter 5 and its implementation on various platforms in Chapter 6, we will now focus on the control application. We present one case study in which unum-based explicit MPC is employed for constrained control. The case study is about anesthesia control problem which is presented with process model, control objectives, MPC problem, and closed-loop simulation results. We show the closed-loop simulation results of floating-point format-based and unum-based explicit MPC implemented on C/C++ application and FPGA platform. Furthermore, the comparison results of computational complexity, optimality, and run-time are presented.

New surgical procedures, increasing prevalence of day surgery and pressure to deliver "value for money" all influence the choice of drugs and techniques for anesthesia. Advanced monitoring of drug effect might help to optimize quality of drug delivery, possibly reduce costs and improve patient outcomes.

Anesthesia is a balance between the amount of anesthetic drug(s) administered and the state of arousal of the patient. In conventional practice, anesthesiologists used to decide initial drug dose by taking into account the patient's physical characteristics, such as gender, age, weight, and height. During the maintenance phase, anesthetists use to regulate drug dose according to the patient's physiologic status such as blood pressure, heart rate, and breathing. Even though hospitals have experienced and skilled anesthesiologists, over and under-dosing of the anesthetic drug can occur. In other words, traditional ways of regulating the drug can violate the constraints associated with patient's safety and health. An automatic control system which can regulate drug infusion rate based on the anesthetic level can potentially improve the quality of surgical operations, patient's safety, and reduce clinician's workload. However, to design anesthesia closed loop system, a reliable mathematical model of a patient to observe the dynamics of the drug in the body is required. In addition to the patient model, dedicated hardware to run the algorithms, Depth of Anesthesia (DoA) monitor and actuators are the key components of closed loop control system (Bibian et al., 2003).

In past years, efforts have been made to design and implement anesthesia closed loop control system considering various electroencephalography (EEG) indices to overcome traditional clinical practices to secure patient safety. At the beginning of 20th century, classical control schemes designed for this application are mainly based on fixed gain controllers such as Proportional-Integral (PI) and Proportional-Integral-Derivative controller (PID) (Ejaz and Yang, 2004) and more recently in Padula et al. (2017). Knowledge-based control system using fuzzy logic was implemented in Méndez et al. (2016) considering Bispectral Index (BIS) as a measure of DoA. These controllers achieve suboptimal performance in case of Multi-Input Multi-Output (MIMO) system, variable time delay systems, robustness, stability, and constraint handling. Therefore, model-based advanced control strategies, like General Predictive Control (GPC) (Bamdadian et al., 2008) and model predictive control (Yelneedi et al., 2009), (Naşcu et al., 2015), (Chang et al., 2015) which performs well in case of inter-variability patient model, disturbance rejection and constraints satisfaction are attempted recently and can be the attractive choice for safety-critical applications including anesthesia, diabetes and artificial pancreas control.

The design of a model predictive controller for regulating anesthesia requires a reliable mathematical model of the patient to represent anesthesia dynamics and also, appropriate hardware devices to measure and monitor the depth of anesthesia. In order to realize a desirable hypnosis control, an effort has been taken to establish a reliable patient model that relates the drug inputs to the outcomes (Schüttler and Ihmsen, 2000), (Sawaguchi et al., 2008). Due to significant deviations in physical conditions, age, weight, metabolism, pre-existing medical conditions, and surgical procedures, patient dynamics demonstrate non-linearity and large variations in their responses to drug infusion. One potential remedy is to individualize the patient model in real-time based on the individual clinical data collected during a procedure. Good collection of various models used in research can be found in Furutani et al. (2015).

When modeling biological systems for drug distribution, several methods are common and all have their significance. The pharmacology of anesthetic drugs includes linear pharmacokinetic effects as well as non-linear pharmacodynamic effects. Pharmacokinetics represent the dynamic process of drug distribution in the body while pharmacodynamics represents the description of drug effect on the body. Three main forms of models are mostly used for modeling anesthesia patient, i.e., empirical, compartmental, physiological models, and all share many characteristics of the other representations. Empirical models are black box model, which relate the inputs to outputs by analytical expressions, such as the sums of exponentials. Compartmental models are formulated on the basis of the minimal number of compartments that adequately fits observed data. Physiologically based models are the most realistic representation of drug kinetics because the parameters relate directly to physiology, anatomy, and biochemistry. Out of all the modeling options available, the standard modeling paradigm that has been commonly used is compartmental models.

7.1 Compartmental Models

Compartmental models are mainly based on the assumption that different parts of the body can be represented by virtual compartments disregarding the physical properties of the described tissues. Standard compartmental models often consist of two interacting parts: a pharmacokinetic compartment model and a pharmacodynamic model.

7.1.1 Pharmacokinetic Modeling

As for the Pharmacokinetic (PK) model, we use four-compartment model including the Propofol (drug used in anesthesia) effect-site compartment based on the large-scale multi-center study by Schüttler and Ihmsen in Schüttler and Ihmsen (2000) which is further extended in Sawaguchi et al. (2008). The model's parameters were determined based on patient's real-time data. This model incorporates the patient's age and body weight (BW), so it can take individual differences into account to a certain extent.

Fig. 7.1 shows the three-compartment model with an effect-site compartment. It consists of central, shallow peripheral (fast), deep peripheral (slow), and virtual compartment regarded as effect-site. Peripheral compartment comprises muscle, fat, and other organs and tissues of the body which are metabolically inert as far as the drug is concerned. Shallow peripheral compartment represents tissues with a rich blood supply, and deep peripheral compartment represents tissues with inferior blood supply. Here, x_i is the concentration of Propofol in compartment

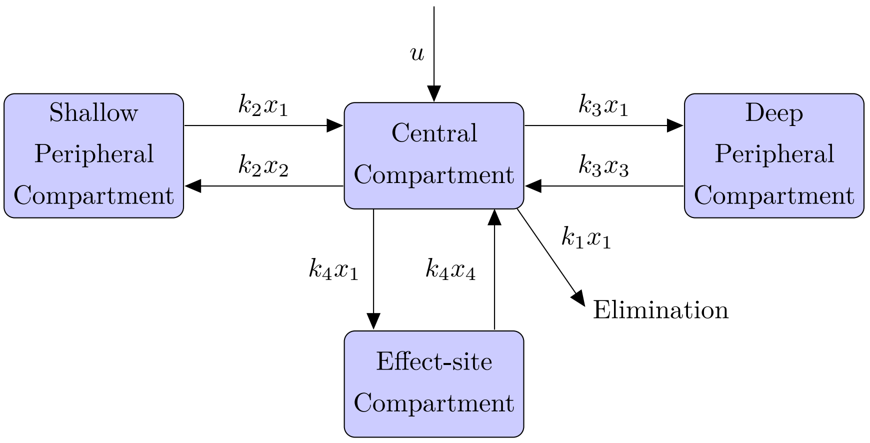

Figure 7.1: Three-compartment pharmacokinetic model with the effect-site compartment.

i; compartments $1, 2, 3$ and 4 correspond, to the central, shallow peripheral, deep peripheral and effect-site compartments, respectively. In addition, u is the infusion rate of Propofol drug, and k_i and V_i are the clearance and volume of the compartment, respectively, given as functions of the patient's age and weight, as given in Table 7.1. The drug is infused in central compartment and then distributed to the slow and fast compartment and eliminated through metabolism. The mass balance in the different compartments are described below:

Central Compartment

The central compartment is the volume in which initial mixing of the drug occurs, and thus can be thought to include the vascular system (blood volume) and for some drugs the interstitial fluid as well as highly perfused organs such as heart, brain,

Table 7.1: Pharmacokinetic parameter values given as a function of patient's age(years) and BW(Kg)(Schüttler and Ihmsen, 2000).

Parameter	**Value**
k_1	$0.0595\text{BW}^{0.75}$ L/min (age ≤ 60)
	$(0.0595\text{BW}^{0.75} - 0.45\text{age} + 2.7)$ L/min (age > 60)
k_2	$0.0969\text{BW}^{0.62}$ L/min
k_3	$0.0889\text{BW}^{0.55}$ L/min
k_4	0.12 L/min
V_1	$1.72\text{BW}^{0.71}\text{age}^{-0.39}$ L
V_2	$3.32\text{BW}^{0.61}$ L
V_3	266 L
V_4	$0.01V_1$ L

kidney, and liver. The concentration of drug within the central compartment is given by:

$$V_1\frac{dx_1}{dt} = -(k_1 + k_2 + k_3 + k_4)x_1 + k_2x_2 + k_3x_3 + k_4x_4 + u. \tag{7.1}$$

Shallow Peripheral Compartment

The shallow or fast peripheral compartment represents a compartment of the body that absorbs drug rapidly from the central compartment, and thus can be thought of as comprising tissues of the body that are well-perfused (such as muscles and vital organs). The concentration of Propofol within the shallow peripheral compartment is given by:

$$V_2\frac{dx_2}{dt} = k_2x_1 - k_2x_2. \tag{7.2}$$

Deep Peripheral Compartment

The deep or slow peripheral compartment is used to represent mathematically, a compartment into which re-distribution occurs more slowly, and thus can be thought of as including tissues with a poor blood supply (such as adipose tissue). Re-distribution of drug in deep tissue is given by

$$V_3 \frac{dx_3}{dt} = k_3 x_1 - k_3 x_3. \tag{7.3}$$

Effect-Site Compartment

It is virtually created compartment to measure the effect of the drug on cardiac output and cerebral blood flow. The rate of plasma/effect-site equilibration depends on factors that determine the rate of drug delivery to the effect-site and pharmacological properties that determine the rate of drug transfer across the blood–brain barrier (lipid solubility, the degree of ionization, etc.). The time course of effect-site equilibration can be mathematically described by a first-order rate constant typically referred to as the k_4. The effect-site compartment accounts for the equilibration time between targeted plasma drug concentration and central nervous system (brain) concentration. The effect-site concentration and targeted plasma drug concentration are related by a first-order lag given as follows (Sawaguchi et al., 2008), (Yelneedi et al., 2009).

$$V_4 \frac{dx_4}{dt} = k_4 x_1 - k_4 x_4. \tag{7.4}$$

The pharmacokinetic model of the patient can be described in state-space form as

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{7.5a}$$

$$y(t) = Cx(t). \tag{7.5b}$$

where $x(t) \in \mathbb{R}^{n_x}$ is the vector of Propofol concentrations at the current time instant, $u(t) \in \mathbb{R}^{n_u}$ is the infusion rate of Propofol, $y(t) \in \mathbb{R}^{n_y}$ is the effect-site concentration. System matrix $A \in \mathbb{R}^{n_x \times n_x}$, input matrix $B \in \mathbb{R}^{n_x \times n_u}$ and output matrix $C \in \mathbb{R}^{n_y \times n_x}$ are the pharmacokinetic parameters given as

$$A = \begin{bmatrix} -\frac{k_1+k_2+k_3+k_4}{V_1} & \frac{k_2}{V_1} & \frac{k_3}{V_1} & \frac{k_4}{V_1} \\ \frac{k_2}{V_2} & -\frac{k_2}{V_2} & 0 & 0 \\ \frac{k_3}{V_3} & 0 & -\frac{k_3}{V_3} & 0 \\ \frac{k_4}{V_4} & 0 & 0 & -\frac{k_4}{V_4} \end{bmatrix}, \quad B = \begin{bmatrix} \frac{1}{V_1} \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

$$C = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}.$$

7.1.2 Pharmacodynamic Modeling

As discussed in above section, the PK model is limited to the kinetics of elimination and distribution of the drug. To reflect the observed clinical effect of the drug, another set of mathematical description is required. These are known as pharmacodynamic models. These are used to describe the relationship between drug concentration and the observed clinical effect; effect signals may be any number of patient vital signs, EEG signals, or blood glucose levels, as examples. In this case, it is frequently said that PD models address “what the drug does to the body”. These models are typically given by static non-linear functions, which are used to describe the equilibrium relationship between the drug concentration (x_4), and drug effect E as shown in Fig. 7.2. In this work, we are using BIS as a measure of DoA. A commonly used pharmacodynamic model structure is given by the well-known Hill equation:

$$E(t) = E_0 - E_{\max}\frac{y(t)^{\gamma}}{y(t)^{\gamma} + c_{50}^{\gamma}}, \tag{7.6}$$

where E_0 is the BIS value before starting the Propofol infusion, $E_{\max}$ is the change of the BIS index corresponding to the infinite Propofol concentration, C_{50} is the effect-site concentration corresponding to $\frac{E_{\max}}{2}$, and γ is the Hill’s coefficient. In this paper, we assume $E_{\max} = E_0$. The control objective is to manipulate the Propofol infusion rate such that the BIS index tracks a prescribed reference. In the above model, we assumed baseline value equal to maximal output value and default values of C_{50} and γ are taken from Sawaguchi et al. (2008).

The pharmacodynamic model typically describes the nonlinear dynamics of BIS, Mean Atrial Pressure (MAP) and Heart Rate (HR) to the effect-site concentration $y(t)$. For the control purpose, BIS is taken as key measure of DoA (reference for controller) whereas MAP and HR are used to observe the corresponding blood pressure and heart rate during surgery. Hill’s Sigmoid E_{max} model for BIS index is given as

$$\mathrm{BIS}(t) = \mathrm{BIS}_0 - \mathrm{BIS}_{\max}\frac{y(t)^{\gamma}}{y(t)^{\gamma} + c_{50}^{\gamma}}, \tag{7.7}$$

where BIS_0 is the base value at no-drug which is around 100, $\mathrm{BIS}_{\max}$ is the maximal-effect intensity. BIS is an EEG-derived index which indicates the effect of drug on the body and it is measured on the scale of $0-100$, see Fig. 7.3. BIS values near 100 represent an “awake” clinical state while 0 denotes the maximal EEG effect possible (i.e., an isoelectric EEG) which means the patient is in the dead state. In general

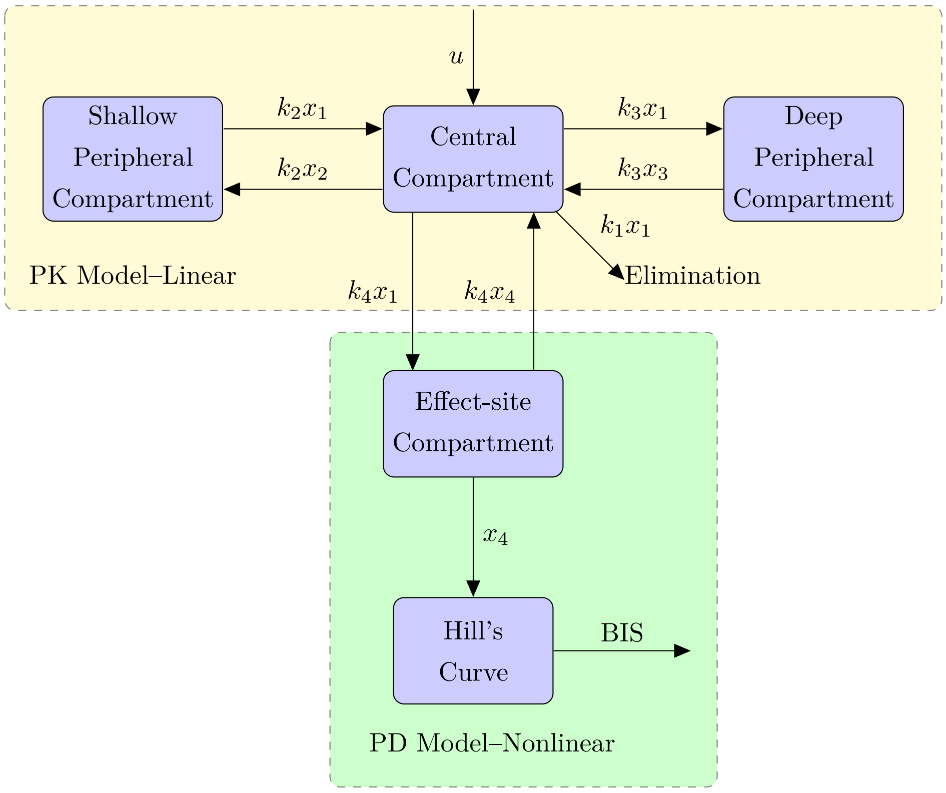

Figure 7.2: Three-compartment pharmacokinetic model with the effect-site compartment and pharmacodynamic model.

surgery practice, the BIS value is maintained in the range of $40 - 60$ to ensure adequate hypnotic effect during balanced general anesthesia while improving the recovery process. Drug doses below 60 is regarded as under dose where the patient can respond to surgical stimuli and can feel pain. BIS index values lower than 40 signify a greater effect of the drug on EEG of a patient and drug dose is regarded as an overdose.

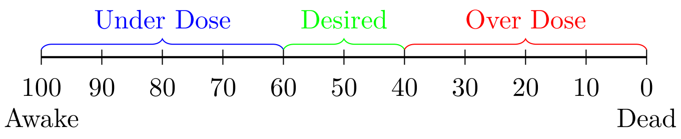

Figure 7.3: BIS scale to indicate the level of DoA in the patients.

Patient's Mean Arterial Blood Pressure (MAP) can also be taken as a measure of anesthesia, Hill's Sigmoid model for MAP is given as

$$\mathrm{MAP}(t) = \mathrm{MAP}_0 - \mathrm{MAP}_{\max}\frac{y(t)^\gamma}{y(t)^\gamma + c_{50}^\gamma}, \tag{7.8}$$

where, MAP_0 and $\mathrm{MAP}_{\max}$ are the baseline value at zero input which is on average 100 and the maximum value at infinite input respectively. MAP has to be maintained in the desired limit for heart problem patients.

Third important index is heart rate, which has opposite dynamics than BIS and MAP. This model exhibits a mild increase in heart rate to increase in the effect-site concentration. The HR model is described as

$$\mathrm{HR}(t) = \mathrm{HR}_0 + \mathrm{HR}_{\max}\frac{y(t)^\gamma}{y(t)^\gamma + c_{50}^\gamma}, \tag{7.9}$$

where HR_0 is the output at zero input and $\mathrm{HR}_{\max}$ stands for output at maximal possible input. On an average HR_0 is 55.

7.2 Drug Delivery in Anesthesia Control

From the perspective of the control system, we can distinguish drug delivery in three types as described below

7.2.1 Open-Loop

The basic procedure is the open-loop practice in which the anesthetist, according to the parameters of the patient (age and weight) directly uses predefined infusion rates of drugs. According to the response observed through his vital signs, the drug rates can be modified. Here the anesthetist act like the controller.

7.2.2 Target Controlled Infusion (TCI)

In TCI the infusion rate is calculated from models of the pharmacokinetic of the patient, as shown in Figure 7.4. Thus, the objective in TCI is to achieve a pre-set

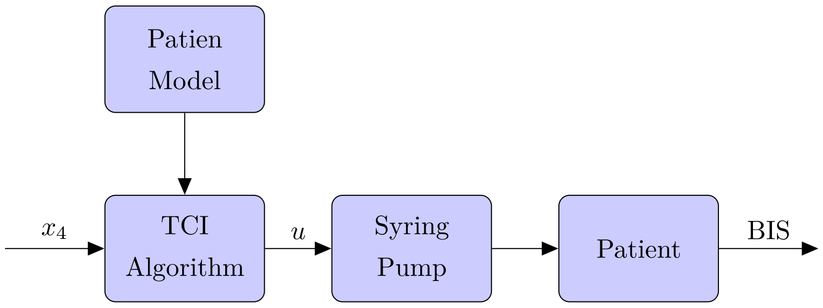

Figure 7.4: Block diagram of the target controlled infusion system.

target plasma concentration x_4. According to the model of the patient, the TCI system (normally implemented in the infusion pump) delivers the adequate drug doses to achieve the objective. There is a clear weakness in TCI related to the fact that the real plasma concentration cannot be on-line measured to compute the infusion rate. That is, TCI is also an open-loop control strategy.

7.2.3 Closed-Loop

The main idea in closed-loop control is to use information about the state of the patient to adjust the drug dosing automatically . Fig. 7.5 shows a schematic view of a closed-loop control scheme for drug delivery in anesthesia.

The control plant is the patient, the actuators are syringe pumps, and the sensors are of several kinds, e.g., BIS, ECG, blood oxygenation, or temperature. The dashed lines and arrows depict indirect actions, while the solid lines and arrows depict direct actions. As an example, the anesthesiologist can only observe the

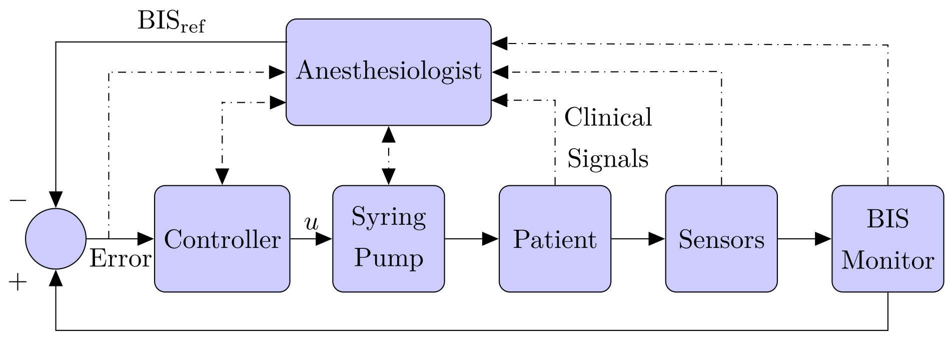

Figure 7.5: Schematic view of a closed-loop control scheme for drug delivery in anesthesia.

tracking error, but cannot exert a direct impact to change it. The anesthesiologist's action on the error has to be indirect via changes of the set-points of the measured effect. The variables and quantities depicted by the arrows can be vector-valued, means, e.g., several sensors are placed on the patient, so that several effect measurements are performed.

7.3 Problem Set-up

7.3.1 Control Objective

The objective of MPC controller is to track desired reference taking into account the physical constraints like drug delivery rate. To achieve said objective, we are using PK-PD model (described in Section 7.1) of the patient to get the effect of the drug on the body. Infusion rate is calculated based on the BIS value and MAP and HR are observed to keep patient at low-risk. The value of $\gamma = 1.87$ and $c_{50} = 3.75$ were taken from Sawaguchi et al. (2008) and kept same to obtain the values of BIS (7.7), MAP (7.8), and HR (7.9). Considering the control objective in the following, we show the MPC problem set-up.

7.3.2 MPC Problem Set-up

In this section, the unum-based explicit MPC is constructed using MPT and `munum`. The control objective is to track the desired BIS reference without violating the input constraints. The continuous time PK model of a patient having age 25 years and weight 60 kg is discretized with sampling time $T_\mathbf{s} = 60$ s. The discrete time state space PK model is given by

$$x_{k+T_\mathbf{s}} = Ax_k + Bu_k \tag{7.10a}$$

$$y_k = Cx_k \tag{7.10b}$$

with

$$A = \begin{bmatrix} 0.0139 & 0.1325 & 0.3076 & 0.0001 \\ 0.0295 & 0.3214 & 0.2416 & 0.0003 \\ 0.0104 & 0.0366 & 0.8741 & 0.0001 \\ 0.0140 & 0.1342 & 0.3074 & 0.0001 \end{bmatrix},$$
$$B = \begin{bmatrix} 0.4255 & 0.3175 & 0.0614 & 0.4243 \end{bmatrix}^T,$$
$$C = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}.$$

Considering the discrete PK model (7.10), the reference tracking problem is formulated as a constrained optimization problem with constraints on the drug input rate as given below,

$$\min_{U} \sum_{k=0}^{N-1} (y_k - y_\mathrm{r})^T Q (y_k - y_\mathrm{r}) + \Delta u_k^T R \Delta u_k \tag{7.11a}$$

$$\text{s.t.}\quad x_{k+T_s} = Ax_k + Bu_k, \quad k = 0, \dots, N-1, \tag{7.11b}$$

$$y_k = Cx_k, \quad k = 0, \dots, N-1, \tag{7.11c}$$

$$\Delta u_k = u_k - u_{k-1}, \quad k = 0, \dots, N-1, \tag{7.11d}$$

$$0 \leq u_k \leq 20, \quad k = 0, \dots, N-1, \tag{7.11e}$$

$$u_{-1} = u(t - T_\mathbf{s}), \tag{7.11f}$$

$$x_0 = x(t), \tag{7.11g}$$

where weighting matrices Q and R was set to 1 and 0.001, respectively.

7.4 Simulation Results: C Implementation

In this section, the software-based results of floating-point and unum-based explicit MPC is presented. The anesthesia control problem for reference tracking was constructed in MATLAB using multi-parametric toolbox for above settings (T_s, Q, R).

The constructed mp-QP problem was subsequently exported in low-level C language routines using `munum` for several prediction horizons ($N = 2, \dots, 15, 20$) with sequential search point location method described in Algorithm 1. The exported code is then synthesized in C application software for performing simulations.

Fig. 7.6 shows the performance of explicit MPC for prediction horizon, 5. It shows the responses of effect-site concentration and optimal drug input rate needed to achieve concentration corresponding to the desired BIS value. Fig. 7.7 depicts the response of BIS, MAP, and HR during simulation.

Table 7.2 summarizes the numerical analysis of explicit MPC for different prediction horizons ($N = 2, 3, 4, 5$). It shows the settling-time (τ_s), mean square error (MSE) and integral squared control efforts (ISCE, $\sum \Delta u^2$). It can be clearly seen that the performance of controller goes on increasing as horizon increases which means in ideal case it is better to have controller with long horizon in real-time applications. Unfortunately, it is difficult to deploy long horizon controller on embedded devices due to the lack of on-chip memory to store controller data. In the next, we will show that with the unums it can be possible to deploy long horizon controller on low-end embedded devices to achieve better performance.

Table 7.2: Performance comparison of floating-point based explicit MPC for different prediction horizons.

N	τ_s [min]	MSE [μg/mL]	ISCE [mg/kg/h]
2	37	15.0100	1.5541
3	28	14.7035	1.5199
4	23	14.4796	1.4906
5	18	14.1014	1.4813

7.4.1 Controller Complexity

In this section, we will compare the floating-point and unum-based controller and show their complexity in terms of memory and execution time required for the set of prediction horizons. For the comparison purpose, we considered double precision

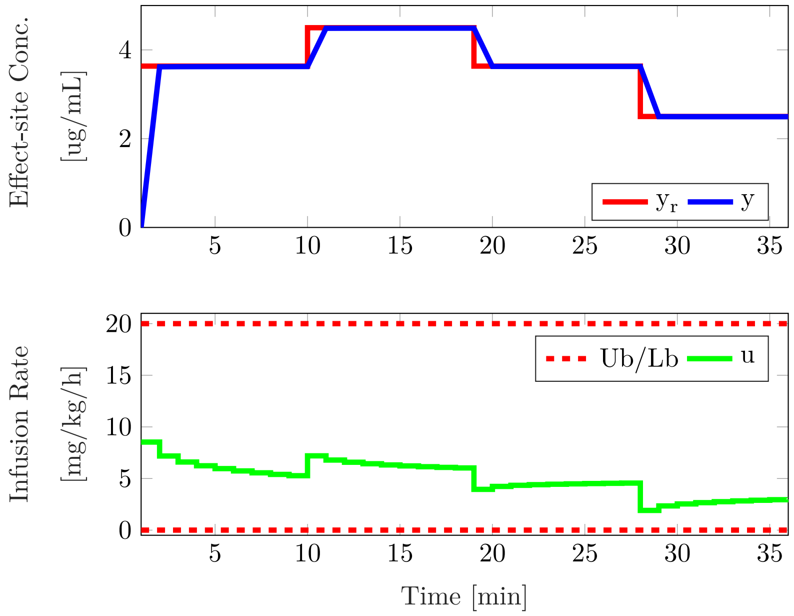

Figure 7.6: Response of effect-site concentration and corresponding drug input rate controlled by explicit MPC.

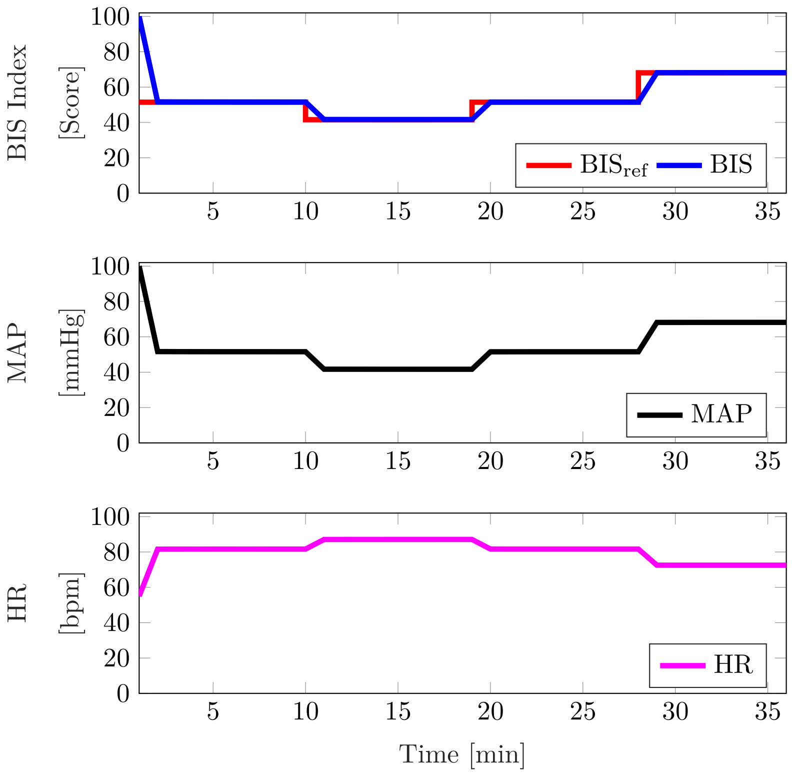

Figure 7.7: Response of measured BIS, MAP, and HR for BIS reference tracking.

floating-point format and universal number format with two *environments*, i.e., unum$\{3, 2\}$ and unum$\{3, 4\}$. All the controllers were exported to C language code using MPT and `munum`. The memory required to store controller data in the form of floating-point format was calculated by (3.14) and for unums it is obtained by calling the function

```
memory_empc(),
```

from `munum`. Table 7.3 shows the number of regions, the total number of FP numbers required to store all the regions, total number of bits as per the format used, total memory required to store all the data and memory savings with unums to store same data. It can be observed from the table that the number of regions and the total number of FP numbers grows exponentially as the length of prediction goes on increasing, as a consequence of that, the number of bits in the floating-point format increases hugely. On the other hand, with the use of unums the number of bits increases slowly. Last two columns of the table shows the memory saving as compared to the FP format and it can be observed that the unum with env$\{3, 2\}$ saves $81 - 84\%$ memory and unum with env$\{3, 4\}$ saves $70 - 78\%$ memory. Interesting to notice is that the byte size of the unum-encoded solution for $N = 10$ (97.86 kB and 724 regions) is roughly equal to the size of floating-point representation for $N = 6$ (96.14 kB and 133 regions). It follows that by using unums, one can fit more data into the same space and thus can be able to use larger horizons in MPC. The unum based controller takes on an average 11-bits in env$\{3, 2\}$ and 18-bits in env$\{3, 4\}$ to represent each number. Roughly, we can store up to 24 kB on low-end PLCs where it is possible to store up to 30 regions with the floating-point format. But, unum gives freedom to store up to 120 regions which means one can implement the controller with $N \geq 5$.

7.4.2 Execution Time and Optimality

This section presents the compression of execution time taken by both the number formates when implemented in low-level C language using `cunum` prototype of the unum. As computing hardware, a personal computer with an Intel Core i7 CPU with 2 GHz processor and 8 GB memory was used. As an operating system and compiler, 64-bit Windows 7 with `Cygwin` were used. The exported explicit MPC controllers with different horizons from MPT and `munum` were executed in C environment for several initial conditions and references. Table 7.4 summarizes the

Table 7.3: Comparison of memory footprints between double precision floating- point and universal number-based explicit MPC.

N	Number of	Total	Number of Bits			Memory [kB]			Memory Savings [%]	
	Regions	Numbers	FP (64-bits)	unum$\{3,2\}$	unum$\{3,4\}$	FP (64-bits)	unum$\{3,2\}$	unum$\{3,4\}$	unum$\{3,2\}$	unum$\{3,4\}$
2	3	272	17472	2874	3934	2.13	0.35	0.48	83.55	77.48
3	9	819	52416	8969	12989	6.39	1.09	1.58	82.88	75.21
4	27	2457	157248	28022	42710	19.19	3.42	5.21	82.18	72.84
5	65	5999	383936	69826	108736	46.86	8.52	13.27	81.81	71.68
6	133	12306	787584	144972	228414	96.14	17.69	27.88	81.59	71.00
7	230	21378	1368192	253257	400997	167.01	30.91	48.95	81.49	70.69
8	359	33362	2135168	196542	629483	260.64	48.40	76.84	81.43	70.52
9	522	48496	3103744	577606	918481	378.87	70.51	112.12	81.39	70.41
10	724	67027	4301248	801691	1276449	525.05	97.86	155.81	81.36	70.32
11	955	88697	5676608	1059128	1687326	692.94	129.29	205.97	81.34	70.27
12	1029	112301	7187264	1341730	2138614	877.35	163.78	261.06	81.33	70.24
13	1501	139321	8916544	1665661	2656034	1088.40	203.33	324.22	81.32	70.21
14	1816	168581	10789184	2015791	3214285	1317.00	246.07	392.37	81.32	70.20
15	2153	199843	12789952	2389026	3808257	1561.30	291.63	464.87	81.31	70.22
20	4176	388213	24845632	4642363	7400568	3032.90	566.69	903.39	81.31	70.22

average execution time in ms for double precision floating-point and unum format. It is interesting to notice that the time taken by FP controller remains almost same for all the prediction horizons whereas for unum controller it goes on an increase and this is because of the unum arithmetic. Table 7.5 enlists the optimal values (up to 4 decimal of accuracy) obtained by floating-point format and unum-based explicit MPC controller for different prediction horizons. It can be seen from the table that the values obtained using unum are 100% optimal as compared to double precision FP. Also, to get the same accuracy as double FP unum needs only 33-bits, i.e., $\{3, 4\}$ which almost half of the 64-bits which means using unum we can get the same accuracy with fewer bits.

7.5 HIL Co-Simulation Results: FPGA Implementation

This section deals with the hardware implementation results of floating-point and unum-based explicit MPC for anesthesia control problem (7.11). For hardware implementation we use the sequential search algorithm and controller data were exported in the C code from MPT (FP data) and `munum` toolbox (unum data) from inside the MATLAB. To synthesis unum-based controller we used unum arithmetic library developed for hardware implementation as discussed in Section 6.3.

7.5.1 Floating-point Explicit MPC

The design flow for implementing double precision FP explicit MPC on FPGA is similar to that of unum-based EMPC implementation. For implementation, we kept control problem same as in (7.11) with same settings for Q and R. The exported C code from MPT were deployed on FPGA using Vivado HLS and PROTOIP toolbox. For floating-point arithmetic, Vivado supports standard math library (math.h) which is optimized for hardware. The EMPC codes were synthesized using synthesis tool and verified in software. Then the controller was employed in HIL co-simulation to track the output of anesthesia patient model running inside MATLAB.

Table 7.4: Comparison of execution-time taken by floating-point (64-bit) and universal number-based-explicit MPC.

	Execution Time [ms]		
N	FP (64-bit)	unum$\{3,2\}$	unum$\{3,4\}$
2	1.03	8.10	10.40
3	0.95	23.4	11.40
4	1.09	28.00	13.60
5	1.02	15.50	16.90
6	0.89	19.60	18.20
7	0.81	21.00	26.80
8	0.94	25.30	27.10
9	1.03	34.00	38.00
10	1.10	47.10	51.30
11	2.77	63.40	67.89
12	2.08	70.10	68.50
13	2.17	94.60	88.90
14	1.78	92.60	93.50
15	1.53	99.20	104.80
20	1.80	100.00	110.45

Table 7.5: Comparison of optimal values obtained by floating-point (64-bit) and universal number-based explicit MPC.

	Optimal Values [mg/kg/h]		
N	FP (64-bit)	unum$\{3, 2\}$	unum$\{3, 4\}$
2	8.5616	(8,9)	(8.5615,8.5617)
3	8.5608	(8,9)	(8.5605,8.5609)
4	8.5607	(8,9)	(8.5605,8.5612)
5	8.5606	(8,9)	(8.5606,8.5616)

7.5.2 Unum-based Explicit MPC

The unum EMPC is implemented on FPGA using env $\{3, 2\}$ which saves almost 14 bits for each number as compared to env $\{3, 4\}$. The exported data from `munum` in the form of env $\{3, 2\}$ is then stored on FPGA memory using unum datatype which actually stores unum sub-fields in required bits, and it varies as per the individual number. As we set env $\{3, 2\}$ for each number, maximum number of bits allowed for number storage is limited to 19. The unum explicit MPC algorithm along with the unum data was synthesized in Vivado HLS tools from MATLAB using PROTOIP toolbox. Then the synthesized algorithm was deployed on FPGA (Zed board) using JTAG cable. Next, the standalone unum-based EMPC algorithm were used in closed-loop HIL co-simulation for reference tracking problem of anesthesia control.

Now, we will show the comparison of memory and resource utilization for both implementations for same settings of Q and R.

7.5.3 Memory Comparison

For memory comparison, several explicit controllers were stored on FPGA memory with different values of prediction horizons $(2, 3, \ldots, 8)$. Table 7.6 shows the number of block RAMs used for different prediction horizons. On the FPGA data is stored in Block RAM (BRAM) which is a function of the configuration parameters for:

memory address range, number of byte-write enables, the data width, and the targeted architecture. A ZedBoard comprised of total 280 blocks of BRAM where each block is of 18kbits. The controller data is stored in arrays which are then mapped by FPGA to store it on BRAMs. Fig. 7.8 shows the bar chart of BRAM

Table 7.6: Comparison of memory taken by floating-point (64-bit) and universal number based-explicit MPC.

	Block RAM Utilization [#]	
N	FP (64-bit)	unum$\{3, 2\}$
2	13	5
3	27	12
4	39	17
5	79	40
6	151	76
7	163	91
8	262	120

usage for FP and unum controller. It can be seen that the data stored in double precision FP form needs more number of BRAMs, i.e., memory as compared to unum data for each value of prediction horizon. It is interesting to observe that the actual utilization of memory on FPGA is much more than that of the theoretical memory shown in table 7.3. This is due to the fact that memory shown in the Table 7.6 is total memory used for configuration parameters listed above.

7.5.4 On-chip Memory and Cost Relationship

As process technologies continue to shrink and memory size and design complexity grow, it has become increasingly difficult to achieve high manufacturing yield. Embedded memories are the densest components within a system-on-chip (SoC), accounting for more than 50% of the chip area. Implemented using aggressive de-

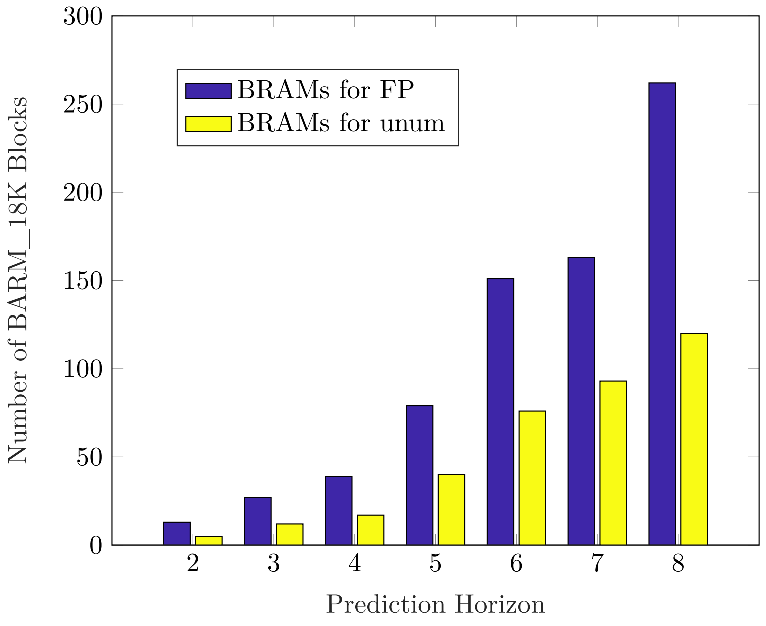

Figure 7.8: Comparison of BRAM utilization for floating-point and unum-based explicit MPC with different prediction horizons.

sign rules, embedded memories tend to be more prone to manufacturing defects and field reliability problems than any other core on the chip. Therefore, the overall yield of SoC depends heavily on the memory yield, and securing high memory yield is critical to achieving lower silicon cost.

Today's demanding applications require SoCs that are bigger and faster, more area, timing, and power sensitive than ever before, resulting in a shift from the logic-dominant chips of the past to memory-dominant ones. Fig. 7.9 shows embedded memory projections from Semico Research Corporation. In 2008, embedded memories accounted for more than half of the die area in a typical SoC. It predicted that the amount of space they occupy on the die would continue to increase, reaching up to 70% by the end of 2017. This growing percentage is mainly to ever increased demand of performance and higher memory bandwidth requirement to minimize latency (Kaushik and Zorian, 2012). The unum is one of the solutions to reduce memory, bandwidth, and energy. If we design small memory chips, we can save lots of money on this. In general cost of FPGA and microprocessors mainly depends

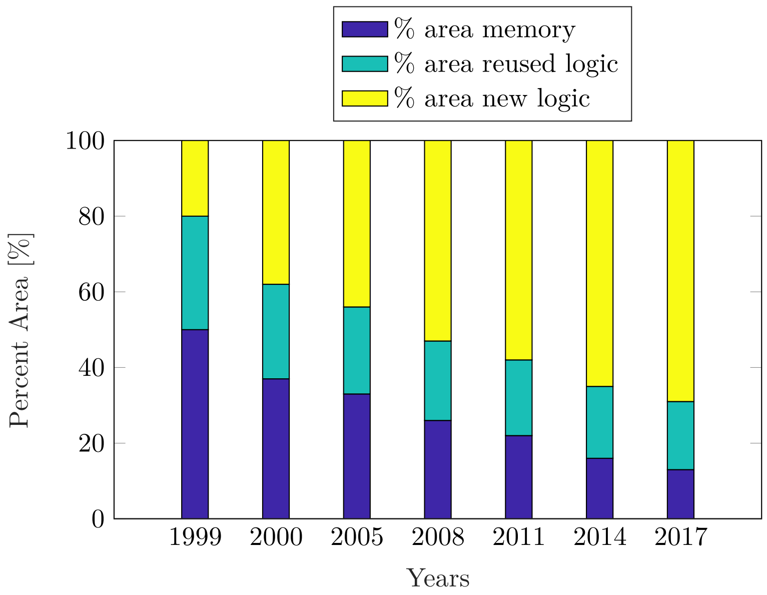

Figure 7.9: Prediction of embedded memories on the die area of a typical SoC devices.

on silicon chip area/size used in that device. The number of chips that can be produced on that wafer depends on the die size: The smaller the die, the more of them can fit onto the wafer. Fig. 7.10 shows the effect of die size on die yield process. On a silicon wafer if we fit many small dies the yield increases, and if die size is large, it decreases.

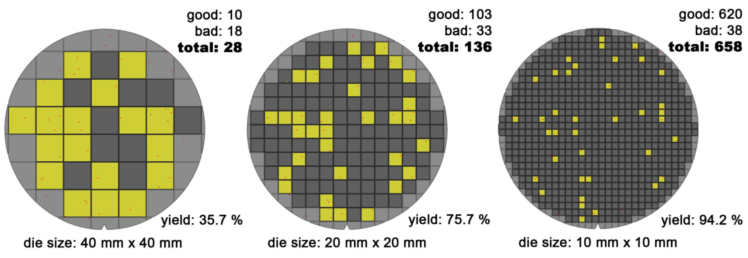

Figure 7.10: Picture of silicon wafer showing working (good) and non-working (bad) die. (Source: www.neogaf.com)

7.5.5 Resource Utilization

The design was synthesized and placed and routed using Xilinx Vivado tools targeting a ZedBoard. Table 7.7 summarize the resource utilization of floating-point and unum-based explicit MPC. The resources utilized for all controllers (with a set of different prediction horizons) are constant as it mainly used for arithmetic units and independent on controller data. Available resources in ZedBoard are, Digital Signal Processor (DSP): 220, Flip-Flop (FF): 106400 and Look-Up Table (LUT):53200. The percentages laid in the table are based on total available resources. It can be seen from the table that floating-point arithmetic takes less resources as compared to unum arithmetic. This is due to the unums require more logic than FP for a hardware implementation, but transistors have become so inexpensive that we welcome anything that gives them useful new things to do. One has to make a trade-off between memory and resources. Fig. 7.11 depicts the trade-off parameters of floating-point and unum-based explicit MPC implementation on FPGA. It's simple either go for less resources which saves less money or go for less memory which will save more money.

Table 7.7: Comparison of resources utilized for floating-point and unum-based explicit MPC implementation of ZedBard.

	Resource Utilization [%]		
	DSP	FF	LUT
FP	7	3	10
Unum	13	67	96

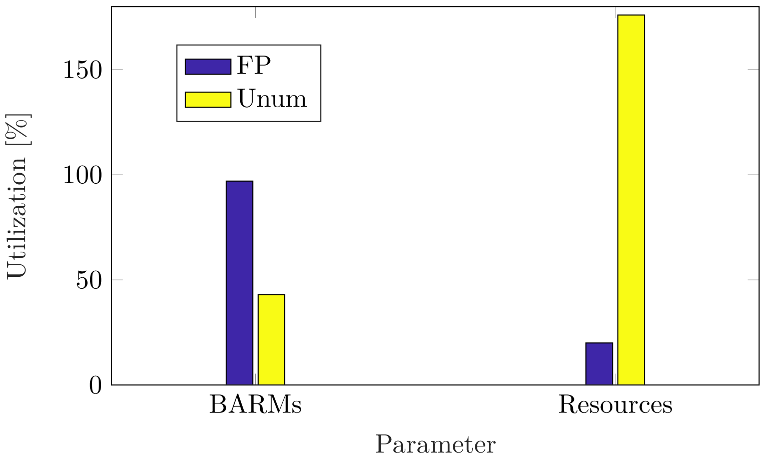

Figure 7.11: Trade-off parameters of explicit MPC implementation of FPGA. (Resources is sum of DSP, FF, and LUT.)

7.6 Summary

In this chapter we have shown memory-efficient implementation of unum-based explicit MPC with anesthesia control problem which was formulated as a MPC problem and explicit MPC problem was constructed in MPT. Consequently, a unum-based explicit MPC is exported in low level C language and simulated in C application for several level of controller complexity. For the C implementation we analyzed memory, run-time and optimality and compared the results with floating-point-based explicit MPC. Then we employed unum-based explicit MPC implemented on FPGA same control problem with different complicity. HIL co-simulation results for floating-point and unum-based explicit are shown. Detailed analysis of BRAM and resource utilization is carried to show tread-off between memory and resource utilization. In next chapter we discuss future research directions and open questions.

Chapter 8

Conclusions and Future Research Directions

8.1 Conclusions

This thesis has proposed a new memory reduction technique for the implementation of explicit model predictive control feedback law on an embedded platform with the objective of increasing the application domain of explicit MPC. The technique is based on representing the controller data by universal number format which takes fewer bits as compared to the “one-size fits all” IEEE-754 floating-point standard. Unum encompasses all standard floating-point formats and gets more accurate answers than floating-point arithmetic with less number of bits, which saves memory and bandwidth. Unlike floating-point numbers, unums make no rounding errors and cannot overflow or underflow. To show the applicability of unum arithmetic in optimization and control field, we have developed two toolboxes, first is `munum` for MATLAB and another `cunum` for algorithms running in C/C++ . With the help of available software tools for explicit MPC, we developed an automatic tool chain to export low-memory unum-based explicit controller in C code. Further, the unum arithmetic and explicit MPC is developed and implemented on FPGA targeting the ZedBoard Zynq-7000 ARM/FPGA SoC Development Board - Xilinx. The feasibility of unums in C application and on FPGA is demonstrated with two case studies. The closed-loop hardware-in-the-loop co-simulation results of unum-based explicit MPC are presented for different prediction horizons. The resulting memory footprints and resource usage are compared with those of the floating-point (double precision)-based explicit MPC approach, and it is observed that the unum-based explicit MPC can reduce memory footprints by $40-50\%$ on FPGA and by 80% in software. Furthermore, the execution time of floating-point and unum controller were compared in C application. With the use of the appropriate `environment` in unum, it is possible to use longer prediction horizons and large system, while achieving lossless closed-loop performance and a significant reduction in memory footprints.

8.2 Future Research Directions

The memory reduction technique presented in this thesis is based on the variable bit size universal number format which is an attractive alternative to one size fits all format, i.e., IEEE floating-point to deal with memory demands in explicit MPC. The work presented here is mainly focused on two things, developing software tools to export controller data in unums and demonstrate the applicability of unums on reconfigurable devices like FPGAs. But, there is definitely more to do on the side of reducing memory using unums. In the following we will list some of the possible future research directions and open problems:

- Continue the studies on large real-time systems with the purpose of verifying the results obtained in this thesis.
- In this thesis, we are mainly focusing on unum implementation on FPGA device, a detailed study on exploring the features of unums on other embedded devices will require.
- The issue with the PLCs and microcontrollers is that there is no flexibility of storing data in variable bits and unum stores data in variable bits. So, to take the advantages of unums one research on new number format which should include nice features (no overflow, no underflow, no rounding, high range, etc.) of unums but store number in a fixed number of bits as in IEEE standard. By doing so, one can target low-cost, low-end embedded hardware.
- In FPGA implementation we are storing controller data in BRAMs which is comprised of fixed size blocks. One can achieve even more memory reduction

by optimally storing the data on each block, so that every BRAM block get fully utilized.

- One can reduce resource utilization by optimizing unum arithmetic codes and parallelism them.
- On the theoretical side, research is required to prove the stability designed controller.
- Unums can be used in other methods of optimization which demands memory storage or more accuracy.
- The comparison of unums with fixed-point format will be needed.
- In the end, the open question is “How many number of bits do we need to get satisfactory closed-loop performance?”.

Appendix A

Author's Publications

Following is the list of publications which I have co-authored during my PhD study. The list is structured based on the categorization of Slovak Accreditation Committee.

- Accreditation Category A, AFC- IFAC World Congress Proceedings

  1. **Ingole D.**, Kvasnica M., De Silva H., Gustafson J., "Reducing Memory Footprints in Explicit Model Predictive Control using Universal Numbers", *In Preprints of the 20th IFAC World Congress*, IFAC, Toulouse, France, vol. 20, pp. 12100–12105, 2017.

- Accreditation Category B, AFC- Conference Proceedings

  2. Dani S., Sonawane D., **Ingole D.**, and Patil S., "Performance Evaluation of PID, LQR and MPC for DC Motor Speed Control", *In Proceedings of International Conference for Convergence in Technology (I2CT)*, IEEE, Pune, India, pp. 1–7, 2017.

  3. **Ingole D.** and Kvasnica M., "FPGA Implementation of Explicit Model Predictive Control for Closed Loop Control of Depth of Anesthesia", *In Preprints of the 5th Conference on Nonlinear Model Predictive Control*, IFAC, Seville, Spain, pp. 484–489, 2015.

  4. Kvasnica M., Holaza J., Takács B., and **Ingole D.**, "Design and Verification of Low-Complexity Explicit MPC Controllers in MPT3. *In Pro-*

ceedings of the 14th European Control Conference (ECC 2015), IEEE, Linz, Austria, pp. 2600-2605, 2015.

- Accreditation Category B, AFD- Conference Proceedings

5. **Ingole D.**, Drgoňa J., Kalúz, M., Klaučo, M., Bakošová, M., Kvasnica M., "Model Predictive Control of a Combined Electrolyzer-Fuel Cell Educational Pilot Plant", *In Proceedings of the 21th International Conference on Process Control*, IEEE, Štrbské Pleso, Slovakia, pp. 147-154, 2017.
6. **Ingole D.**, Drgoňa J., and Kvasnica M., "Offset-Free Hybrid Model Predictive Control of Bispectral Index in Anesthesia", *In Proceedings of the 21th International Conference on Process Control*, IEEE, Štrbské Pleso, Slovakia, pp. 422-427, 2017.
7. Sharma A., Drgoňa J., **Ingole D.**, Holaza J., Valo R., Koniar S., Kvasnica M., "Teaching Classical and Advanced Control of Binary Distillation Column", *In Preprints of the 11th IFAC Symposium on Advances in Control Education*, IFAC, Bratislava, Slovakia, vol. 11, pp. 348–353, 2016.
8. **Ingole D.**, Holaza J., Takács B., and Kvasnica M., "FPGA-Based Explicit Model Predictive Control for Closed-Loop Control of Intravenous Anesthesia", *In Proceedings of the 20th International Conference on Process Control*, IEEE, Štrbské Pleso, Slovakia, pp. 42-47, 2015.
9. Kvasnica M., Takàcs B., Holaza J., and **Ingole D.**,"Reachability Analysis and Control Synthesis for Uncertain Linear Systems in MPT", *In Proceedings of the 8th Symposium on Robust Control Design*, IFAC, Bratislava, Slovak Republic, no. 8, pp. 302–307, 2015.

- Accreditation Category B, AFG- Abstract in Conference Proceedings

10. **Ingole D.**, Drgoňa J., Kalúz, M., Klaučo, M., Bakošová, M., Kvasnica M., "Explicit Model Predictive Control of a Fuel Cell", *In The European Conference on Computational Optimization*, Leuven, Belgium, vol. 4, 2016.

- Miscellaneous

11. Kvasnica M., Holaza J., Takács B., **Ingole D.**, “Design and Verification of Low-Complexity Explicit MPC Controllers in MPT3 (Extended version)”, 2015.
12. Sonawane D., **Ingole D.**, and Naik V., “FPGA implementation of linear model predictive controller for real-time position control of DC motor”, *International Journal of Circuits and Architecture Design*, Inderscience, vol. 1, issue 4, pp. 281-294, 2015.

Appendix B

Curriculum Vitae

Deepak Ingole

Date of Birth: September 22, 1988
Citizenship: Indian
Email: deepak.ingole@stuba.sk
Homepage: http://www.kirp.chtf.stuba.sk/∼ingole

Education

Ph.D., Process Control September 2017 (expected)

- Marie Curie Early Stage Researcher at Slovak University of Technology in Bratislava, Slovakia
  - Project: Training in Embedded Predictive Control and Optimization (TEMPO) a Marie Curie Initial Training Network, October 2014 – September 2017
  - Major: Embedded implementation of explicit MPC
  - Minor: Optimization, control system, embedded systems, and universal numbers

Master of Technology, Instrumentation and Control June 2012

- College of Engineering Pune, India
  - Major: Implementation of active set method for closed-loop control of intravenous anesthesia

Bachelor of Engineering, Instrumentation and Control August 2010

- University of Pune, India
  - Major: Design of PLC-based MU-G10 starter test bench

Research Experience

Researcher October 2014 to present

- Slovak University of Technology in Bratislava, Slovakia
  - Working on predictive control and embedded optimization

Visiting Researcher March 2017 to May 2017

- Imperial College London, United Kingdom
  - FPGA Implementation of unum-based explicit MPC

Visiting Researcher December 2016 to March 2017

- University of Oxford, United Kingdom
  - Implementation of unums in C/C++ for general purpose optimization solver

Project Engineer August 2012 to June 2014

- Virtual Labs Project, College of Engineering Pune, India
  - Developed mathematical model of industry grade pilot plants and their control using predictive controller

Research Interests

- Predictive control, embedded optimization, embedded systems, process control, and anesthesia control.

Awards and Funding

- Xilinx University Program Donation: softwares and development board 2016
- University rector's scholarship for Support of Young Researchers 2015
- Best paper award at *Third International Conference on Control, Communication and Power Engineering*, Springer, Bangalore, India 2012

Computer Skills

- FPGA Tools: Xilinx ISE and Vivado, System Generator, Quartus Prime, ModelSim
- Programming Languages: VHDL/Verilog, C, C++ , MATLAB/Simulink, Julia, Python, HTML
- Toolboxes: YALMIP, MPT, PROTOIP, ACADO Toolkit
- Writing and Drawing: LaTeX, TikZ
- Version Control Tools: GitHub, Bitbucket, RhodeCode
- Project Management Tool: Trello

Trainings and Workshops

- Professional courses on Business Management in Action, ABB University, Baden, Switzerland 2016
- Presentation Skills Workshop, NTNU, Trondheim, Norway 2016
- TEMPO Professional Development, Entrepreneurship, and Complementary Skills Workshop, Imperial College London, UK 2016

- TEMPO Spring School on Theory and Numerics for Nonlinear MPC, University of Freiburg, Germany 2015
- TEMPO Summer School on Numerical Optimal Control and Embedded Optimization, University of Freiburg, Germany 2015
- Automotive Embedded Control Workshop, Renault, Paris, France 2015
- Certificate Course on Embedded System Design, COEP, India 2011
- Industrial Training on Automation, COEP, India 2008

Bibliography

Alamir, M., Murilo, A., Amari, R., Tona, P., Fürhapter, R., and Ortner, P. (2010). On the use of parameterized nmpc in real-time automotive control. In *Automotive model predictive control*, pages 139–149. Springer.

Almurib, H. A., Askari, M., and Moghavvemi, M. (2010). Hard constraints explicit model predictive control of an inverted pendulum. In *Energy, Power and Control (EPC-IQ), 2010 1st International Conference on*, pages 28–32. IEEE.

Ameen, N. A., Galal, B., Kennel, R., and Kanchan, R. (2012). The explicit solution of model-based predictive control by considering the nonlinearities in drive applications. In *Power Electronics and Motion Control Conference (EPE/PEMC), 2012 15th International*, pages DS2a–1. IEEE.

ANSI/IEEE Std 1985 (1985). *IEEE Standard for Binary Floating Point Arithmetic.* IEEE.

Avnet, I. (2014). *Zynq^{TM}Evaluation and Development Hardware User's Guide*, 2.2 edition.

Bamdadian, A., Towhidkhah, F., and Moradi, M. H. (2008). Generalized predictive control of depth of anesthesia by using a pharmocokinetic-pharmacodynamic model of the patient. In *Bioinformatics and Biomedical Engineering, 2008. ICBBE 2008. The 2nd International Conference on*, pages 1276–1279. IEEE.

Bank, B., Guddat, J., Klatte, D., Kummer, B., and Tammer, K. (1982). Non-linear parametric optimization. *Akademie-Verlag, Berlin.*

Baotić, M. (2002). An efficient algorithm for multiparametric quadratic programming. Technical report, ETH, Zurich.

Bayat, F., Johansen, T. A., and Jalali, A. A. (2011). Combining truncated binary search tree and direct search for flexible piecewise function evaluation for explicit mpc in embedded microcontrollers. *IFAC Proceedings Volumes*, 44(1):1332–1337.

Beccuti, A., Papafotiou, G., Frasca, R., and Morari, M. (2007). Explicit hybrid model predictive control of the dc-dcboost converter. In *Power Electronics Specialists Conference, 2007. PESC 2007. IEEE*, pages 2503–2509. IEEE.

Beccuti, A. G., Mariéthoz, S., Cliquennois, S., Wang, S., and Morari, M. (2009). Explicit model predictive control of dc–dc switched-mode power supplies with extended kalman filtering. *IEEE Transactions on Industrial Electronics*, 56(6):1864–1874.

Behrooz, P. (2000). Computer arithmetic: Algorithms and hardware designs. *Oxford University Press*, 19:512583–512585.

Bemporad, A. (2004). Hybrid Toolbox - User's Guide.

Bemporad, A. (2006). Model predictive control design: New trends and tools. In *Decision and Control, 2006 45th IEEE Conference on*, pages 6678–6683. IEEE.

Bemporad, A., Borrelli, F., Morari, M., et al. (2002). Model predictive control based on linear programming˜ the explicit solution. *IEEE Transactions on Automatic Control*, 47(12):1974–1985.

Bemporad, A. and Filippi, C. (2003). Suboptimal explicit receding horizon control via approximate multiparametric quadratic programming. *Journal of optimization theory and applications*, 117(1):9–38.

Bemporad, A., Morari, M., Dua, V., and Pistikopoulos, E. N. (2000). The explicit solution of model predictive control via multiparametric quadratic programming. In *American Control Conference, 2000. Proceedings of the 2000*, volume 2, pages 872–876. IEEE.

Bibian, S., Ries, C. R., Huzmezan, M., and Dumont, G. A. (2003). Clinical anesthesia and control engineering: Terminology, concepts and issues. In *European Control Conference*, pages 2465–2474.

Borrelli, F., Baotić, M., Pekar, J., and Stewart, G. (2009). On the complexity of explicit mpc laws. In *Control Conference (ECC), 2009 European*, pages 2408–2413. IEEE.

Borrelli, F., Bemporad, A., and Morari, M. (2015). Predictive control for linear and hybrid systems, 2015. *preparation, available online at http://www. mpc. berkeley. edu/mpc-course-material.*

Camacho, E. F. and Alba, C. B. (2013). *Model predictive control.* Springer Science & Business Media.

Carver, J. C. (2012). Software engineering for computational science and engineering. *Computing in Science & Engineering*, 14(2):8–11.

Chang, J. J., Syafiie, S., Kamil, R., and Lim, T. A. (2015). Automation of anaesthesia: a review on multivariable control. *Journal of clinical monitoring and computing*, 29(2):231–239.

Christofides, P. D., Scattolini, R., de la Pena, D. M., and Liu, J. (2013). Distributed model predictive control: A tutorial review and future research directions. *Computers & Chemical Engineering*, 51:21–41.

Csekő, L. H., Kvasnica, M., and Lantos, B. (2015). Explicit mpc-based rbf neural network controller design with discrete-time actual kalman filter for semiactive suspension. *IEEE Transactions on Control Systems Technology*, 23(5):1736–1753.

Cychowski, M. T. and O'Mahony, T. (2005). Efficient off-line solutions to robust model predictive control using orthogonal partitioning. *IFAC Proceedings Volumes*, 38(1):129–134.

de la Peña, D. M., Ramirez, D., Camacho, E., and Alamo, T. (2005). Application of an explicit min-max mpc to a scaled laboratory process. *Control Engineering Practice*, 13(12):1463–1471.

de Oliveira, N. and Biegler, L. T. (1994). Constraint handing and stability properties of model-predictive control. *AIChE journal*, 40(7):1138–1155.

D'Errico, J. (2012). High precision floating point arithmetic - a big decimal class. *MATLAB central file exchange.*

Dirscherl, C., Hackl, C., and Schechner, K. (2015). Explicit model predictive control with disturbance observer for grid-connected voltage source power converters. In *Industrial Technology (ICIT), 2015 IEEE International Conference on*, pages 999–1006. IEEE.

Drgoňa, J., Klaučo, M., Janeček, F., and Kvasnica, M. (2017). Optimal control of a laboratory binary distillation column via regionless explicit mpc. *Computers & Chemical Engineering*, 96:139–148.

Drgona, J., Kvasnica, M., Klauco, M., and Fikar, M. (2013). Explicit stochastic mpc approach to building temperature control. In *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, pages 6440–6445. IEEE.

Dua, V., Bozinis, N. A., and Pistikopoulos, E. N. (2002). A multiparametric programming approach for mixed-integer quadratic engineering problems. *Computers & Chemical Engineering*, 26(4):715–733.

Ejaz, K. and Yang, J.-S. (2004). Controlling depth of anesthesia using pid tuning: a comparative model-based study. In *Control Applications, 2004. Proceedings of the 2004 IEEE International Conference on*, volume 1, pages 580–585. IEEE.

El Hadef, J., Olaru, S., Rodriguez-Ayerbe, P., Colin, G., Chamaillard, Y., and Talon, V. (2013). Explicit nonlinear model predictive control of the air path of a turbocharged spark-ignited engine. In *Control Applications (CCA), 2013 IEEE International Conference on*, pages 71–77. IEEE.

Farooq, U., Marrakchi, Z., and Mehrez, H. (2012). Fpga architectures: An overview. In *Tree-based Heterogeneous FPGA Architectures*, pages 7–48. Springer New York.

Feller, C. and Johansen, T. A. (2013). Explicit mpc of higher-order linear processes via combinatorial multi-parametric quadratic programming. In *Control Conference (ECC), 2013 European*, pages 536–541. IEEE.

Feller, C., Johansen, T. A., and Olaru, S. (2013). An improved algorithm for combinatorial multi-parametric quadratic programming. *Automatica*, 49(5):1370–1376.

Finch, S. (2003). *Mathematical constants*. Cambridge University Press.

Findeisen, R. and Allgöwer, F. (2002). An introduction to nonlinear model predictive control. In *21st Benelux Meeting on Systems and Control*, volume 11, pages 119–141. Technische Universiteit Eindhoven Veldhoven Eindhoven, The Netherlands.

Fletcher, R. (2013). *Practical Methods of Optimization.* John Wiley & Sons, 2nd edition.

Forbes, M. G., Patwardhan, R. S., Hamadah, H., and Gopaluni, R. B. (2015). Model predictive control in industry: Challenges and opportunities. *IFAC-PapersOnLine*, 48(8):531–538.

Furutani, E., Sakai, C., Takeda, T., and Shirakami, G. (2015). Comparison of pharmacokinetic models for hypnosis control based on effect-site propofol concentration to maintain appropriate hypnosis. *Automat Control Physiol State Func*, 2(104):2.

Gerkšič, S. and de Tommasi, G. (2013). Vertical stabilization of iter plasma using explicit model predictive control. *Fusion Engineering and Design*, 88(6):1082–1086.

Geyer, T., Torrisi, F. D., and Morari, M. (2008). Optimal complexity reduction of polyhedral piecewise affine systems. *Automatica*, 44(7):1728–1740.

Grancharova, A., Johansen, T. A., and Kocijan, J. (2003). Explicit model predictive control of gas-liquid separation plant. In *European Control Conference (ECC), 2003*, pages 2475–2480. IEEE.

Grancharova, A., Johansen, T. A., and Kocijan, J. (2004). Explicit model predictive control of gas–liquid separation plant via orthogonal search tree partitioning. *Computers & chemical engineering*, 28(12):2481–2491.

Granlund, T. (2016). *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6.1.2 edition. `http://gmplib.org/`.

Gupta, A., Bhartiya, S., and Nataraj, P. (2011). A novel approach to multiparametric quadratic programming. *Automatica*, 47(9):2112–2117.

Gustafson, J. (2015). *The End of Error: Unum Computing.* CRC Press.

Herceg, M., Kvasnica, M., Jones, C., and Morari, M. (2013a). Multi-parametric toolbox 3.0. In *Proceedings of the European control conference*, number EPFL-CONF-186265.

Herceg, M., Kvasnica, M., Jones, C. N., and Morari, M. (2013b). Multi-parametric toolbox 3.0. In *Control Conference (ECC), 2013 European*, pages 502–510. IEEE.

Hickey, T., Ju, Q., and Van Emden, M. H. (2001). Interval arithmetic: From principles to implementation. *Journal of the ACM (JACM)*, 48(5):1038–1068.

Honek, M., Kvasnica, M., Szűcs, A., Šimončič, P., Fikar, M., et al. (2015). A low-complexity explicit mpc controller for afr control. *Control Engineering Practice*, 42:118–127.

Hovland, S., Gravdahl, J. T., and Willcox, K. E. (2008). Explicit model predictive control for large-scale systems via model reduction. *Journal of guidance, control, and dynamics*, 31(4):918–926.

Hredzak, B., Agelidis, V. G., and Demetriades, G. (2015). Application of explicit model predictive control to a hybrid battery-ultracapacitor power source. *Journal of Power Sources*, 277:84–94.

Hrovat, D., Di Cairano, S., Tseng, H. E., and Kolmanovsky, I. V. (2012). The development of model predictive control in automotive industry: A survey. In *Control Applications (CCA), 2012 IEEE International Conference on*, pages 295–302. IEEE.

IEEE Std 2008 (2008). Ieee standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70.

Jiang, H., Lin, J., Song, Y., You, S., and Zong, Y. (2016). Explicit model predictive control applications in power systems: an agc study for an isolated industrial system. *IET Generation, Transmission & Distribution*, 10(4):964–971.

Johansen, T. A. (2014). Toward dependable embedded model predictive control. *IEEE Systems Journal*.

Johansen, T. A. and Grancharova, A. (2003). Approximate explicit constrained linear model predictive control via orthogonal search tree. *IEEE Transactions on Automatic Control*, 48(5):810–815.

Jones, C. and Morari, M. (2009). Approximate explicit mpc using bilevel optimization. In *Control Conference (ECC), 2009 European*, pages 2396–2401. IEEE.

Jones, C. N. and Kerrigan, E. (2015). Predictive control for embedded systems. *Optimal Control Applications and Methods*, 36(5):583–584.

Kahan, W. (1996). Ieee standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE*, 754(94720-1776):11.

Karush, W. (1939). Minima of functions of several variables with inequalities as side conditions. Master's thesis, University of Chicago, Chicago, Illinois.

Kaushik, S. and Zorian, Y. (2012). Embedded memory test and repair optimizes soc yields. *Synopsys, Mountain View, CA, USA, Technical Report.*

Kilts, S. (2007). *Advanced FPGA design: architecture, implementation, and optimization.* John Wiley & Sons.

Kirubakaran, V., Radhakrishnan, T., and Sivakumaran, N. (2016). Fuzzy aggregation based multiple models explicit multi parametric mpc design for a quadruple tank process. *IFAC-PapersOnLine*, 49(1):555–560.

Kirubakaran, V., Radkakrishnan, T., and Sivakumaran, N. (2013). Blood glucose concentration regulation in type 1 diabetics using multi model multi parametric model predictive control: An empirical approach. *IFAC Proceedings Volumes*, 46(31):291–296.

Klaučo, M., Kalúz, M., and Kvasnica, M. (2017). Real-time implementation of an explicit mpc-based reference governor for control of a magnetic levitation system. *Control Engineering Practice*, 60:99–105.

Knuth, D. (1985). Dynamic Huffman Coding. *J. Algorithms*, 6(2):163–180.

Koehler, S. and Borrelli, F. (2013). Building temperature distributed control via explicit mpc and "trim and respond" methods. In *Control Conference (ECC), 2013 European*, pages 4334–4339. IEEE.

Krogstad, T., Gravdahl, J., and Tondel, P. (2005). Explicit model predictive control of a satellite with magnetic torquers. In *Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation*, pages 491–496. IEEE.

Kuon, I., Tessier, R., and Rose, J. (2008). Fpga architecture: Survey and challenges. *Foundations and Trends in Electronic Design Automation*, 2(2):135–253.

Kvasnica, M. and Fikar, M. (2012). Clipping-based complexity reduction in explicit MPC. *IEEE Transactions on Automatic Control*, 57(7):1878–1883.

Kvasnica, M., Grieder, P., Baotić, M., and Morari, M. (2004). Multi-parametric toolbox (mpt). In *International Workshop on Hybrid Systems: Computation and Control*, pages 448–462. Springer.

Kvasnica, M., Hledík, J., and Fikar, M. (2012). Reducing the memory footprint of explicit mpc solutions by partial selection. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 4537–4542. IEEE.

Kvasnica, M., Hledík, J., Rauová, I., and Fikar, M. (2013). Complexity reduction of explicit model predictive control via separation. *Automatica*, 49(6):1776–1781.

Kvasnica, M., Holaza, J., Takács, B., and Ingole, D. (2015). Design and verification of low-complexity explicit MPC controllers in MPT3. In *Control Conference (ECC), 2015 European*, pages 2595–2600. IEEE.

Kvasnica, M., Rauová, I., and Fikar, M. (2011). Simplification of explicit mpc feedback laws via separation functions. *IFAC Proceedings Volumes*, 44(1):5383–5388.

Kwon, W. H. and Han, S. H. (2006). *Receding horizon control: model predictive control for state models*. Springer Science & Business Media.

Ławryńczuk, M. (2009). Explicit nonlinear predictive control of a distillation column based on neural models. *Chemical engineering & technology*, 32(10):1578–1587.

Lazar, M. (2006). Model predictive control of hybrid systems: Stability and robustness.

Lee, C. F. and Line, C. M. C. (2008). Explicit nonlinear mpc of an automotive electromechanical brake. *IFAC Proceedings Volumes*, 41(2):10758–10763.

Lee, J. H. (2011). Model predictive control: Review of the three decades of development. *International Journal of Control, Automation and Systems*, 9(3):415–424.

Liu, C., Chen, W.-H., and Andrews, J. (2011). Piecewise constant model predictive control for autonomous helicopters. *Robotics and Autonomous Systems*, 59(7):571–579.

Liu, C., Chen, W.-H., and Andrews, J. (2012). Tracking control of small-scale helicopters using explicit nonlinear mpc augmented with disturbance observers. *Control Engineering Practice*, 20(3):258–268.

Liu, C., Lu, H., and Chen, W.-H. (2015). An explicit mpc for quadrotor trajectory tracking. In *Control Conference (CCC), 2015 34th Chinese*, pages 4055–4060. IEEE.

Maasoumy, M., Razmara, M., Shahbakhti, M., and Vincentelli, A. S. (2014). Handling model uncertainty in model predictive control for energy efficient buildings. *Energy and Buildings*, 77:377–392.

Maciejowski, J. M. (2002). *Predictive control: with constraints*. Pearson education.

Maeder, U., Borrelli, F., and Morari, M. (2009). Linear offset-free model predictive control. *Automatica*, 45(10):2214–2222.

Mariethoz, S., Domahidi, A., and Morari, M. (2009). Sensorless explicit model predictive control of permanent magnet synchronous motors. In *Electric Machines and Drives Conference, 2009. IEMDC'09. IEEE International*, pages 1250–1257. IEEE.

Mariethoz, S., Domahidi, A., and Morari, M. (2012). High-bandwidth explicit model predictive control of electrical drives. *IEEE Transactions on Industry Applications*, 48(6):1980–1992.

Mayne, D. Q. (2014). Model predictive control: Recent developments and future promise. *Automatica*, 50(12):2967–2986.

Mayne, D. Q., Rawlings, J. B., Rao, C. V., and Scokaert, P. O. (2000). Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814.

Méndez, J. A., Marrero, A., Reboso, J. A., and León, A. (2016). Adaptive fuzzy predictive controller for anesthesia delivery. *Control Engineering Practice*, 46:1–9.

Mohammadkhani, M. A., Bayat, F., and Jalali, A. A. (2014). Design of explicit model predictive control for constrained linear systems with disturbances. *International Journal of Control, Automation and Systems*, 12(2):294–301.

Mönnigmann, M. and Kastsian, M. (2011). Fast explicit mpc with multiway trees. *IFAC Proceedings Volumes*, 44(1):1356–1361.

Monsson, P. K. (2008). Combined binary and decimal floating-point unit. Master's thesis, Technical University of Denmark, DTU, DK-2800 Kgs. Lyngby, Denmark.

Montague, R. G., Bingham, C., and Atallah, K. (2013). Magnetic gear pole-slip prevention using explicit model predictive control. *IEEE/ASME Transactions on Mechatronics*, 18(5):1535–1543.

Moore, R. E. (1979). *Methods and applications of interval analysis.* SIAM.

Moore, R. E., Kearfott, R. B., and Cloud, M. J. (2009). *Introduction to interval analysis.* SIAM.

Muller, J., Brisebarre, N., De Dinechin, F., Jeannerod, C., Lefevre, V., Melquiond, G., Revol, N., Stehlé, D., and Torres, S. (2009). *Handbook of floating-point arithmetic.* Springer Science & Business Media.

Muske, K. R. and Rawlings, J. B. (1993). Model predictive control with linear models. *AIChE Journal*, 39(2):262–287.

Naşcu, I., Diangelakis, N. A., Oberdieck, R., Papathanasiou, M. M., and Pistikopoulos, E. N. (2016). Explicit mpc in real-world applications: the paroc framework. In *American Control Conference (ACC), 2016*, pages 913–918. IEEE.

Naşcu, I., Krieger, A., Ionescu, C. M., and Pistikopoulos, E. N. (2015). Advanced model-based control studies for the induction and maintenance of intravenous anaesthesia. *IEEE Transactions on biomedical engineering*, 62(3):832–841.

Naus, G., Ploeg, J., Van de Molengraft, M., Heemels, W., and Steinbuch, M. (2010). Design and implementation of parameterized adaptive cruise control: An explicit model predictive control approach. *Control Engineering Practice*, 18(8):882–892.

Nielsen, I. and Axehill, D. (2016). Reduced memory footprint in multiparametric quadratic programming by exploiting low rank structure. In *Decision and Control (CDC), 2016 IEEE 55th Conference on*, pages 3654–3661. IEEE.

Nocedal, J. and Wright, S. (2006). *Numerical Optimization.* Springer-Verlag, USA, 2$^{\text{nd}}$ edition.

Oberdieck, R., Diangelakis, N. A., Papathanasiou, M., Nascu, I., and Pistikopoulos, E. (2016). Pop–parametric optimization toolbox. *Industrial & Engineering Chemistry Research*, 55(33):8979–8991.

Oberdieck, R., Diangelakis, N. A., and Pistikopoulos, E. N. (2017). Explicit model predictive control: a connected-graph approach. *Automatica*, 76:103–112.

Oldewurtel, F., Gondhalekar, R., Jones, C. N., and Morari, M. (2009). Blocking parameterizations for improving the computational tractability of affine disturbance feedback mpc problems. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 7381–7386. IEEE.

Oliveri, A., Naus, G. J., Storace, M., and Heemels, W. (2011). Low-complexity approximations of pwa functions: A case study on adaptive cruise control. In *Circuit Theory and Design (ECCTD), 2011 20th European Conference on*, pages 669–672. IEEE.

Oravec, J., Blažek, S., and Kvasnica, M. (2013). Simplification of explicit mpc solutions via inner and outer approximations. In *Process Control (PC), 2013 International Conference on*, pages 389–394. IEEE.

Overton, M. L. (2001). *Numerical computing with IEEE floating point arithmetic*. SIAM.

Padgett, W. T. and Anderson, D. V. (2009). Fixed-point signal processing. *Synthesis Lectures on Signal Processing*, 4(1):1–133.

Padula, F., Ionescu, C., Latronico, N., Paltenghi, M., Visioli, A., and Vivacqua, G. (2017). Optimized pid control of depth of hypnosis in anesthesia. *Computer Methods and Programs in Biomedicine*, 144:21–35.

Pannocchia, G. and Rawlings, J. B. (2003). Disturbance models for offset-free model-predictive control. *AIChE journal*, 49(2):426–437.

Parhami, B. (1999). *Computer arithmetic*, volume 20. Oxford university press.

Parisio, A., Fabietti, L., Molinari, M., Varagnolo, D., and Johansson, K. H. (2014). Control of hvac systems via scenario-based explicit mpc. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 5201–5207. IEEE.

Pistikopoulos, E. (2009). Perspectives in multiparametric programming and explicit model predictive control. *AIChE journal*, 55(8):1918–1925.

Pistikopoulos, E., Galindo, A., Dua, V., Kikkinides, E. S., Papageorgiou, L., Jorisch, W., Benz, K.-W., Neumann, W., Köhler, M., Fritzsche, W., et al. (2007a). *Multi-Parametric Programming: Theory, Algorithms and Applications*, volume 1. Weinheim: WileyVCH.

Pistikopoulos, E. N., Dominguez, L., Panos, C., Kouramas, K., and Chinchuluun, A. (2012). Theoretical and algorithmic advances in multi-parametric programming and control. *Computational Management Science*, pages 1–21.

Pistikopoulos, E. N., Galindo, A., Dua, V., Kikkinides, E. S., Papageorgiou, L., Jorisch, W., Benz, K.-W., Neumann, W., Köhler, M., Fritzsche, W., et al. (2007b). *Multi-Parametric Model-Based Control: Theory and Applications*, volume 2. Weinheim: WileyVCH.

Pu, Y. and Yu-hong, W. (2015). Explicit model predictive control of cstr system based on pwa model. In *Control Conference (CCC), 2015 34th Chinese*, pages 2304–2308. IEEE.

Puig, V., Rosich, A., Ocampo-Martínez, C., and Sarrate, R. (2007). Fault-tolerant explicit mpc of pem fuel cells. In *Decision and Control, 2007 46th IEEE Conference on*, pages 2657–2662. IEEE.

Qin, S. J. and Badgwell, T. A. (2000). An overview of nonlinear model predictive control applications. *Nonlinear model predictive control*, pages 369–392.

Qin, S. J. and Badgwell, T. A. (2003). A survey of industrial model predictive control technology. *Control engineering practice*, 11(7):733–764.

Rawlings, J. B. (2000). Tutorial overview of model predictive control. *IEEE Control Systems*, 20(3):38–52.

Rossiter, J. A. (2003). *Model-based predictive control: a practical approach.* CRC press.

Rossiter, J. A. and Grieder, P. (2005). Using interpolation to improve efficiency of multiparametric predictive control. *Automatica*, 41(4):637–643.

Sahu, C., Radhakrishnan, T., and Sivakumaran, N. (2015). Real time closed loop data based estimation and explicit model based control of an air conditioning system implemented in hardware in loop scheme. In *Robotics, Automation, Control and Embedded Systems (RACE), 2015 International Conference on*, pages 1–7. IEEE.

Sanchez-Cossio, J., Ortega-Alvarez, J. D., and Ocampo-Martinez, C. (2015). Temperature regulation of a pilot-scale batch reaction system via explicit model predictive control. In *Automatic Control (CCAC), 2015 IEEE 2nd Colombian Conference on*, pages 1–6. IEEE.

Sawaguchi, Y., Furutani, E., Shirakami, G., Araki, M., and Fukuda, A. (2008). A model-predictive hypnosis control system under total intravenous anesthesia. *Biomedical Engineering, IEEE Transactions on*, 55(3):874–887.

Schüttler, J. and Ihmsen, H. (2000). Population pharmacokinetics of propofol: a multicenter study. *Anesthesiology*, 92(3):727–738.

Scibilia, F., Olaru, S., and Hovd, M. (2009). Approximate explicit linear mpc via delaunay tessellation. In *Control Conference (ECC), 2009 European*, pages 2833–2838. IEEE.

Seron, M. M., Goodwin, G. C., and De Doná, J. A. (2002). Finitely parameterised implementation of receding horizon control for constrained linear systems. In *American Control Conference, 2002. Proceedings of the 2002*, volume 6, pages 4481–4486. IEEE.

Shen, Y., Xie, L., and Li, X. (2013). Explicit hybrid model predictive control of the forward dc-dc converter. In *Control and Decision Conference (CCDC), 2013 25th Chinese*, pages 638–642. IEEE.

Sites, M. (2008). Ieee standard for floating-point arithmetic.

Snyman, J. (2005). *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*, volume 97. Springer Science & Business Media.

Spjøtvold, J., Kerrigan, E. C., Jones, C. N., TøNdel, P., and Johansen, T. A. (2006). On the facet-to-facet property of solutions to convex parametric quadratic programs. *Automatica*, 42(12):2209–2214.

Stephens, M. A., Manzie, C., and Good, M. C. (2011). Explicit model predictive control for reference tracking on an industrial machine tool. *IFAC Proceedings Volumes*, 44(1):14513–14518.

Suardi, A., Kerrigan, E. C., and Constantinides, G. A. (2015). Fast fpga prototyping toolbox for embedded optimization. In *European Control Conference (ECC)*, pages 2589–2594.

Suardi, A., Longo, S., Kerrigan, E., and Constantinides, G. (2014). Robust explicit mpc design under finite precision arithmetic. *IFAC Proceedings Volumes*, 47(3):2939–2944.

Suardi, A., Longo, S., Kerrigan, E. C., and Constantinides, G. (2016). Explicit MPC: Hard constraint satisfaction under low precision arithmetic. *Control Engineering Practice*, 47:60–69.

Swartzlander, E. E. (2015). Computer arithmetic. In *Computer Arithmetic: Volume I*, pages 1–398. World Scientific.

Szücs, A., Kvasnica, M., and Fikar, M. (2011). A memory-efficient representation of explicit MPC solutions. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 1916–1921. IEEE.

Takács, B., Števek, J., Valo, R., and Kvasnica, M. (2016a). Python code generation for explicit mpc in mpt. In *Control Conference (ECC), 2016 European*, pages 1328–1333. IEEE.

Takács, G., Batista, G., Gulan, M., and Rohal'-Ilkiv, B. (2016b). Embedded explicit model predictive vibration control. *Mechatronics*, 36:54–62.

Tøndel, P. and Johansen, T. A. (2002). Complexity reduction in explicit linear model predictive control. *IFAC Proceedings Volumes*, 35(1):189–194.

TøNdel, P., Johansen, T. A., and Bemporad, A. (2003). An algorithm for multiparametric quadratic programming and explicit mpc solutions. *Automatica*, 39(3):489–497.

Ulbig, A., Olaru, S., and Dumur, D. (2008). Explicit model predictive control for a magnetic levitation system. In *Control and Automation, 2008 16th Mediterranean Conference on*, pages 1544–1549. IEEE.

UNE, L. and Pannek, J. (2011). *Nonlinear Model Predictive Control: Theory and Algorithms, Communications and Control Engineering.* Springer,.

Wang, L. (2009). *Model predictive control system design and implementation using MATLAB®.* Springer Science & Business Media.

Wang, Y. and Boyd, S. (2010). Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, 18(2):267–278.

Wen, C., Ma, X., and Ydstie, B. E. (2009). Analytical expression of explicit mpc solution via lattice piecewise-affine function. *Automatica*, 45(4):910–917.

Xilinx. Xilinx. Available at `http://www.xilinx.com`.

Yelneedi, S., Samavedham, L., and Rangaiah, G. (2009). Advanced control strategies for the regulation of hypnosis with propofol. *Industrial & Engineering Chemistry Research*, 48(8):3880–3897.

Yu-Geng, X., De-Wei, L., and Shu, L. (2013). Model predictive control—status and challenges. *Acta Automatica Sinica*, 39(3):222–236.

Zanini, F., Atienza, D., Benini, L., and De Micheli, G. (2009). Multicore thermal management with model predictive control. In *Circuit Theory and Design, 2009. ECCTD 2009. European Conference on*, pages 711–714. IEEE.

Zhang, J., Cheng, X., and Zhu, J. (2016). Control of a laboratory 3-dof helicopter: Explicit model predictive approach. *International Journal of Control, Automation, and Systems*, 14(2):389.

Zhao, D., Liu, C., Stobart, R., Deng, J., Winward, E., and Dong, G. (2014). An explicit model predictive control framework for turbocharged diesel engines. *IEEE Transactions on Industrial Electronics*, 61(7):3540–3552.

Zong, Y., Böning, G. M., Santos, R. M., You, S., Hu, J., and Han, X. (2017). Challenges of implementing economic model predictive control strategy for buildings interacting with smart energy systems. *Applied Thermal Engineering*, 114:1476–1486.