TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Realzeit-Computersysteme

Resource-Aware Automotive Control Systems Design

Wanli Chang

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr.-Ing. Gerhard Rigoll
Prüfer der Dissertation:	1. Prof. Dr. Samarjit Chakraborty
	2. Prof. Anuradha Annaswamy, Ph.D.

Die Dissertation wurde am 03.06.2016 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 17.08.2017 angenommen.

Abstract

As the automotive industry is entering the smart era through advances in sensing, computation, storage, communication, and actuation technologies, a larger number of more complex control applications with better performances are expected to be on board. This requires an implementation platform with abundant resources, which is undesired in the cost-sensitive automotive domain. The implementation platform, often embedded in an Electronic Control Unit (ECU) and shared by multiple applications to save the cost, is mainly comprised of a processor for computation, memory for storing instructions and data, and bus for internal and external communication. Conventionally, automotive control systems are designed using model-based approaches, where the details of the implementation platform are ignored. Techniques that integrate the characteristics of implementation resources into control algorithms design are largely missing. Such a separate design paradigm is too conservative in resources dimensioning and utilization for modern vehicles.

This thesis presents novel approaches in automotive control systems design that take implementation resources into consideration, aiming to improve the control performances for a given amount of resources, or equivalently, realize the required control performances with fewer resources. While communication resources have started to get explored in the literature of networked embedded control systems, the focus of this thesis is on memory and computation resources. As Electric Vehicles (EVs) have become a new trend in the automotive industry, energy resources of EVs, i.e., the batteries, are also investigated.

In the typical two-level memory hierarchy of an onboard embedded implementation platform, on-chip cache is the most costly component. A memory-aware sampling order, which is proposed to run each application consecutively multiple times instead of all applications in a conventional round-robin fashion, increases the cache reuse and improves control performances. In particular, two techniques are presented. First, a memory analysis technique is used to compute the guaranteed Worst-Case Execution Time (WCET) reduction due to the cache reuse between two consecutive runs of one application. Second, a controller design technique is tailored for non-uniform sampling with sensor-to-actuator delays shorter than or equal to the sampling periods, which results from the WCETs of the memory-aware sampling order. The approach to find the optimal sampling order that maximizes the overall control performance is reported.

On the embedded implementation platform often runs a Time-Triggered (TT) Operating System (OS) due to the safety-critical nature of automotive systems. Typically, a TT OS only supports a limited set of predefined periods, from which the sampling period of a feedback control application has to be chosen. For a given performance requirement, the optimal sampling period usually does not fall in this set, which forces the controller to be designed with a shorter

sampling period than the optimal one — leading to an unnecessarily higher utilization of the computation resources. In this thesis, a novel multirate controller that switches between the available sampling periods offered by the TT OS is proposed, towards achieving an average sampling period closer to the optimal one. The benefit is a lower processor utilization, while the control performance requirement and the system constraints are still satisfied. The challenge lies in the performance-oriented controller design under the non-uniform sampling scheme with negligible sensor-to-actuator delays.

Battery is a key component of EVs. The effective battery capacity depends on the discharging current profile, which varies with different control strategies. For a given battery, it is desired to increase the battery usage while satisfying the control performance requirement. A novel optimization framework is proposed in this thesis to explore the Pareto front between battery usage and the control performance. Designers can then select the Pareto point according to the specific requirement and preference. As the processor in the embedded implementation platform ages, the sampling period of the feedback control application is prolonged. This leads to control performance deterioration, which is highly undesirable for safety-critical control applications in EVs. Using the same framework with slight modification, the controller is re-optimized to ensure that the control performance is kept with a modest compromise in the battery usage.

Throughout this thesis, state-feedback control applications with both linear and non-linear control law are considered and naturally robust to uncertainties. Several well-established optimization approaches are taken and novel optimization techniques are developed on top of them. The iterative interior-point method is deployed for convex problems. Particle Swarm Optimization (PSO) and gradient-based Sequential Quadratic Programming (SQP) are typically used to solve non-convex single-objective problems. The non-dominated sorting genetic algorithm (NSGA) is taken for non-convex multi-objective problems. A number of real-world applications that are detailed in the appendix validate the resource-aware automotive systems design techniques proposed in this thesis.

Acknowledgements

I would like to express my gratitude to my supervisor, Prof. Samarjit Chakraborty, for offering me the opportunity to work with him on a number of exciting problems that eventually constitute this thesis. I am constantly surprised by his intelligence, creativity and sense of responsibility. I always feel fortunate to have his support and advice along the way.

I also thank Dr. Anuradha Annaswamy for hosting me at MIT. We had many interesting and insightful discussions that broadened my view of research.

Dr. Dip Goswami has been very helpful, especially in giving me technical suggestions in a number of occasions.

It was a great pleasure working with Dr. Jason Xue in Hong Kong, where I benefited from him and his group in knowledge building and skill development.

I appreciate that Prof. Gerhard Rigoll chaired my defense.

Contents

1 Introduction

Performance and reliability of automobiles are influenced by feedback control applications implemented on board. At the inception of the first horseless carriage, some form of control was already applied to motor vehicles. Engine idle speed control, which can be found in every modern vehicle powered by an Internal Combustion Engine (ICE), traces back to the Watt's governor in 1769. This device marking the origin of both feedback control and the industrial revolution can be viewed as a mechanical idle speed feedback controller for a steam engine [BBC+07].

Over the last century, control has been applied to almost every aspect of vehicle operation, from engine to drivetrain, from steering to braking. For instance, electronic powertrain control was introduced in automobiles in the 1970s, aiming to substantially reduce emissions. As a result, cars on the street today are 99% cleaner than they were when emission regulations were first introduced in the 1960s. Applications including Anti-Lock Braking System (ABS), traction control, Electronic Stability Control (ESC), and active safety systems have decreased the number and severity of accidents.

With advances in sensing, computation, storage, communication, and actuation technologies, more complex automotive control applications targeting better performances have emerged. In the combustion engine control, homogeneous charge compression ignition has been developed to reduce NO_x emission. These engines have a higher level of Exhaust Gas Recirculation (EGR), yet the involved control technique is more sophisticated due to the fragile combustion stability in load transients [CSJ07]. The level of EGR of conventional turbocharged diesel engines can be increased as well. The operating range of the engine is partitioned in [OdR07]. Linear models are identified for each partition and Model Predictive Control (MPC) is applied to take account of actuator saturation.

In the powertrain control, a Design for Six Sigma (DFSS) analysis approach is used to determine automatic transmission gear content, aiming at fuel consumption minimization for various powertrain systems [Rob14]. Conventional powertrain Active Noise Control (ANC) systems have difficulty in tracking the fast engine speed variations. A number of modified

Filtered-x Least Mean Squares (FxLMS) algorithms are reported in [SFX+15] to achieve a balanced noise reduction performance over a broad frequency range.

A unified chassis control strategy integrating Active Front Steering (AFS) and ESC is proposed in [CCK+12] to improve agility, maneuverability, and vehicle lateral stability. Integrated chassis control is also reported in [HJYK15] and [CLP+14] to enhance high speed cornering performance and on-center handling behavior, respectively. In [HJYK15], individual chassis control systems, such as ESC, Four Wheel Drive (4WD), Active Roll Control System (ARS), and Electronic Control Suspension (ECS), are involved. Various chassis modules are analyzed to verify the proposed integrated control strategy. In [CLP+14], MPC is used for optimal allocation of sub-chassis control systems.

Along the direction of autonomous driving, which can be classified into five levels depending on the extent of automation as follows [NHTSA13], new control applications have debuted in the modern premium cars.

- **Level 0 (no automation):** The driver is in complete and sole control of the primary vehicle controls (brake, steering, throttle, and motive power) at all times, and is solely responsible for monitoring the roadway and for safe operation of all vehicle controls. Vehicles that have certain driver support/convenience systems but do not have control authority over steering, braking, or throttle would still be considered “level 0" vehicles. Examples include systems that provide only warnings (e.g., forward collision warning, lane departure warning, blind spot monitoring) as well as systems providing automated secondary controls such as wipers, headlights, turn signals, hazard lights, etc.

- **Level 1 (function-specific automation):** Automation at this level involves one or more specific control functions; if multiple functions are automated, they operate independently from each other. The driver has overall control, and is solely responsible for safe operation, but can choose to cede limited authority over a primary control, the vehicle can automatically assume limited authority over a primary control, or the automated system can provide added control to aid the driver in certain normal driving or crash-imminent situations.

- **Level 2 (combined-function automation):** This level involves automation of at least two primary control functions designed to work in unison to relieve the driver of control of those functions. Vehicles at this level of automation can utilize shared authority when the driver cedes active primary control in certain limited driving situations. The driver is still responsible for monitoring the roadway and safe operation and is expected to be available for control at all times and on short notice. The system can relinquish control with no advance warning and the driver must be ready to control the vehicle safely.

- **Level 3 (limited self-driving automation):** Vehicles at this level of automation enable the driver to cede full control of all safety-critical functions under certain traffic or environmental conditions and in those conditions to rely heavily on the vehicle to monitor for changes in those conditions requiring transition back to driver control. The driver is expected to be available for occasional control, but with sufficiently comfortable transition time.

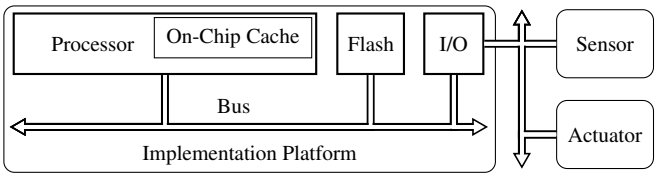

Figure 1.1: A typical embedded implementation platform for automotive control applications. The processor executes control programs. Instructions and data are stored in the flash memory. On-chip cache accelerates the memory access. Programmable I/O peripherals are used for communication with sensors and actuators.

- **Level 4 (full self-driving automation):** The vehicle is designed to perform all safety-critical driving functions and monitor roadway conditions for an entire trip. Such a design anticipates that the driver will provide destination or navigation input, but is not expected to be available for control at any time during the trip. This includes both occupied and unoccupied vehicles. By design, safe operation rests solely on the automated vehicle system.

Nowadays, most vehicles with autonomous features in the market fall into Level 1 or 2. For instance, Adaptive Cruise Control (ACC) enables the driver to cede limited authority over a primary control. Lane keeping automatically assumes limited authority over a primary control. Dynamic brake support provides added control to aid the driver in emergencies. Other emerging control applications include automated parking [MDG07], path tracking [Sni09], obstacle avoidance [VdNMP07], and vehicle control at friction limits [Kri12].

1.1 Motivation

Automotive control applications are implemented on a platform embedded in an Electronic Control Unit (ECU). A feedback control loop has three operations:

- **Measurement:** Sensors measure the states of the physical plants. This is also called sampling.
- **Computation:** Taking the data from sensors, control programs are executed and compute the control input.
- **Actuation:** The control input is sent to actuators, aiming to achieve certain desired behavior of the plants.

A typical embedded implementation platform for automotive control applications is shown in Figure 1.1. There are often programs of multiple control applications executed on one processor, which necessitates an Operating System (OS) for coordination. The flash memory stores all instructions and data. The on-chip cache accelerates the memory access. Programmable Input/Output (I/O) peripherals are used for communication with sensors and actuators.

The implementation platform considerably impacts the control performances via, e.g., sampling periods and sensor-to-actuator delays. The sampling period is defined as the time duration between two consecutive measurements (or samplings) of the plant states under control. The sensor-to-actuator delay is defined as the time duration between the measurement and the actuation of one feedback control loop. If the processor or the memory access is not fast enough, the execution time of the control program might be too long to meet the desired sampling periods and sensor-to-actuator delays. Therefore, a larger number of more complex control applications calls for an implementation platform with abundant resources, which contradicts the cost-sensitive nature of the automotive industry.

The algorithms development for control applications from a control-theoretical perspective is well-established. The controller design methods can be drawn from a large pool of research and practical experience that have been accumulated in the control community. However, little attention has been paid to the embedded implementation platform. Control theorists and embedded system engineers make model-based assumptions of the other side. Since most control applications are safety-critical, such assumptions in this separate design paradigm are inevitably conservative to guarantee the required control performances. As a result, the resources on the embedded implementation platform, such as communication, computation and memory, are inefficiently utilized. This thesis presents new techniques in automotive control systems design that take implementation resources into consideration, aiming to improve the control performances for a given amount of resources, or equivalently, realize the required control performances with fewer resources.

Motivated by the increasing worldwide efforts to reduce Greenhouse Gas (GHG) emissions[1], automotive manufacturers have been struggling in upgrading their ICEs. It is challenging to reduce emissions while keeping the engine performance. An alternative solution is an Electric Vehicle (EV). Another major advantage of an EV is its independence of fossil fuels[2]. Although the petrol price is quite low as this thesis is written, it might be hard to predict its future trend. Besides, the torque and noise performances of an electric motor are generally better than an ICE of the similar size at low speeds. Most major car manufacturers have presented their mass-produced EVs, including Nissan Leaf, BMWi3, Volkswagen e-Golf, Chevy Volt, and Tesla Model S.

One major issue that impedes the market acceptance of EVs is the range anxiety. The energy resource is the major factor determining the driving range of an EV. Given a fixed battery pack, it is desired to maximize the battery usage (i.e., maximize the effective battery capacity and minimize the energy consumption of a control task instance), which is directly related to the driving range. Different control strategies result in different discharging current profiles and the battery usage depends on the discharging current profile. This thesis investigates the control systems design in an EV taking the energy resource into consideration. The influence of

[1]For instance, the U.S. Environmental Protection Agency (EPA) and the Department of Transportation's National Highway Traffic Safety Administration (NHTSA) have set standards to reduce GHG emissions and improve fuel economy that model years 2017-2025 cars and light trucks must reach 4.3 liters per 100 kilometers [Age12].

[2]It is noted that both GHG emissions reduction and fossil fuels independence also depend on the way of electricity generation.

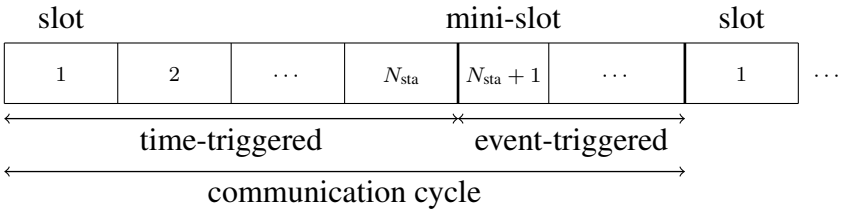

Figure 1.2: FlexRay bus with both time-triggered static and event-triggered dynamic segments

processor aging in the implementation platform on both the control performance and the battery usage is explored, and countermeasures are proposed.

1.2 Resources

In this section, communication, memory, computation, and energy resources are described, where there have been a number of works on communication-aware embedded control systems design in the literature and the focus of this thesis is on the latter three types of resources.

1.2.1 Communication Resources

The number of bits that can be transmitted per unit of time over a communication network is limited by the bandwidth. Precise characterization of the communication resources for automotive control systems is protocol-specific. The communication protocols are broadly classified into two groups — Event-Triggered (ET) and Time-Triggered (TT) networks. For instance, Controller Area Network (CAN) is ET and has been widely used since its first official release in 1986 [Bos91]. FlexRay, which was designed about a decade ago to be faster and more reliable than CAN, can be found in most premium cars [Fle05]. Media access control in FlexRay is based on communication cycles of equal and predefined length in time. Each communication cycle is divided into a TT static and an ET dynamic segment as shown in Figure 1.2. Messages can be sent with FlexRay over either the TT or ET segment using a bandwidth of 10 Mbit/s.

The TT static segment follows a Timing Division Multiple Access (TDMA) policy for media access control, where the entire segment is divided into multiple slots with the same predefined length in time. In every segment, slots are statically indexed starting from 1 to N_{sta}, which is the total number of TT slots in the static segment. Each application involved in the TT communication is assigned a dedicated index number and only uses the TT slot of this index to transmit messages. This allows a predictable temporal behavior, since in every communication cycle, an application is able to access the TT segment once, and the interval between two consecutive allowed transmissions is fixed. If no messages from an application need to be sent on its given slot, then the network is idle for this period of time, resulting in an inefficient utilization of bandwidth.

In the ET dynamic segment, media access control is priority-based and the entire segment is divided into mini-slots. Every application involved in the ET communication is associated with an index and in effect, a priority. In a segment, each mini-slot is dynamically assigned an

index. The starting index is $N_{\text{sta}} + 1$. The application matching the mini-slot index is allowed to transmit a message. A message can be transmitted over multiple mini-slots and the mini-slots transmitting the same message have the same index. After the transmission ends, the mini-slot index is incremented. If the message is not ready when its mini-slot starts, the mini-slot goes idle and the index is incremented. Since a mini-slot is typically much shorter than a TT slot, the ET segment offers more efficient utilization of bandwidth compared to the TT segment. The ET segment generally does not provide temporal guarantee due to its priority-based nature of arbitration. The timing of a message over ET communication depends on the presence of messages with higher priorities.

The research on networked control systems dates back to [HR88, Nil98, WYB99, ZBP01]. A number of recent efforts have been made to address the communication-aware embedded control systems design. An aperiodic strategy for dynamic allocation of bandwidth according to the current state of the plants and available resources is proposed in [AT09]. Control loops closed over CAN are discussed and illustrated on a train car. In [AGL15], an event-based control-scheduling co-design strategy involving a set of continuous-time linear time-invariant plants is proposed to address the challenges of variable communication delays, access constraints and resource constraints. In [SCEP09], communication delay and jitter resulting from complex timing behavior are considered. A method integrating controller design and message scheduling is developed to optimize the overall control performance. In [LWH$^+$14], the network management problem is addressed to reduce communication jitter and improve control performance. A predictive compensator co-located with the actuator is proposed in [HSJ08] to deal with communication outages. When a new control command is not received, a replacement one based on the history of past control commands is suggested.

There are mainly two challenges in the communication-control co-design. First, the design space can be too large to be tractable. There are many parameters to determine in the design of the controller and the communication network. A combined design space can be difficult to handle. This is aggravated by the increase of system size. Second, the trade-off between the control performance and the communication resource utilization, which enables more design freedom, has not been explored. Some first efforts have been made in [RZC$^+$16] to address these challenges.

1.2.2 Memory Resources

In the two-level memory architecture shown in Figure 1.1 (such as the XC23xxB Series microcontroller [Inf09] from Infineon that is popular in automotive systems), the flash memory has a large size and can thus store all the application programs and data, but experiences high read/write latencies (hundreds of processor cycles). The cache is faster with low read/write latencies (several processor cycles), but usually limited in size due to its high cost. It is assumed that the access times of cache and flash memory are t_c and t_m, respectively, where $t_c \ll t_m$. In this thesis, the focus is on instruction memory, since control applications are typically not data-intensive.

When a processor executes an instruction, it checks the cache first. If this instruction is located in the cache, it is a cache hit and the access time is t_c. If this instruction is not in the cache, the memory block containing it is fetched from the flash memory and then written into

cache. This is a cache miss and the access time is t_m. Afterwards, when the same instruction is called again by the processor, the access time is t_c if it is still in the cache without being replaced. This is a cache reuse.

A program usually has different execution paths resulting in different execution times. The Worst-Case Execution Time (WCET) is defined to be the maximum length of time a program takes to be executed. The WCET constrains the sampling period of a control application. There are two general methods to reduce the WCET of a program — increasing the cache size and/or cache reuse. In resource-aware automotive control systems design, it is desirable to minimize the cache size while satisfying the performance requirement, or equivalently, improve the performance for the given memory resources. Therefore, the cache reuse should be maximized.

Given a collection of control applications (e.g., $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$), it is conventional to run the control loops of them in a round-robin fashion ($\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$, $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$, $\cdots$). Since the codes for different control applications are different, the on-chip cache is frequently refreshed in this process. This results in poor cache reuse and long WCET. In order to address this issue, a new sampling order for the control applications is proposed, using which cache reuse is improved and the WCET of each application is reduced. In particular, a non-uniform sampling scheme is studied, where the control loop of each application is consecutively run multiple times — in order to increase cache reuse, before moving on to the next application.

An example memory-aware sampling order ($\mathcal{C}_1(1)$, $\mathcal{C}_1(2)$, $\mathcal{C}_1(3)$, $\mathcal{C}_2(1)$, $\mathcal{C}_2(2)$, $\mathcal{C}_2(3)$, $\mathcal{C}_3(1)$, $\mathcal{C}_3(2)$, $\mathcal{C}_3(3)$, $\cdots$) is illustrated in Figure 1.3, where $\mathcal{C}_i(j)$ denotes the jth execution of the control application $\mathcal{C}_i$. Before the first execution $\mathcal{C}_i(1)$, the cache is either empty (i.e., cold cache) or filled with instructions from other applications, that are not used by $\mathcal{C}_i$ (equivalent to cold cache). The WCET of $\mathcal{C}_i(1)$ can be computed by a number of existing standard techniques [Wea08, AGS+13, WGR+09]. Before the second execution $\mathcal{C}_i(2)$, the instructions in the cache are from the same application $\mathcal{C}_i$ and thus can be reused. This results in more cache hits and hence shorter WCET. Depending on which execution path the program takes, the amount of WCET reduction varies. Therefore, a technique is required to compute the guaranteed WCET reduction of $\mathcal{C}_i(2)$ and $\mathcal{C}_i(3)$ relative to $\mathcal{C}_i(1)$, independent of the path taken.

Control parameters of the applications, such as sampling periods and sensor-to-actuator delays, can be derived from the WCET results. A controller must be tailored for the memory-aware non-uniform sampling orders, so that the control performance can be improved. In summary, two main techniques are required — (i) cache analysis to compute the guaranteed WCET reduction between two consecutive executions of one program; (ii) controller design for the non-uniform sampling with sensor-to-actuator delays shorter than or equal to the sampling periods. On top of showing that an example sampling order under memory-aware automotive control systems design results in better control performance, the approach to find the optimal sampling order that maximizes the overall control performance is reported.

1.2.3 Computation Resources

For a given processor with a certain operating frequency, computation resources usually mean the available execution time. When multiple applications share one processor, in general, the performance of an application can be improved if it is allowed to access the processor for a longer period of time. Computation-aware automotive control systems design aims to reduce

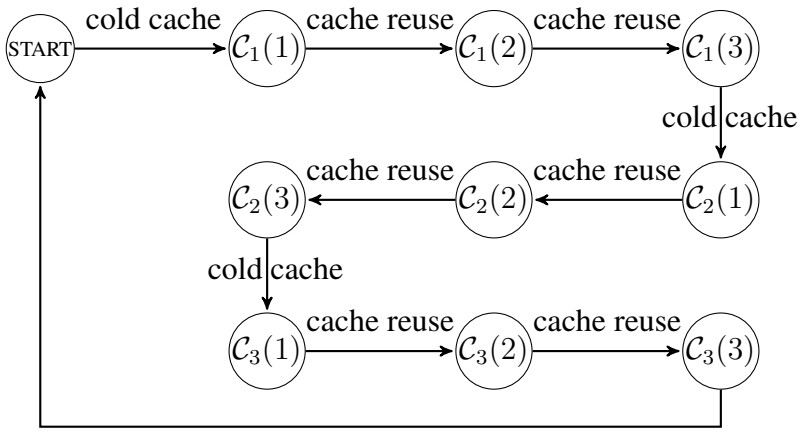

Figure 1.3: An example memory-aware sampling order with three control applications. Each application is consecutively executed three times. After the first execution $\mathcal{C}_i(1)$, some instructions in the cache can be reused and thus the WCETs of the following two executions are shortened.

the execution time of a control application, while still satisfying its performance requirement. In this way, more applications can be mapped to the processor, thereby saving the cost. This is the recent trend of ECU consolidation in the automotive industry.

Generally, a shorter sampling period allows the controller to respond to its plant more frequently, and is thus potentially able to achieve better control performance with an appropriately designed controller. The obvious downside is a higher processor utilization, which is defined to be the WCET of an application divided by its sampling period. This prevents more functions and applications from being integrated onto the processor. Therefore, the controller should use the largest possible sampling period that is able to fulfill the control performance requirement and satisfy the system constraints.

Due to the safety-critical nature of the automotive domain, TT OS usually runs on the processor. For instance, OSEK/VDX (Open Systems and Their Corresponding Interfaces for Automotive Electronics/Vehicle Distributed Executive) OS [Con05, Fei03] is widely used in automobiles and considered in this thesis. OSEK/VDX OS only offers a limited set of predefined periods, which implies that the sampling periods of control applications have to be taken from this set. In most cases, the optimal sampling period is not directly realizable on the OS. The conventional way to handle it is to use the largest sampling period offered by the OS that is smaller than the optimal one. This is a straightforward method, yet leads to a waste of computational resources. It is desirable to minimize the processor utilization of an application, while still satisfying the performance requirement.

Towards this goal, a multirate controller that switches between available sampling periods offered by OSEK/VDX OS is proposed. A typical example with sampling periods of $2ms$, $5ms$ and $10ms$ on OSEK/VDX OS is illustrated in Figure 1.4. Switching between two sampling periods can only occur at the common multiplier of them. For instance, switching between $2ms$ and $5ms$ is possible at the time instant of $10ms$, $20ms$, and so on. Therefore, possible sequences of sampling periods are $\{2ms, 2ms, 2ms, 2ms, 2ms, 5ms, 5ms, repeat\}$,

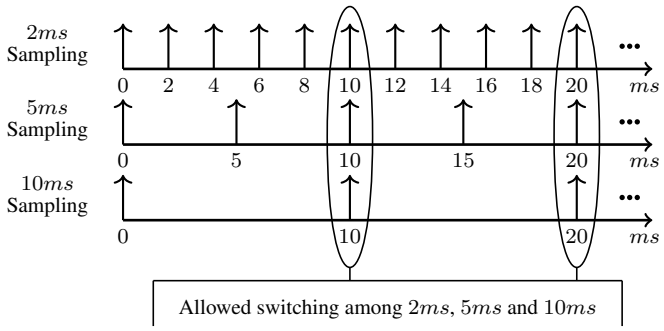

Figure 1.4: Allowed switching instants among multiple sampling periods

$\{5ms, 5ms, 10ms, repeat\}$, and so on. The main challenge lies in the performance-oriented multirate controller design under the non-uniform sampling scheme with negligible sensor-to-actuator delays, aiming to reduce the processor utilization.

1.2.4 Energy Resources

For all practical purposes, a longer driving range is desired in EVs to increase their usability. A battery pack with large capacity is needed to offer a long driving range. However, with larger capacity, the battery weight also increases leading to higher energy consumption. Moreover, the capacity is restricted by the space that can be allocated to the battery pack in EVs. One potential solution to the above problem is to design the controller in such a way that the energy consumption of a control task instance can be minimized.

All off-the-shelf battery packs are labeled with a nominal capacity. However, due to the rate capacity effect, the effective capacity or Full Charge Capacity (FCC) of a battery pack, which is defined to be the amount of electric charges that can be delivered from the battery after it is fully charged, actually varies with different discharging current profiles [DS06, KQ11]. Generally speaking, larger discharging current tends to reduce the effective capacity. For most common lithium-ion batteries in the market, the capacity could potentially get significantly compromised if the rate capacity effect is not properly considered in the control systems design.

In this thesis, an optimization framework considering the control performance as one design objective and battery usage as the other is presented. The trade-off between these two design objectives is explored by generating a Pareto front. The battery usage is quantified by the number of times the control system can reach a steady state after a disturbance occurs powered up by a fully charged battery pack. In order to maximize the battery usage, the energy consumption of a control task instance, i.e., the disturbance rejection, should be small and the battery effective capacity should be increased by generating a battery-friendly discharging current profile.

In this context, the other important design aspect is processor aging. As a processor ages, the switching time of its transistors increases, resulting in longer path delays. On-chip monitors could be used to measure the delay of the critical path. It always has to be guaranteed that the signal transmission can be complete along any path within one clock cycle [LBS10]. Therefore, the processor operating frequency is reduced based on the new critical path delay.

As discussed above, a shorter sampling period can potentially provide a better control performance. Therefore, with a smaller processor operating frequency, the sampling period increases and the control performance gets deteriorated, which is dangerous and thus highly unwanted for safety-critical applications in EVs, such as electric motor control. To deal with the above situation, the same optimization framework can be slightly modified to re-optimize the controller with the longer sampling period, which results from processor aging, aiming to ensure that the control performance is kept with an inconsiderable compromise on battery usage.

1.3 Organization

This thesis comprises six chapters. Chapter 1 is the introduction. The mathematical background, including the necessary basics of control theory and optimization techniques, is presented in Chapter 2. Feedback control applications are first described, following which are the linear state-feedback control law and the non-linear MPC. The relationship between the control performance and the sampling period is shown based on an Electronic Wedge Brake (EWB) developed by Siemens [FRBW+07]. The presented optimization techniques include the iterative interior-point method, which can be deployed to solve convex problems, and Particle Swarm Optimization (PSO), gradient-based Sequential Quadratic Programming (SQP), and genetic algorithms for non-convex single- and multi-objective problems.

Chapter 3 discusses memory-aware automotive control systems design. The memory analysis technique that computes the guaranteed WCET reduction due to consecutive executions of one control program is first given. A motivational example is used for the illustration purpose. The control parameters, such as sampling periods and sensor-to-actuator delays, are then derived based on the WCET results. The controller design techniques for both the conventional memory-oblivious uniform sampling scheme and the proposed memory-aware non-uniform sampling scheme are elaborated. The approach to compute the optimal sampling order is explained. Experimental results are reported at the end of the chapter. Partial contents of this chapter are included in [CGC+17].

Chapter 4 discusses computation-aware automotive control systems design. The OS used in automobiles is described and the restriction on the choice of sampling periods is addressed. The multirate controller design technique is presented to reduce the processor load while satisfying the control performance requirement and system constraints. Contents of this chapter also appear in [CGCHng].

Chapter 5 discusses battery- and aging-aware automotive control systems design. The design objective of battery usage is introduced with battery characteristics. The processor aging and its influence on the control system is analyzed. Then the optimization framework and flow are shown. Experimental results can be found at the end of the chapter and have been published in [CPG+14].

The conclusion of this thesis is given in Chapter 6 and possible future work is discussed. All real-world applications used in this thesis are detailed in the appendix. Relevant information can also be found in [CFC+16, CC16, CPG+15].

1.4 List of Publications

Parts of the main contributions presented in this thesis appear in the following publications:

- **Wanli Chang**, Alma Pröbstl, Dip Goswami, Majid Zamani, Samarjit Chakraborty, *Battery- and Aging-Aware Embedded Control Systems for Electric Vehicles,* IEEE Real-Time Systems Symposium (RTSS), 2014.

- **Wanli Chang**, Dip Goswami, Samarjit Chakraborty, Jason Xue, Lei Ju, Sidharta Andalam, *Memory-Aware Embedded Control Systems Design,* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 36(4): 586-599, 2017.

- **Wanli Chang**, Dip Goswami, Samarjit Chakraborty, Arne Hamann, *OS-Aware Automotive Controller Design Using Non-Uniform Sampling,* ACM Transactions on Cyber-Physical Systems, forthcoming.

The following publications are generally related to the area of automotive control systems and implementation resources:

- Martin Lukasiewycz, Sebastian Steinhorst, Florian Sagstetter, **Wanli Chang**, Peter Waszecki, Matthias Kauer, Samarjit Chakraborty, *Cyber-Physical Systems Design for Electric Vehicles,* Euromicro Conference on Digital System Design (DSD), 2012.

- Martin Lukasiewycz, Sebastian Steinhorst, Sidharta Andalam, Florian Sagstetter, Peter Waszecki, **Wanli Chang**, Matthias Kauer, Philipp Mundhenk, Shreejith Shanker, Suhaib Fahmy, Samarjit Chakraborty, *System Architecture and Software Design for Electric Vehicles,* Design Automation Conference (DAC), 2013.

- **Wanli Chang**, Martin Lukasiewycz, Sebastian Steinhorst, Samarjit Chakraborty, *Dimensioning and Configuration of EES Systems for Electric Vehicles with Boundary-Conditioned Adaptive Scalarization,* IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2013. (**Best Paper Nominee**)

- **Wanli Chang**, Alma Pröbstl, Dip Goswami, Majid Zamani, Samarjit Chakraborty, *Reliable CPS Design for Mitigating Semi-Conductor and Battery Aging in Electric Vechicles,* IEEE International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA), 2015.

- Debayan Roy, Licong Zhang, **Wanli Chang**, Dip Goswami, Samarjit Chakraborty, *Multi-Objective Co-Optimization of FlexRay-based Distributed Control Systems,* IEEE Real-Time Embedded Technology & Applications Symposium (RTAS), 2016.

- Debayan Roy, Licong Zhang, **Wanli Chang**, Samarjit Chakraborty, *Automated Synthesis of Cyber-Physical Systems from Joint Controller/Architecture Specifications,* Forum on Specification & Design Languages (FDL), 2016.

- **Wanli Chang**, Debayan Roy, Licong Zhang, Samarjit Chakraborty, *Model-based Design of Resource-Efficient Automotive Control Software,* International Conference on Computer-Aided Design (ICCAD), 2016.
- Samarjit Chakraborty, Mohammad Abdullah Al Faruque, **Wanli Chang**, Dip Goswami, Marilyn Wolf, Qi Zhu, *Automotive Cyber-Physical Systems: A Tutorial Introduction*, IEEE Design & Test, 33(4): 92-108, 2016.
- **Wanli Chang**, Samarjit Chakraborty, *Resource-Aware Automotive Control Systems Design: A Cyber-Physical Systems Approach,* Foundations and Trends® in Electronic Design Automation, 10(4): 249-369, 2016.
- **Wanli Chang**, Licong Zhang, Debayan Roy, Samarjit Chakraborty, *Control/Archi-tecture Co-Design for Cyber-Physical Systems,* Handbook of Hardware/Software Codesign, Springer, 2017.
- S. Ramesh, Birgit Vogel-Heuser, **Wanli Chang**, Debayan Roy, Licong Zhang, Samarjit Chakraborty, *Specification, Verification and Design of Evolving Automotive Software,* Design Automation Conference (DAC), 2017.
- Michael Balszun, Debayan Roy, Licong Zhang, **Wanli Chang**, Samarjit Chakraborty, *Effectively Utilizing Elastic Resources in Networked Control Systems,* IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2017. (**Best Paper Award**)
- Debayan Roy, Licong Zhang, **Wanli Chang**, Sanjoy Mitter, Samarjit Chakraborty, *Ensuring Safety through Semantics-Preserving Co-Synthesis of Cyber-Physical Systems,* Proceedings of the IEEE, forthcoming.

2 Mathematical Background

The necessary mathematical background of this thesis includes the basics of the control theory and the optimization techniques. Feedback control applications are described from the state-space modelling, discretization and controllability, to control performance metrics and constraints. For the linear state-feedback control, the closed-loop system stability is analyzed. The feedback and feedforward gains are computed. The pole-placement technique is briefly discussed. For the non-linear MPC, both the unconstrained and constrained cases are considered. A small case study based on the EWB developed by Siemens is shown to illustrate the relationship between the control performance and the sampling period.

The novel optimization techniques proposed in this thesis are developed on top of four methods. The iterative interior-point method is used for convex problems. In particular, it solves the quadratic programming problem in the constrained MPC. PSO, gradient-based SQP and genetic algorithms are all deployed for the non-convex pole-placement problem. The general aim is to optimize the control performance, while respecting all constraints. In the context of battery-aware automotive control systems design, there is another objective — the battery usage — to optimize, which makes it a multi-objective optimization problem.

2.1 Control Theory

2.1.1 Feedback Control Applications

Plant dynamics: A control application is responsible for controlling a plant or dynamic system. In particular, linear Single-Input Single-Output (SISO) control applications are considered, where the dynamic behavior is modelled by a set of differential equations,

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t), \\ y(t) &= Cx(t), \end{aligned} \tag{2.1}$$

where $x(t) \in \mathbb{R}^l$ is the system state, $\dot{x}(t)$ is the derivative of $x(t)$ with respect to time, $y(t)$ is the system output and $u(t)$ is the control input. The number of system states is l. The system (or state) matrix is A. The input matrix is B. The output matrix is C. These matrices A, B and C are physical properties of the plant. System poles are eigenvalues of A. In a state-feedback control algorithm, $u(t)$ is computed utilizing $x(t)$ (feedback signals) and then applied to the plant, which is expected to achieve certain desired behavior.

Discretized dynamics: In most applications, the controller is implemented in a digital fashion on a computer. This implies that the system states must be sampled when measured by the sensors. Assuming the sampling period to be h, the sampled system state is denoted as

$$x[k] = x(t_k), \quad t_k = kh, \ k = 0, 1, 2, 3, \cdots . \tag{2.2}$$

Similarly, the sampled system output is

$$y[k] = y(t_k). \tag{2.3}$$

The control input taking discrete values is denoted as $u[k]$, which is passed through a Zero-Order Hold (ZOH) and applied to the plant. The output of the ZOH is given by

$$u(t) = u[k], \quad t_k \leq t < t_{k+1}. \tag{2.4}$$

Now the discretized dynamics of (2.1) can be derived. First of all,

$$y[k] = y(t_k) = Cx(t_k) = Cx[k]. \tag{2.5}$$

The solution to (2.1) is

$$x(t) = e^{A(t-t_0)}x(t_0) + \int_{t_0}^{t} e^{A(t-\tau)}Bu(\tau)d\tau. \tag{2.6}$$

Taking $t_0 = t_k$ and $t = t_{k+1}$, since $t_{k+1} - t_k = h$,

$$\begin{aligned} x[k+1] &= e^{Ah}x[k] + \int_{t_k}^{t_{k+1}} e^{A(t_{k+1}-\tau)}Bu(\tau)d\tau \\ &= e^{Ah}x[k] + \int_{t_k}^{t_{k+1}} e^{A(t_{k+1}-\tau)}d\tau Bu[k]. \end{aligned} \tag{2.7}$$

Defining a new variable $\tau' = t_{k+1} - \tau$, there is $d\tau' = -d\tau$. As τ ranges from t_k to t_{k+1}, τ' ranges from h to 0. Therefore,

$$\int_{t_k}^{t_{k+1}} e^{A(t_{k+1}-\tau)}d\tau B = \int_{h}^{0} e^{A\tau'}(-d\tau')B = \int_{0}^{h} e^{A\tau'}d\tau' B. \tag{2.8}$$

Then, the discretized dynamics is

$$\begin{aligned} x[k+1] &= A_d x[k] + B_d u[k], \\ y[k] &= Cx[k], \end{aligned} \tag{2.9}$$

where

$$A_d = e^{Ah},\ B_d = \int_0^h (e^{A\tau'} d\tau')B. \tag{2.10}$$

If A is invertible, the integral in B_d can be got rid of. Utilizing

$$\frac{d}{dt}e^{Ah} = e^{Ah}A, \tag{2.11}$$

B_d can be derived to be

$$B_d = A^{-1}\int_0^h Ae^{A\tau'}d\tau' B = A^{-1}e^{A\tau'}\Big|_{\tau'=0}^{h} B = A^{-1}(e^{Ah} - \mathbf{I})B, \tag{2.12}$$

where **I** is the identity matrix.

System controllability: Controllability of a discrete system is defined as the ability to transfer the system from any initial state $x[0] = x_0$ to any desired final state $x[k_f] = x_f$. Taking $k = 1, 2, \ldots, n$,

$$\begin{aligned} x[1] &= A_d x[0] + B_d u[0] \\ x[2] &= A_d x[1] + B_d u[1] = A_d^2 x[0] + A_d B_d u[0] + B_d u[1] \\ &\vdots \\ x[n] &= A_d^n x[0] + A_d^{n-1} B_d u[0] + \cdots + B_d u[n-1]. \end{aligned} \tag{2.13}$$

$x[n]$ can be written in a matrix form as

$$x[n] - A_d^n x[0] = [B_d \quad A_d B_d \quad \cdots \quad A_d^{n-1} B_d] \begin{bmatrix} u[n-1] \\ u[n-2] \\ \cdots \\ u[0] \end{bmatrix}, \tag{2.14}$$

where $\mathcal{CO} = [B_d \quad A_d B_d \quad \cdots \quad A_d^{n-1} B_d]$ is the square controllability matrix. If $\mathcal{CO}$ is non-singular, the unique solution for the control input sequence in (2.14) is given by

$$\begin{bmatrix} u[n-1] \\ u[n-2] \\ \cdots \\ u[0] \end{bmatrix} = \mathcal{CO}^{-1}(x[n] - A_d^n x[0]). \tag{2.15}$$

In this case, for any $x[n] = x_f$, the solution in (2.15) determines the control input sequence that transfers the initial system state x_0 to the desired system state x_f in n steps. It follows that the controllability condition is equivalent to the non-sigularity of the controllability matrix $\mathcal{CO}$.

Control performance: Settling time is a widely used metric to quantify the control performance, especially for real-time control applications. The time it takes for the system output $y[k]$ to reach and stay in a closed region around the reference value r (e.g., $0.98r$ to $1.02r$) is

the settling time of a control loop and denoted as t_s. Shorter settling time implies better control performance.

Another popular performance metric in the control context is a quadratic cost function, which is given by

$$J = \sum_{k=0}^{N-1} [x^T[k]Qx[k] + u^T[k]Ru[k]], \tag{2.16}$$

assuming that the system state $x[k]$ is expected to stabilize at 0. Q is a positive semi-definite matrix and R is a positive definite matrix. Since SISO applications are considered in this thesis, $u^T[k] = u[k]$. In some cases, there is also a term $x^T[N]Sx[N]$ added in the end, where S is a positive semi-definite matrix. Q, R and S are all weight matrices. To optimize the control performance, J is minimized. Among the three terms,

- $x^T[k]Qx[k]$ penalizes the transient state deviation;
- $u^T[k]Ru[k]$ penalizes the control effort;
- $x^T[N]Sx[N]$ penalizes the final state deviation.

System constraints: In almost every real-world system, due to the physical constraint of the actuator, there is some maximum available control input signal and the controller needs to be designed such that the maximum value of $u[k]$ does not exceed this limit U_{max}, i.e., $u[k] \leq U_{max}$. This is the constraint of the input saturation. Another constraint is on the peak overshoot, which is defined as

$$y_{max} - r \leq \phi_0 r, \tag{2.17}$$

where y_{max} is the maximum system output and ϕ_0 is the overshoot threshold. The constraint on the steady-state error has been discussed when defining the settling time. The system output $y[k]$ has to reach and stay in a closed region around r, i.e., the system has to settle. If the region is $[0.98r, 1.02r]$, then the steady-state error tolerance $\phi_e = 2\%$. Besides, in many real-time control applications, there is a maximum allowed settling time t_s^0 [LLB+12]. That is, $t_s \leq t_s^0$ must be satisfied.

2.1.2 State-Feedback Control Law

In the state-feedback control, the control input $u[k]$ is computed based on the system state $x[k]$. There can be both linear and non-linear controllers, depending on the relationship between $u[k]$ and $x[k]$.

Linear controller: The general structure of a linear controller is as follows,

$$u[k] = Kx[k] + Fr, \tag{2.18}$$

where K is the feedback gain and F is the feedforward gain. Clearly, the relationship between $u[k]$ and $x[k]$ is linear.

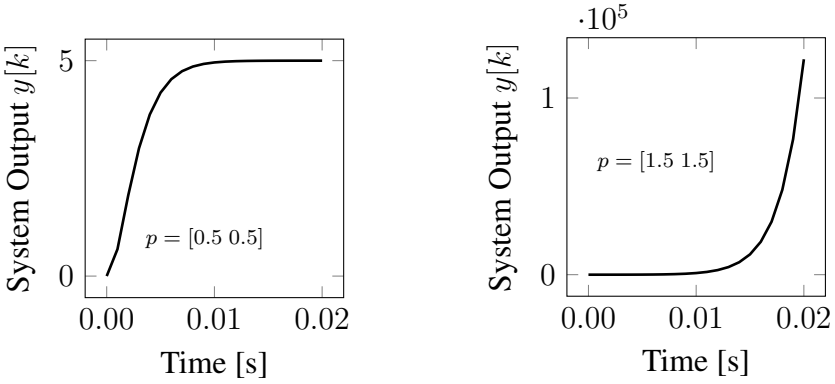

Figure 2.1: Different system ouput responses for stable and unstable poles

Closed-loop system: With the linear controller as shown in (2.18), the system dynamics in (2.9) becomes

$$x[k+1] = (A_d + B_d K)x[k] + B_d F r, \tag{2.19}$$

i.e., closed-loop dynamics.

Pole-placement: Different locations of closed-loop system poles, i.e., eigenvalues of $(A_d + B_d K)$, result in different system behaviors. In pole-placement, poles are placed in desired locations (eigenvalues are set) often to fulfill various high-level goals, such as control performance maximization and system constraints satisfaction. The desired poles p can be decided with empirical or optimization techniques. This method is feasible since there is the freedom to choose the feedback gain K.

Feedback and feedforward gain: Once pole locations are decided, the following characteristics equation of z can be constructed with these poles as roots:

$$z^n + \gamma_1 z^{n-1} + \gamma_2 z^{n-2} + \cdots + \gamma_n = 0. \tag{2.20}$$

Then the following is defined,

$$\gamma_c(A_d) = A_d^n + \gamma_1 A_d^{n-1} + \gamma_2 A_d^{n-2} + \cdots + \gamma_n \mathbf{I}. \tag{2.21}$$

According to Ackermann's formula [AU98], the feedback gain used to stabilize the closed-loop system is calculated as

$$K = [0 \ \cdots \ 0 \ 1]\, \mathcal{C}\mathcal{O}^{-1}\, \gamma_c(A_d). \tag{2.22}$$

The static feedforward gain F used to make the system output $y[k]$ track the reference r is computed by

$$F = 1/(C_d(\mathbf{I} - A_d - B_d K)^{-1} B_d). \tag{2.23}$$

System stability: All eigenvalues of $(A_d + B_d K)$ must have absolute values of less than unity in order to ensure system stability [AM09]. This is illustrated with a double integrator example as follows,

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 \end{bmatrix}. \tag{2.24}$$

The initial state is $\begin{bmatrix} 0 & 0 \end{bmatrix}$ and the reference value is 5. The sampling period is set as $h = 0.001$s. The system output responses for two sets of poles $p = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}$ and $p = \begin{bmatrix} 1.5 & 1.5 \end{bmatrix}$ are shown in Figure 2.1.

Restricted pole-placement: If the system is controllable, i.e., $\mathcal{CO}$ has full rank, there is no restriction on pole locations. The feedback gain K can be determined with (2.22). If $\mathcal{CO}$ does not have full rank, some of the poles cannot be modified with any choice of K and thus are uncontrollable. Since $\mathcal{CO}$ is not invertible, (2.22) does not work, either. In this case, if the uncontrollable poles are stable (with absolute values of less than unity), then the system is stabilizable. Restricted pole-placement can be used for stabilizable systems in the way that only controllable poles are placed in the desired locations and uncontrollable poles remain untouched. Therefore, for the automotive control systems design discussed in this thesis, the system is required to be at least stabilizable, if not controllable.

Non-linear MPC: In MPC, the relationship between the control input $u[k]$ and the system state $x[k]$ is non-linear. Considering the quadratic cost function in (2.16), the goal is to find a sequence of control inputs $u[0], u[1], \ldots, u[N-1]$ that minimizes this cost function. Assuming that $S = Q$ and expanding (2.16),

$$\begin{aligned} J = x^T[0]Qx[0] + [x^T[1]\ x^T[2]\ \ldots\ x^T[N]] \begin{bmatrix} Q & 0 & \cdots & 0 \\ 0 & Q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & Q \end{bmatrix} \begin{bmatrix} x[1] \\ x[2] \\ \vdots \\ x[N] \end{bmatrix} \\ + [u^T[0]\ u^T[1]\ \ldots\ u^T[N-1]] \begin{bmatrix} R & 0 & \cdots & 0 \\ 0 & R & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & R \end{bmatrix} \begin{bmatrix} u[0] \\ u[1] \\ \vdots \\ u[N-1] \end{bmatrix}. \end{aligned} \tag{2.25}$$

Computing the system states as

$$\begin{bmatrix} x[1] \\ x[2] \\ \vdots \\ x[N] \end{bmatrix} = \begin{bmatrix} B_d & 0 & \cdots & 0 \\ A_dB_d & B_d & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A_d^{N-1}B_d & A_d^{N-2}B_d & \cdots & B_d \end{bmatrix} \begin{bmatrix} u[0] \\ u[1] \\ \vdots \\ u[N-1] \end{bmatrix} + \begin{bmatrix} A_d \\ A_d^2 \\ \vdots \\ A_d^N \end{bmatrix} x[0], \tag{2.26}$$

the quadratic cost function becomes

$$J = x^T[0]Qx[0] + (\bar{S}U + \bar{T}x[0])^T\bar{Q}(\bar{S}U + \bar{T}x[0]) + U^T\bar{R}U, \tag{2.27}$$

where

$$U = \begin{bmatrix} u[0] \\ u[1] \\ \vdots \\ u[N-1] \end{bmatrix}, \quad \bar{Q} = \begin{bmatrix} Q & 0 & \cdots & 0 \\ 0 & Q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & Q \end{bmatrix}, \quad \bar{R} = \begin{bmatrix} R & 0 & \cdots & 0 \\ 0 & R & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & R \end{bmatrix},$$
$$\bar{T} = \begin{bmatrix} A_d \\ A_d^2 \\ \vdots \\ A_d^N \end{bmatrix}, \quad \bar{S} = \begin{bmatrix} B_d & 0 & \cdots & 0 \\ A_d B_d & B_d & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A_d^{N-1} B_d & A_d^{N-2} B_d & \cdots & B_d \end{bmatrix}. \tag{2.28}$$

Expanding (2.27),

$$\begin{aligned} J &= x^T[0] Q x[0] + (U^T \bar{S}^T + x^T[0] \bar{T}^T) \bar{Q} (\bar{S} U + \bar{T} x[0]) + U^T \bar{R} U \\ &= \frac{1}{2} U^T 2(\bar{S}^T \bar{Q} \bar{S} + \bar{R}) U + 2 x^T[0] \bar{T}^T \bar{Q} \bar{S} U + x^T[0] (Q + \bar{T}^T \bar{Q} \bar{T}) x[0]. \end{aligned} \tag{2.29}$$

Defining

$$H = 2(\bar{S}^T \bar{Q} \bar{S} + \bar{R}), \quad F = 2 \bar{T}^T \bar{Q} \bar{S}, \quad Y = \bar{T}^T \bar{Q} \bar{T}, \tag{2.30}$$

there is

$$J = \frac{1}{2} U^T H U + x^T[0] F U + x^T[0] Y x[0]. \tag{2.31}$$

It is clearly seen that this quadratic cost function J depends on the control input sequence U, for a given initial syste state $x[0]$. H, F and Y involve physical properties of the plant under control (A_d and B_d) and the weights defined in the cost function (Q and R). It is important to note that H is semi-definite, since Q is positive semi-definite, R is positive definite, and S has full column rank.

Solving unconstrained MPC: If there are no constraints considered, computing the first derivative can be used to minimize the cost function (2.31). Solving

$$\frac{\partial J}{\partial U} = H U + F^T x[0] = 0, \tag{2.32}$$

the optimal control input sequence is

$$U^* = -H^{-1} F^T x[0]. \tag{2.33}$$

Since

$$\frac{\partial^2 J}{\partial U^2} = H, \tag{2.34}$$

is positive semi-definite, the obtained U^* in (2.33) is a minimum. If H is non-singular (i.e., positive definite), the optimal U^* is unique.

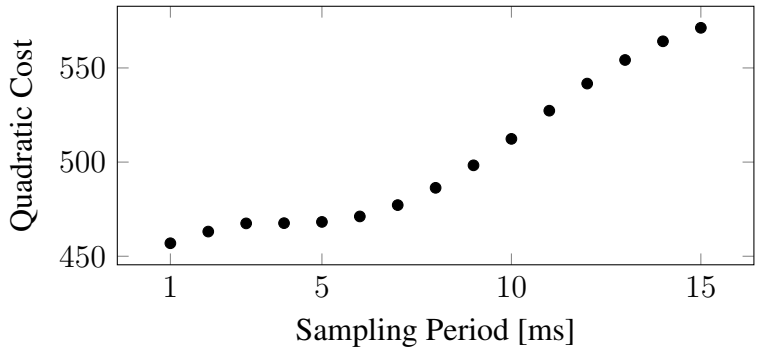

Figure 2.2: The relationship between the control performance and the sampling period for the EWB

Solving constrained MPC: In practice, there are often system constraints. For instance, the input saturation might require

$$u_{min} \leq u[k] \leq u_{max}. \tag{2.35}$$

This is a constrained convex quadratic programming problem and can be solved by a number of optimization techniques. Both the active-set method and the interior-point method are widely used. In this thesis, the interior-point method is used and will be elaborated in the next section.

2.1.3 Control Performance and Sampling Period

The relationship between the control performance and the sampling period is explored with the EWB described in Section A. The input saturation constraint is that the supplied voltage of the motor cannot exceed $12V$. MPC is used in designing the control algorithm. The quadratic cost function as in (2.16) serves as the control performance metric. The term of the control input is excluded, since the input saturation is explicitly considered. The number of steps to look ahead N is set to be 4. The weight matrix Q is assumed to be an identity matrix. Varying the sampling period from $1ms$ to $15ms$ in integers, results are reported in Figure 2.2. It can be clearly seen that as the sampling period gets longer, the quadratic cost increases, which means that the control performance deteriorates. The underlying reason is that the controller is able to interact with the plant more frequently under a shorter sampling period.

2.2 Optimization Techniques

2.2.1 Interior-Point Method

The interior-point method is widely used for convex optimization problems. In this thesis, it is taken to solve the quadratic programming problem in MPC with inequality constraints, which

is formulated as follows,

$$\begin{aligned} \min_x q(x) &= \tfrac{1}{2}x^T G x + x^T d, \\ &\text{subject to} \\ Ax &\geq b, \end{aligned} \tag{2.36}$$

where G is symmetric and positive semi-definite.

Specializing the general Karush-Kuhn-Tucker (KKT) conditions for constrained optimization [NW06], the necessary conditions for (2.36) can be obtained: If x^* is a solution of (2.36), there is a Lagrange multiplier vector λ^* such that the following conditions are satisfied for $(x, \lambda) = (x^*, \lambda^*)$,

$$\begin{aligned} Gx - A^T\lambda + d &= 0, \\ Ax - b &\geq 0, \\ (Ax - b)_i\lambda_i &= 0, \quad i = 1, 2, \ldots, m, \\ \lambda &\geq 0. \end{aligned} \tag{2.37}$$

Introducing $y = Ax - b$, these conditions in (2.37) can be rewritten as

$$\begin{aligned} Gx - A^T\lambda + d &= 0, \\ Ax - y - b &= 0, \\ y_i\lambda_i &= 0, \quad i = 1, 2, \ldots, m, \\ (y, \lambda) &\geq 0. \end{aligned} \tag{2.38}$$

Since both the objective function and the feasible region are convex, these necessary conditions are also sufficient.

Defining

$$F(x, y, \lambda) = \begin{bmatrix} Gx - A^T\lambda + d \\ Ax - y - b \\ Y\Lambda e \end{bmatrix}, \quad (y, \lambda) \geq 0, \tag{2.39}$$

where

$$Y = \text{diag}(y_1, y_2, \ldots, y_m), \quad \Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_m), \quad e = (1, 1, \ldots, 1)^T, \tag{2.40}$$

Solving

$$F(x, y, \lambda) = 0, \quad (y, \lambda) \geq 0, \tag{2.41}$$

generates the optimal x^*. There are a number of iterative ways to solve (2.41), including Newton's method with slight modification [NW06] and Mehrotra's predictor-corrector algorithm [Meh92].

2.2.2 Particle Swarm Optimization

PSO is a popular method to solve non-convex non-linear optimization problems and can be used to locate the poles that maximize the control performance. A group of particles are randomly

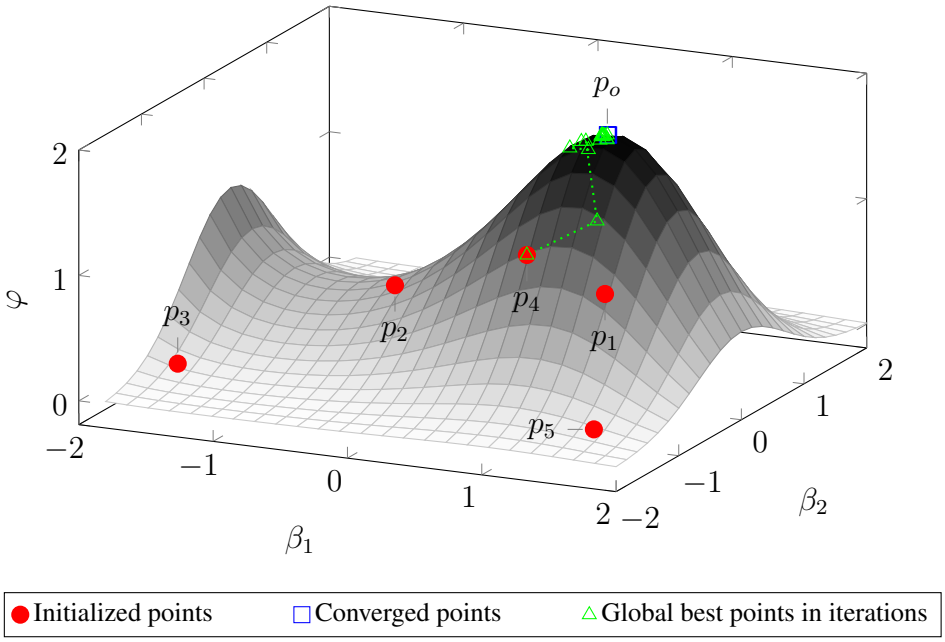

Figure 2.3: With the particle swarm optimization method, five particles are randomly initilized and converge to the optimum p_o.

initialized in the decision space with positions and velocities. They search for the optimum by iteratively updating their positions. The search is led by two points. The first is the local best point (in terms of the objective value) that has been reached by a particle. Every particle has its own local best point. The second is the global best point that has been reached by all particles.

The velocity of a particle is determined by the following equation,

$$V_{\text{new}} = \alpha_0 V_{\text{current}} + \alpha_1 \text{rand}(0,1)(P_{\text{lbest}} - P_{\text{current}}) + \alpha_2 \text{rand}(0,1)(P_{\text{gbest}} - P_{\text{current}}), \tag{2.42}$$

where V_{new} is the new velocity, V_{current} is the current velocity, P_{current} is the current position, P_{lbest} is the local best point of this particle and P_{gbest} is the global best point of all particles. $\text{rand}(0,1)$ is a random number with uniform distribution from the open interval $(0,1)$. α_0 is the weight inertia. α_1 and α_2 are cognitive and social scaling parameters. Widely used values for these parameters are

$$\alpha_0 = 0.4, \quad \alpha_1 = \alpha_2 = 2, \tag{2.43}$$

which have been shown to have good performance in many optimization scenarios. The new position of this particle is

$$P_{\text{new}} = P_{\text{current}} + V_{\text{new}}. \tag{2.44}$$

The algorithm is terminated once all particles have converged or the maximum number of iterations has been reached.

A numerical example is used to illustrate the PSO. The formulation is as follows,

$$\begin{gathered} \max_{\mathbb{D}} \varphi = e^{-\frac{1}{3}{\beta_1}^3 + \beta_1 - {\beta_2}^2} \\ \text{subject to} \\ \mathbb{D} = \{(\beta_1, \beta_2) | -1.8 \leq \beta_1 \leq 2, \ -2 \leq \beta_2 \leq 2\}, \end{gathered} \tag{2.45}$$

where β_1 and β_2 are two continuous decision variables, constrained in the decision space $\mathbb{D}$. The objective to maximize is φ. As shown in Figure 2.3, five particles are randomly initialized at p_1, p_2, p_3, p_4 and p_5. After 22 iterations, all five particles converge to the points around the optimum p_o $(1.0006, 0.0033, 1.9477)$. The path showing how the global best point evolves iteratively is drawn, with certain points that are too close to others omitted for better illustration.

2.2.3 Sequential Quadratic Programming

SQP is a popular gradient-based optimization technique for single-objective non-convex problems. The entire process begins with a chosen starting point in the decision space, from which SQP searches for a sequence of points, hopefully closer and closer to the optimal point until either the optimal point is reached or termination criteria are satisfied. The question is then how to iteratively find the next point, which is supposed to be better than the current point in terms of the objective value. It needs to be known what is the search direction and how far to go along this direction. Neither of them is easy to compute for a non-convex problem with significant non-linearity. Therefore, an approximated local quadratic model is built around this current point as

$$f(x_p) = \frac{1}{2} x_p^T H x_p + c_q^T x_p, \tag{2.46}$$

where x_p is the current point. $f(x_p)$ is the objective value, H is the symmetric Hessian matrix at x_p. Both c_q and x_p are column vectors with the same number of elements.

The search direction can be computed in various ways. For instance, the direction of steepest descent d_s with potentially the best local improvement in the objective value f can be used. d_s is the opposite of the gradient as,

$$d_s = -\frac{df}{dx_p} = -x_p^T H - c_q^T. \tag{2.47}$$

After computing the search direction, the next step is to determine how far to go in this direction. Assuming that Δ is the radius of the region where the confidence in the accuracy of the approximated local quadratic model is high, a list of steps $\{\frac{\Delta}{n_{ss}}, \frac{2\Delta}{n_{ss}}, \ldots, \Delta\}$ are tried, where n_{ss} is the total number of steps and can be decided customarily. The step size with the best objective value f and all constraints fulfilled is chosen. Then a new quadratic model is established around this point and the search process is iterated. When no step size gives improvement in f, the optimization algorithm is terminated. The other termination criterion is that the set maximum number of iterations is exceeded. It is noted that solving the quadratic model in an exact manner as in Section 2.2.1 is not of much interest, since the model itself is only an approximation of the original problem in a restricted space. Instead, the aim is to compute the search direction and the step size that locally find a next point better than the current point.

SQP is originally meant for single-objective optimization problems, yet can also be extended to multi-objective optimization problems, where all objectives are converted into one. One common way to do so is constructing a new objective f_0 to be minimized with a weighted sum of all objectives. Taking two objectives f_1 and f_2 as as an example,

$$f_0 = w_1 \times f_1 + w_2 \times f_2, \tag{2.48}$$

where w_1 and w_2 are weights of the two objectives f_1 and f_2 and

$$w_1 + w_2 = 1. \tag{2.49}$$

In multi-objective optimization, if a point is worse than another point in every objective, it is said to be dominated. If a point is not dominated by any other point in the feasible decision space, then this point is called a Pareto point. There can be multiple Pareto points, which form the Pareto front. If SQP is able to reach the global optimum, when f_0 in (2.48) is minimized as the objective, the solution point must be on the Pareto front. This can be proven by contradiction. If there is another point x', which dominates the solution point x_0, then

$$f_1(x') < f_1(x_0), \quad f_2(x') < f_2(x_0). \tag{2.50}$$

From (2.48),

$$f_0(x') < f_0(x_0). \tag{2.51}$$

This contradicts that x_0 is the global optimum for minimizing f_0.

2.2.4 Non-Dominated Sorting Genetic Algorithm

There is a variety of stochastic methods developed to solve non-convex optimization problems. Among them, the Non-Dominated Sorting Genetic Algorithm (NSGA) mimicking the process of natural selection is widely applied to deal with multiple objectives [SD94]. The second version of NSGA has three advantages over its predessor: (i) the computational complexity of non-dominated sorting is reduced; (ii) elitism speeds up the performance; (iii) the sharing parameter is no longer needed [DPAM02]. The entire flow of NSGA is illustrated in Figure 2.4, consisting of four major steps — initilization, genetic operation, environmental selection, and termination.

At first, an initial population is generated. This process is often random. Offsprings are then produced with genetic operators including mainly crossover and mutation. The crossover function tries to keep the good genes of parents, which generally means that the offsprings are close to parents in the decision space. The mutation function deviates from parents and produces offsprings in unexplored regions of the objective space. It is not necessary that an offspring only has two parents, as inspired in biology. The population is now composed of these offsprings.

During each successive generation including this one, a proportion of the existing population is selected as parents to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions as measured by a fitness function are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as the former process may be very time-consuming. The optimization objective function is often used for fitness evaluation. In some problems, it is hard or even impossible to explicitly define the fitness expression and a simulation may be used to determine the fitness function value.

Elitism can be implemented for environmental selection, so that the next generation is selected among both parents and offsprings. This not only speeds up convergence but also ensures that good solutions will not be lost once they are found. In order to deal with multiple objectives,

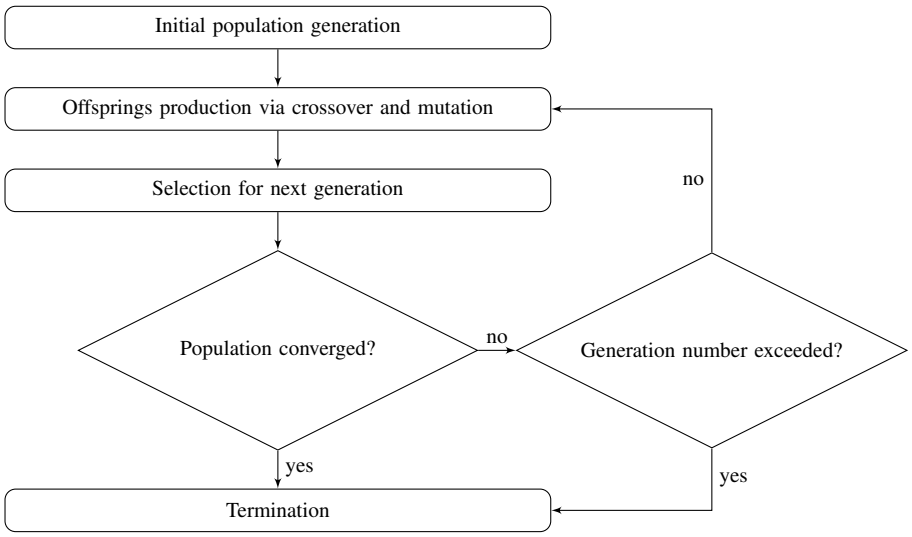

Figure 2.4: Optimization flow of the non-dominated sorting genetic algorithm

all parents and offsprings are sorted and ranked by its level of domination. For each solution, if it is not dominated by any other solution, then it is given a rank of 1. All solutions with the rank of 1 form the first front. Then the first front is removed and the second front is formed with non-dominated solutions among the rest. All solutions on the second front have a rank of 2. This process is iterated until all solutions are assigned a rank. The new generation is filled in the way that solutions with lower ranks have priorities. This non-dominated sorting feature in environmental selection favors those solutions on or close to the Pareto front. There are two termination conditions — whether the population has converged and whether the maximum allowed number of generations has been exceeded.

3

Memory-Aware Automotive Control Systems Design

In the cost-sensitive and resource-constrained embedded platforms that are used to implement automotive control algorithms, memory subsystems constitute an important component, and on-chip caches contribute significantly towards their cost. There have been many efforts on cache reuse maximization, for improving the real-time performance, such as the worst-case execution time (WCET), of embedded software [BW08]. However, the characteristics of control systems and metrics of control performance have not been directly incorporated into these techniques.

In this chapter, a Cyber-Physical System (CPS)-oriented approach brings two very disparate classes of techniques — cache modeling and program analysis on one hand, and controller design on the other hand — together and quantifies the resulting benefit. The idea is to shorten the sampling periods of the control applications with execution (i.e., sampling) orders that increase the cache reuse. Such memory-aware execution of control applications implies non-uniform sampling. Generally, the non-uniform sampling scheme is undesirable in control systems, since the controller design has to deal with switching instability, making it challenging to optimize the control performance. In this presented method, the key is that the sampling order is a design parameter and known in the controller design phase. Exploiting the knowledge of the sampling order, a controller design technique is proposed to improve the control performance.

The organization of this chapter is as follows. Section 3.1 reviews the literature in memory analysis and control theory. Section 3.2 discusses the memory analysis technique to compute the guaranteed WCET reduction due to consecutive executions of one control program. A motivational example is presented to illustrate the technique. Section 3.3 derives the control timing parameters, such as sampling periods and sensor-to-actuator delays, from WCETs. Controller design methods for both uniform and non-uniform sampling are illustrated in Section 3.4, including the pole-placement technique with the hybrid PSO. Section 3.5 computes the optimal

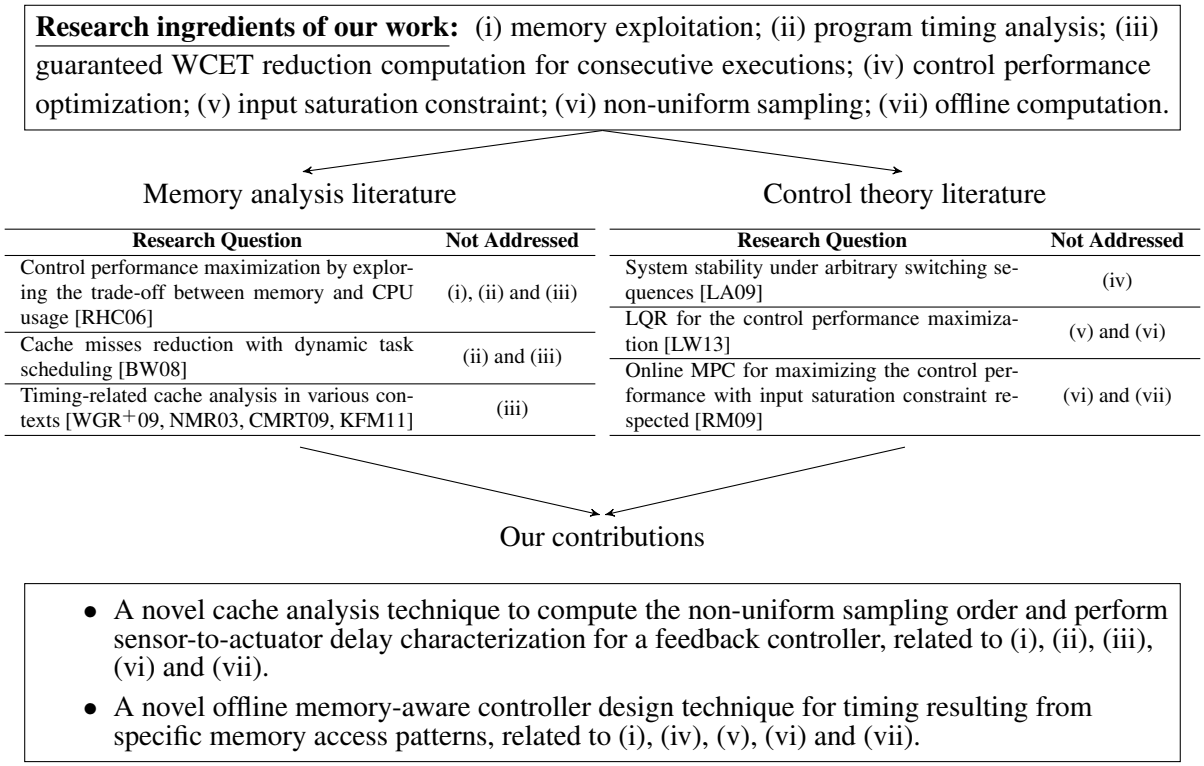

Figure 3.1: Position of the work in Chapter 3 in the memory analysis and control theory literature

sampling order. Experimental results are reported in Section 3.6 and Section 3.7 makes some concluding remarks of this chapter.

3.1 Related Work

The position of Chapter 3 in the memory analysis and control theory literature is illustrated in Figure 3.1. The trade-off between memory and CPU usage in a feedback scheduling system, which dynamically adjusts the sampling periods of control tasks to maximize the overall control performance, is explored in [RHC06]. However, it is not considered how memory can be exploited to improve the system performance. In [BW08], the round-robin scheduling of multiple tasks on an embedded operating system is dynamically tuned during program execution, adapting to changes in work load and external input stimulus. As a result, cache misses are reduced and the system performance is improved.

Cache analysis for timing-related computation has been studied in a number of works [WGR+09, NMR03, CMRT09, KFM11]. The architectural influence on static timing analysis of embedded hard real-time systems is described in [WGR+09]. The cache-related preemption delay is analyzed in [NMR03] for a multi-task embedded system with preemption, and extended to streaming applications in [CMRT09] for cache-aware timing estimation. The set-associative cache is considered in [KFM11]. In this chapter, the existing approach is modified to com-

pute guaranteed WCET reduction due to cache reuse between two consecutive executions of the same application, which is then exploited by the tailored controller design method in the memory-aware sampling order to achieve better control performance.

Works in control theory literature with non-uniform sampling focus on guaranteeing stability of the resulting switched system [LA09]. Generally, theoretical tools such as Common Quadratic Lyapunov Function (CQLF) and Switched Lyapunov Function (SLF) tackle arbitrary switching between sampling periods to assure stability of the overall closed-loop system. In this chapter, as opposed to arbitrary switching, the switching order is precisely known in the design phase, i.e., from the memory-aware sampling order. The aim is further performance optimality by exploiting this additional knowledge about the switching behavior.

In the field of optimal control, techniques such as Linear Quadratic Regulator (LQR) [LW13] are well-developed. By adjusting the weights in the quadratic cost, a trade-off between the input magnitude and the settling time can be achieved. However, these existing optimal control methods cannot be directly applied in the memory-aware automotive control systems design, since first, they are not specifically tuned for switched systems, and second, they do not explicitly consider the constraint on the input signal, which exists in all real-life systems.

The combination of performance optimization and input constraints is addressed by MPC techniques [RM09] — another well-developed area. First, MPC performs online optimization in every sampling period, making it computationally heavy and unsuitable for being implemented on certain resource-constrained embedded platforms. Second, the control law in MPC techniques is non-linear in nature due to online optimization in every sample and existing literature on MPC does not explicitly handle switching between multiple linear sub-systems. Building upon these previous works discussed above, the presented method in this chapter formulates an optimal pole-placement problem, where poles of sub-systems are decision variables, the input constraint is explicitly respected and the settling time is the optimization objective. Unlike MPC, the controller design is performed off-line making scalability a less important aspect.

3.2 Memory Analysis

As discussed in Chapter 1, consecutively executing the same control application increases the cache reuse and decreases the WCET, which potentially improves the control performance with an appropriately designed controller. An example memory-aware sampling order is ($\mathcal{C}_1(1)$, $\mathcal{C}_1(2)$, $\mathcal{C}_1(3)$, $\mathcal{C}_2(1)$, $\mathcal{C}_2(2)$, $\mathcal{C}_2(3)$, $\mathcal{C}_3(1)$, $\mathcal{C}_3(2)$, $\mathcal{C}_3(3)$, $\cdots$), as illustrated in Figure 1.3 In this section, the guaranteed WCET reduction for the second and subsequent executions of a control application ($\mathcal{C}_i(2)$ and $\mathcal{C}_i(3)$ in the above example sampling order) is computed. The technique is derived from previous research [NMR03, CMRT09] and modified to suit this work. Starting from a Control Flow Graph (CFG), equations to compute the Reaching Cache States (RCS) and Live Cache States (LCS) of each node are set up, based on which the fixed-point computation is performed. Afterwards, the guaranteed WCET reduction can be calculated. Figure 3.2 presents a motivational example to illustrate this approach. Only instruction cache is considered in this chapter.

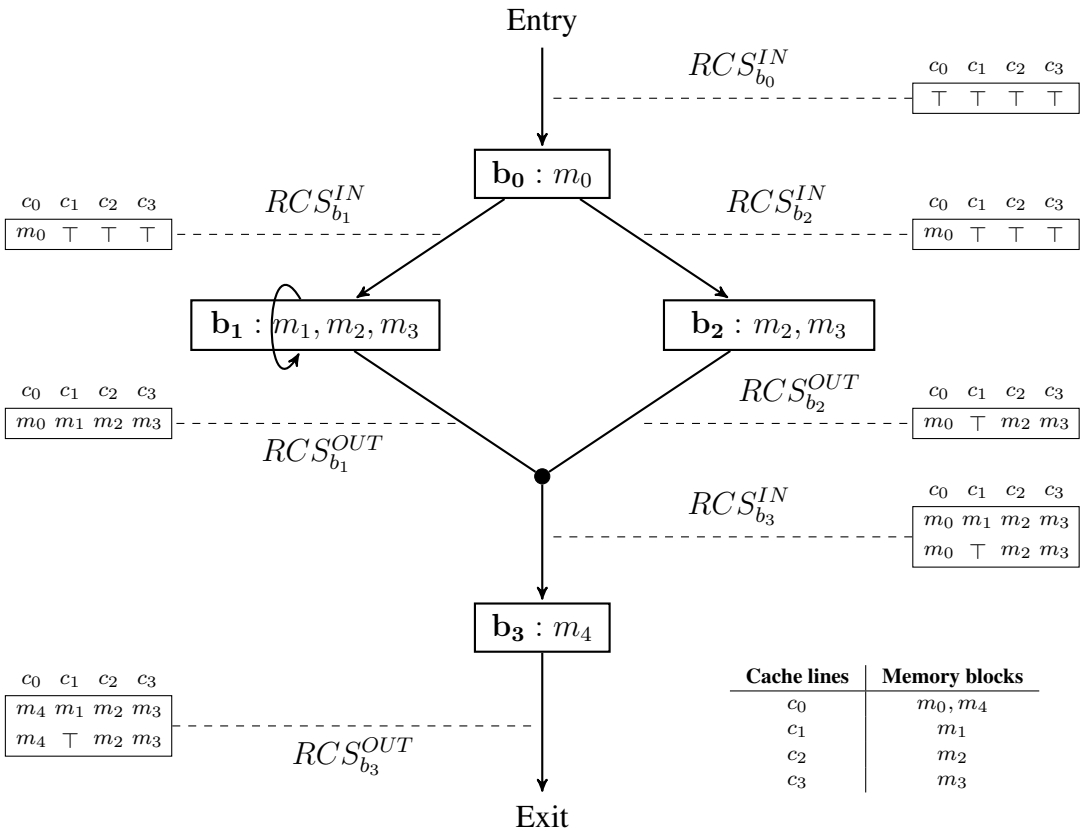

Figure 3.2: A motivational example for memory analysis. Five memory blocks are mapped to four cache lines. Memory blocks executed by each basic block are shown. RCS^{IN} and RCS^{OUT} in the initialization phase are illustrated.

3.2.1 Basic Definitions

In the two-level memory hierarchy of the typical automotive control system shown in Figure 1.1, there are N_c cache lines, denoted as $CL = \{c_0, c_1, \ldots, c_{N_c-1}\}$ and the flash main memory has N_m blocks, denoted as $M = \{m_0, m_1, \ldots, m_{N_m-1}\}$. Each memory block is mapped to a fixed cache line. The example in Figure 3.2 has four cache lines and five memory blocks. A basic block is a straight-line sequence of code with only one entry point and one exit point. This restriction makes a basic block highly amenable for program analysis. The presented CFG, consisting of four basic blocks $B = \{b_0, b_1, b_2, b_3\}$, has all the three key elements of a control program, i.e., sequential basic blocks, branches and a loop. Therefore, it is suitable for illustrating the memory analysis technique.

There are three key terms in memory analysis that are described as follows:

- *cache states*: A cache state cs is described as a vector of N_c elements. Each element $cs[i]$, where $i \in \{0, 1, \ldots, N_c - 1\}$, represents the memory block in the cache line c_i. When the cache line c_i holds the memory block m_j, where $j \in \{0, 1, \ldots, N_m - 1\}$, $cs[i] = m_j$.

If c_i is empty, it is denoted as $cs[i] = \perp$. If the memory block is unknown, it is denoted as $cs[i] = \top$. CS is the set of all possible cache states.

- *reaching cache states*: RCS of a basic block b_k, denoted as RCS_{b_k}, is the set of all possible cache states when b_k is reached via any incoming path.
- *live cache states*: LCS of a basic block b_k, denoted as LCS_{b_k}, is the set of all possible first memory references to cache lines at b_k via any outgoing path.

Since the focus is on WCET reduction between two consecutive executions of $\mathcal{C}_i$, e.g., $\mathcal{C}_i(1)$ and $\mathcal{C}_i(2)$, it is necessary to compute the RCS of the exit point in $\mathcal{C}_i(1)$ and the LCS of the entry point in $\mathcal{C}_i(2)$. By comparing all possible pairs of cache states, the guaranteed number of cache hits and thus WCET reduction can be calculated. In the following, equations for the RCS and LCS computation are first discussed.

3.2.2 Computation of Cache States

In the RCS computation, gen_{b_k} is first defined as the cache state describing the last executed memory block in every cache line for the basic block b_k. Assuming that b_0 in Figure 3.2 executes m_0 and then m_4, instead of only m_0, the last executed memory block in c_0 is m_4. Therefore, gen_{b_0} is $[m_4, \perp, \perp, \perp]$. For the example in Figure 3.2,

$$\begin{aligned} gen_{b_0} &= [m_0, \perp, \perp, \perp], \quad gen_{b_1} = [\perp, m_1, m_2, m_3], \\ gen_{b_2} &= [\perp, \perp, m_2, m_3], \quad gen_{b_3} = [m_4, \perp, \perp, \perp]. \end{aligned} \tag{3.1}$$

There are two equations involved in the RCS computation that calculate RCS^{IN} and RCS^{OUT}, where RCS^{IN} of a basic block b_k is the RCS before b_k is executed and RCS^{OUT} is the set of all possible cache states after b_k is executed. First, $RCS^{OUT}_{b_k}$ can be calculated from $RCS^{IN}_{b_k}$ as

$$RCS^{OUT}_{b_k} = \{\mathcal{T}(b_k, cs) | cs \in RCS^{IN}_{b_k}\}, \tag{3.2}$$

where $\mathcal{T}$ is a transfer function defined as follows: For any cache state $cs \in CS$ and basic block $b_k \in B$, there is a cache state $cs' = \mathcal{T}(b_k, cs)$, where for any cache line $c_i \in CL$ and $i \in \{0, 1, \ldots, N_c - 1\}$,

$$cs'[i] = \begin{cases} cs[i] & : \ if \ gen_{b_k}[i] = \perp; \\ gen_{b_k}[i] & : \ otherwise. \end{cases} \tag{3.3}$$

$RCS^{IN}_{b_k}$ can be calculated as

$$RCS^{IN}_{b_k} = \bigcup_{p \in predecessor(b_k)} RCS^{OUT}_p, \tag{3.4}$$

where $predecessor(b_k)$ is the set of all immediate predecessors of b_k.

The RCS computation is composed of two phases: initialization and fixed-point computation. As illustrated with the example in Figure 3.2, the initialization phase starts from the entry

Table 3.1: Computation of RCS^{IN} for the motivational example

	Basic Block	RCS^{IN}
Initialization	b_0	$\{[\top, \top, \top, \top]\}$
	b_1	$\{[m_0, \top, \top, \top]\}$
	b_2	$\{[m_0, \top, \top, \top]\}$
	b_3	$\{[m_0, m_1, m_2, m_3], [m_0, \top, m_2, m_3]\}$
Fixed-Point Computation Results	b_0	$\{[\top, \top, \top, \top]\}$
	b_1	$\{[m_0, \top, \top, \top], [m_0, m_1, m_2, m_3]\}$
	b_2	$\{[m_0, \top, \top, \top]\}$
	b_3	$\{[m_0, m_1, m_2, m_3], [m_0, \top, m_2, m_3]\}$

Table 3.2: Computation of RCS^{OUT} for the motivational example

	Basic Block	RCS^{OUT}
Initialization	b_0	$\{[m_0, \top, \top, \top]\}$
	b_1	$\{[m_0, m_1, m_2, m_3]\}$
	b_2	$\{[m_0, \top, m_2, m_3]\}$
	b_3	$\{[m_4, m_1, m_2, m_3], [m_4, \top, m_2, m_3]\}$
Fixed-Point Computation Result	b_0	$\{[m_0, \top, \top, \top]\}$
	b_1	$\{[m_0, m_1, m_2, m_3]\}$
	b_2	$\{[m_0, \top, m_2, m_3]\}$
	b_3	$\{[m_4, m_1, m_2, m_3], [m_4, \top, m_2, m_3]\}$

basic block b_0 with $RCS^{IN}_{b_0} = \{[\top, \top, \top, \top]\}$. The element is $\top$ since the analysis is independent of the program executed before b_0. According to (3.2), $RCS^{OUT}_{b_0}$ is calculated to be $\{[m_0, \top, \top, \top]\}$. Since b_0 is the only immediate predessor of b_2, $RCS^{IN}_{b_2}$ is equal to $RCS^{OUT}_{b_0}$ based on (3.4). Due to the self loop, b_1 has both itself and b_0 as immediate predessors. However, since $RCS^{OUT}_{b_1}$ has not been initialized yet, $RCS^{IN}_{b_1}$ is equal to $RCS^{OUT}_{b_0}$. In the same manner, $RCS^{OUT}_{b_1}$, $RCS^{OUT}_{b_2}$, $RCS^{IN}_{b_3}$ and $RCS^{OUT}_{b_3}$ are computed, following the program flow as shown both in Figure 3.2, Table 3.1 and Table 3.2. The initialization phase is completed once all basic blocks have been visited.

The next phase is fixed-point computation. RCS^{IN} and RCS^{OUT} of all basic blocks are computed iteratively with (3.4) and (3.2). This phase is terminated once the fixed point is reached, i.e., RCS^{IN} and RCS^{OUT} of all basic blocks remain unchanged. The program RCS is defined as the RCS^{OUT} of the exit basic block, i.e., $RCS = RCS^{OUT}_{b_3}$. Results are reported in Table 3.1 and Table 3.2.

The LCS computation can be done in a similar fashion. gen_{b_k} is defined as the cache state describing the first executed memory block in every cache line for the basic block b_k. For the sake of explanation, taking the same assumption when defining gen_{b_k} for RCS computation that b_0 in Figure 3.2 executes m_0 and then m_4, instead of only m_0, the first executed memory block in c_0 is m_0. Therefore, gen_{b_0} is $[m_0, \perp, \perp, \perp]$. LCS^{IN} of a basic block b_k is the LCS after b_k is

Table 3.3: Computation of LCS^{IN} for the motivational example

	Basic Block	LCS^{IN}
Initialization	b_3	$\{[\top, \top, \top, \top]\}$
	b_2	$\{[m_4, \top, \top, \top]\}$
	b_1	$\{[m_4, \top, \top, \top]\}$
	b_0	$\{[m_4, m_1, m_2, m_3], [m_4, \top, m_2, m_3]\}$
Fixed-Point Computation Results	b_3	$\{[\top, \top, \top, \top]\}$
	b_2	$\{[m_4, \top, \top, \top]\}$
	b_1	$\{[m_4, \top, \top, \top], [m_4, m_1, m_2, m_3]\}$
	b_0	$\{[m_4, m_1, m_2, m_3], [m_4, \top, m_2, m_3]\}$

Table 3.4: Computation of LCS^{OUT} for the motivational example

	Basic Block	LCS^{OUT}
Initialization	b_3	$\{[m_4, \top, \top, \top]\}$
	b_2	$\{[m_4, \top, m_2, m_3]\}$
	b_1	$\{[m_4, m_1, m_2, m_3]\}$
	b_0	$\{[m_0, m_1, m_2, m_3], [m_0, \top, m_2, m_3]\}$
Fixed-Point Computation Results	b_3	$\{[m_4, \top, \top, \top]\}$
	b_2	$\{[m_4, \top, m_2, m_3]\}$
	b_1	$\{[m_4, m_1, m_2, m_3]\}$
	b_0	$\{[m_0, m_1, m_2, m_3], [m_0, \top, m_2, m_3]\}$

executed and can be derived from

$$LCS_{b_k}^{IN} = \bigcup_{s \in successor(b_k)} LCS_s^{OUT}, \tag{3.5}$$

where $successor(b_k)$ is the set of all immediate successors of b_k. LCS^{OUT} of b_k is the LCS before b_k is executed with

$$LCS_{b_k}^{OUT} = \{\mathcal{T}(b_k, cs) | cs \in LCS_{b_k}^{IN}\}. \tag{3.6}$$

LCS computation also comprises two phases of initialization and fixed-point computation. The only difference from the RCS computation is that the initialization phase starts from the exit basic block and ends in the entry basic block. Detailed results for the motivational example are reported in Table 3.3 and Table 3.4. The program LCS is defined to be the LCS^{OUT} of the entry basic block, i.e., $LCS = LCS_{b_0}^{OUT}$. It is noted that since the presented cache analysis technique is based on the fixed-point computation over the program CFG, it inherently handles loop structures.

3.2.3 Guaranteed WCET Reduction

Conceptually, the program RCS is the set of all possible cache states after the program finishes execution by any execution path, and the program LCS is the set of all cache states, where

each cache state contains memory blocks that may be firstly referenced after the program starts execution, for any execution path to follow. Both the RCS and LCS could contain multiple cache states. Each pair with one cache state cs from the program RCS and one cache state cs' from the program LCS represents one possible execution path between the two consecutive executions. For any cache line c_i in a pair, if $cs[i]$ is equal to $cs'[i]$ and they are not equal to $\top$, then there is certainly a hit and thus WCET reduction. Whether there is a hit for a particular cache line can be determined by the function $\mathcal{H}$ defined as follows,
$\forall cs \in CS$, $cs' \in CS$ and $c_i \in CL$, where $i \in \{0, 1, \ldots, N_c - 1\}$,

$$\mathcal{H}(cs, cs', c_i) = \begin{cases} 1 : if\ cs[i] = cs'[i] \wedge cs[i] \neq \bot; \\ 0 : otherwise. \end{cases} \tag{3.7}$$

The number of hits can be counted with the function $\mathcal{HT}$ defined as,
$\forall cs \in CS$ and $cs' \in CS$,

$$\mathcal{HT}(cs, cs') = \sum_{i=0}^{N_c-1} \mathcal{H}(cs, cs', c_i). \tag{3.8}$$

The guaranteed number of hits among all possibilities is calculated as

$$\mathcal{G}(RCS, LCS) = \min_{cs \in RCS, cs' \in LCS} (\mathcal{HT}(cs, cs')). \tag{3.9}$$

Given that the main memory access time and the cache access time are respectively t_m and t_c, the guaranteed WCET reduction is computed as

$$\bar{E}^g = \mathcal{G}(RCS, LCS) \times (t_m - t_c) \approx \mathcal{G}(RCS, LCS) \times t_m, \tag{3.10}$$

where the approximation can be taken if $t_c \ll t_m$.

For the motivational example, there are two cache states in the RCS ($RCS_{b_3}^{OUT}$) and two cache states in the LCS ($LCS_{b_0}^{OUT}$). In total, there are four pairs and the number of hits are calculated to be 3, 2, 2 and 2 with (3.8). For instance, $\mathcal{HT}([m_4, m_1, m_2, m_3], [m_0, m_1, m_2, m_3]) = 3$. Therefore, the guaranteed number of hits is 2 according to (3.9), no matter which path the program takes. From (3.10), the guaranteed WCET reduction is $2 \times (t_m - t_c)$, or approximately $2 \times t_m$, when $t_c \ll t_m$. It is noted that this result is obtained from the small example used for illustration. More WCET reduction is expected for larger realistic programs.

Using the memory analysis technique presented in this section to compute the guaranteed WCET reduction due to consecutive executions of one program, together with standard WCET analysis approaches to compute the default WCET of one program without considering the execution order, the effective WCETs of a memory-aware sampling order can be calculated and used to determine the control timing parameters, such as sampling periods and sensor-to-actuator delays. This will be shown in the next section.

3.3 Control Timing Parameters

As discussed in Chapter 1, the overall control loop in an embedded implementation platform performs three operations: measure, compute and actuate. The general timing model of a control loop is illustrated in Figure 3.3. The compute operation executes the control program, which

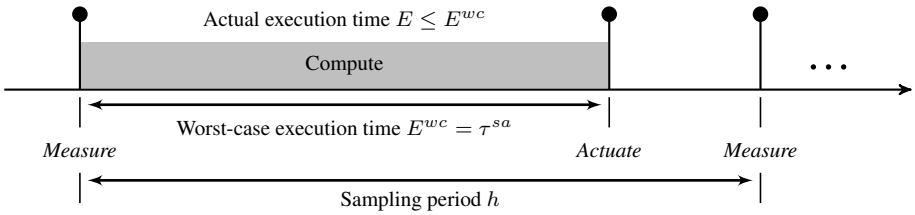

Figure 3.3: The general timing model of a control loop

takes E time units. The sampling period is denoted by h. The time interval between the measure and the corresponding actuate operations in the same sampling period is the sensor-to-actuator delay τ^{sa}, which is equal to the WCET of the control program E^{wc}.

Two exapmle sampling orders are used to show the derivation of control timing parameters from the WCET results. As illustrated in Figure 3.4, S1 is the conventional memory-oblivious scheme and summarized as follows,

S1: $\mathcal{C}_1(1) \rightarrow \mathcal{C}_2(1) \rightarrow \mathcal{C}_3(1) \rightarrow \mathcal{C}_1(2) \rightarrow \mathcal{C}_2(2) \rightarrow \mathcal{C}_3(2) \rightarrow \mathcal{C}_1(3) \rightarrow \mathcal{C}_2(3) \rightarrow \mathcal{C}_3(3) \rightarrow \cdots$.

There is no cache reuse in S1 between consecutive executions, considering that different control applications typically have different instructions to execute. In other words, when $\mathcal{C}_i(j)$ starts execution, all instructions of $\mathcal{C}_i$ need to be brought into the cache from the flash memory. Therefore,

$$E_i^{wc}(1) = E_i^{wc}(2) = \cdots = E_i^{wc}, \tag{3.11}$$

where $E_i^{wc}(j)$ is the WCET of the jth execution for $\mathcal{C}_i$. The WCET of the application $\mathcal{C}_i$ is denoted by E_i^{wc}, since all executions of the same application have equal WCET. This can be computed with standard WCET analysis techniques, as discussed before. Clearly, all control applications run with a uniform sampling period of

$$h = \sum_{i=1,2,3} E_i^{wc}. \tag{3.12}$$

Moreover, for the sensor-to-actuator delay,

$$\tau_i^{sa} = E_i^{wc}. \tag{3.13}$$

As illustrated in Figure 3.5, S2 is an example memory-aware sampling order and summarized as,

S2: $\mathcal{C}_1(1) \rightarrow \mathcal{C}_1(2) \rightarrow \mathcal{C}_1(3) \rightarrow \mathcal{C}_2(1) \rightarrow \mathcal{C}_2(2) \rightarrow \mathcal{C}_2(3) \rightarrow \mathcal{C}_3(1) \rightarrow \mathcal{C}_3(2) \rightarrow \mathcal{C}_3(3) \rightarrow \cdots$.

The effective WCET taking into account the cache reuse between consecutive executions is denoted as $\bar{E}_i^{wc}(j)$. From the above discussion,
$\forall i \in \{1, 2, 3\}$,

$$\bar{E}_i^{wc}(1) = E_i^{wc}, \tag{3.14}$$

since there is no cache reuse from the previous program for the first execution of every application $\mathcal{C}_i(1)$. $\bar{E}_i^{wc}(2)$ and $\bar{E}_i^{wc}(3)$ are shorter than $\bar{E}_i^{wc}(1)$ due to cache reuse. The amounts of

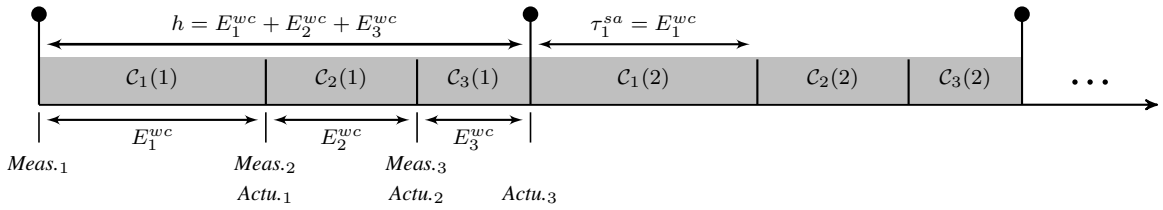

Figure 3.4: In S1, there is no cache reuse between consecutive executions. The WCET of all executions for the same application E_i^{wc} remains constant. The sampling period of every control application h is uniform under this scheme. The sensor-to-actuator delay τ_i^{sa} is equal to E_i^{wc}.

cache reuse are the same for $\mathcal{C}_i(2)$ and $\mathcal{C}_i(3)$ in the worst case. Denoting the guaranteed WCET reduction as $\bar{E}_i^g$,
$\forall i \in \{1, 2, 3\}$,

$$\bar{E}_i^{wc}(2) = \bar{E}_i^{wc}(3) = \bar{E}_i^{wc}(1) - \bar{E}_i^g. \tag{3.15}$$

From these varying WCETs, the sampling periods of all three applications can be calculated. Taking $\mathcal{C}_1$ as an example, there are three sampling periods $h_1(1)$, $h_1(2)$ and $h_1(3)$, which repeat themselves periodically,

$$h_1(1) = \bar{E}_1^{wc}(1), \quad h_1(2) = \bar{E}_1^{wc}(2), \quad h_1(3) = \bar{E}_1^{wc}(3) + \Delta, \tag{3.16}$$

where Δ is computed as

$$\Delta = \sum_{i=2,3} \sum_{j=1,2,3} \bar{E}_i^{wc}(j). \tag{3.17}$$

Similar derivation can be done for $\mathcal{C}_2$ and $\mathcal{C}_3$. The average sampling period of an application h_{avg} is

$$h_{\text{avg}} = \frac{\sum_{i=1,2,3} \sum_{j=1,2,3} \bar{E}_i^{wc}(j)}{3} < h. \tag{3.18}$$

According to (3.14) and (3.15),

$$h_{\text{avg}} < \frac{\sum_{i=1,2,3} 3 \times E_i^{wc}}{3}. \tag{3.19}$$

From (3.12),

$$h_{\text{avg}} < h. \tag{3.20}$$

Moreover, the corresponding sensor-to-actuator delay $\tau_i^{sa}(j)$ also varies with cache reuse as
$\forall i \in \{1, 2, 3\}$,

$$\tau_i^{sa}(1) = h_i(1) = \bar{E}_i^{wc}(1), \quad \tau_i^{sa}(2) = h_i(2) = \bar{E}_i^{wc}(2), \quad \tau_i^{sa}(3) = \bar{E}_i^{wc}(3). \tag{3.21}$$

As all control timing parameters have been derived, it can be seen that the sampling period $h_i(j)$ of a control application is non-uniform for the memory-aware scheme. The average sampling period of S2 is shorter than the uniform sampling period of S1 as shown in (3.18), due to

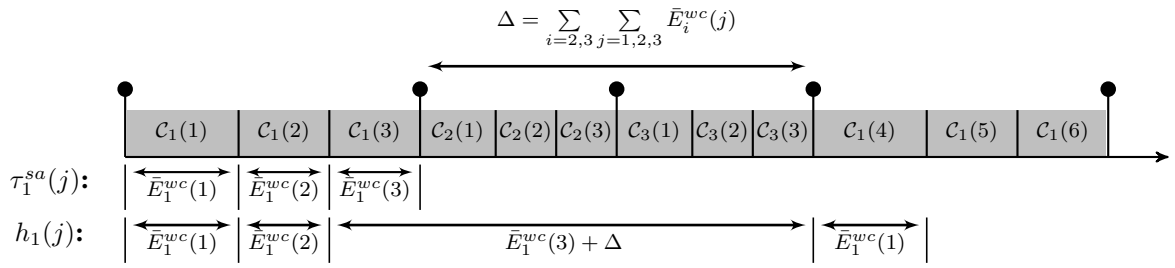

Figure 3.5: In S2, the WCETs of the same control application vary, due to cache reuse. The sampling period for a control application is non-uniform.

the WCET reduction resulting from cache reuse. The sensor-to-actuator delay $\tau_i^{sa}(j)$ varies as shown in (3.21). The next task is developing a controller design method to exploit shortened non-uniform sampling periods and achieve better control performance, which is reported in the next section. The controller design method for the memory-oblivious uniform sampling scheme S1 is also presented.

3.4 Controller Design

3.4.1 Controller Design with Uniform Sampling

As can be derived from (3.12) and (3.13), for an application $\mathcal{C}_i$ under the conventional memory-oblivious sampling scheme S1, the constant sampling period h is larger than the constant sensor-to-actuator delay τ_i^{sa}. Therefore, the discrete-time system in (2.9) becomes

$$x[k+1] = A_d x[k] + B_1(\tau_i^{sa})u[k-1] + B_0(\tau_i^{sa})u[k], \tag{3.22}$$

where

$$B_0(\tau_i^{sa}) = \int_0^{h-\tau_i^{sa}} e^{At}dt \cdot B, \;\; B_1(\tau_i^{sa}) = \int_{h-\tau_i^{sa}}^{h} e^{At}dt \cdot B. \tag{3.23}$$

In (3.22), it is assumed that $u[-1] = 0$ for $k = 0$. Clearly, the system dynamics depends on both $u[k]$ and $u[k-1]$. Thus, a new system state is defined as $z[k] = \left[\begin{array}{cc} x[k] & u[k-1] \end{array} \right]^T$ and the transformed system becomes

$$\begin{aligned} z[k+1] &= A_{S1}z[k] + B_{S1}u[k], \\ y[k] &= C_{S1}z[k], \end{aligned} \tag{3.24}$$

where

$$A_{S1} = \left[\begin{array}{cc} A_d & B_1(\tau_i^{sa}) \\ 0 & 0 \end{array} \right], \;\; B_{S1} = \left[\begin{array}{cc} B_0(\tau_i^{sa}) & \mathbf{I} \end{array} \right]^T, \;\; C_{S1} = \left[\begin{array}{cc} C & 0 \end{array} \right]. \tag{3.25}$$

A_{S1} is a square matrix.

Next, the following input signal is applied,

$$u[k] = K_{S1}z[k] + F_{S1}r. \tag{3.26}$$

The closed-loop system is then

$$z[k+1] = (A_{S1} + B_{S1}K_{S1})z[k] + B_{S1}F_{S1}r. \tag{3.27}$$

In order to find the poles resulting in the best control performance with the pole-placement technique, a constrained optimization problem is formulated. Decision variables are the controllable closed-loop system poles, i.e., the controllable eigenvalues of $(A_{S1} + B_{S1}K_{S1})$. The optimization objective is the control performance. One constraint is that the closed-loop system is stable, i.e., the decision variables have absolute values of less than unity. Another constraint is the input saturation. As discussed in Chapter 2, constraints on the overshoot and steady-state accuracy are also considered. The method to solve this challenging non-convex optimization problem is elaborated later in this section. After the poles are placed, the feedback gain K_{S1} is then calculated according to (2.22). The feedforward gain F_{S1} is computed by (2.23). As long as (A_{S1}, B_{S1}) is stabilizable, i.e., uncontrollable poles have absolute values of less than unity, the above design is feasible. This design method is adapted from sampled-data systems literature and is suitable for the systems with known sensor-to-actuator delay shorter than the sampling period [BK08].

3.4.2 Controller Design with Non-Uniform Sampling

For the convenient discussion, the notation of sampling orders is first defined. The number of consecutive executions for any application $\mathcal{C}_i$, where $i \in \{1, 2, \ldots, n\}$, in one period is denoted by m_i. Then, the periodically repeating sampling order is denoted by $(m_1, m_2, \ldots, m_n)$. For the ease of understanding, a simple sampling order $(2, 2, 2)$ of three control applications is considered in illustrating two controller design techniques under non-uniform sampling — one aims for performance optimality and the other targets better scalability on the number of consecutive executions m_i. Generalization to any periodic sampling order is straightforward and briefly discussed.

Holistic design: The controller for an application $\mathcal{C}_i$ with l system states can be designed in a holistic way that results in the optimal control performance. As shown in Figure 3.6, there are two sampling periods $h_i(1)$ and $h_i(2)$, which are repeated periodically. The two switching systems are

$$\begin{aligned} x[k+1] &= A_1 x[k] + B_1 u[k], \\ x[k] &= A_2 x[k-1] + B_2^1 u[k-1] + B_2^2 u[k], \end{aligned} \tag{3.28}$$

where B_2^1 and B_2^2 depend on the second sampling period $h_i(2)$ and the sensor-to-actuator delay of the second execution $\tau_i^{sa}(2)$. The system output is $y[k] = Cx[k]$. It should be noted that $x[k]$ is influenced by both $u[k-1]$ and $u[k]$, since $\tau_i^{sa}(2)$ is smaller than $h_i(2)$, i.e., $u[k]$ is applied before the sensing of $x[k]$.

Introducing a new state $z[k] = \begin{bmatrix} x[k] & u[k] \end{bmatrix}^T$, the system becomes

$$\begin{aligned} z[k+1] &= A_1^{\text{hol}} z[k] + B_1^{\text{hol}} u[k+1], \\ z[k] &= A_2^{\text{hol}} z[k-1] + B_2^{\text{hol}} u[k], \end{aligned} \tag{3.29}$$

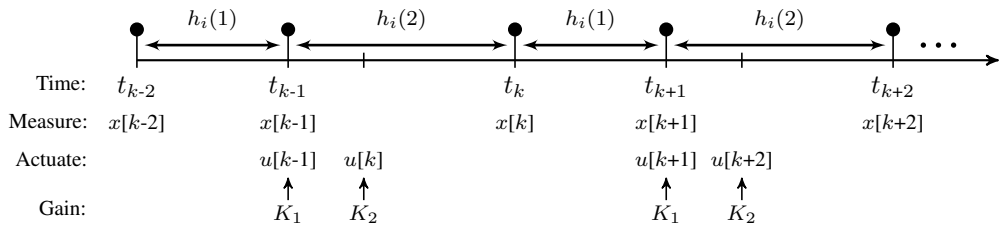

Figure 3.6: Periodically switched sampling periods for $\mathcal{C}_i$ in the schedule $(2, 2, 2)$

where

$$A_1^{\text{hol}} = \begin{bmatrix} A_1 & B_1 \\ 0 & 0 \end{bmatrix}, \quad B_1^{\text{hol}} = \begin{bmatrix} 0 \\ \mathbf{I} \end{bmatrix}, \quad A_2^{\text{hol}} = \begin{bmatrix} A_2 & B_2^1 \\ 0 & 0 \end{bmatrix}, \quad B_2^{\text{hol}} = \begin{bmatrix} B_2^2 \\ \mathbf{I} \end{bmatrix}. \tag{3.30}$$

Both A_1^{hol} and A_2^{hol} are square matrices. The system output is $y[k] = \begin{bmatrix} C & 0 \end{bmatrix} z[k]$.

There are two control inputs that need to be designed within one period,

$$\begin{aligned} u[k+1] &= K_1 z[k] + F_1 r, \\ u[k] &= K_2 x[k-1] + F_2 r. \end{aligned} \tag{3.31}$$

The closed-loop system dynamics are then,

$$\begin{aligned} z[k+1] &= (A_1^{\text{hol}} + B_1^{\text{hol}} K_1) z[k] + B_1^{\text{hol}} F_1 r, \\ z[k] &= (A_2^{\text{hol}} + B_2^{\text{hol}} K_2) z[k-1] + B_2^{\text{hol}} F_2 r. \end{aligned} \tag{3.32}$$

The number of poles to place, i.e., the number of eigenvalues in $(A_1^{\text{hol}} + B_1^{\text{hol}} K_1)$ and $(A_2^{\text{hol}} + B_2^{\text{hol}} K_2)$, is $2l + 2$. This is a constrained non-convex non-linear optimization problem. The objective to maximize is the control performance of $\mathcal{C}_i$. Decision variables are the poles and thus the number of dimensions in the decision space is $2l + 2$. The technique to solve this pole-placement optimization problem is presented later in this section. Once poles are placed, K_1 and K_2 are computed with (2.22). Then, F_1 and F_2 can be calculated as per (2.23). With this holistic method, both feedback gains are designed together taking all the information into account. The maximum control performance can be obtained if the optimization technique returns the optimal poles. However, when $\mathcal{C}_i$ is consecutively executed m_i times in a sampling order, the number of dimensions in the decision space becomes $m_i(l + 1)$, which compromises the scalability.

Scalable design: Since there are two control inputs resulting from two consecutive executions within one period for $\mathcal{C}_i$ in the sampling order $(2, 2, 2)$, the controller can be designed for each of them separately to achieve better scalability. Considering the first execution and ignoring the second[1], without $u[k]$ and $u[k + 2]$, as illustrated in Figure 3.7, the system dynamics becomes,

$$x[k] = A_0 x[k-2] + B_0^1 u[k-3] + B_0^2 u[k-1]. \tag{3.33}$$

[1]When the number of consecutive executions is larger than 2, similarly only the first execution is considered and all the rest are ignored. Later, the following executions are brought back one by one as described in the next paragraph.

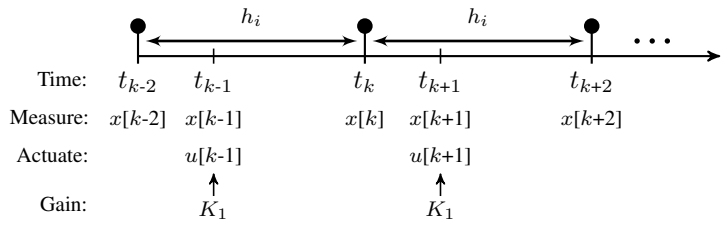

Figure 3.7: Timing of the scalable controller design technique when only the first execution is considered in a period

The system output remains unchanged.

Introducing a new state $v[k] = \begin{bmatrix} x[k] & u[k-1] \end{bmatrix}^T$,

$$v[k] = A^{\text{sca}} v[k-2] + B^{\text{sca}} u[k-1], \tag{3.34}$$

where

$$A^{\text{sca}} = \begin{bmatrix} A_0 & B_0^1 \\ 0 & 0 \end{bmatrix}, \quad B^{\text{sca}} = \begin{bmatrix} B_0^2 \\ \mathbf{I} \end{bmatrix}. \tag{3.35}$$

The system output is $y[k] = \begin{bmatrix} C & 0 \end{bmatrix} v[k]$.

The control input is,

$$u[k-1] = K_1 v[k-2] + F_1 r, \tag{3.36}$$

which closes the loop and thus the system dynamics becomes,

$$v[k] = (A^{\text{sca}} + B^{\text{sca}} K_1) v[k-2] + B^{\text{sca}} F_1 r. \tag{3.37}$$

The number of poles, i.e., the number of eigenvalues in $(A^{\text{sca}} + B^{\text{sca}} K_1)$ to place is $l+1$. The optimization problem to solve is similar to the one in the holistic method, yet the number of dimensions in the decision space is $l+1$. Once poles are placed, K_1 and F_1 can be calculated as per (2.22) and (2.23).

Given K_1 and F_1 for $u[k-1]$ ($u[k+1]$) as calculated, the second execution can be brought back, taking $u[k]$ ($u[k+2]$) into account. Considering the accurate system dynamics is (3.32), only the eigenvalues of $(A_2^{\text{hol}} + B_2^{\text{hol}} K_2)$ need to be determined. Therefore, the number of poles to place is $l+1$, which is also the number of dimensions in the decision space. After the pole-placement, K_2 and F_2 can be computed accordingly. When $\mathcal{C}_i$ is consecutively executed more than twice in a sampling order, i.e., $m_i > 2$, following the above process, the third and following executions can be brought back one by one. Such an optimization problem with $l+1$-dimensional decision space needs to be solved m_i times to obtain the maximum control performance. Therefore, the advantage of this controller design method is the scalability on m_i. The disadvantage is that the poles might not be optimal in maximizing the control performance, since control inputs are not designed together but considered separately. Specifically, when designing K_1 and F_1 for $u[k-1]$, there is not all the information (the second execution and $u[k]$ ignored). Then, K_2 and F_2 for $u[k]$ are designed based on K_1 and F_1. It is important to note that the stability of the overall closed-loop dynamics is still guaranteed while these gains are not necessarily optimal in the control performance.

3.4.3 Pole-Placement with Hybrid PSO

As discussed in Chapter 2, PSO is a popular method to solve non-convex optimization problems and can be used in locating the poles that maximize the control performance. One major issue with the conventional PSO is its tendency for fast and premature convergence before the global optimum is found, since its search is highly directional [SM09]. This problem gets more severe as the number of dimensions in the decision space grows larger. A technique is proposed in [YS05], in which the search is led by the opposite direction of the worst point that has been visited. The advantage is that the decision space can be better explored, while the search is still conducted in the direction where good points (not necessarily the best points) are expected to be located. However, the drawback is that particles may have difficulty to converge.

In this thesis, a hybrid PSO method, which is able to well explore the decision space and converge to local optima, is proposed. The entire search process is divided into two phases. Phase I includes the first k iterations. The velocity update equation is as follows,

$$\begin{aligned} V_{\text{new}} = \alpha_0 V_{\text{current}} &+ \alpha_1 \text{rand}(0,1)(P_{\text{lbest}} - P_{\text{current}}) + \alpha_2 \text{rand}(0,1)(P_{\text{gbest}} - P_{\text{current}}) \\ &+ \alpha_3 \text{rand}(0,1)(P_{\text{lworst}} - P_{\text{current}}) + \alpha_4 \text{rand}(0,1)(P_{\text{gworst}} - P_{\text{current}}), \end{aligned} \tag{3.38}$$

where P_{lworst} is the worst point of this particle and P_{gworst} is the worst point of all particles. The parameters α_3 and α_4 are larger than α_1 and α_2. Phase I tries to explore the decision space while avoiding premature convergence. Phase II includes the rest iterations. The velocity update equation is

$$\begin{aligned} V_{\text{new}} = \alpha_0 V_{\text{current}} &+ \alpha_3 \text{rand}(0,1)(P_{\text{lbest}} - P_{\text{current}}) + \alpha_4 \text{rand}(0,1)(P_{\text{gbest}} - P_{\text{current}}) \\ &+ \alpha_1 \text{rand}(0,1)(P_{\text{lworst}} - P_{\text{current}}) + \alpha_2 \text{rand}(0,1)(P_{\text{gworst}} - P_{\text{current}}). \end{aligned} \tag{3.39}$$

This phase helps all particles to converge. The number k can be determined empirically.

The numerical example in (2.45) with a different set of initial particles is used to show the difference between the proposed hybrid PSO technique and the conventional one. Results are shown in Figure 3.8 and 3.9. In the conventional PSO, five particles are randomly initialized at p_1, p_2, p_3, p_4 and p_5. After 13 iterations, all five particles converge to points around the local optimum p_l. The path showing how the global best point evolves iteratively is drawn, with certain points that are too close to others omitted for better illustration. It can be seen that the search is highly directed towards the global best point in each iteration. The global optimum is not found and particles quickly converge to the local optimum before exploring the decision space sufficiently enough.

In the proposed hybrid PSO method, the same initial points are used. After 22 iterations, all five particles converge to points around the global optimum p_g. Both paths showing how the global best and worst points evolve iteratively are drawn, with certain points too close to others omitted. The search is driven by both the best and worst points. The decision space is better explored and the global optimum is found. It is noted that the proposed method takes more iterations for all particles to converge. This numerical example is meant to show the difference between the hybrid PSO method and the conventional one. In the experiments later shown in this chapter, equal computational efforts are made for fair comparison. That is, there will be more particles for the conventional method.

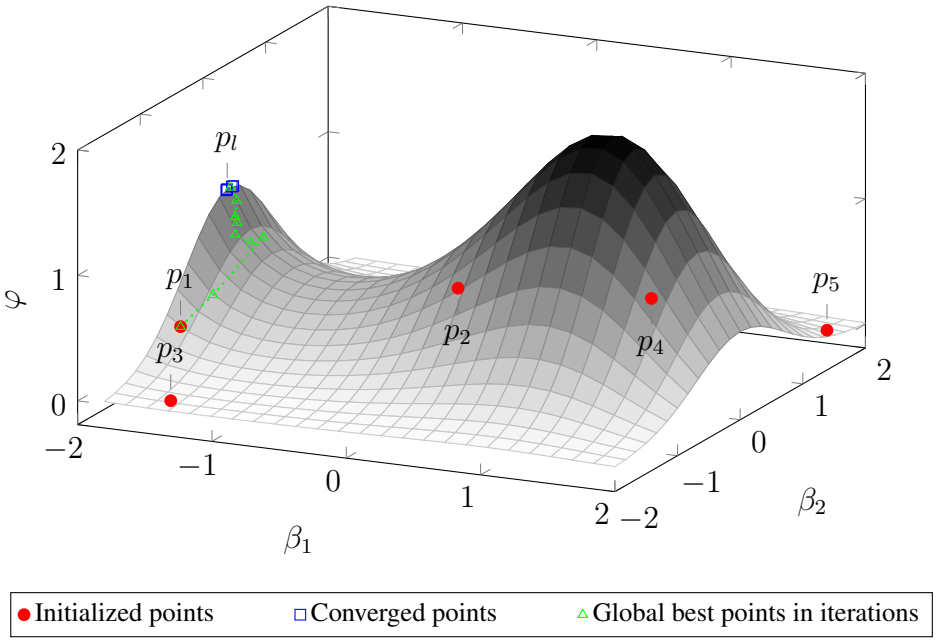

Figure 3.8: With the conventional particle swarm optimization method, five particles are randomly initilized and converge to the local optimum p_l.

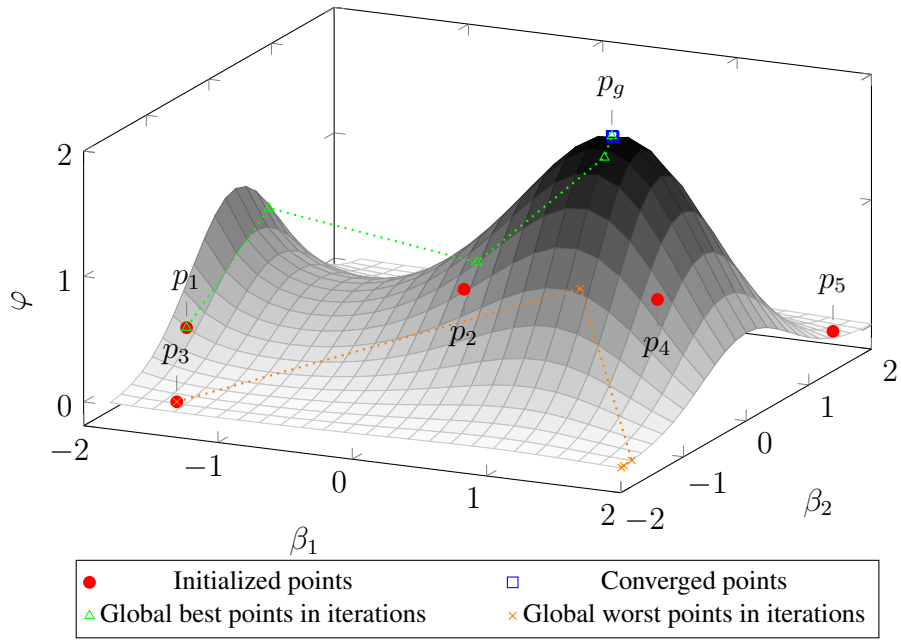

Figure 3.9: With the novel hybrid particle swarm optimization method, five particles are randomly initialized and converge to the global optimum p_g.

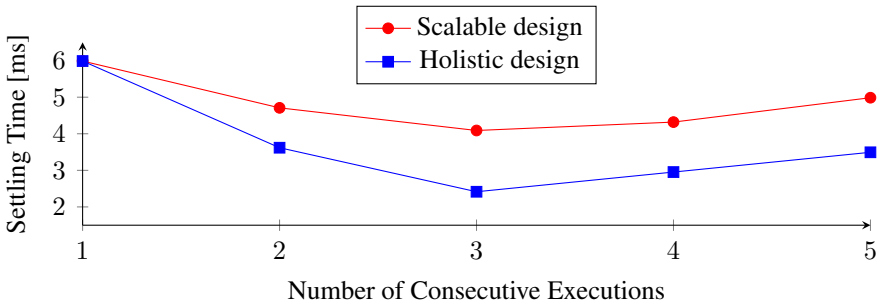

Figure 3.10: Comparison of two design methods in the achieved optimal control performance

3.4.4 Comparison of Controller Design Methods

With the optimization technique presented above, the two controller design methods described in Section 3.4.2 can be evaluated by maximizing the control performance of a double integrator presented in (2.24). The initial state is $\begin{bmatrix} 0 & 0 \end{bmatrix}$ and the reference value r is 0.3. The comparison between these two controller design methods in the achieved optimal control performance as the number of consecutive executions increases from 1 to 5 is shown in Figure 3.10. The sampling order (m_1, m_2, m_3) with three applications is considered and it is assumed that $m_1 = m_2 = m_3$. The WCET of the first execution is assumed to be 2μs and the WCET of the following consecutive executions is assumed to be 1μs. The control performance is quantified by the settling time. Although the PSO technique does not guarantee the optimal solution, these results of settling time in Figure 3.10 fulfill the following two criteria:

- all particles have converged;
- when the number of particles is increased by 100 times, the result remains unchanged.

The holistic method is able to achieve better control performance.

The comparison between these two controller design methods in computational efforts, quantified by the computation time taken in the optimization process maximizing the control performance, is shown in Figure 3.11. The experiments are done with an Intel i5 processor operating at 2.6GHz and with 4GB RAM. It can be seen that the scalable method indeed exhibits excellent scalability on the number of consecutive executions. The computational time exceeds 20 minutes for the holistic method, when considering the sampling order $(5, 5, 5)$. The number of consecutive executions m_i is typically not large, since there is often a constraint on the maximum allowed idle time of an application, in order to prevent the system from being driven to an unsafe state by perturbations. Since the control performance maximization for a given sampling order is an offline task, the computational efforts required by the holistic method are often acceptable. However, when m_i is large, evaluating the control performance of one sampling order could take long time with the holistic method. When the number of applications n is large, there can be too many sampling orders requiring control performance evaluations, since the number of sampling orders is exponential on n. In both situations, the scalable method can be applied to reduce the computation time to a reasonable level.

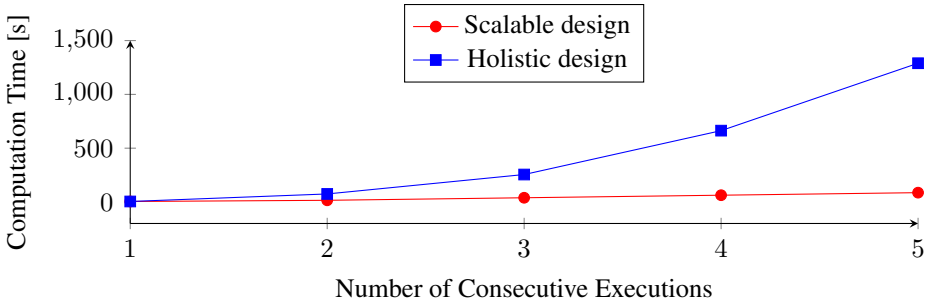

Figure 3.11: Comparison of two design methods in computional efforts

3.5 Optimal Sampling Order Computation

After presenting the method to evaluate the control performance of one application, the overall control performance of one sampling order can be defined as a weighted sum of application control performances. When the settling time t_s is used as the metric to quantify the control performance J, for an application $\mathcal{C}_i$,

$$J_i = 1 - \frac{t_{s,i}}{t^0_{s,i}}, \tag{3.40}$$

where $t^0_{s,i}$ is the requirement of the settling time and serves as the normalization reference. Since both $t_{s,i}$ and $t^0_{s,i}$ are positive numbers, J_i is less than 1. The overall control performance is then,

$$J_{\text{all}} = \sum_{i=1}^{n} w_i J_i = \sum_{i=1}^{n} w_i (1 - \frac{t_{s,i}}{t^0_{s,i}}), \tag{3.41}$$

where w_i is the weight of the application $\mathcal{C}_i$ and,

$$\sum_{i=1}^{n} w_i = 1. \tag{3.42}$$

The next stage is to find the optimal one among all sampling orders. Following the assumption that there are n applications, the formulation is

$$\begin{gathered} \max_{\{m_1,m_2,\ldots,m_n\}} J_{\text{all}} \\ \text{subject to} \\ \{m_i \in \mathbb{N}^+ | i \in \{1,2,\ldots,n\}\}. \end{gathered} \tag{3.43}$$

The objective to optimize is the overall control performance.

Besides all the constraints on individual applications that have been discussed, there is a constraint on the timing of the sampling order that for an application $\mathcal{C}_i$, the maximum allowed idle time is t_i^{idle} to prevent the application from being driven to an unsafe state by perturbations. The idle time is defined as the interval between two consecutive sampling instants and thus

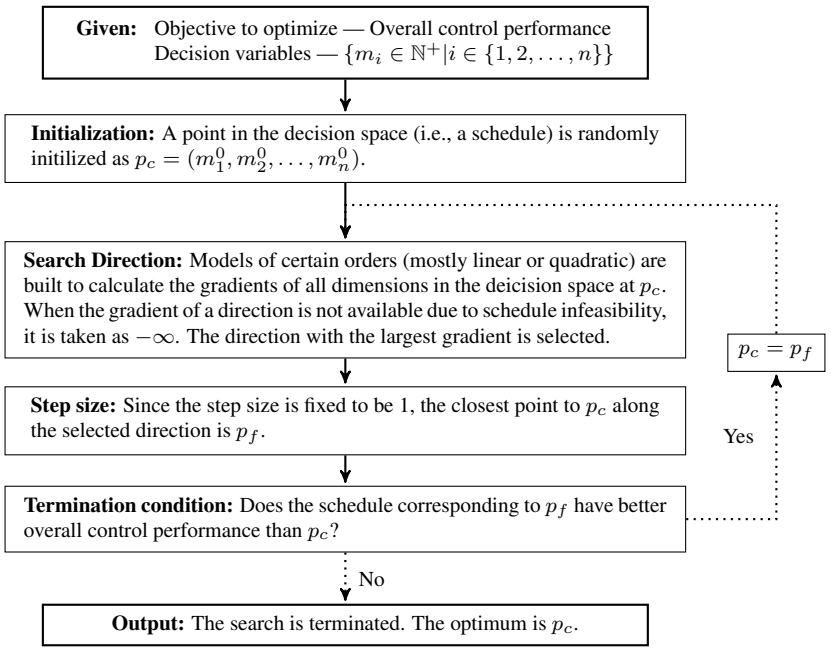

Figure 3.12: The gradient-based search algorithm for discrete decision space

equal to the sampling period. Denoting $h_i^{\max}$ to be the longest sampling period of $\mathcal{C}_i$ in a periodic sampling order,
$\forall i \in \{1, 2, \ldots, n\}$,

$$h_i^{\max} \leq t_i^{\text{idle}}. \tag{3.44}$$

The number of dimensions in the decision space is equal to n. This is a nonlinear discrete optimization problem and the simplest method to solve it is exhaustive search. Denoting the number of values that m_i can take with $|m_i|$, the total number of schedules to evaluate is $\prod_{i=1}^{n} |m_i|$. Considering that the overall control performance evaluation is computationally intensive, a more efficient method than brute force is required. One popular class of techniques to solve nonlinear discrete optimization problems is based on the continuation approach. The discrete problem is transformed to a continuous problem, which can be solved by the modified Newton method [Ng02]. The requirement is that the objective function can be continuously evaluated. However, the overall control performance can only be evaluated discretely. For example, a sampling order like $(2.5, 3.5, 4.8)$ does not have any practical sense and there is no overall control performance associated with it. Therefore, the continuation approach is not suitable in this context.

A gradient-based search algorithm is proposed to find the optimal sampling order efficiently. It is similar to SQP, yet applied in the discrete decision space. As discussed in Chapter 2, in SQP, an approximate quadratic model is built to derive the search direction. Building the n-dimensional quadratic model is not suitable in this context for two reasons. First, it is very likely that the direction computed from the quadratic model is not available in the discrete decision space. Second, in order to build the quadratic model in the n-dimensional decision space, the

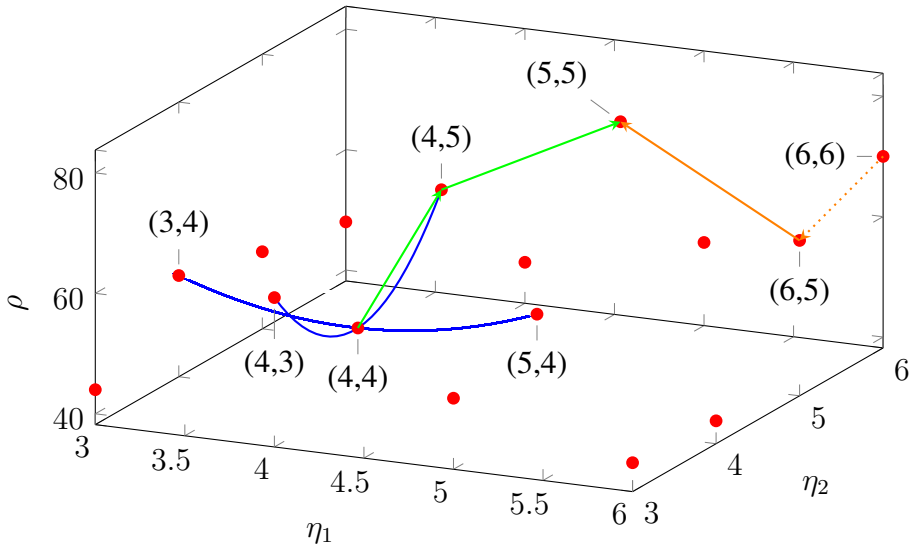

Figure 3.13: A motivational example with two decision variables to illustrate the gradient-based search algorithm. The global optimum is at $(\eta_1 = 5, \eta_2 = 5)$.

overall control performance needs to be evaluated $2n + \binom{n}{2}$ times, which is non-polynomial on the number of applications n.

In the proposed approach, for every dimension of the decision space, one model is built, which can be of any order depending on the characteristics of applications. The gradient of each dimension can then be calculated. The direction with the largest positive gradient is selected. For instance, besides the current point whose overall control performance is already known, one quadratic model requires evaluating the overall control performance of two points (one on each side) of the current point. Since there are n models, the search direction determination process takes $2n$ evaluations of overall control performance, at the maximum. If some overall control performance values have already been computed, this number can be smaller than $2n$. The step size is fixed to be 1. The termination condition is that the closest point along the selected direction does not give any improvement of the objective value. The current point is then the optimum. This gradient-based search algorithm for discrete decision space is summarized in Figure 3.12 and illustrated with a motivational example in Figure 3.13.

In this example, there are two integer decision variables $3 \leq \eta_1 \leq 6$ and $3 \leq \eta_2 \leq 6$. The objective to maximize is ρ. The starting point is randomly initialized at $(\eta_1 = 4, \eta_2 = 4)$. The quadratic model is used to calculate the search direction. Since the number of dimensions in the decision space is 2, two quadratic models are built. One is from the points $(3, 4)$, $(4, 4)$ and $(5, 4)$ along the dimension of η_1. The other is from the points $(4, 3)$, $(4, 4)$ and $(4, 5)$ along the dimension of η_2. Comparing the two gradients at $(4, 4)$ of these two quadratic models, the one along the positive direction of η_2 is larger and thus the move is in this direction by 1 to $(4, 5)$. With the same process, the next point after $(4, 5)$ is computed to be $(5, 5)$, from which no improvement can be achieved in any direction. Therefore, the search is terminated and the optimum point is $(5, 5)$, which matches the result obtained from the exhaustive search.

Table 3.5: Experimental configuration for memory analysis

Clock Frequency	Number of Cache Lines	Cache Line Size	Hit/Miss Penalty
20 MHz	128	16 Byte	$1/100$ cycle

The gradient-based search algorithm does not guarantee finding the global optimum. There are two methods to improve the search performance with a trade-off in computational efforts. First, parallel searches can be conducted. As the number of initialized points is increased, the chance that the global optimum can be found rises. Second, the termination condition can be modified to avoid being trapped at the local optimum. For instance, when the randomly initialized point is at $(6, 6)$ in Figure 3.13, the negative η_2 direction has the largest gradient. However, the next point along this direction $(6, 5)$ does not have a better objective value than $(6, 6)$. It is noted that the linear model is used to calculate the gradient, since $(6, 6)$ is on the boundary and neighbor points only exist on one side. The search will terminate at the local optimum $(6, 6)$ and fails to find the global optimum $(5, 5)$.

If the improvement is not necessary to continue the search process and slight deterioration is tolerated, it is likely that the search will not be trapped at a local optimum. For example, the tolerance threshold γ is set to be 10% and the rule that the search does not revisit any point is enforced. From $(6, 6)$, the next point is $(6, 5)$, since the objective value of $(6, 5)$ is larger than 0.9 multiplying the objective value of $(6, 6)$. The next point after $(6, 5)$ is $(5, 5)$. It can be seen that the search is not trapped at the local optimum $(6, 6)$ anymore and finds the global optimum $(5, 5)$. Although a larger γ increases the chance of getting out of local optima, there is also a higher risk that the search will not terminate at the global optimum. Therefore, the tolerance threshold is set according to the objective space.

3.6 Experimental Results

In order to evaluate the proposed memory-aware automotive control systems design, three control applications are considered: $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$. $\mathcal{C}_1$ is position control of a servo motor that can be used, e.g., in a steer-by-wire system [Yih05]. Details can be found in Section B. $\mathcal{C}_2$ is speed control of a DC motor that can be used in electric vehicle cruise control [CPG+14]. Details are presented in Section C. $\mathcal{C}_3$ is control of the EWB system developed by Siemens as a brake-by-wire solution [FRBW+07], which has been discussed in Chapter 2 and elaborated in Section A. All three control applications run on the same processor.

As shown in Table 3.5, the processor clock frequency is 20MHz. The cache is set to have 128 cache lines and each cache line is 16 bytes. When there is a cache hit, it takes 1 clock cycle to fetch the instruction and when there is a cache miss, it takes 100 clock cycles. WCET results with and without cache reuse for all three applications are shown in Table 3.6. It is noted that the WCET reduction is guaranteed in all cases.

The two example sampling orders S1 and S2 in Section 3.3 are taken to illustrate the derivation of control timing parameters. Based on standard WCET analysis techniques applied to the control programs, the WCETs of all three applications without any cache reuse in S1 are

computed to be

$$E_1^{wc} = 907.55\mu s, \quad E_2^{wc} = 645.25\mu s, \quad E_3^{wc} = 749.15\mu s,$$

Which are summarized in Table 3.6. Thus, the uniform sampling period as in (3.12) is

$$h = \sum_{i=1,2,3} E_i^{wc} = 2301.95\mu s.$$

The constant sensor-to-actuator delay τ_i^{sa} is given by (3.13),
$\forall i \in \{1, 2, 3\}$,

$$\tau_i^{sa} = E_i^{wc}.$$

In S2, according to (3.14),
$\forall i \in \{1, 2, 3\}$,

$$\bar{E}_i^{wc}(1) = E_i^{wc}.$$

Based on the memory analysis approach presented in Section 3.2, the guaranteed numbers of cache hits for the three applications are

$$\mathcal{G}_1 = 92, \quad \mathcal{G}_2 = 95, \quad \mathcal{G}_3 = 104.$$

From the memory configuration in Table 3.5, the cache access time is 1 processor clock cycle, i.e., $t_c = 0.05\mu s$, and the main memory access time is 100 processor clock cycles, i.e., $t_m = 5\mu s$. Therefore, according to (3.10), the guaranteed WCET reduction due to cache reuse for $\mathcal{C}_1$ can be calculated as

$$\bar{E}_1^g = \mathcal{G}_1 \times (t_m - t_c) = 92 \times (5 - 0.05)\mu s = 455.4\mu s.$$

Similarly,

$$\bar{E}_2^g = 470.25\mu s, \quad \bar{E}_3^g = 514.8\mu s.$$

According to (3.15), the reduced effective WCETs are
$\forall j \in \{2, 3\}$,

$$\bar{E}_1^{wc}(j) = 452.15\mu s, \bar{E}_2^{wc}(j) = 175\mu s, \bar{E}_3^{wc}(j) = 234.35\mu s.$$

These are summarized in Table 3.6. Then the sampling periods can be obtained. Taking $\mathcal{C}_1$ as an example, using (3.16) and (3.17),

$$h_1(1) = 907.55\mu s, \quad h_1(2) = 452.15\mu s, \quad h_1(3) = 2665.25\mu s.$$

As in (3.18), the average sampling period of all three applications in S2 is calculated to be $1341.65\mu s$ with 42% of reduction compared to the uniform sampling period in S1. The corresponding sensor-to-actuator delay $\tau_i^{sa}(j)$ is obtained with (3.21).

The weights, settling deadlines and maximum allowed idle times of all three applications are presented in Table 3.7. The holistic design method and the hybrid PSO are used to evaluate the overall control performance of one sampling order. The process of finding the optimal schedule with the gradient-based search algorithm is illustrated in Figure 3.14. Two sampling orders are initialized as $(4, 2, 2)$ ($\mathcal{C}_1$ is consecutively executed 4 times, followed by $\mathcal{C}_2$ twice and $\mathcal{C}_3$ twice)

Table 3.6: WCET results with and without cache reuse for all three control applications. The WCET reduction is guaranteed in all cases.

Application	WCET w/o Cache Reuse	WCET Reduction	WCET w/ Cache Reuse
$\mathcal{C}_1$	$907.55\ \mu$s	$455.40\ \mu$s	$452.15\ \mu$s
$\mathcal{C}_2$	$645.25\ \mu$s	$470.25\ \mu$s	$175.00\ \mu$s
$\mathcal{C}_3$	$749.15\ \mu$s	$514.80\ \mu$s	$234.35\ \mu$s

Table 3.7: Application parameters

Application	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$
Weight (w_i)	0.4	0.4	0.2
Settling deadline [ms] ($t^0_{s,i}$)	45	20	17.5
Maximum allowed idle time [ms] (t_i^{idle})	3.4	3.9	3.5

and $(1, 2, 1)$, with overall control performances 0.166 and 0.107, respectively. From $(4, 2, 2)$, there are three dimensions and four directions to go, since $(5, 2, 2)$ and $(4, 2, 1)$ are infeasible sampling orders. Infeasiblity results from violation of constraints that have been discussed. Taking the sampling order $(5, 2, 2)$ as an example, the longest sampling period $h_3^{\max}$ is 3.77 ms, based on the WCET results shown in Table 3.6. The constraint on the timing of the sampling order in (3.44) is violated, since $h_3^{\max} > t_3^{\text{idle}}$ as shown in Table 3.7. The positive direction along m_3 has the largest gradient and thus the move is from $(4, 2, 2)$ to $(4, 2, 3)$. Following the same process, the move is made from $(4, 2, 3)$ to $(3, 2, 3)$. At $(3, 2, 3)$, no improvement can be achieved in any direction and thus the search is terminated. The sampling order $(3, 2, 3)$ is indeed the global optimum, verified by the exhaustive search. The overall control performance is 0.195.

The other search path starting from $(1, 2, 1)$ moves to $(1, 2, 2)$ and then $(2, 2, 2)$. At $(2, 2, 2)$, the positive direction along m_3 gives the largest gradient. However, the overall control performance of $(2, 2, 3)$ is smaller than that of $(2, 2, 2)$. Thus, the search will be terminated, trapped at the local optimum without reaching the global optimum $(3, 2, 3)$. As discussed before, the tolerance threshold γ can be used to address this issue. In this case , γ is set to be 6%. Since the overall control performance of $(2, 2, 3)$ (0.137) is better than that of $(2, 2, 2)$ multiplied with 0.94 ($0.145 \times 0.94 = 0.136$), the search continues and moves to $(2, 2, 3)$. Following that, the global optimum $(3, 2, 3)$ is reached. It is noted that the positive direction along m_1 gives the largest gradient at $(3, 2, 3)$. Since the overall control performance of $(4, 2, 3)$ (0.181) is less than that of $(3, 2, 3)$ multiplied with 0.94 ($0.195 \times 0.94 = 0.183$), the search is terminated at the global optimum.

Comparison of the system output response between the conventional memory-oblivious sampling order S1 and the optimal memory-aware sampling order $(3, 2, 3)$ for all the three control applications is presented in Figure 3.15, 3.16 and 3.17. Comparison of control performances quantified by the settling times between the conventional memory-oblivious sampling order S1 and the optimal memory-aware sampling order $(3, 2, 3)$ for all the three control applications is reported in Table 3.8. It can be seen that with the memory-aware automotive control systems

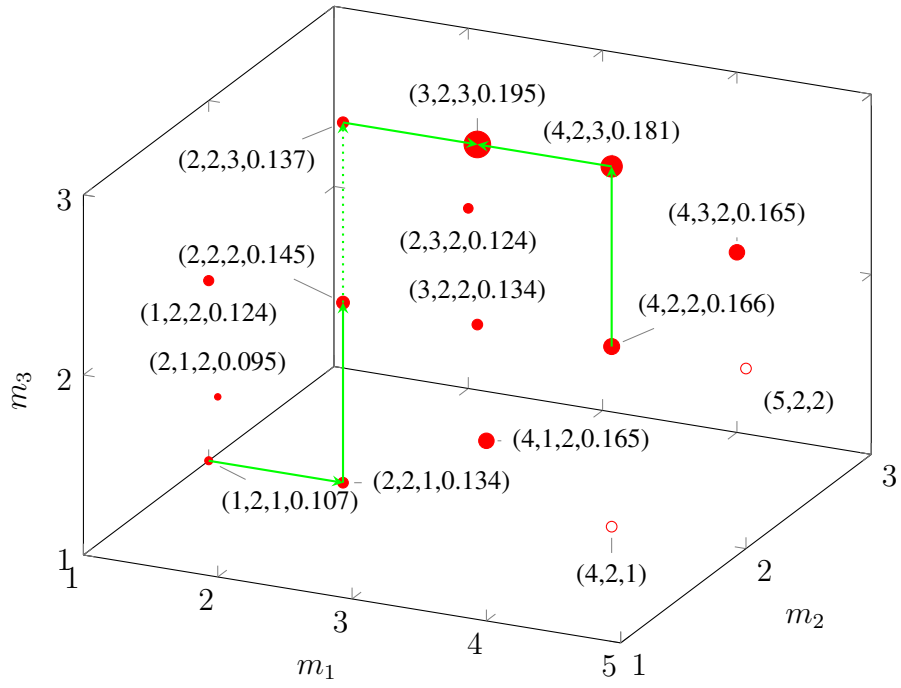

Figure 3.14: The gradient-based search to find the optimal sampling order of the automotive control systems case study with two initial sampling orders. Each point is denoted with $(m_1, m_2, m_3, J_{\text{all}})$. The point size indicates the overall control performance of the sampling order, yet not to scale. Empty circles represent infeasible sampling orders. Only selected feasible sampling orders are shown.

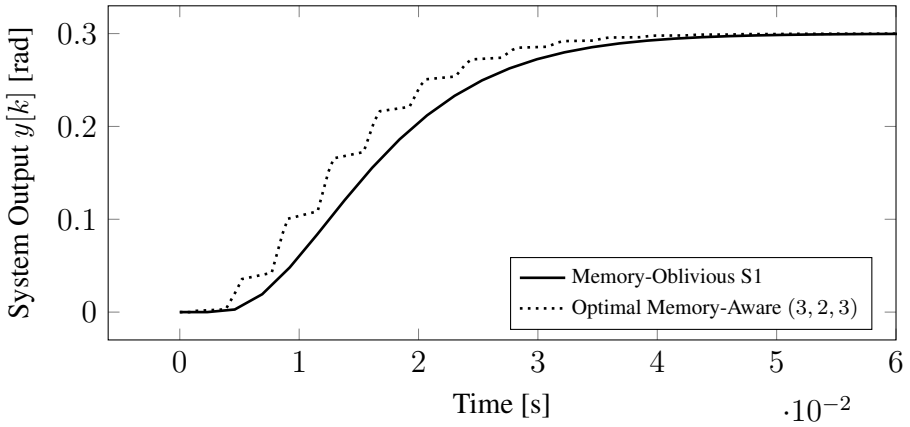

Figure 3.15: Control system output of the memory-oblivious and optimal memory-aware sampling orders for the control application $\mathcal{C}_1$

design, the control performances can be improved by $10 - 20\%$, which is quite significant for the cost-sensitive domains.

In this case study, there are 76 sampling orders to evaluate when the brute force is deployed to find the optimal sampling order, including 74 feasible ones and 2 infeasible ones violat-

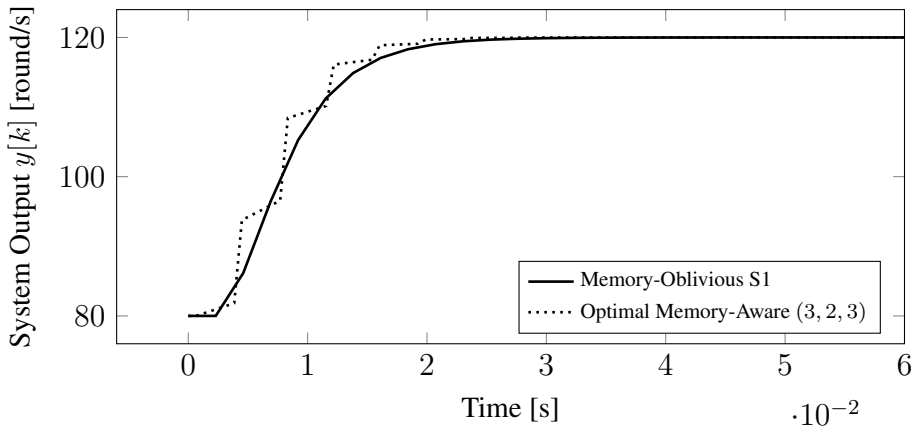

Figure 3.16: Control system output of the memory-oblivious and optimal memory-aware sampling orders for the control application $\mathcal{C}_2$

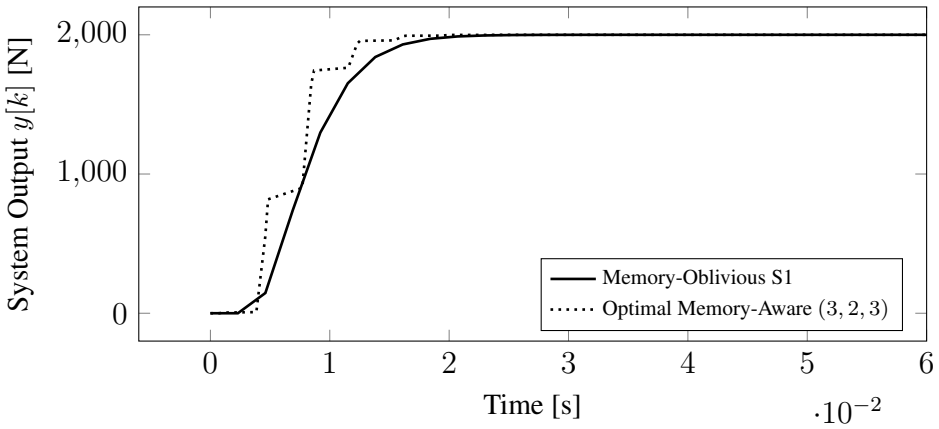

Figure 3.17: Control system output of the memory-oblivious and optimal memory-aware sampling orders for the control application $\mathcal{C}_3$

ing the requirement on the settling time, which are known only after the control performance evaluation. Using the computer with an Intel i5 processor operating at 2.6GHz and with 4GB RAM, evaluating the application control performance takes from seconds (when $m_i = 1$) to hours (when $m_i > 5$). Completing the exhaustive search of all 76 sampling orders costs days. With the proposed gradient-based search algorithm, the search starting from $(4, 2, 2)$ evaluates 9 sampling orders, which is 11.8% of the 76 schedules using brute force. The search starting from $(1, 2, 1)$ evaluates 18 sampling orders. Furthermore, the searches starting from all feasible sampling orders are able to reach the global optimum, with the tolerance threshold of 6%.

The control performances are also evaluated with the conventional PSO method for comparison to the hybrid PSO technique. For all feasible sampling orders, the overall control performance obtained by the hybrid PSO is at least not worse than that from the conventional PSO. Comparison for the optimal sampling order $(3, 2, 3)$ is shown in Table 3.9. It is noted that the hybrid technique takes more iterations to converge with the same number of particles. For fair

Table 3.8: Control performance comparison for all three applications between the conventional memory-oblivious sampling order S1 and the optimal memory-aware sampling order $(3, 2, 3)$

Application	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$
Settling time for S1 [ms]	43.2	17.7	17.3
Settling time for $(3, 2, 3)$ [ms]	37.7	15.3	14.4
Control performance improvement of $(3, 2, 3)$ compared to S1 [%]	13	14	17

Table 3.9: Control performance comparison for the optimal sampling order $(3, 2, 3)$ between the conventional and hybrid PSO

PSO	$\mathcal{C}_1$ settling time	J_1	$\mathcal{C}_2$ settling time	J_2	$\mathcal{C}_3$ settling time	J_3	J_{all}
Conventional	38.74 ms	0.139	15.29 ms	0.236	16.11 ms	0.079	0.166
Hybrid	37.68 ms	0.163	15.28 ms	0.236	14.44 ms	0.175	0.195

comparison, the computational efforts are equalized by increasing the number of particles in the conventional PSO method. It can be seen that the hybrid PSO technique achieves better control performances of all three applications for the optimal sampling order $(3, 2, 3)$.

Using the overall control performances obtained from the conventional PSO method, searches from certain starting sampling orders might not be able to reach the optimal one $(3, 2, 3)$. One example is shown in Figure 3.18. The initial sampling order is $(4, 2, 2)$, from which there are three dimensions and four directions to go, since $(5, 2, 2)$ and $(4, 2, 1)$ are infeasible sampling orders. The positive direction along m_3 has the largest gradient. However, the overall control performance of $(4, 2, 3)$ (0.152) is worse than that of $(4, 2, 2)$ multiplied with 0.94 ($0.163 \times 0.94 = 0.153$). Therefore, the search will not continue and $(4, 2, 2)$ becomes the optimal sampling order.

3.7 Remarks

This chapter proposes techniques of memory-aware automotive control systems design and considers multiple applications running on a single processor with shared cache. While exploiting existing program analysis techniques in conjunction with cache modeling, the analysis focuses on estimating the guaranteed WCET reduction due to consecutive executions of the same program, which is required in computing the non-uniform sampling order for a feedback controller. Estimating such WCET reduction has not been studied before, mostly since until now there has been no useful context for studying it. In addition, controller design aiming for optimal control performance with non-uniform sampling relies on the proposed technique that exploits the shortened WCET in the memory-aware sampling order.

The overall control performance is maximized by an optimal choice of sampling order taking into account the effects of cache reuse, in an integrated framework of sampling order computation and controller optimization. As has been shown, this leads to a two-stage optimization problem. First, the optimal controller poles are located maximizing the overall control performance of a given sampling order. Second, the optimal sampling order among all feasible ones is found.

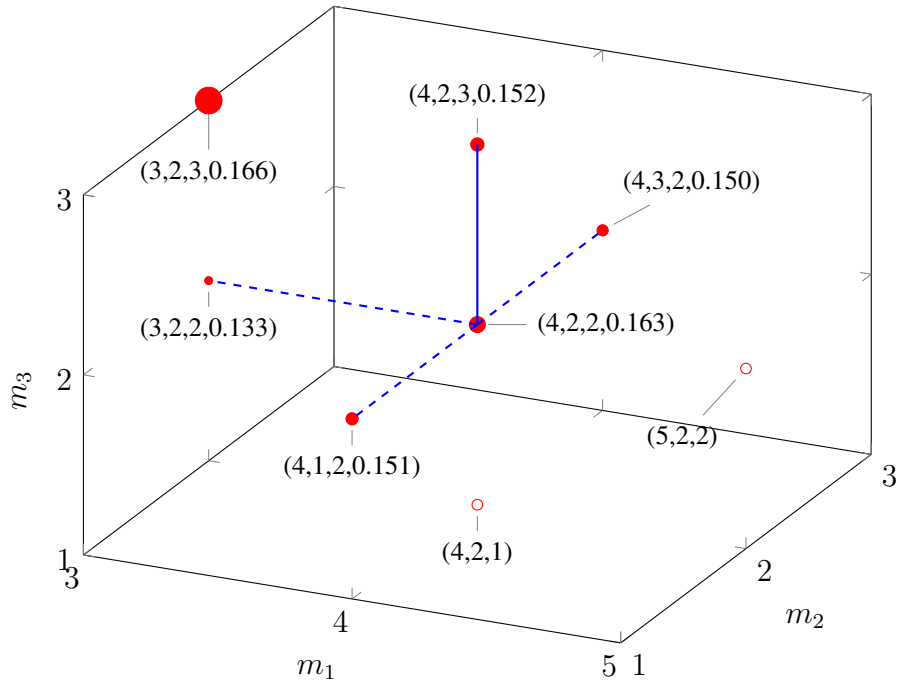

Figure 3.18: When the conventional PSO is used, the search starting from $(4, 2, 2)$ is not able to reach the global optimum $(3, 2, 3)$.

In the existing memory-conscious algorithms design (e.g., in real-time tasks), the programs are treated as black boxes and their functionality is not considered. This chapter explicitly considers the properties of control algorithms. In particular, the optimal choice of the control algorithms parameters, such as the gain values, are dependent on their sampling periods and sensor-to-actuator delays, which in turn are determined by the WCETs they experience. In the case of other algorithms, their parameter values are not updated on the basis of the WCETs. Hence, while memory-conscious algorithms design for other domains stops at WCET minimization, whether and how simultaneously modifying the controller parameters in response to the reduced WCET leads to improved control performance, is an open question. This chapter makes the first efforts to answer this question.

While the direct-mapped cache (i.e., 1-way set-associative cache) is assumed, the presented technique can be adapted to handle set-associative cache. For example, considering fully associative cache in the example of Figure 3.2, when computing $RCS_{b_3}^{OUT}$ from $RCS_{b_3}^{IN}$, the memory block m_4 can be loaded to any cache line, which gives $RCS_{b_3}^{OUT}$ five more cache states, i.e., $[m_0, m_4, m_2, m_3]$, $[m_0, m_1, m_4, m_3]$, $[m_0, \top, m_4, m_3]$, $[m_0, m_1, m_2, m_4]$, and $[m_0, \top, m_2, m_4]$. From this, it can be seen that the number of cache states in RCS and LCS is larger for set-associative cache, which means that the guaranteed WCET reduction could be smaller. Details can be found in [KFM11].

The number of possible data inputs is very large in the state-feedback control under consideration. In addition, a control program typically contains a large number of branches, which are dynamically evaluated according to the data values. Small value changes of some data input may trigger different execution paths and overall execution times. Hence, estimating the WCET

of the control software on the basis of exploring all possible inputs or a portion of them, instead of following the static memory analysis technique as is shown in this chapter, is not practically feasible. Actual execution time larger than the WCET cannot be tolerated by safety-critical control applications.

The contribution of this chapter is the synergistically integrated framework of memory-aware automotive control systems design, which further provides new insights and design options in line with the CPS-oriented design philosophy, where the goal is to study control theory and embedded systems with the same footing. Accounting for on-chip cache behavior motivates new techniques for control algorithms design, that are otherwise not used. These techniques open up a number of possibilities for co-design and co-optimization of control algorithms, code placement, and memory-aware control tasks scheduling, that will be pursued in the future.

4

Computation-Aware Automotive Control Systems Design

ECU consolidation is a recent trend in the automotive industry triggering the need to maximize the number of functions and applications that share the limited computation resources on an ECU. In this chapter, a common automotive setup that a feedback control application is integrated on an ECU with other applications is studied. ECUs usually run TT OS due to the safety-critical nature of the automotive domain. OSEK/VDX OS, which is a class of widely used TT OS in automobiles providing services like task management, resource management, and error treatment, is considered[1]. The goal of computation-aware automotive control systems design is to minimize the processor utilization of an application, while still satisfying the performance requirement, system constraints and restriction on sampling periods from the OS. This enables an ECU to accommodate more applications.

The main technical challenge is designing a controller for the non-uniform schedule, which switches between available sampling periods offered by the OS. Building upon existing techniques in control systems with non-uniform sampling and optimal control theory, a novel controller design approach for the linear state-feedback control is proposed to optimize the settling time, and explicitly respects the hard physical constraint of the input signal. A new PSO technique with adaptive parameterization is used in the controller pole-placement. The proposed idea is evaluated on the real-life Electro-Mechanical Braking (EMB) system. The multirate schedule is also tested in MPC with the EWB system. The number of applications that can be implemented on an ECU is doubled, which is significant resource saving for the cost-sensitive automotive domain.

[1]OSEK is a joint project in the German automotive industry founded in 1993 with initial partners of BMW, Bosch, DaimlerChrysler, Opel, Siemens and Volkswagen. It was later joined by the French car manufacturers PSA and Renault introducing their VDX approach. The goal is to define an industry standard for an open-ended architecture for distributed control units in vehicles.

The organization of this chapter is as follows. Section 4.1 discusses the related work on computation-aware embedded control systems design, together with some recent contributions on optimal control and non-uniform sampling. Section 4.2 describes the OSEK/VDX OS. The multirate linear state-feedback controller design is presented in Section 4.3, with the optimal pole-placement technique using adaptively parameterized PSO. An alternative controller design is explained for better scalability. Besides the linear controller, non-uniform non-linear MPC is illustrated. Section 4.4 reports the experimental results. The linear state-feedback controller is evaluated on the EMB system and the MPC is evaluated on the EWB system. Section 4.5 makes concluding remarks of this chapter.

4.1 Related Work

A number of works have been reported on computation-aware embedded control systems design [CMV+06, MFFR02, GFB11, SEP+10]. A resource management strategy adjusting the task periods at runtime considering the response over a finite time horizon of the plants is proposed in [CMV+06] to maximize the control performance. Flexible timing constraints are proposed in [MFFR02] to achieve faster reaction by adaptively choosing the controller sampling rate and completion time upon transient perturbations, while wastage of resources is avoided when the system is in equilibrium. In [GFB11], a novel controller design technique based on a hierarchy of controllers is proposed, so that when the allocated execution time is short, a low-level computationally light controller is activated to achieve basic control performance and when the execution time is long, a high-level computationally intensive controller is used aiming for better control performance. The trade-off between control performance and CPU usage is explored in [SEP+10] by dynamic scheduling of multiple self-triggered control tasks executed on one processor. None of the efforts above address the restriction from the OS.

There have been continuous efforts in research of optimal control and non-uniform sampling. Notable ones include [BB14] and [CBMC11]. The optimal sampling problem is tackled in [BB14], where the sampling instants and control inputs are selected to minimize a given function of the system state and control input. In particular, a necessary condition for the optimality of a set of sampling instants is derived and a quantization-based sampling strategy is proposed to be computationally tractable. The optimality of this method is proved in first-order systems. However, the sampling periods in this work can be arbitrarily chosen and no constraints (e.g., from the OS as is studied in this chapter) are taken into account. In addition, the proposed method is only applied in the LQR problem, which is relatively easy to analyze due to its quadratic cost function. In this chapter, the settling time, which is an important performance metric for real-time control applications, is also considered.

Online optimal sampling period assignment is investigated in [CBMC11] in order to maximize the control performance. A feedback scheduler is developed to periodically assigns new sampling periods based on the current plant states. It is shown that most computation can be done offline and stored in a look-up table. Again, only the quadratic cost function is considered and the selection of sampling periods is not restricted. Besides, since the switching is not fixed, yet occurs depending on the plant states in real-time, stability cannot be guaranteed. Building upon these previous works discussed above, the method for the linear state-feedback control that

Table 4.1: An example OSEK/VDX OS time table of applications release

Time	Release
$0ms$	Applications with periods of $2ms$/$5ms$/$10ms$
$2ms$	Applications with the period of $2ms$
$4ms$	Applications with the period of $2ms$
$5ms$	Applications with the period of $5ms$
$6ms$	Applications with the period of $2ms$
$8ms$	Applications with the period of $2ms$
$10ms$	**Repeat actions at** $0ms$

is presented in this chapter formulates an optimal pole-placement problem for a non-uniform schedule known in the design time, where the input signal saturation is explicitly respected, the settling time is minimized and the stability is ensured.

4.2 OSEK/VDX Operating System

OSEK/VDX OS is a class of real-time OS widely used in the automotive industry. In general, an OSEK/VDX OS supports preemptive fixed-priority scheduling. That is, priorities are assigned to applications and at any point in time, the task with the highest priority among all active ones is executed. Tasks can be triggered by events (e.g., interrupts, alarms, etc.) or by time. In the TT scheme, each application gets released and is allowed to access the processor periodically. There are various periods of release times and each application is assigned one. Different applications may have different periods. Every time an application is released, its program gets the chance to be executed.

A time table containing all the periodic release times within the alleged hyperperiod (i.e., the minimum common multiple of all periods) needs to be configured. An example with a set of three periods $2ms$, $5ms$ and $10ms$ is illustrated in Table 4.1. The hyperperiod is equal to $10ms$ and the time table repeats itself every $10ms$ by reseting the timer. Independent of the triggering mode (i.e., be it ET or TT), the assigned priority will still determine the execution order of applications. In the TT scheme, a higher priority is typically assigned to the application released with a shorter period, since this generally results in a more efficient use of the processor.

An example with two applications C_1 and C_2 sharing one ECU is illustrated in Figure 4.1. C_1 has a period of $2ms$ and C_2 has a period of $5ms$. The execution time of C_1 is assumed to be $0.7ms$ and the execution time of C_2 is assumed to be $2ms$. C_1 has a higher priority than C_2. Within a hyperperiod of $10ms$, C_1 is released at $0ms$, $2ms$, $4ms$, $6ms$, $8ms$ and $10ms$. C_2 is released at $0ms$, $5ms$ and $10ms$. It can be seen that C_2 is executed only when C_1 does not require to access the ECU. For instance, at $0ms$, both C_1 and C_2 are released and require access to the ECU. C_1 is permitted to be executed while C_2 has to wait. At $0.7ms$, C_1 completes its execution and C_2 gets the access to the ECU.

It is assumed that the set of available periods restricted by OSEK/VDX is ϕ. As discussed in Chapter 1, control applications have to be sampled with one period or a combination of multiple periods from ϕ. In the latter case, switching between two sampling periods can only occur at the

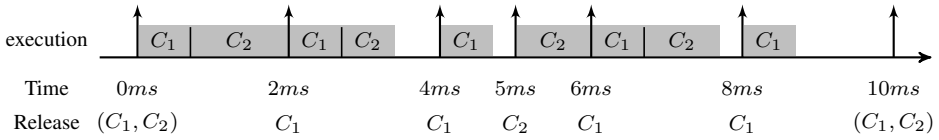

Figure 4.1: Release and execution time of two applications sharing one ECU. C_1 with a sampling period of $2ms$ has a higher priority than C_2 with a sampling period of $5ms$. Execution times of C_1 and C_2 are $0.7ms$ and $2ms$, respectively.

common multiplier of them, as has been illustrated in Figure 1.4. Often, the optimal sampling period for a control application does not belong to the set ϕ. The simple and straightforward method used in practice is to select the largest sampling period in ϕ that is smaller than the optimal one. Taking the example in Table 4.1, assuming that the optimal sampling period is $7.5ms$, then $5ms$ is chosen as the sampling period to be used. This results in a higher processor utilization, which is another important design aspect.

Denoting E_i^{wc} to be the worst-case execution time (WCET) of a control application C_i, if the uniform sampling period is h, the processor utilization for C_i is

$$L_i = \frac{E_i^{wc}}{h}. \tag{4.1}$$

The upper bound on the utilization of any processor is 1. Considering a single processor p,

$$\sum_{\{i|C_i \text{ runs on } p\}} L_i \leq 1. \tag{4.2}$$

Clearly, increasing the sampling period of a control application decreases its processor utilization, and thus potentially enables more applications to be integrated on the ECU.

4.3 Multirate Controller Design

The design problem for a control application C_i in this chapter is to reduce the processor utilization L_i, while satisfying the settling time requirement $t_{s,i}^0$, the system stability and the input saturation constraint $U_{max,i}$. Towards this, a multirate controller switching between multiple sampling periods in ϕ is proposed. In this chapter, the sensor-to-actuator delay is neglected. When the sensor-to-actuator delay has to be considered, the controller design technique presented in Chapter 3 can be used.

The cyclic sequence of sampling periods for a control application defines a schedule S,

$$S = \{T_1, T_2, T_3, \ldots, T_N\}, \tag{4.3}$$

where $\forall j \in \{1, 2, \ldots, N\},\ T_j \in \phi$. It implies the sequence of sampling periods as,

$$T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_N \rightarrow T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_N \rightarrow repeat$$

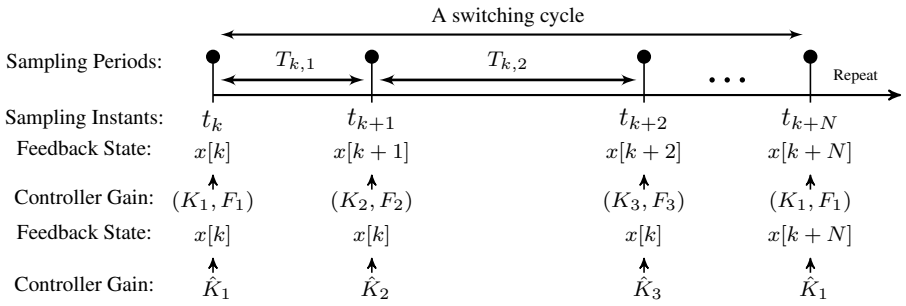

Figure 4.2: Cyclically switched linear systems

Following the assumption in (4.1) that the WCET of C_i is E_i^{wc}, the processor utilization for C_i over S is

$$L_i = \frac{NE_i^{wc}}{\sum_{j=1}^{N} T_j}. \tag{4.4}$$

4.3.1 Linear State-feedback Controller

Dictated by the schedule S, N systems switch cyclically in a deterministic fashion. When $t_{k+1} - t_k = T_{k,j}$, the dynamics is

$$x[k+1] = A_d(T_{k,j})x[k] + B_d(T_{k,j})K_j x[k] + B_d(T_{k,j})F_j r, \tag{4.5}$$

as referred to (2.19) The key in the linear state-feedback controller design is to compute the feedback gain K_j for each system with pole-placement, based on which, the static feedforward gain F_j can be derived with (2.23).

As seen in Figure 4.2, after the first sampling interval of a switching cycle,

$$x[k+1] = A_d(T_{k,1})x[k] + B_d(T_{k,1})\hat{K}_1 x[k] + B_d(T_{k,1})F_1 r. \tag{4.6}$$

It is noted that K_1 is the feedback gain based on the most recent system state $x[k]$ and used to compute the control input. $\hat{K}_1$ is the equivalent feedback gain based on the starting system state $x[k]$ of a switching cycle. In this case that only one sampling period is considered, $\hat{K}_1 = K_1$. The feedforward gain F_1, which is related to K_1, is also based on the most recent system state and used to compute the control input. The closed-loop system matrix is denoted as $A_{cl,1}$ and

$$A_{cl,1} = A_d(T_{k,1}) + B_d(T_{k,1})\hat{K}_1. \tag{4.7}$$

If the pair $(A_d(T_{k,1}), B_d(T_{k,1}))$ is controllable, $\hat{K}_1$ can be designed by pole-placement. Poles to place are eigenvalues of $A_{cl,1}$. The number of poles is thus lN, where l is the number of system states and N is the number of sampling periods in the schedule. F_1 is computed as per (2.23).

After the second sampling interval,

$$x[k+2] = A_d(T_{k,2})x[k+1] + B_d(T_{k,2})K_2 x[k+1] + B_d(T_{k,2})F_2 r. \tag{4.8}$$

To consider the overall dynamics of the first two sampling periods, the relation between $x(t_{k+2})$ and $x(t_k)$ is derived as

$$\begin{aligned} x[k+2] = {} & A_d(T_{k,2})A_{cl,1}x[k] + B_d(T_{k,2})K_2A_{cl,1}x[k] \\ & + (A_d(T_{k,2}) + B_d(T_{k,2})K_2)B_d(T_{k,1})F_1r + B_d(T_{k,2})F_2r. \end{aligned} \tag{4.9}$$

Letting

$$\hat{K}_2 = K_2A_{cl,1}, \tag{4.10}$$

then (4.9) becomes

$$\begin{aligned} x[k+2] = {} & A_d(T_{k,2})A_{cl,1}x[k] + B_d(T_{k,2})\hat{K}_2x[k] \\ & + (A_d(T_{k,2}) + B_d(T_{k,2})K_2)B_d(T_{k,1})F_1r + B_d(T_{k,2})F_2r. \end{aligned} \tag{4.11}$$

Similar to (4.7),

$$A_{cl,2} = A_d(T_{k,2})A_{cl,1} + B_d(T_{k,2})\hat{K}_2. \tag{4.12}$$

It is noted that (4.11) has the same form as (4.6). If the pair $(A_d(T_{k,2})A_{cl,1}, B_d(T_{k,2}))$ is controllable, $\hat{K}_2$ can be designed by pole-placement and K_2 is derived with (4.10), as long as $A_{cl,1}$ is non-singular. Poles to place are eigenvalues of $A_{cl,2}$. F_2 is computed as per (2.23). Continuing the above analysis, the following can be defined as
$\forall j \in \{1, 2, \ldots, N\}$,

$$A_{cl,j} = A_d(T_{k,j})A_{cl,j-1} + B_d(T_{k,j})\hat{K}_j, \tag{4.13}$$

and $A_{cl,0} = \mathbf{I}$. If the pair $(A_d(T_{k,j})A_{cl,j-1}, B_d(T_{k,j}))$, where $j \in \{1, 2, \ldots, N\}$, is controllable, $\hat{K}_j$ can be designed by pole-placement. Poles to place are eigenvalues of $A_{cl,j}$. As long as $A_{cl,j-1}$ is non-singular, K_j is derived by

$$K_j = \hat{K}_jA_{cl,j-1}^{-1}. \tag{4.14}$$

F_j is computed as per (2.23).

4.3.2 Optimal Pole-Placement

Now the optimization problem for the pole-placement can be formulated as,

$$\begin{gathered} \min_{\mathbb{D}} t_s \\ \text{subject to} \\ u[k] \leq U_{max}; \quad t_s \leq t_s^0; \quad \forall j \in \{1, 2, \ldots, N\},\ rank(A_{cl,j}) = l, \end{gathered} \tag{4.15}$$

where poles are decision variables and the settling time t_s is to be minimized as the objective. There are four constraints. First, the input saturation has to be respected. Second, the settling time requirement has to be satisfied. Third, all closed-loop system matrices $A_{cl,j}$ must be non-singular. Fourth, $\mathbb{D}$ is a domain of poles with absolute values less than unity and ensures the system stability.

Table 4.2: Randomly initialized points in the numerical example of PSO

Initial Point	p_1	p_2	p_3	p_4	p_5
β_1	-1.7	-0.8	-1.4	0.5	1.9
β_2	-1	1.5	-1.8	1.8	1.6
φ	0.3456	0.0562	0.0241	0.0619	0.0525

It is challenging to solve such a constrained non-convex optimization problem with significant non-linearity. The efficient PSO technique as discussed in Chapter 2 is used. Constraints are handled in the following way. In comparison, a point that respects all constraints is always better than a point that violates at least one constraint, no matter what their objective values are. When comparing two points that both either respect all constraints or violate at least one constraint, the point with a shorter settling time is better.

As mentioned in Chapter 3, one major issue with PSO is its tendency for fast and premature convergence before the global optimum has been found, since its search is highly directional. This problem gets more severe as the number of dimensions in the decision space grows larger. The cognitive and social scaling parameters α_1 and α_2 have a significant impact on the search behavior and convergence of PSO. If α_1 is larger than α_2, the PSO tends to have better local searches around the points, yet converges more slowly. If α_2 is larger than α_1, the PSO often converges fast before thoroughly searching the local area around each point, which might miss the global optimum. This is a trade-off between optimality and efficiency. It is challenging to achieve both simultaneously.

There have been a number of works investigating the parameterization of PSO [JJ13, Ped10, NES11]. Various existing strategies for PSO parameters setting are summarized in [JJ13], which discusses some future research directions. A list of good parameter choices for several benchmarks is reported in [Ped10]. An adaptive inertia weight is proposed in [NES11] and uses the success rate of the swarm as its feedback parameter to ascertain the particles' situation in the search space. In this chapter, an adaptive parameterization approach for the cognitive and social scaling parameters with a constant sum is proposed. As the iteration number increases, α_1 is decreased and α_2 increases. The basic idea is that at the beginning of the optimization when particles are more disperse, local areas are better searched aiming to explore a larger space. When the optimization approaches to the end, particles are close to one another, and the focus is put on convergence. The goal is to achieve optimality and efficiency at the same time.

Assuming that the iteration number is q ($0 < q \leq q_{max}$, where q_{max} is the maximum number of iterations), the cognitive and social scaling parameters can be computed as,

$$\alpha_2 = f(\frac{q}{q_{max}}), \quad \alpha_1 = 4 - \alpha_2, \tag{4.16}$$

where the constant sum of α_1 and α_2 is taken as 4. f is a function that can be customarily decided. In this chapter, an exponential function is used as,

$$f(x) = 0.5e^{2x} + 0.1. \tag{4.17}$$

The numerical example in Chapter 3 is used to show the advantage of the proposed adaptively parameterized PSO technique. Five particles are randomly initialized at p_1, p_2, p_3, p_4 and

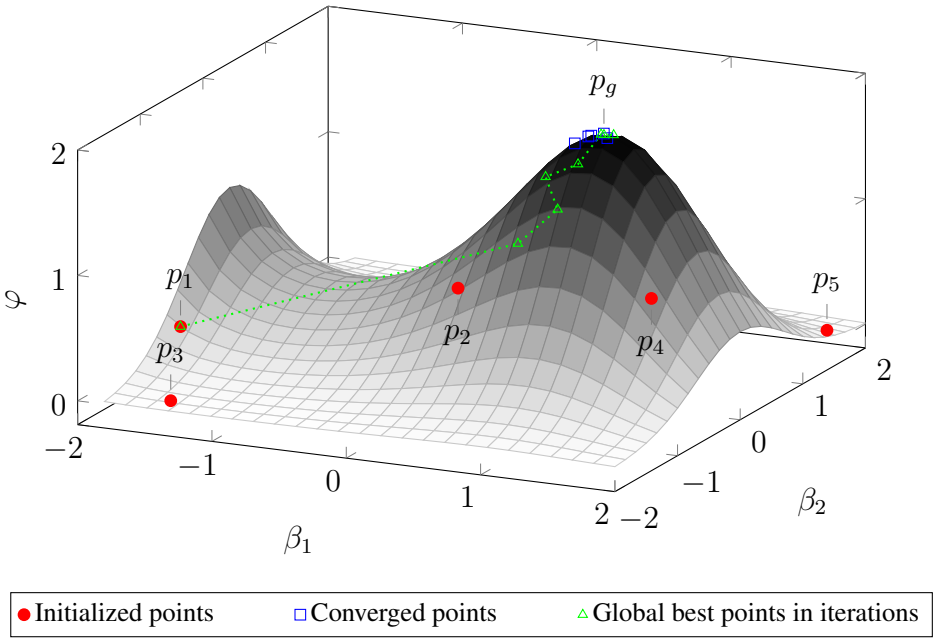

Figure 4.3: With the proposed novel particle swarm optimization method with adaptive parameterization, five particles are randomly initialized and converge to the global optimum p_g.

p_5, as shown in Table 4.2. Among them, p_1 has the best objective value. These initial points are the same as in Chapter 3 and the result with the conventional PSO technique has been illustrated in Figure 3.8. After 13 iterations, all five particles converge to points around the local optimum p_l ($\beta_1 = -1.7997, \beta_2 = 8.3188 \times 10^{-3}, \varphi = 1.1540$). The search is highly directed towards the global best point in each iteration. The global optimum is not found and particles quickly converge to the local optimum before exploring the decision space sufficiently.

The proposed novel PSO method with adaptive parameterization is illustrated in Figure 4.3. The same initial points are used as in Table 4.2. After 13 iterations, all five particles converge to points around the global optimum p_g ($\beta_1 = 9.8765 \times 10^{-1}, \beta_2 = -3.9853 \times 10^{-3}, \varphi = 1.9474$). The path showing how the global best point evolves iteratively is drawn, with certain points too close to others omitted. It can be seen that the decision space is better explored and the global optimum is found. This numerical example demonstrates the advantage of the novel PSO method over the conventional one.

4.3.3 Alternative Controller Design for Scalability

As discussed before, the number of dimensions in the decision space using the controller design presented in Section 4.3.1 is lN. When the number of sampling periods in a schedule N is very large, solving the pole-placement optimization problem could be computationally too heavy, even for an offline task. The PSO-based technique naturally offers a solution — decreasing the number of particles and iterations. However, this renders the result stochastic with a large variation and considerably dependent on the choices of initialization, which is often undesirable.

Here, an alternative controller design aiming for better scalability on the number of sampling periods is provided.

The complexity of the proposed controller in Section 4.3.1 comes from that the closed-loop dynamics of all the sampling periods are considered and optimized. A simpler design technique is to assume identical dynamics for the systems with the same sampling period. That is, if $T_j = T_{j'}$, the poles and feedback/feedforward gains of these two systems are the same. Clearly, the solution is suboptimal, since the assumption that poles are identical for systems with the same sampling period does not necessarily hold. The advantage is a smaller decision space. The number of decision variables (i.e., poles to place) becomes lN' — the number of states of the plant multiplied by the number of distinctive sampling periods in the schedule. Therefore, when the number of sampling periods N in the schedule S is very large and the number of distinctive sampling periods N' is relatively small, this alternative controller has better scalability on the number of sampling periods.

4.3.4 Non-Uniform MPC

In order to show that the proposed multirate controller is effective in both the linear and non-linear control, non-uniform MPC is analyzed. Following the derivation in Chapter 2, the quadratic cost remains as (2.25). Taking a schedule $(10ms, 10ms, 20ms)$ as an example and considering the next three sampling periods, the system states can be computed as

$$\begin{aligned}\begin{bmatrix} x[1] \\ x[2] \\ x[3] \end{bmatrix} &= \begin{bmatrix} B_d(10ms) & 0 & 0 \\ A_d(10ms)B_d(10ms) & B_d(10ms) & 0 \\ A_d(20ms)A_d(10ms)B_d(10ms) & A_d(20ms)B_d(10ms) & B_d(20ms) \end{bmatrix} \\ &\times \begin{bmatrix} u[0] \\ u[1] \\ u[2] \end{bmatrix} + \begin{bmatrix} A_d(10ms) \\ A_d(10ms)^2 \\ A_d(10ms)^2A_d(20ms) \end{bmatrix} x[0],\end{aligned} \tag{4.18}$$

if the next three sampling periods are $10ms$, $10ms$, and $20ms$.

The quadratic cost function then becomes

$$J = x^T[0]Qx[0] + (\bar{S}_1U + \bar{T}_1x[0])^T\bar{Q}(\bar{S}_1U + \bar{T}_1x[0]) + U^T\bar{R}U, \tag{4.19}$$

which is similar to (2.27), except that $\bar{S}$ and $\bar{T}$ are replaced by $\bar{S}_1$ and $\bar{T}_1$, respectively, where

$$\begin{aligned}\bar{S}_1 &= \begin{bmatrix} B_d(10ms) & 0 & 0 \\ A_d(10ms)B_d(10ms) & B_d(10ms) & 0 \\ A_d(20ms)A_d(10ms)B_d(10ms) & A_d(20ms)B_d(10ms) & B_d(20ms) \end{bmatrix} \\ \bar{T}_1 &= \begin{bmatrix} A_d(10ms) \\ A_d(10ms)^2 \\ A_d(10ms)^2A_d(20ms) \end{bmatrix}.\end{aligned} \tag{4.20}$$

Table 4.3: Settling time and processor utilization of three schedules

Schedule	Settling Time	Requirement	Processor Utilization
$S1 = \{5ms\}$	$253.69ms$	Violated	14%
$S2 = \{2ms\}$	$110.44ms$	Satisfied	35%
S^0 (novel PSO)	$128.6ms$	Satisfied	24.5%
S^0 (conventional PSO)	$154.05ms$	Violated	24.5%

When the next three sampling periods are $10ms$, $20ms$, and $10ms$,

$$\bar{S}_2 = \begin{bmatrix} B_d(10ms) & 0 & 0 \\ A_d(20ms)B_d(10ms) & B_d(20ms) & 0 \\ A_d(20ms)A_d(10ms)B_d(10ms) & A_d(10ms)B_d(20ms) & B_d(10ms) \end{bmatrix}$$
$$\bar{T}_2 = \begin{bmatrix} A_d(10ms) \\ A_d(10ms)A_d(20ms) \\ A_d(10ms)^2A_d(20ms) \end{bmatrix}. \tag{4.21}$$

In the cost function (4.19), $\bar{S}_2$ and $\bar{T}_2$ replace $\bar{S}_1$ and $\bar{T}_1$, respectively. When the next three sampling periods are $20ms$, $10ms$, and $10ms$,

$$\bar{S}_3 = \begin{bmatrix} B_d(20ms) & 0 & 0 \\ A_d(10ms)B_d(20ms) & B_d(10ms) & 0 \\ A_d(10ms)^2B_d(20ms) & A_d(10ms)B_d(10ms) & B_d(10ms) \end{bmatrix}$$
$$\bar{T}_3 = \begin{bmatrix} A_d(20ms) \\ A_d(10ms)A_d(20ms) \\ A_d(10ms)^2A_d(20ms) \end{bmatrix}. \tag{4.22}$$

In the cost function (4.19), $\bar{S}_3$ and $\bar{T}_3$ replace $\bar{S}_1$ and $\bar{T}_1$, respectively. In total, there are three constrained quadratic programming problems that can be solved by the interior-point method presented in Chapter 2.

4.4 Experimental Results

The proposed multirate controller design technique is evaluated with an EMB system used in automobiles, as shown in Section D. The set of available sampling periods offered by OSEK/VDX OS is

$$\phi = \{1ms, 2ms, 5ms, 10ms, 20ms, 50ms, 100ms, 200ms, 500ms, 1sec\}. \tag{4.23}$$

As shown in Table 4.3 and Figure 4.4, the schedule $S1 = \{5ms\}$ cannot meet the settling time requirement. The largest sampling period smaller than $5ms$ in ϕ is $2ms$. The schedule $S2 = \{2ms\}$ is able to fulfill all the requirements. According to (4.1), using the WCET requirement in Table D.3, the processor utilization of $S2$ is 35%. As discussed before, this number can be unnecessarily large and prevents more applications from sharing the ECU.

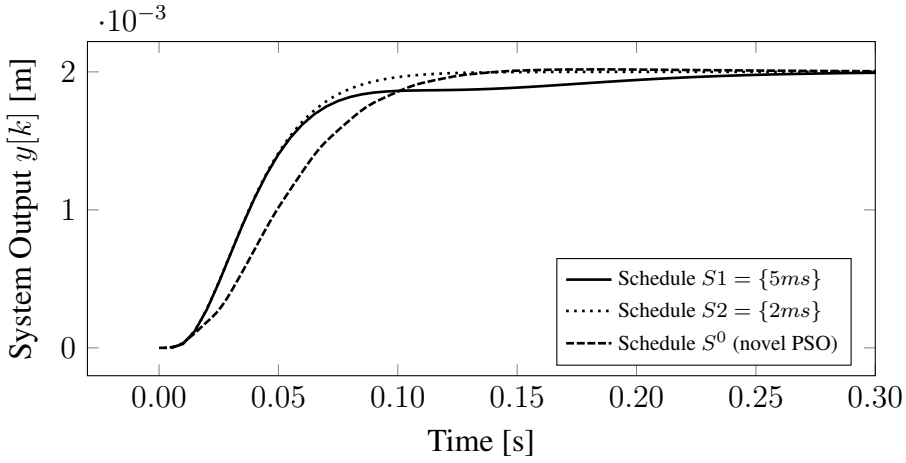

Figure 4.4: System output of three different schedules. The proposed novel PSO technique is used for S^0.

Table 4.4: Poles of closed-loop system matrices

Closed-loop system matrix	Poles
$A_{cl,1}$	[0.9202 0.6460 0.8334 0.6243 0.6802]
$A_{cl,2}$	[0.6902 0.1039 0.4454 0.7232 0.5911]
$A_{cl,3}$	[0.4671 0.2641 0.6241 0.1696 0.5199]
$A_{cl,4}$	[0.4060 0.4473 0.7508 0.0751 0.1369]
$A_{cl,5}$	[0.8570 0.2319 0.7864 0.3954 0.4922]
$A_{cl,6}$	[0.6598 0.0952 0.7009 0.7579 0.8847]
$A_{cl,7}$	[0.7284 0.2338 0.1384 0.4873 0.0415]

Then the schedule $S^0 = \{2ms, 2ms, 2ms, 2ms, 2ms, 5ms, 5ms\}$ switching between $2ms$ and $5ms$ is evaluated. This sequence of sampling periods satisfies the OSEK/VDX OS requirement as discussed in Section 4.2. The multirate controller is designed as proposed in Section 4.3 and the novel PSO with adaptive parameterization as in Section 4.3.2 is used for optimal pole-placement. 50 particles are used and converge after 16 iterations. Increasing the number of particles beyond 50 does not further improve the control performance. 35 poles of seven closed-loop system matrices are reported in Table 4.4. S^0 has a slightly longer settling time than $S2$, but still fulfills the requirement. According to (4.4), the processor utilization is 24.5%, achieving a 30% reduction compared to $S2$. The settling time of S^0 is also evaluated using the conventional PSO technique. 50 particles are used and the convergence also takes 16 iterations. As reported in Table 4.3, the solution does not satisfy the requirement.

Comparison of the optimal controller presented in Section 4.3 and the scalable controller in Section 4.3.3 is shown in Table 4.5. The settling time generated by the scalable controller is longer than that of the optimal one and violates the requirement. However, it only takes 15 particles that converge after 9 iterations. Increasing the number of particles beyond 15 does

Table 4.5: Comparison of the optimal and scalable controller design

Controller	Settling Time	Requirement	Particle	Iteration	Time
Optimal	$128.6ms$	Satisfied	50	16	$1448s$
Scalable	$157.05ms$	Violated	15	9	$36s$

not further improve the control performance. The total computation time on a computer with an Intel i5 processor operating at 2.6GHz with 4GB RAM is $36s$, compared to $1448s$ for the optimal controller. Although $1448s$ sounds acceptable for an offline task, when a schedule has more sampling periods than S^0, the computation could take hours or even days due to the increase of the decision space dimensions. In such a case, if the number of distinctive sampling periods is small, the scalable controller design can be used first to check whether the requirements can be satisfied.

Now we consider a case that multiple applications are to be implemented on ECUs. For the convenience of illustration, all applications are assumed to be identical to the EMB system discussed before with the WCET of $0.7ms$. As reported above, the schedule $S2 = \{2ms\}$ is able to satisfy the control performance requirement and system constraints. Under $S2$, an ECU is able to accommodate two applications according to (4.2). Under the multirate schedule S^0, four applications can share one ECU, where detailed invocation timing is presented in Figure 4.5 and Table 4.6. While the schedule for C_1 and C_2 is $\{2ms, 2ms, 2ms, 2ms, 2ms, 5ms, 5ms\}$, the schedule for C_3 and C_4 is $\{5ms, 5ms, 2ms, 2ms, 2ms, 2ms, 2ms\}$. The switching of sampling periods occurs every $10ms$ for both schedules. When an application is in the shorter sampling period ($2ms$ in this case), it occupies the ECU more often. Therefore, these two variants of S^0 are essentially opposite to each other, so that when some applications require more frequent access to the ECU, others are in the longer sampling period ($5ms$ in this case), requesting the execution less often. In this way, the number of applications packed onto the ECU can be maximized.

It is noted that preemption is supported in OSEK/VDX. For instance, at $1.4ms$ when both C_1 and C_2 finish their first invocations, C_3 is started and allowed to access the processor for $0.6ms$. After that, C_3 is suspended, waiting for the second invocations of C_1 and C_2. Then, C_3 resumes and completes its first invocation. It can be seen that the number of applications that are accommodated by an ECU is doubled with the proposed OS-aware multirate controller design, which is significant for the cost-sensitive automotive domain.

MPC is used to show that the proposed multirate controller is effective in both linear and non-linear control. The EWB system presented in Section A is evaluated. The schedule $S'^0 = \{10ms, 10ms, 20ms\}$ is taken as an example. It is assumed that the quadratic cost of every ms is computed and that the sum is expected to be below 550.

As shown in Table 4.7, the schedule $S4 = \{20ms\}$ cannot meet the quadratic cost requirement. The largest sampling period smaller than $20ms$ in ϕ is $10ms$. The schedule $S3 = \{10ms\}$ is able to fulfill all the requirements. Then the schedule $S'^0 = \{10ms, 10ms, 20ms\}$ switching between $10ms$ and $20ms$ is evaluated. This sequence of sampling periods satisfies the OSEK/VDX OS requirement as discussed in Section 4.2. The multirate controller is designed

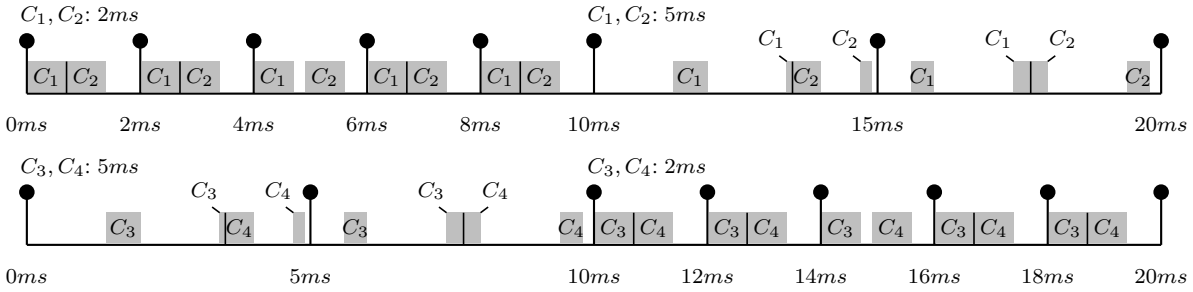

Figure 4.5: Invocation timing of four control applications under the schedule S^0. The schedule for C_1 and C_2 is $\{2ms, 2ms, 2ms, 2ms, 5ms, 5ms\}$, and the schedule for C_3 and C_4 is $\{5ms, 5ms, 2ms, 2ms, 2ms, 2ms\}$. Preemption is allowed in OSEK/VDX OS.

Table 4.6: Exact invocation starting times of four control applications under the multirate schedule S^0. When one invocation is preempted, two starting times are separated by a forward slash and the number in the bracket indicates the duration. The timing unit is ms.

Invocation	C_1	C_2	C_3	C_4
1	0	0.7	1.4(0.6)/3.4(0.1)	3.5(0.5)/4.7(0.2)
2	2	2.7	5.6(0.4)/7.4(0.3)	7.7(0.3)/9.4(0.4)
3	4	4.9	10	10.7
4	6	6.7	12	12.7
5	8	8.7	14	14.9
6	11.4(0.6)/13.4(0.1)	13.5(0.5)/14.7(0.2)	16	16.7
7	15.6(0.4)/17.4(0.3)	17.7(0.3)/19.4(0.4)	18	18.7

Table 4.7: Quadratic cost of three schedules

Schedule	Quadratic Cost	Requirement
$S3 = \{10ms\}$	512.33	Satisfied
$S4 = \{20ms\}$	580.16	Violated
$S'^0 = \{10ms, 10ms, 20ms\}$	513.40	Satisfied

as discussed in Section 4.3.4. S'^0 has a slightly larger quadratic cost than $S3$, but still fulfills the requirement.

4.5 Remarks

This chapter discusses computation-aware automotive control systems design. In particular, to deal with the restriction imposed by the OS on sampling periods for control applications, a novel performance-oriented multirate controller design technique is proposed, in which an adaptively parameterized PSO is used for optimal pole-placement. It reduces the processor

utilization, while satisfying the control performance requirement and system constraints. This saves the scarce computational resources on the automobiles and enables integration of more functions and applications along the philosophy of ECU consolidation. A case study shows that the number of control applications that share an ECU can be doubled with the proposed OS-aware controller design.

In the controller design, the sensor-to-actuator delay is neglected. The method dealing with the sensor-to-actuator delay shorter than or equal to one sampling period has been presented in Chapter 3. Two new PSO-based techniques are illustrated in Chapter 3 and this chapter, aiming to address the issue of premature convergence. The technique in Chapter 3 makes use of the worst point in deciding the search direction and the technique in this chapter adaptively adjusts the cognitive and social scaling parameters to encourage local exploration in the beginning search phase and global convergence in the ending search phase.

When the control performance is desired to be maximized instead of serving as a requirement to be satisfied, using the proposed multirate controller, there can be various schedules with non-uniform sampling that are able to achieve a trade-off between control performance and the use of computational resources. Synthesizing the optimal computation-aware schedule can be a direction of future work.

While in this chapter the focus is on single-core ECUs, the presented approach can be extended to multi-core architectures. There are several challenges to be addressed. First of all, it might be necessary due to load balancing requirements to distribute different parts of complex control applications to different cores. This introduces additional delays for sensor to actuator cause-effect chains that need to be taken into account during controller design to ensure stability. Moreover, the placement of shared instructions and data to local and global memories including memory arbitration effects needs to be considered, since this has major influence on the execution times of control programs.

5

Battery- and Aging-Aware Automotive Control Systems Design

A reduced emission, independence from fossil fuels, improvement of energy conversion efficiency, together with better torque and noise performances at low speeds have made EVs a potential alternative of conventional vehicles with ICEs. Battery is a key component in an EV. In order to achieve a larger driving range, it is desirable to make the most use of the battery with a given nominal capacity. Therefore, on one hand, the energy consumption of a control task should be minimized and on the other hand, the discharging current profile should maximize the effective capacity of the battery.

In this chapter, an optimization framework with both control performance and battery usage considered as design objectives is proposed. The control performance metric is the settling time and the battery usage is quantified by the number of times the control system can reach a steady state after a disturbance occurs powered up by a fully charged battery pack. With gradient-based and stochastic methods implemented, this battery-aware controller design offers a Pareto front of well-distributed and non-dominated solutions. The trade-off between these two objectives is explored.

Besides battery, another important design aspect is processor aging. As briefly discussed in Chapter 1, the processor operating frequency is decreased along with its use. This results in a longer sampling period for a control application and potentially degraded control performance, which cannot be tolerated in the safety-critical domain. The same optimization framework as above is used with slight modification. The controller is re-optimized to mitigate the processor aging effect in the way that the control performance is kept not deterioratead with an inconsiderable compromise on the battery usage. Throughout this chapter, the electric motor control presented in Section C is used as an example application.

The organization of this chapter is as follows. Section 5.1 discusses the literature in optimal sampling period selection, battery rate capacity effect and processor aging. In Section 5.2, two

design aspects for embedded control systems in EVs are discussed. Battery usage is introduced as the other design objective with the rate capacity effect taken into account, besides the control performance indexed by settling time. Processor aging and its influence on the control system sampling period are analyzed. Section 5.3 presents the optimization framework and flow. Experimental results are reported in Section 5.4 and Section 5.5 makes some remarks of this chapter.

5.1 Related Work

There have been a number of works on choosing the optimal sampling period for feedback control applications [BC08, ZSWM08]. The delay is part of the linear cost function in [BC08] and estimated using an approximate response-time analysis. The optimal period assignment problem is solved analytically and the performance improvement is verified. The task scheduling problem for feedback control applications is studied in [ZSWM08] in order to optimize the performance of both the computing unit and the physical plant. The control law and the task scheduling algorithm are co-designed for predictable performance and power consumption. However, none of the works have directly investigated the relationship of the control system with properties of the input energy source (i.e., battery) and effects caused by the underlying processing platform aging, which are two important aspects in embedded control systems design for EVs.

In battery modelling, one important issue to address is the rate capacity effect that the capacity of a battery cell varies with different discharging current profiles. In general, researchers have developed three types of battery models to characterize this effect. An electrochemical model is based on chemical processes taking place within the battery [SRW07]. These models describe the battery processes in great detail, which makes them the most accurate among all types of models. However, highly detailed description makes the models complex and difficult to configure. Electrical-circuit models use circuit elements to simulate battery behavior. The first such model was proposed in [Hag93]. They are simpler than the electrochemical models and thus computationally less expensive. However, it still takes considerable efforts to configure the electrical-circuit models. Especially the lookup tables used in the model require lots of experimental data on the battery behavior. An analytical model describes the battery at a higher level of abstraction than the other two types. The major properties of the battery are modeled by a couple of equations, which makes this type of model much easier to use. In this chapter, the extended Peukert's law presented in [RV01] is used to characterize battery capacity rate effect. With carefully chosen parameters based on experimental data, reasonable accuracy of about 10% error can be achieved.

As a processor ages, the transistor switching time is increased and the path delay gets prolonged. Signals may not be able to go through some of the paths within one clock cycle, which results in timing constraint violation and false calculation. This issue can be handled by safety margins, supply voltage regulation or clock frequency scaling [MKB$^+$12]. Currently, the most widely-used method is implementing a frequency or voltage guard band. The processor is operated at a lower frequency or higher voltage that meets worst-case path delays. It is noted that a higher supply voltage makes transistor switching faster. In this way, as the processor ages and

gets slower, the system does not have to be changed. However, the guard band can be quite big, leading to a passive design and a waste of resources[LDF+11]. In cost-sensitive domains like EVs, there is a strong motivation to make full use of resources and minimize the guard band, while reliable functionality is maintained. Instead of having a fixed setting that deals with worst-case conditions, some newer approaches suggest changing settings dynamically. By monitoring the critical path, delays are detected and the supply voltage of the processor is adjusted accordingly [BTW+09]. These approaches are effective but can be difficult to implement. The supply voltage is usually limited by the maximum allowable input current in the circuit, cooling constraints and other temperature-dependent reliability issues, which are sensitive in the automotive domain. In this chapter, theautonomous frequency scaling is used to deal with processor aging [MKB+12]. On-chip monitors could be used to watch the critical path delay, based on which, the processor operating frequency is adjusted by a frequency synthesis circuit to ensure that signals can go through all paths within one clock cycle. Timing constraints are respected and functions are guaranteed to be correct.

5.2 Design Aspects of Electric Vehicles

In this section, two important design aspects of EVs — battery usage and processor aging — are discussed. Besides the settling time t_s as one design objective for control performance, r is made the other design objective to quantify battery usage. r is defined to be the number of times the control system can reach a steady state after a disturbance occurs with a fully charged battery pack, taking account of the rate capacity effect. Then processor aging and its influence on the control performance are analyzed.

5.2.1 Battery Rate Capacity Effect

The battery pack is one of the most important components in EVs. Battery capacity is the major factor determining the driving range of an EV. A natural objective of embedded control systems design for EVs is to minimize the energy consumption of each control task invocation. However, this is not enough to thoroughly and accurately consider the energy impact from control systems design. Due to battery rate capacity effect, the FCC of a battery pack varies depending on discharging current profiles. This effect needs to be considered in embedded control systems design for EVs since different controller designs result in different current profiles. The rate capacity effect is described by extended Peukert's law [RV01] as shown below,

$$L_t = \frac{a}{\left(\frac{\sum_{k=1}^{n} I_k (t'_{k+1} - t'_k)}{L_t}\right)^d}, \tag{5.1}$$

which is able to handle non-constant loads. $t'_1 = 0$ is the starting time stamp and $L_t = t'_{n+1}$ is the total duration that the battery can be used before next charging and divided into n slots. In this chapter, each slot is equal to one sampling period of the control system. For the kth time slot, I_k is the average current for the sampling period from t_k to t_{k+1}. a and d are determined by experimental data. It is noted that this is an approximation of the battery rate capacity effect

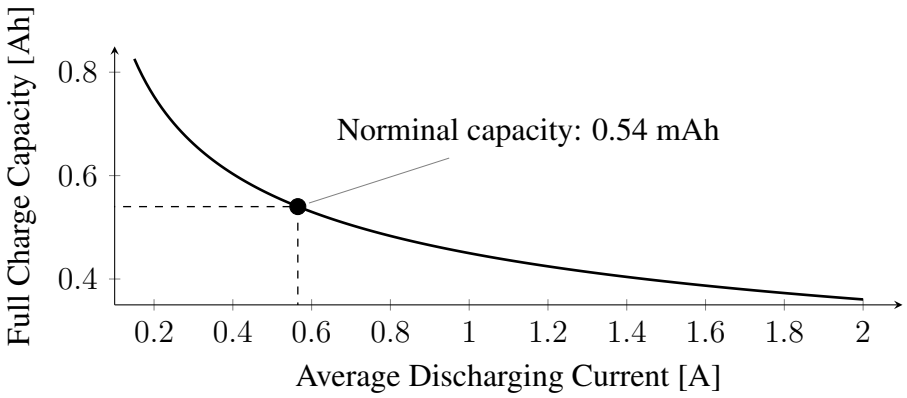

Figure 5.1: Relationship between battery FCC and average discharging current

by considering the average discharging current. As discussed in Section 5.1, more accurate methods can be used as well, and the entire framework presented in this chapter still holds. The battery capacity under the discharging current profile $\{(I_1, t_1), (I_2, t_2), \ldots, (I_n, t_n)\}$ is

$$Q_{FCC} = \sum_{k=1}^{n} I_k(t_{k+1} - t_k). \tag{5.2}$$

Ideally, both Q_{FCC} and a are equal to the nominal battery capacity Q_{nom} and $d = 1$.

In this chapter, the lithium-ion rechargeable batteries with lithium cobalt oxide cathode and graphite anode produced by Sony (UP383562) is used. The operating voltage of each cell is $3.7V$. One constraint on the battery pack is that the current drawn from each cell cannot exceed a certain value I_c^{max}. From the experimental data in [Son], a and d can be calculated as 0.45 and 1.32, respectively. The relationship between the FCC of a battery cell and the average discharging current is shown in Figure 5.1. The nominal capacity labeled with the battery is $0.54Ah$.

Besides t_s as a design objective to quantify the control performance, battery usage is proposed as the other objective in the embedded control systems design for EVs, which is characterized by the number of times r the control system can reach a steady state after a disturbance occurs with a fully charged battery pack. In order to maximize r, (i) the energy consumption of each single control task invocation has to be small, and (ii) the average discharging current is desirable to be small to improve the FCC. Assuming that the battery only powers the electric motor control task, which is presented in Section C, based on control system description in Chapter 2 and (5.1), r can be calculated as

$$r = \frac{a}{\left(\frac{\sum_{k=1}^{n_{sp}} I_k(t_{k+1}-t_k)}{t_s}\right)^d \times t_s}, \tag{5.3}$$

where t_s is the time taken to complete one single control task invocation, i.e., settling time, as discussed before. $t_{k+1} - t_k$ is the sampling period of the control system, i.e., h. These time slots are constant since control systems with constant sampling period are dealt with in this

chapter. n_{sp} is the total number of sampling periods in each invocation. Referring to Chapter 2 and Section C, when the controller design is decided, from experiments the value of t_s and the supplied current profile can be obtained, based on which the sum of current $\sum_{k=1}^{n_{sp}} I_k(t_{k+1} - t_k)$ can be derived. In general, with different controller designs, i.e., eigenvalue selections, these two objectives t_s and r take different values. The relationship between these two objectives is not explicit since the sum of current also varies depending on how the controller is designed. Therefore, it is necessary to develop an optimization technique to optimize both t_s and r while all constraints must be satisfied.

5.2.2 Processor Aging in Embedded Control Systems

Electronics plays a major role in embedded control systems of modern vehicles. A processor is the key part ensuring correct functioning and timing behavior in control. As the transistor size keeps shrinking, processors are becoming more and more susceptible to aging, which could get control performance degraded. This is dangerous and highly undesirable in safety-critical applications like EVs. The main transistor aging mechanisms are Hot Carrier Injection (HCI) and Negative Bias Temperature Instability (NBTI) [MIMG08]. They cause degradation in the electrical characteristics of transistors, such as a shift of the threshold voltage [SB03]. As a result, the switching times of transistors are prolonged.

There are many paths for digital signal transmission in a processor and each path consists of a number of transistors. The time that a signal takes to travel through the path is called path delay. As transistors require more time for switching, the path delay is increased [LBS10]. It has to be ensured that the signal transmission along any path can be completed within one clock cycle. Therefore, the operating frequency of the processor is determined by the worst-case path delay, which is the path with the longest delay and called the critical path. With the above analysis, it is known that as a processor ages, the signal may not be able to travel through certain paths within one clock cycle if the operating frequency remains fixed. This could produce incorrect functional results, that cannot be tolerated by safety-critical applications like motor control in EVs.

As discussed in Section 5.1, one appropriate way to keep functions correct in the domain of EVs is reducing the operating frequency. Before everything, the extend of aging needs to be known. An on-chip aging monitor could be implemented to watch the delay of the critical path. Shrinking transistor dimensions make larger within-die and die-to-die variations [PA12]. The static timing analysis at the circuit level is not able to precisely determine which path among all is the critical one in modern processors. However, all critical path candidates, which are a subset of all paths in the processor that can become the critical one during the runtime, can be identified. To avoid interference on real functions run on the processor, CPRs, which are replicas of these candidates, are fabricated on the chip [PA12]. The aging monitor then sends signals to CPRs to get the current critical path delay, based on which it is known whether the operating frequency needs to be reduced and if so, by how much. A frequency generator changes the operating frequency accordingly, if necessary.

When the processor operating frequency is decreased, the execution time of control programs gets longer. The sampling period is mainly constrained by the control program WCET. Therefore, the sampling period of the control system is increased. It is possible to avoid the

aging effect by inserting a safety margin between the WCET and the sampling period. However, this is a waste of resources and not desirable for cost-sensitive applications like EVs. In general, a shorter sampling period means more frequent response from the controller and thus potentially better control performance. When the increased sampling period due to processor aging is fed into the control system, the results showing how control performance changes can be obtained.

Based on the method presented in [MKB+12], it can be estimated how much processor aging can be expected in EVs. The processor only ages when it is used, i.e., in the non-idle state. Taking an electric taxi as an example, the drivers take shifts and the car is driven approximately 20 hours every day. The processor is always in the non-idle state while the car is driven. After 4 years of use, the overall operation time of the processor is approximately $\frac{5}{6} \times 4 \approx 3.33$ years. According to the results presented in [MKB+12], the processor is roughly 10% slower.

5.3 Optimization Framework

As discussed in Section 5.2, three design aspects — control performance, battery usage and processor aging — are considered. In this section, the optimization framework for battery-aware controller design is first presented. This is a constrained bi-objective optimization problem with control performance and battery usage as objectives. Both gradient-based and stochastic methods are used to solve it. They complement each other with their respective advantages to obtain well-distributed non-dominated Pareto points. An algorithm is proposed to further improve the distribution quality. The trade-off between the control performance Q_b and battery usage R_b can then be explored. Afterwards, the framework is slightly modified for battery- and aging-aware controller design to ensure that the control performance keeps not getting degraded, i.e., $Q_b' \geq Q_b$.

5.3.1 Battery-Aware Controller Design

In this battery-aware controller design optimization problem, the two closed-loop eigenvalues (i.e., poles) p_1 and p_2 in the closed-loop system dynamics of electric motor control are decision variables to tune. The space for the eigenvalues is restricted in the real positive plane — which is the case in most of the real-life design problems. This is particularly acceptable considering the possible oscillation with complex eigenvalues in speed control of EVs. Therefore,

$$\forall i \in \{1, 2\}, \;\; 0 \leq p_i < 1. \tag{5.4}$$

It can be seen that the design space is continuous. As discussed before, the two objectives to optimize are

- the settling time of the control task t_s and
- the number of times r the control system can reach a steady state after a disturbance occurs with a fully charged battery pack.

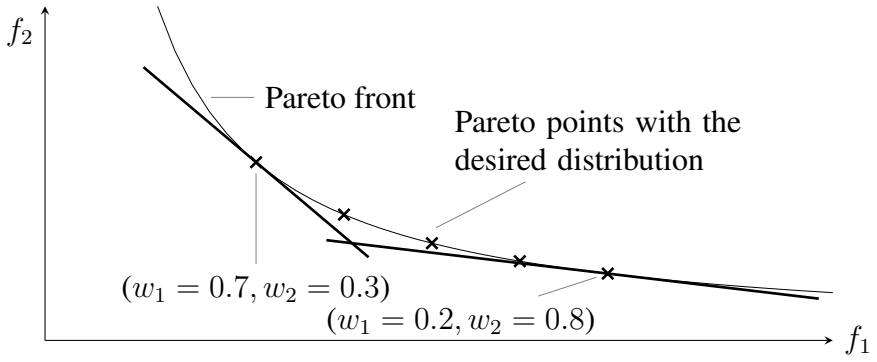

Figure 5.2: Illustration of how SQP locates Pareto points with various pairs of weights

t_s should be minimized and r should be maximized. Usually an optimization technique takes objectives either to minimize or maximize but not both. Therefore, in this chapter, $f_1 = t_s$ and $f_2 = -r$ are to be minimized.

There are several constraints to be satisfied. As shown in (C.18), the control input, i.e., the duty cycle of the PWM control signals c is constrained as $0 \leq c \leq 1$. As shown in (C.19), the current in the motor cannot exceed the maximum allowed current, i.e., I_m^{max}. The current drawn from each cell cannot be larger than I_c^{max} as explained in Section 5.2.1. For automotive control applications, timing is critical. It is often necessary to guarantee that each control task meets the settling time requirement, leading to the constraint $t_s \leq t_s^0$ as discussed before. The last constraint is related to overshoot and explained in (2.17).

In solving a constrained bi-objective optimization problem, there are typically two goals to achieve:

- the solution point is on the Pareto surface and
- there is a good distribution of solution points.

The first goal ensures that the solution point is not dominated by any other point. In other words, no other point is better than the solution in both objectives. The second goal gives designers more freedom to choose solutions under different circumstances. In this controller design optimization problem with significant non-linearity and non-convexity, the design space is continuous with infinite design choices. There is no relationship between objectives and decision variables that can be explicitly formulated. Therefore, it is a challenging task to achieve both optimization goals.

5.3.2 Optimization Techniques

There are generally two types of heuristic techniques to solve such a constrained bi-objective optimization problem as presented above: gradient-based and stochastic methods [KSD07]. They have different advantages and disadvantages. As discussed in Chapter 2, SQP is a popular gradient-based optimization technique for single-objective non-convex problems and can be extended to solve multi-objective problems by scalarization. However, SQP is good at finding

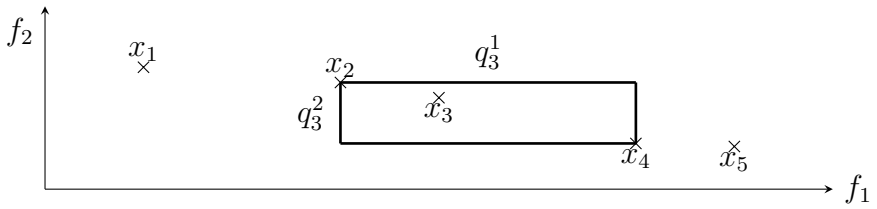

Figure 5.3: Illustration of the crowding distance calculation for Pareto points distribution quality quantification

local optima and also easily gets trapped by them, which makes it lose track of global optima. A common way to improve the global awareness of SQP is taking multiple starting points to explore different regions, yet it is often a challenging task to select an appropriate list of starting points covering the entire objective space. In practice, starting point selection is usually on the random base and thus the objective space is only partially searched.

SQP is also weak in locating a good distribution of Pareto points in multi-objective optimization. Taking two objectives as an example, different pairs of weights (w_1, w_2) find different Pareto points. For example, optimizing f_0 constructed by $(w_1 = 1, w_2 = 0)$ and $(w_1 = 0, w_2 = 1)$ finds the solution optimizing f_1 and f_2, respectively. And $(w_1 = 0.5, w_2 = 0.5)$ corresponds to the point with equal emphasis on both objectives. However, a set of equally distant pairs of weights does not necessarily produce a set of equally distant Pareto points. As shown in Figure 5.2, when applying a pair (w_1, w_2), the straight line with the gradient of $-\frac{w_1}{w_2}$ starts from the lower-left corner and approaches the Pareto front. The first intercepting point between the straight line and the Pareto front is then the Pareto point corresponding to this pair (w_1, w_2). Given the convex Pareto front in Figure 5.2, theoretically, an appropriate set of weight pairs is able to locate a good distribution of Pareto points, yet it is often difficult to identify such a set since the information regarding the Pareto front is always missing. It becomes especially difficult when the gradient change along the Pareto surface is small. Besides, Pareto points in the non-convex region of the Pareto front cannot be located by SQP at all.

Stochastic methods have also been developed to solve such a constrained bi-objective optimization problem. One example is the popular NSGA discussed in Chapter 2. The advantage of NSGA over SQP lies in its stochastic nature. More of the objective space can be explored, so that it is very likely that a good distribution of Pareto points can be selected from the final converged population. NSGA is not easily trapped by local optima and thus has better global awareness. This is also its weakness that converged solutions are not guaranteed to be local optima. Considering that both gradient-based and stochastic optimization methods have different advantages, both of them are implemented and the solutions found by them are combined. The overall runtime is in minutes and scalability is not an issue, since the number of objectives is fixed to be two and the controller design is an offline task.

There are various ways to quantify the distribution quality for a set of Pareto points. In this problem of embedded controller design for EVs, the distribution in the objective space instead of decision variable space is investigated since it is important to see the trade-off between control performance and battery usage. The popular method of crowding distance calculation described in [DPAM02] is modified to suit this context. As illustrated in Figure 5.3, assuming that there

Algorithm 1: Removal of less representative solution points according to crowding distance ranking

Input: $S = \{x_1, x_2, \ldots, x_n\}$, n_d, $\{\rho_1, \rho_2\}$, $\{f_1, f_2\}$
Output: $S_d = \{x_1, x_2, \ldots, x_{n_d}\}$
for $j \in \{1, 2, \ldots, n - n_d\}$ **do**
  for $i \in \{1, 2, \ldots, n - j + 1\}$ **do**
    calculate q_i^1 and q_i^2 for the element x_i as in (5.5);
  end
  for $k \in \{1, 2\}$ **do**
    sort S based on q_i^k from maximum to minimum;
    for $i \in \{1, 2, \ldots, n - j + 1\}$ **do**
      assign the position value (1 for maximum to $n - j + 1$ for minimum) of x_i in S to r_i^k;
    end
  end
  for $i \in \{1, 2, \ldots, n - j + 1\}$ **do**
    calculate r_i^0 as in (5.7);
  end
  Sort S based on r_i^0 from maximum to minimum to be $\{x_1, x_2, \ldots, x_{n-j+1}\}$;
  Remove x_{n-j+1} from S;
end
$S_d = S$;

are two objectives $\{f_1, f_2\}$ and n solution points $\{x_1, x_2, \ldots, x_n\}$ ordered by the value of either objective, for each point x_i, $i \in \{1, 2, \ldots, n\}$, that is not at the end of this point sequence, the crowding distance of x_i in terms of the objective f_k, $k \in \{1, 2\}$ can be calculated as

$$q_i^k = |f_k(x_{i+1}) - f_k(x_{i-1})|, \tag{5.5}$$

where x_{i+1} and x_{i-1} are the two closest points to x_i on each side, respectively. Since a set of Pareto points that are non-dominated is dealt with, x_{i+1} and x_{i-1} are closest to x_i in terms of both objectives. Both end points of the point sequence are excluded from crowding distance calculation. The overall crowding distance for the objective f_k is

$$\bar{q}^k = \frac{1}{n-2} \sum_{i=2}^{n-1} q_i^k. \tag{5.6}$$

A smaller crowding distance $\bar{q}^k$ indicates that these solution points are more crowded in terms of the objective f_k and thus the distribution quality is low. Larger crowding distances are desirable since they represent a good distribution where all solution points are more representative.

Optimization techniques are required to explore the objective space as much as possible in order to find more Pareto points. However, among all found points, some can be very close to others. They are not representative and thus should be removed. The question is then which

points to remove so that the overall crowding distance can be maximized. First, for each point, the two crowding distances corresponding to the two objectives are computed. Two ranks r^1 and r^2 are assigned to it based on the comparison in crowding distances with other points. For instance, if the point x_i has the maximum crowding distance in terms of f_1 among all n points, then $r_i^1 = 1$. If x_i has the minimum crowding distance, then $r_i^1 = n$. The overall rank of x_i is

$$r_i^0 = \rho_1 r_i^1 + \rho_2 r_i^2, \tag{5.7}$$

where ρ_1 and ρ_2 are importance factors of two objectives. These values depend on the application and

$$\rho_1 + \rho_2 = 1. \tag{5.8}$$

For example, if in an application, f_1 is the single key objective, then ρ_1 can be set to 1 and ρ_2 to 0. In this case, r_i^0 is equal to r_i^1 and all points are ranked according to their crowding distances in terms of f_1. After each point x_i has an overall rank r_i^0, the point with the smallest r_i^0 is removed from the solution set. The entire process starting from crowding distance calculation is iterated until the desirable number of points n_d is reached. Both end points of the point sequence are always kept in the set to maintain the coverage of the solution set. The algorithm is shown in Algorithm 1. It is noted that there are two objectives in this embedded controller design problem for EVs. Everything presented in this section can be easily extended to a multi-objective optimization problem.

5.3.3 Battery- and Aging-Aware Controller Design

As discussed in Section 5.2, processor aging prolongs the control system sampling period, opening up the possibility of control performance deterioration. This has to be prevented for safety-critical applications like motor control in EVs as it endangers lives of both passengers and the driver. For instance, in ACC, if it takes more than original time to change the motor speed after the vehicle is driven for some time, accidents might occur. In order to ensure that the control performance does not get degraded, i.e., $Q_b' \geq Q_b$, the optimization framework for battery-aware controller design is modified. One constraint is added for every generated solution that t_s must not get longer. The constraint $t_s \leq t_s^0$ is then not needed any more. Battery usage r becomes the single objective in this battery- and aging-aware controller design optimization problem. The goal will only be approaching the global optimum as closely as possible. In order to achieve it, the objective space needs to be fully explored, to find local optima and compare them. Therefore, both SQP and the genetic algorithm are used for their respective advantages. For SQP, no objective conversion is necessary. A simpler genetic algorithm is implemented to replace NSGA in the sense that sorting is only based on the objective value r in environmental selection. The algorithm to improve distribution quality is removed as only one single solution is looked for and the distribution is no longer relevant.

5.4 Experimental Results

In this section, the optimization results of the battery-aware embedded control systems design for EVs are first shown. Solutions produced by SQP and NSGA are combined and sorted by

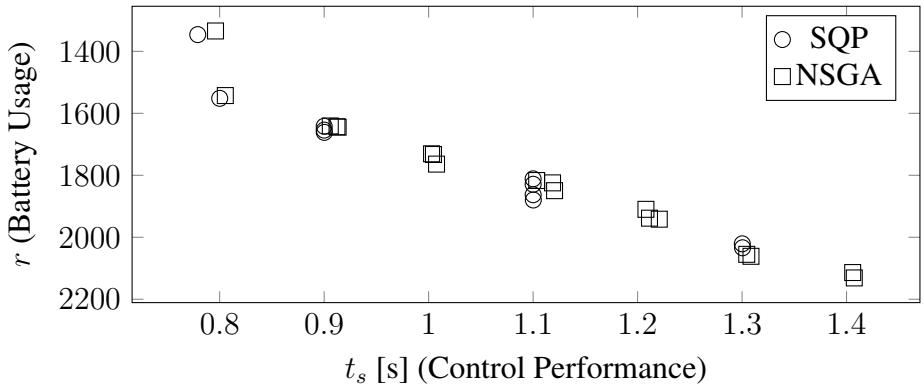

Figure 5.4: Non-dominated solutions found by SQP and converged solutions found by NSGA. For SQP, 20 random starting points and 3 pairs of weights for objective conversion are used. For NSGA, 20 random starting points are used and the maximum allowed number of generations is 10. Many NSGA solutions are dominated by SQP solutions but NSGA covers more of the objective space.

domination. Only non-dominated ones are kept. The algorithm is illustrated to improve the distribution quality of the solution set. The trade-off between control performance Q_b and battery usage R_b is explored. Then processor aging effects are reported in terms of both design objectives: control performance Q'_b and battery usage R'_b. It is shown that with controller design re-optimization in battery- and aging-aware controller design, the safety-critical control performance can be kept not getting degraded with a slight compromise on battery usage, i.e., $Q'_b \geq Q_b$.

The electric motor control presented in Section C is used for evaluation with a starting sampling period of $0.1s$ before processor aging. The battery pack consists of 100 cells in series and 30 cells in parallel. Each cell has a unit voltage of $3.7V$, as presented in Section 5.2.1. The settling time is expected to be below $1.5s$ and the maximum allowed current in the battery cell I_c^{max} is $5A$.

In the battery-aware controller design, solutions produced by both SQP and NSGA in the objective space are shown in Figure 5.4. For SQP, three pairs of weights $(w_1 = 1, w_2 = 0)$, $(w_1 = 0, w_2 = 1)$ and $(w_1 = 0.5, w_2 = 0.5)$ are used for objective conversion and 20 starting points are randomly selected. All points converge to local optima and are sorted by domination. Only non-dominated solutions remain. For NSGA, 20 random starting points are used and the maximum allowed number of generations is 10. Converged solutions are automatically non-dominated. From the results, it can be seen that SQP tends to find better points locally but fails to explore certain regions of the objective space. Some of the solutions generated by NSGA are not locally optimal and dominated by solutions from SQP. But NSGA covers more of the objective space and provides more choices to select for a better distribution. These findings match our analysis on both optimization techniques in Section 5.3. All solutions from SQP and NSGA are combined and sorted by domination. Only non-dominated ones are left and presented in Figure 5.5. Compared with solutions generated by SQP or NSGA alone, combined results are better in objective values and cover more of the objective space.

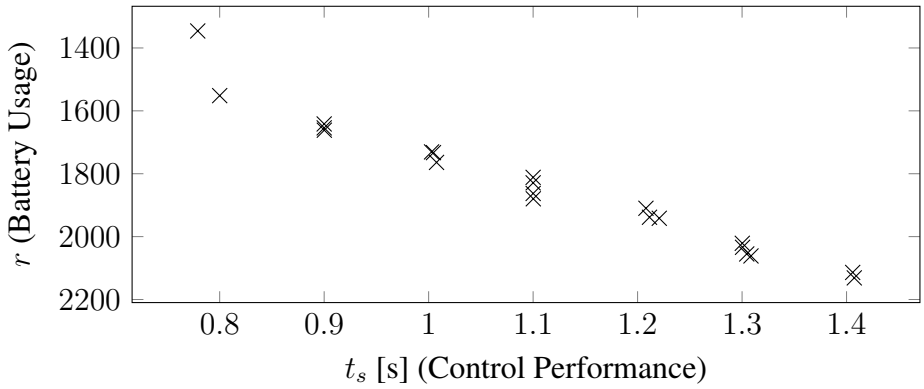

Figure 5.5: Combined solutions found by both SQP and NSGA. Dominated ones are removed.

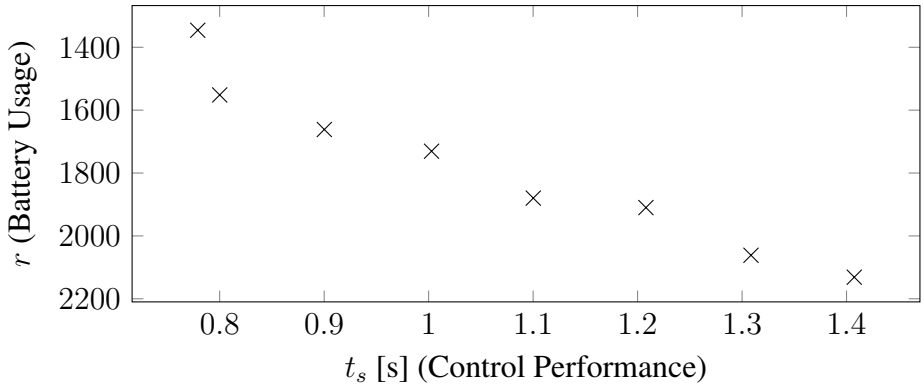

Figure 5.6: Final non-dominated solutions for the battery-aware embedded control system design. The distribution quality improvement algorithm is applied.

From the distribution point of view, some of the points in Figure 5.5 are very close to others and thus redundant. There is still space to improve the distribution quality. Based on the method explained in Section 5.3, the two crowding distances are computed to be $\bar{q}^1 = 0.065$ and $\bar{q}^2 = 70.925$. Then Algorithm 1 is taken to improve the distribution quality. In the embedded controller design for EVs, the objective of control performance $f_1 = t_s$ is safety-critical and more relevant to the vehicle performance. Therefore, $\rho_1 = 1$ and $\rho_2 = 0$ are set as the distribution quality in f_1 is more of a concern. It is noted that other pairs of importance factors for distribution quality can be used as well based on the designer's choice. Assuming that the desirable number of solutions n_d is 8, Algorithm 1 is executed and the final results are shown in Figure 5.6. It is visually obvious that the distribution quality is improved. Quantitatively, the two crowding distances become $\bar{q}^1 = 0.190$ and $\bar{q}^2 = 215.860$, respectively. Approximately, for every interval of $0.1s$ in settling time, there is one solution to choose. The trade-off between control performance Q_b and battery usage R_b can be clearly seen in Figure 5.6. If a solution point with better control performance is desired, then battery usage has to be compromised, and vice versa. The designer can then make the selection based on different cases.

The change in processor operating frequency leads to a decreased sampling frequency of the controller, for which the safety-critical design objective control performance could get de-

Table 5.1: Aging effects in settling time and battery usage with and without controller design re-optimization. Degradation is calculated with respect to objective values before aging. Negative numbers indicate improvement. Degr. stands for Degradation and re-opt. stands for re-optimization.

Point Number	1	2	3	4	5	6	7	8
t_s before aging [s]	0.779	1.100	0.900	0.800	1.003	1.407	1.309	1.208
t_s after aging w/o re-opt. [s]	0.846	1.199	0.979	0.869	1.091	1.536	1.427	1.317
Degr. in t_s w/o re-opt. [%]	8.55	8.96	8.78	8.62	8.82	9.11	9.05	9.02
t_s after aging with re-opt. [s]	0.753	0.980	0.809	0.796	0.870	1.310	1.200	1.090
Degr. in t_s with re-opt. [%]	-3.29	-10.92	-10.11	-0.56	-13.2	-6.89	-8.28	-9.77
r before aging	1346	1880	1662	1552	1731	2131	2062	1910
r after aging w/o re-opt.	1467	1822	1575	1450	1862	2245	2182	2034
Degr. in r w/o re-opt. [%]	-8.99	3.09	5.23	6.54	-7.57	-5.34	-5.83	-6.48
r after aging with re-opt.	1336	1787	1539	1543	1683	2073	2008	1883
Degr. in r with re-opt. [%]	0.75	4.92	7.38	0.55	2.76	2.72	2.59	1.42

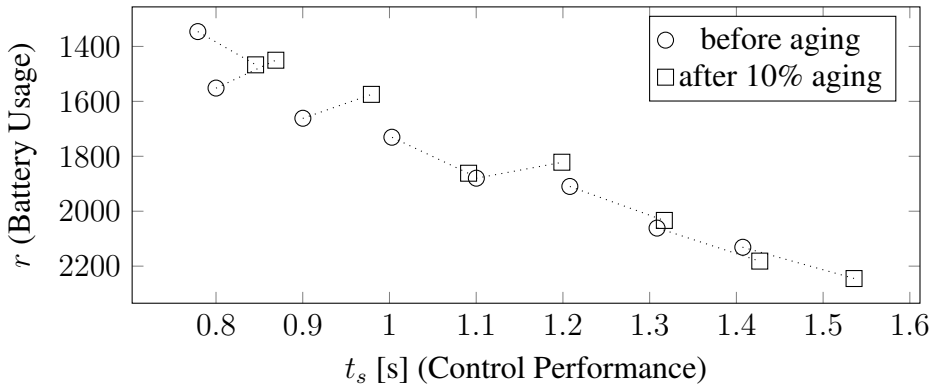

Figure 5.7: Aging effects in both settling time and battery usage. Two points for the same design case before and after aging are connected with a dotted line. Control performance gets deteriorated for all eight solutions generated in battery-aware controller design if the controller design does not change. Exact values can be found in Table 5.1.

graded. In the following experiments, it is assumed that the sampling period is increased by 10%, as discussed in Section 5.2.2 from $0.1s$ to $0.11s$. In Figure 5.7 and Table 5.1, the aging effects in settling time and battery usage are shown for the eight controller designs generated for battery-aware controller design in Figure 5.6, assuming that the controller eigenvalues do not change. The two points with the same controller design before and after aging are connected with a dotted line. It is found that the battery usage and settling time get changed when the processor ages and the control system sampling frequency decreases. Compared to R_b, the battery usage R_b' gets degraded in some cases and improved in other cases. Compared to Q_b, the safety-critical design objective QoC Q_b' gets deteriorated for every solution. In one case, even the hard constraint $t_s \leq 1.5s$ is violated.

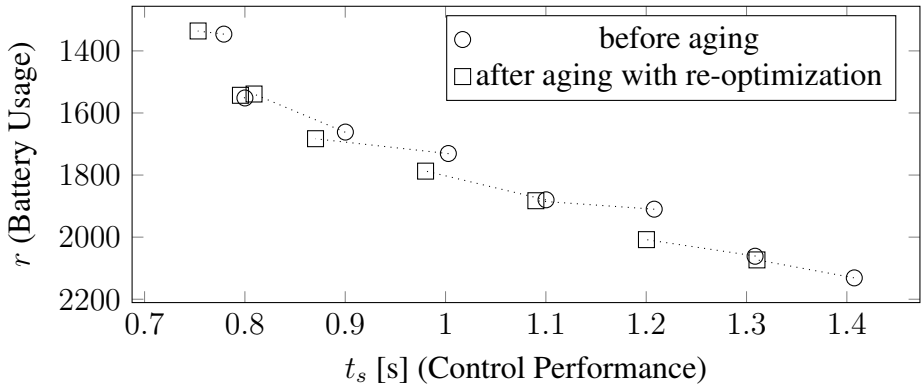

Figure 5.8: Aging effects mitigation with controller design re-optimization. Two points for the same design case before aging and after aging with controller re-optimization are connected with a dotted line. For all eight solutions generated in battery-aware controller design, control performance is kept not deteriorated with a slight compromise on battery usage.

As discussed in Section 5.3, it is necessary to ensure that the control performance does not get worse even when the processor ages, i.e., $Q'_b \geq Q_b$. With controller design re-optimization, results are shown in Figure 5.8 and Table 5.1. The two points connected with a dotted line represent the same design case before aging and after aging with controller re-optimization. One can see that it is possible to find new controller design eigenvalues that ensure no degradation of control performance. The price paid is that the battery usage has slightly deteriorated for all design cases. It is noted that which sampling period is optimal for embedded systems control in EVs is a challenging question to answer motivating future work, since both objectives need to be considered, together with constraints and aging effects. The framework presented in this paper guarantees no compromise in control performance due to processor aging, given any original sampling period. Besides, controller design re-optimization is an offline task. A look-up table with controller eigenvalues corresponding to processor operating frequency is stored in the controller memory. There is a small safety margin to accommodate slight processor aging. When the extend of aging exceeds this safety margin, on-chip monitors realize that it is necessary to reduce the processor frequency. The frequency generator makes the change, and the controller adjusts its eigenvalues accordingly. With a larger look-up table, a smaller safety margin can be set and thus the resource utilization can be improved.

5.5 Remarks

In this chapter, a design optimization framework for embedded control systems in EVs — explicitly studying various trade-offs among (i) control performance (ii) battery usage and (iii) processor aging — is presented. The results reported in battery-aware controller design show that a higher control performance implies a lower battery usage and vice versa. This framework essentially offers the possibility to achieve trade-offs from the Pareto front. Furthermore, when the processor ages, the control performance gets deteriorated. A modified version of the optimization framework is used in battery- and aging-aware controller design and experi-

mented considering 10% reduction in processing speed due to aging. Results show that control performance deterioration due to aging can be mitigated by a little compromise on battery usage. Thus, the presented framework allows to preserve the safety-critical guarantees in spite of processor aging at the cost of a slightly lower battery usage.

In this chapter, electric motor control, which is a safety-critical application, is considered. Therefore, the control performance cannot be deteriorated as the processor ages. It is common that multiple applications run on one ECU. Some of the applications can be comfort-related, such as suspension control. Besides the controller re-optimization method presented in this chapter that keeps control performances of safety-critical applications not degraded with a decrease on the battery usage along the processor aging, control performances of these comfort-related applications could also be compromised to fulfill the same purpose.

Besides battery rate capacity effect considered in this chapter, battery aging also has significant impact on the battery usage and can be influenced by controller design. In order to precisely take battery aging effect into account, battery tests with discharging current profiles under real-world automotive scenarios are required. With these test results, it can be known which controller design better mitigates the battery aging. This makes the Pareto front exploring the trade-off between control performance and battery usage practically more useful.

Building a hybrid electrical energy storage (EES) system consisting of a supercapacitor and a battery could also mitigate battery aging, which is mainly caused by large discharging currents that can be supplied by the supercapacitor [SKS+11], since the life cycles of supercapacitors are considerably larger than those of lithium-ion batteries. There have been works studying EES systems with multiple elements [PCKW10] and how to dimension them. However, none of the papers have considered battery aging in the CPS design context, which can be explored in the future work.

6 Conclusion

The automotive industry is cost-sensitive. In the past decade, the complexity of automotive systems has increased significantly, with modern automobiles equipped with more and more functions, which makes it much more challenging for the implementation to keep cost-efficient. Along the research direction of system dimensioning, this thesis studies control systems taking memory, computation, and energy resources into consideration. The goal is to achieve better control performance with a given implementation platform, or equivalently, reduce the system cost while satisfying all the requirements.

When the memory is shared by multiple applications running on one processor and executed in a round-robin fashion, the cache is frequently refreshed, resulting in long WCET. A sampling order that consecutively executes every application multiple times is proposed in Chapter 3 in order to increase the cache reuse and shorten the WCET. The control performance is improved by the framework integrating memory analysis and non-uniform controller design. The optimal sampling order is computed to fully realize the potential of a memory-aware sampling order.

There have been no previous works in memory-aware embedded control systems design. The efforts in Chapter 3 open up a number of possible topics that can be pursued in the future. For example, in a scratchpad-based multi-core architecture, it is worth of investigating how the scratchpad memory should be partitioned among applications, so that the overall control performances can be maximized. Conventionally, the WCET is used as the optimization objective. However, this WCET-driven solution might not be optimal for control performance. Therefore, new techniques need to be developed for the scratchpad partitioning in the control context.

In the cache-based architecture, besides the memory-aware sampling order, another possible way to improve the control performance is cache locking. It is interesting to know which way behaves better. This could depend on the ratio between the program size and the cache size. The technique to determine which part of the code should be allocated into the cache is required.

Memory and communication resources can be taken account of together in automotive control systems design. For instance, the choices of sampling periods can be restricted by the

communication cycles in the FlexRay bus. Without considering the memory, there is a trade-off between control performance and bus load. That is, to achieve better control performance, the bus load is high. The memory adds another dimension to this problem. To realize a smaller sampling period, the cache or scratchpad size needs to be larger as well. This is a more thorough exploration of the trade-off between the performance and system dimensioning.

As a class of widely used OS in automobiles, OSEK/VDX-compliant OS offer a limited set of periods. A multirate schedule is able to realize the required control performance requirement, while reducing the processor load, thereby enabling more applications to be integrated into an ECU. Such OS-aware automotive control systems design presented in Chapter 4 has not been studied before.

With different memory architectures, sizes, and uses, the WCET of a control program varies. The trade-off between memory resources and computation resources is an interesting direction to pursue in the future, especially for a multi-core ECU. In general, this thesis addresses each individual type of resources of the implementation platform and develops relevant techniques. As all the necessary techniques are ready, all types of resources can be taken into account together, resulting in the holistic resource-aware automotive control systems design.

While the implementation platform with computation, memory and communication resources exists in every modern vehicle, the energy resource is of particular interest for EVs, due to the relatively slow progress in the battery technology development. Since there has been no significant improvement in the battery material and mechanism in the past decade, it is important for control theorists and automotive engineers to make the best use of the limited capability provided by the battery. In particular, it is desirable to minimize the energy consumption of each control task invocation and generate a friendly discharging current profile (low average and peak current). Along this line, a trade-off between the control performance and the battery usage is explored in Chapter 5.

The difference between the energy resource and other resources on the implementation platform is that the trade-off involving the system dimensioning for the processor, memory and communication bus occurs during the design phase. For instance, once the vehicle is on the road, it is rare and difficult to change the size of the memory. However, the battery usage can be adjusted in the real-time, which only requires change of controller poles. This offers more flexibility to handle the negative effects along the use of vehicles, such as processor aging. As the processor ages, the processor operating frequency is decreased and the sampling period is prolonged. Re-optimized controller is able to maintain the control performance with a compromise in the battery usage.

This thesis discusses both linear and non-linear control algorithms, involving techniques of offline contoller optimization. Online optimal control (e.g., selecting the optimal sampling period or poles depending on the real-time system states) is more robust. However, the challenge is mainly on how to avoid heavy computation. One potential direction is to make use of deep learning. For instance, a neural network can be trained offline. The supervised learning can be conducted by MPC. The reinforcement learning can be conducted in experiments. The benefit of deep learning is that once the training is done, the online effort is minimal — similar to looking up a table. All sorts of disturbance can be well handled.

A number of optimization techniques to solve non-convex problems are presented in this thesis. Due to the complexity, heuristics are often deployed. It is barely true that one heuristic

is able to take many real-life problems — with different dimensions, domain shapes and constraints. Therefore, customized techniques developed based on the understanding of the specific problem are necessary to achieve excellent optimality and efficiency.

Automotive systems have always been cost-sensitive and are expected to continue to be so. Conventional design techniques do not consider resources due to the small number of simple control applications implemented on board. As the automotive industry is entering the smart era with more and more functions, resource-aware automotive control systems design will play a key role, aiming to achieve better performance and efficient dimensioning simultaneously. This thesis makes some first theoretic efforts along this research direction, which are validated by real-life control systems. More possibilities are opened up and discussed that can be pursued by researchers and engineers in the automotive, embedded system and control domains.

Appendix

A Electronic Wedge Brake

This section presents a brake-by-wire solution called EWB, developed by Siemens [FRBW$^+$07]. The EWB uses the self-reinforcing wedge principle, thereby making the required actuation force and power much lower than the conventional electromechanical brakes. Experiments show that the EWB outperforms hydraulic systems in the braking distance, especially on surfaces with low friction. The simplified model is illustrated in Figure A.1 and the state-space model is derived as follows.

Along the direction parallel to the braking disc,

$$m\dot{v} = F_M \cos\alpha + F_B - F_R \sin\alpha = F_M \cos\alpha + \mu F_N - F_R \sin\alpha, \tag{A.1}$$

where m is the wedge mass, v is the wedge velocity along the x direction, $\dot{v}$ is the wedge acceleration along x, α is the wedge angle, F_M is the force applied by the motor, F_N is force between the wedge and the disc, and F_R is the force between the wedge and the calliper.

Along the perpendicular direction to x,

$$m\dot{v}\tan\alpha = F_M \sin\alpha + F_R \cos\alpha - F_N. \tag{A.2}$$

Considering the spring connecting the callipers,

$$F_N = K_{cal} x \tan\alpha, \tag{A.3}$$

where x is the wedge position, assuming that F_N is zero when x is zero. Combining (A.1) and (A.2),

$$\dot{v} = \frac{1}{m(1+\tan^2\alpha)}\left(\frac{1}{\cos\alpha}F_M + (\mu - \tan\alpha)\tan\alpha K_{cal} x\right), \tag{A.4}$$

To relate the motor angular position θ_m and ω_m to the force F_M, the connection system is modelled by a stiffness K_{axial} and a viscous damping D_{axial}. Therefore,

$$F_M = K_{axial}\left(\theta_m \frac{L}{2\pi\cos\alpha} - x\right) + D_{axial}\left(\omega_m \frac{L}{2\pi\cos\alpha} - v\right) \tag{A.5}$$

The simplified roller screw has a lead L and a constant torque-to-force efficiency η, where $0 < \eta < 1$. The torque from the motor is then

$$M_M = \frac{L F_M}{2\pi\eta} \tag{A.6}$$

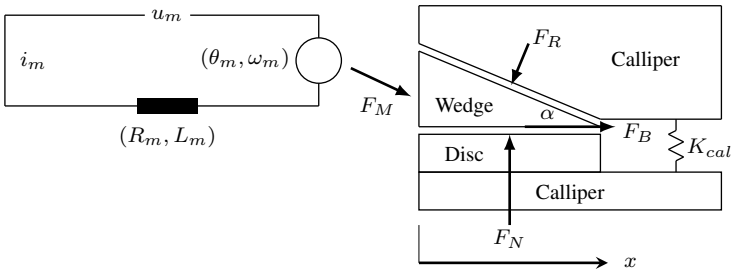

Figure A.1: A simplified model of the electronic wedge braking system

Table A.1: Paramters of the EWB

μ	α	K_{cal}	K_{axial}	m	D_{axial}	L
0.4	0.2π	$4 \times 10^4 N/m$	$8 \times 10^4 N/m$	$0.2kg$	$300Ns/m$	$0.2m$
η	J_m	D_m	K_m	L_m	R_m	u_m
0.6	$10^{-3} kgm^2$	$0.3Nms$	$0.5Nm/A$	$10^{-3}H$	1Ω	up to $12V$

The motor used to drive the wedge is a brushless single-phase DC motor driven by a voltage u_m and modelled with the resistance R_m, inductance L_m, viscous damping D_m as friction, and torque constant k_m as follows,

$$\dot{i_m} = -\frac{R_m}{L_m} i_m - \frac{k_m}{L_m} \omega_m + \frac{1}{L_m} u_m, \tag{A.7}$$

where i_m is the motor current and $\dot{i_m}$ is the first derivative of the motor current with respect to time. Assuming J_m to be the moment of inertia of the rotor, there is

$$J_m \dot{\omega_m} = k_m i_m - M_M - D_m \omega_m. \tag{A.8}$$

Combining (A.4), (A.5), (A.6), (A.7), and (A.8), the state-space model of the EWB can be formulated as in (2.1), where

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ \frac{(\mu - \tan\alpha)\sin\alpha K_{cal} - K_{axial}}{m \sec^2\alpha \cos\alpha} & -\frac{D_{axial}}{m \sec^2\alpha \cos\alpha} & \frac{LK_{axial}}{2\pi m \sec^2\alpha \cos^2\alpha} & \frac{LD_{axial}}{2\pi m \sec^2\alpha \cos^2\alpha} & 0 \\ 0 & 0 & 0 & 1 & 0 \\ \frac{LK_{axial}}{2\pi J_m \eta} & \frac{LD_{axial}}{2\pi J_m \eta} & -\frac{L^2 K_{axial}}{4\pi^2 J_m \cos\alpha \eta} & -\frac{L^2 D_{axial}}{4\pi^2 J_m \cos\alpha \eta} - \frac{D_m}{J_m} & \frac{k_m}{J_m} \\ 0 & 0 & 0 & -\frac{k_m}{L_m} & -\frac{R_m}{L_m} \end{bmatrix}, \tag{A.9}$$

and

$$B = [0, 0, 0, 0, \frac{1}{L_m}]^T, \quad C = [K_{cal} \tan\alpha, 0, 0, 0, 0]. \tag{A.10}$$

The system state is $[x, v, \theta_m, \omega_m, i_m]$. The control input is u_m. The system output is F_N. Parameters are listed in Table A.1.

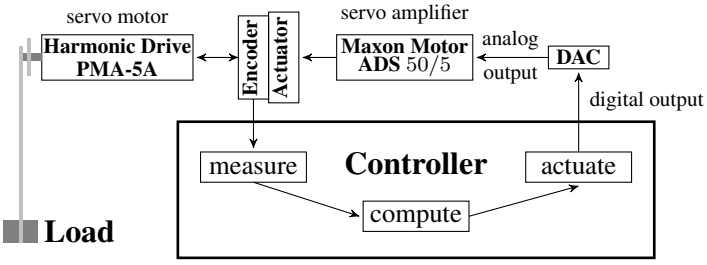

Figure B.2: Experimental setup of servo motor position control

B Servo Motor Position Control

Position control of a servo motor can be found, e.g., in a steer-by-wire system [Yih05]. As shown in Figure B.2, the shaft of the servo motor (Harmonic Drive) [Dri] is attached to a rigid stick with 300g of weight at the end. The position of the motor shaft is measured by digital quadrature encoders attached to the motor shaft. The motor provides a desired amount of torque (computed by the control program) using a Digital-to-Analog Converter (DAC) via a servo amplifier (Maxon Motor) [Mot11].

The above system of servo motor position control has two states: $x_1(t)$, the angular position and $x_2(t)$, the angular velocity of the shaft. Within the control loop, the measure operation reads the quadrature encoder to obtain $x_1(t)$ and $x_2(t)$; the compute operation executes the control program and computes the input current $u(t)$ for the motor and the actuate operation is performed using the DAC. The control objective is to keep the rigid load at a certain angular position. Since the output is the angular position $x_1(t)$, $y(t) = x_1(t)$. The dynamics of the above system is represented as in (2.1) with

$$A = \begin{bmatrix} 0 & 1 \\ 37 & 7.5 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 6450 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 \end{bmatrix}. \tag{B.11}$$

The maximum current that the servo amplifier can supply to the motor is 1.5 ampere. Therefore, $u(t) \leq 1.5$A must be respected by the controller.

C Electric Motor Control

Electric motor control is a key function in EVs. As shown in Figure C.3, a DC motor running in the speed control mode is considered. The controller is supposed to operate the motor at various speeds according to driver input or environmental requirements. V is the DC voltage provided by the battery pack. R and L are resistance and inductance in the armature circuit. e is the Back Electromotive Force (EMF) from the motor. The Insulated Gate Bipolar Transistor (IGBT) works as a switch controlled by Pulse-Width Modulation (PWM) signals at the gate. When the switch is on, V is applied to the armature circuit. When the switch is off, the diode flows out remaining current in the motor and thus the applied voltage is equivalent to zero.

Periodic PWM signals are shown in Figure C.4, where the duty cycle c is calculated as

$$c = \frac{t_{on}}{t_{period}}, \tag{C.12}$$

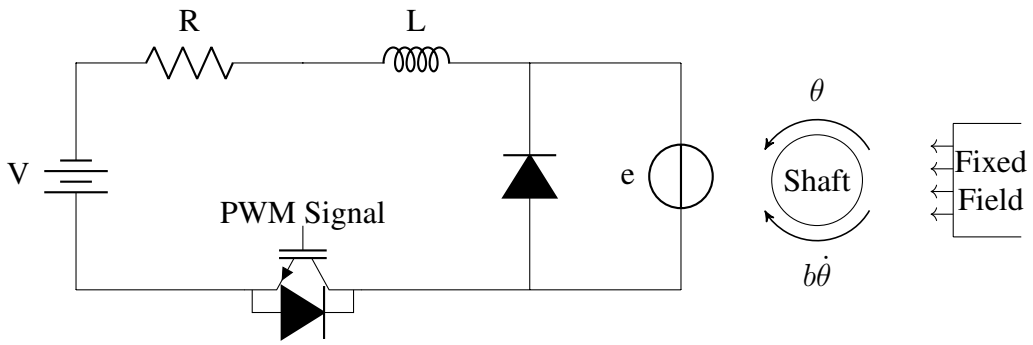

Figure C.3: Modelling DC motor with the armature circuit powered up by a battery pack

and the effective voltage applied in the armature circuit is

$$V_{eff} = cV. \tag{C.13}$$

It can be clearly seen that V_{eff} is adjustable by controlling PWM signals.

In general, the torque T generated by a DC motor is proportional to the armature current i and the strength of the magnetic field. It is assumed that the magnetic field is constant and thus the torque is calculated as

$$T = K_t i, \tag{C.14}$$

where K_t is the motor torque constant. The angular position of the motor is denoted to be θ. The angular velocity and acceleration are then $\dot{\theta}$ and $\ddot{\theta}$, respectively. The back EMF is proportional to the angular velocity of the shaft by a constant factor K_e as follows,

$$e = K_e \dot{\theta}. \tag{C.15}$$

A viscous friction model is assumed and the friction torque is proportional to the shaft angular velocity $\dot{\theta}$ by a factor of b. Now the following governing equations can be derived based on Newton's second law and Kirchhoff's law [RKHB08],

$$J\ddot{\theta} + b\dot{\theta} = K_t i, \quad L\frac{di}{dt} + Ri = cV - K_e\dot{\theta}, \tag{C.16}$$

where J is the moment of inertia of the motor.

The state-space system modelling as in (2.1) becomes

$$\begin{aligned} \frac{d}{dt}\begin{bmatrix} \dot{\theta} \\ i \end{bmatrix} &= \begin{bmatrix} -\frac{b}{J} & \frac{K_t}{J} \\ -\frac{K_e}{L} & -\frac{R}{L} \end{bmatrix}\begin{bmatrix} \dot{\theta} \\ i \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{V}{L} \end{bmatrix} c, \\ y &= \begin{bmatrix} 1 & 0 \end{bmatrix}\begin{bmatrix} \dot{\theta} \\ i \end{bmatrix}. \end{aligned} \tag{C.17}$$

The states are the angular velocity of the motor $\dot{\theta}$ and the armature current i. The control input is the duty cycle c and the system output is $\dot{\theta}$. The control task is to make $\dot{\theta}$ approach a certain value $\dot{\theta}_{ref}$. It is clear that the constraint on the control input is

$$0 \leq c \leq 1. \tag{C.18}$$

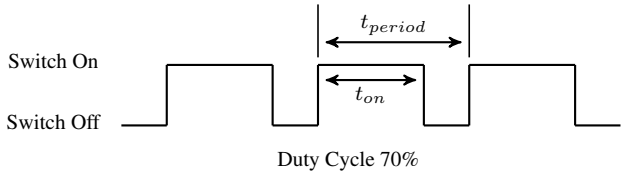

Figure C.4: Pulse-width modulation control signals in the armature circuit to adjust the DC voltage applied to the rotor

Table C.2: Parameters and constraints of the motor control system

$J\ [kgm^2]$	$b\ [Nms]$	$K_t\ [Nm/A]$	$K_e\ [V/(rad \cdot s)]$	$R\ [\Omega]$	$L\ [H]$	$V\ [V]$	$I_m^{max}\ [A]$
0.15	0.03	0.1	0.1	1	0.01	370	300

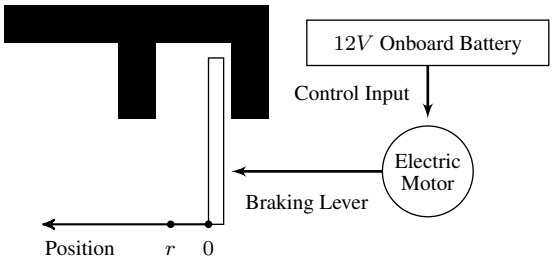

Figure D.5: A simplified model of the electro-mechanical braking system

Another constraint is on the largest current that can be sustained by wires in the motor, i.e.,

$$i \leq I_m^{max}. \tag{C.19}$$

Example parameters of the entire motor control system are shown in Table C.2. The operating voltage V of the battery pack is 370 V. Constraints of the control system design are also shown in Table C.2. The maximum allowed current in the motor I_m^{max} is 300 A, for the sake of thermal safety.

D Electro-Mechanical Braking System

The simplified model of an EMB system from Bosch is shown in Figure D.5. When the EMB is active, the braking lever should reach a reference position r, which is at the braking disc, within the desired settling time t_s^0. This is the position mode. The requirement on the settling time ensures the reactiveness of the system. After that, a certain force is applied in the force mode. The electric motor mobilizing the braking lever is powered by the onboard battery, which has a voltage of $12V$. The position mode of the EMB is of interest in several scenarios: braking, disk wiping, and pre-crash preparations. The system dynamics can be modeled as (D.20) with five system states — motor current, motor angular velocity, motor angular position, lever velocity,

Table D.3: EMB system requirements

Settling time requirement t_s^0	Input saturation U_{max}	Reference r	WCET E^{wc}
$150ms$	$12V$	$2mm$	$0.7ms$

and lever position.

$$A = \begin{bmatrix} -520 & -220 & 0 & 0 & 0 \\ 220 & -500 & -999994 & 0 & 2 \times 10^8 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 66667 & -0.1667 & -1.3333 \times 10^7 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1000 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{D.20}$$

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

Example requirements are summarized in Table D.3.

Bibliography

[Age12] The U.S. Environmental Protection Agency. EPA and NHTSA Set Standards to Reduce Greenhouse Gases and Improve Fuel Economy for Model Years 2017-2025 Cars and Light Trucks , 2012.

[AGL15] S. Al-Areqi, D. Görges, and S. Liu. Event-based Control and Scheduling Codesign: Stochastic and Robust Approaches. *IEEE Transactions on Automatic Control*, 60(5):1291–1303, 2015.

[AGS+13] S. Andalam, A. Girault, R. Sinha, P. Roop, and J. Reineke. Precise Timing Analysis for Direct-Mapped Caches. In *DAC*, 2013.

[AM09] K. Astrom and R. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2009.

[AT09] A. Anta and P. Tabuada. On the Benefits of Relaxing the Periodicity Assumption for Networked Control Systems over CAN. In *RTSS*, pages 3–12, 2009.

[AU98] J. Ackermann and V. Utkin. Sliding Mode Control Design based on Ackermann's Formula. *IEEE Transactions on Automatic Control*, 43(2):234–237, 1998.

[BB14] E. Bini and G. Buttazzo. The Optimal Sampling Pattern for Linear Control Systems. *IEEE Transactions on Automatic Control*, 59(1):78–90, 2014.

[BBC+07] S. Brennan, J. Buckland, U. Christen, I. Haskara, and I. Kolmanovsky. Editorial: Special Issue on Control Applications in Automotive Engineering. *IEEE Transactions on Control Systems Technology*, 15(3):403–405, 2007.

[BC08] E. Bini and A. Cervin. Delay-Aware Period Assignment in Control Systems. In *RTSS*, pages 291–300, 2008.

[BK08] A. Bhave and B. Krogh. Performance Bounds on State-Feedback Controller with Network Delay. In *CDC*, 2008.

[Bos91] Bosch. CAN Specification Version 2.0, 1991.

[BTW+09] K. Bowman, J. Tschanz, C. Wilkerson, S. Lu, T. Karnik, V. De, and S. Borkar. Circuit Techniques for Dynamic Variation Tolerance. In *DAC*, pages 4–7, 2009.

[BW08] K. Batcher and R. Walker. Dynamic Round-Robin Task Scheduling to Reduce Cache Misses for Embedded Systems. In *DATE*, 2008.

[CBMC11] A. Cervin, M. Belasco, P. Marti, and A. Camacho. Optimal Online Sampling Period Assignment: Theory and Experiments. *IEEE Transactions on Control Systems Technology*, 19(4):902–910, 2011.

[CC16] W. Chang and S. Chakraborty. Resource-Aware Automotive Control Systems Design: A Cyber-Physical Systems Approach. *Foundations and Trends® in Electronic Design Automation*, 10:249–369, 2016.

[CCK+12] W. Cho, J. Choi, C. Kim, S. Choi, and K. Yi. Unified Chassis Control for the Improvement of Agility, Maneuverability, and Lateral Stability. *IEEE Transactions on Vehicular Technology*, 61(3):1008–1020, 2012.

[CFC+16] S. Chakraborty, M. Al Faruque, W. Chang, D. Goswami, M. Wolf, and Q. Zhu. Automotive Cyber-Physical Systems: A Tutorial Introduction. *IEEE Design & Test*, 33:92–108, 2016.

[CGC+17] W. Chang, D. Goswami, S. Chakraborty, J. Xue, L. Ju, and S. Andalam. Memory-Aware Embedded Control Systems Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36:586–599, 2017.

[CGCHng] W. Chang, D. Gowami, S. Chakraborty, and A. Hamann. OS-Aware Automotive Controller Design Using Non-Uniform Sampling. *ACM Transactions on Cyber-Physical Systems*, forthcoming.

[CLP+14] S. Chang, B. Lee, Y. Park, H. Cho, and M. Kim. Integrated Chassis Control for Improving On-Center Handling Behavior. *SAE International Journal of Passenger Cars — Mechanical Systems*, 7(3):1002–1008, 2014.

[CMRT09] S. Chakraborty, T. Mitra, A. Roychoudhury, and L. Thiele. Cache-Aware Timing Analysis of Streaming Applications. *Real-Time Systems*, 41(1):52–85, 2009.

[CMV+06] R. Castane, P. Marti, M. Velasco, A. Cervin, and D. Henriksson. Resource Management for Control Tasks based on the Transient Dynamics of Closed-Loop Systems. In *ECRTS*, 2006.

[Con05] OSEK/VDX Consortium. OSEK/VDX Operating System Specification Version 2.2.3, 2005.

[CPG+14] W. Chang, A. Pröbstl, D. Goswami, M. Zamani, and S. Chakraborty. Battery- and Aging-Aware Embedded Control Systems for Electric Vehicles. In *RTSS*, 2014.

[CPG+15] W. Chang, A. Pröbstl, D. Goswami, M. Zamani, and S. Chakraborty. Reliable CPS Design for Mitigating Semiconductor and Battery Aging in Electric Vehicles. In *CPSNA*, 2015.

[CSJ07] C. Chiang, A. Stefanopoulou, and M. Jankovic. Nonlinear Observer-based Control of Load Transitions in Homogeneous Charge Compression Ignition Engines. *IEEE Transactions on Control Systems Technology*, 15(3):438–448, 2007.

[DPAM02] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[Dri] Harmonic Drive. Produktbeschreibung PMA.

[DS06] D. Doerffel and S. Sharkh. A Critical Review of Using the Peukert Equation for Determining the Remaining Capacity of Lead-Acid and Lithium-Ion Batteries. *Journal of Power Sources*, 165(2):395–400, 2006.

[Fei03] P. Feiler. Real-Time Application Development with OSEK: A Review of the OSEK Standards. Technical report, Carnegie Mellon University, 2003.

[Fle05] FlexRay Consortium. The FlexRay Communications System Specifications Version 2.1, 2005.

[FRBW+07] J. Fox, R. Roberts, C. Baier-Welt, L. Ho, L. Lacraru, and B. Gombert. Modeling and Control of a Single Motor Electronic Wedge Brake. Technical report, SAE, 2007.

[GFB11] L. Greco, D. Fontanelli, and A. Bicchi. Design and Stability Analysis for Anytime Control via Stochastic Scheduling. *IEEE Transactions on Automatic Control*, 56(3):571–585, 2011.

[Hag93] S. Hageman. Simple PSpice Models Let You Simulate Common Battery Types. *Electronic Design News*, 38:117–129, 1993.

[HJYK15] H. Heo, E. Joa, K. Yi, and K. Kim. Integrated Chassis Control for Enhancement of High Speed Cornering Performance. *SAE International Journal of Commercial Vehicles*, 8(1):102–109, 2015.

[HR88] Y. Halevi and A. Ray. Integrated Communication and Control Systems: Part I–Analysis. *Journal of Dynamic Systems, Measurement, and Control*, 110:367–373, 1988.

[HSJ08] E. Henriksson, H. Sandberg, and K. Johansson. Predictive Compensation for Communication Outages in Networked Control Systems. In *CDC*, pages 2063–2068, 2008.

[Inf09] Infineon. XC2300B-Series 16/32-bit μC for Automotive Safety, 2009.

[JJ13] A. Jordehi and J. Jasni. Parameter Selection in Particle Swarm Optimization: A Survey. *Journal of Experimental and Theoretical Artificial Intelligence*, 25(4):527–542, 2013.

[KFM11] J. Kleinsorge, H. Falk, and P. Marwedel. A Synergetic Approach to Accurate Analysis of Cache-Related Preemption Delay. In *EMSOFT*, 2011.

[KQ11] T. Kim and W. Qiao. A Hybrid Battery Model Capable of Capturing Dynamic Circuit Characteristics and Nonlinear Capacity Effects. *IEEE Transactions on Energy Conversion*, 26(4):1172–1180, 2011.

[Kri12] K. Kritayakirana. *Autonomous Vehicle Control at the Limits of Handling*. PhD thesis, Stanford University, 2012.

[KSD07] A. Kumar, D. Sharma, and K. Deb. A Hybrid Multi-Objective Optimization Procedure Using PCX based NSGA-II and Sequential Quadratic Programming. In *CEC*, pages 3011–3018, 2007.

[LA09] H. Lin and P. Antsaklis. Stability and Stabilizability of Switched Linear Systems: A Survey of Recent Results. *IEEE Transactions on Automatic Control*, 54(2):308–322, 2009.

[LBS10] D. Lorenz, M. Barke, and U. Schlichtmann. Aging Analysis at Gate and Macro Cell Level. In *ICCAD*, 2010.

[LDF+11] C. Lefurgy, A. Drake, M. Floyd, M. Alle, B. Brock, J. Tierno, and J. Carter. Active Management of Timing Guardband to Save Energy in POWER7. In *MICRO*, pages 1–11, 2011.

[LLB+12] O. Ljungkrantz, H. Lonn, H. Blom, C. Ekelin, and D. Karlsson. Modelling of Safety-Related Timing Constraints for Automotive Embedded Systems. In *SAFECOMP*, 2012.

[LW13] E. Lavretsky and K. Wise. *Robust and Adaptive Control*. Springer London, 2013.

[LWH+14] Q. Leng, Y. Wei, S. Han, A. Mok, W. Zhang, and M. Tomizuka. Improving Control Performance by Minimizing Jitter in RT-WiFi Networks. In *RTSS*, 2014.

[MDG07] B. Müller, J. Deutscher, and S. Grodde. Continuous Curvature Trajectory Design and Feedforward Control for Parking a Car. *IEEE Transactions on Control Systems Technology*, 15(3):541–553, 2007.

[Meh92] S. Mehrotra. On the Implementation of a Primal-Dual Interior Point Method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.

[MFFR02] P. Marti, J. Fuertes, G. Fohler, and K. Ramamritham. Improving Quality-of-Control Using Flexible Timing Constraints: Metric and Scheduling. In *RTSS*, 2002.

[MIMG08] R. Mishra, D. Ioannou, S. Mitra, and R. Gauthier. Effect of Floating-Body and Stress Bias on NBTI and HCI on 65-nm SOI pMOSFETs. *IEEE Electron Device Letters*, 29(3):262–264, 2008.

[MKB+12] A. Masrur, P. Kindt, M. Becker, S. Chakraborty, V. Kleeberger, M. Barke, and U. Schlichtmann. Schedulability Analysis for Processors with Aging-Aware Autonomic Frequency Scaling. In *RTCSA*, pages 11–20, 2012.

[Mot11] Maxon Motor. Product Specifications, 2011.

[NES11] A. Nickabadi, M. Ebadzadeh, and R. Safabakhsh. A Novel Particle Swarm Optimization Algorithm with Adaptive Inertia Weight. *Applied Soft Computing*, 11(4):3658–3670, 2011.

[Ng02] K. Ng. *A Continuation Approach for Solving Nonlinear Optimization Problems with Discrete Variables*. PhD thesis, Stanford University, 2002.

[NHTSA13] U.S. Department of Transportation National Highway Traffic Safety Administration. Preliminary Statement of Policy Concerning Automated Vehicles, 2013.

[Nil98] J. Nilsson. *Real-Time Control Systems with Delays*. PhD thesis, Lund Institute of Technology, 1998.

[NMR03] H. Negi, T. Mitra, and A. Roychoudhury. Accurate Estimation of Cache-Related Preemption Delay. In *CODES*, 2003.

[NW06] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, 2006.

[OdR07] P. Ortner and L. del Re. Predictive Control of a Diesel Engine Air Path. *IEEE Transactions on Control Systems Technology*, 15(3):449–456, 2007.

[PA12] J. Park and J. Abraham. A Fast, Accurate and Simple Critical Path Monitor for Improving Energy-Delay Product in DVS Systems. In *ISLPED*, pages 391–396, 2012.

[PCKW10] M. Pedram, N. Chang, Y. Kim, and Y. Wang. Hybrid Electrical Energy Storage Systems. In *ISLPED*, pages 363–368, 2010.

[Ped10] M. Pedersen. Good Parameters for Particle Swarm Optimization. Technical report, Hvass Laboratories, 2010.

[RHC06] S. Robertz, D. Henriksson, and A. Cervin. Memory-Aware Feedback Scheduling of Control Tasks. In *ETFA*, 2006.

[RKHB08] M. Ruderman, J. Krettek, F. Hoffmann, and T. Bertram. Optimal State Space Control of DC Motor. In *IFAC*, 2008.

[RM09] J. Rawlings and D. Mayne. *Model Predictive Control: Theory and Design*. Nob Hill Publishing, 2009.

[Rob14] D. Robinette. A DFSS Approach to Determine Automatic Transmission Gearing Content for Powertrain-Vehicle System Integration. *SAE International Journal of Passenger Cars — Mechanical Systems*, 7(3):1138–1154, 2014.

[RV01] D. Rakhmatov and S. Vrudhula. An Analytical High-Level Battery Model for Use in Energy Management of Portable Electronic Systems. In *ICCAD*, pages 488–493, 2001.

[RZC+16] D. Roy, L. Zhang, W. Chang, D. Goswami, and S. Chakraborty. Multi-Objective Co-Optimization of FlexRay-based Distributed Control Systems. In *RTAS*, 2016.

[SB03] D. Schroder and J. Babcock. Negative Bias Temperature Instability: Road to Cross in Deep Submicron Silicon Semiconductor Manufacturing. *Journal of Applied Physics*, 94(1):1–18, 2003.

[SCEP09] S. Samii, A. Cervin, P. Eles, and Z. Peng. Integrated Scheduling and Synthesis of Control Applications on Distributed Embedded Systems. In *DATE*, pages 57–62, 2009.

[SD94] N. Srinivas and K. Deb. Multiobjective Optimization Using Nondominated Sorting Genetic Algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[SEP+10] S. Samii, P. Eles, Z. Peng, P. Tabuada, and A. Cervin. Dynamic Scheduling and Control-Quality Optimization of Self-Triggered Control Applications. In *RTSS*, 2010.

[SFX+15] G. Sun, T. Feng, J. Xu, M. Li, and T. Lim. Modified FxLMS Algorithm with Equalized Convergence Speed for Active Control of Powertrain Noise. *SAE International Journal of Passenger Cars — Mechanical Systems*, 8(3):868–872, 2015.

[SKS+11] D. Shin, Y. Kim, J. Seo, N. Chang, Y. Wang, and M. Pedram. Battery-Supercapacitor Hybrid System for High-Rate Pulse Load Applications. In *DATE*, 2011.

[SM09] D. Sedighizadeh and E. Masehian. Particle Swarm Optimization Methods, Taxonomy and Applications. *International Journal of Computer Theory and Engineering*, 1(4):486–502, 2009.

[Sni09] J. Snider. Automatic Steering Methods for Autonomous Automobile Path Tracking. Technical report, Robotics Institute, Carnegie Mellon University, 2009.

[Son] Sony. Lithium Ion Rechargeable batteries Technical Handbook.

[SRW07] K. Smith, C. Rahn, and C. Wang. Control Oriented 1D Electrochemical Model of Lithium Ion Battery. *Energy Conversion and Management*, 48(9):2565–2578, 2007.

[VdNMP07] J. Villagra, B. d'Andrea Novel, H. Mounier, and M. Pengov. Flatness-based Vehicle Steering Control Strategy with SDRE Feedback Gains Tuned via a Sensitivity Approach. *IEEE Transactions on Control Systems Technology*, 15(3):554–565, 2007.

[Wea08] R. Wilhelm and et al. The Worst-Case Execution-Time Problem — Overview of Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems*, 7(3):1–53, 2008.

[WGR+09] R. Wilhelm, D. Grund, J. Reineke, M. Schlickling, M. Pister, and C. Ferdinand. Memory Hierarchies, Pipelines, and Buses for Future Architectures in Time-Critical Embedded Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(7):966–978, 2009.

[WYB99] G. Walsh, H. Ye, and L. Bushnell. Stability Analysis of Networked Control Systems. In *ACC*, pages 2876–2880, 1999.

[Yih05] P. Yih. *Steer-by-Wire: Implications for Veicle Handling and Safety*. PhD thesis, Stanford University, 2005.

[YS05] C. Yang and D. Simon. A New Particle Swarm Optimization Technique. In *ICSE*, 2005.

[ZBP01] W. Zhang, M. Branicky, and S. Phillips. Stability of Networked Control Systems. *IEEE Control Systems*, 21(1):84–99, 2001.

[ZSWM08] F. Zhang, K. Szwaykowska, W. Wolf, and V. Mooney. Task Scheduling for Control Oriented Requirements for Cyber-Physical Systems. In *RTSS*, pages 47–56, 2008.

List of Tables

List of Figures

Abbreviations

ICE Internal Combustion Engine

ABS Anti-Lock Braking System

ESC Electronic Stability Control

EGR Exhaust Gas Recirculation

MPC Model Predictive Control

DFSS Design for Six Sigma

ANC Active Noise Control

FxLMS Filtered-x Least Mean Squares

AFS Active Front Steering

4WD Four Wheel Drive

ARS Active Roll Control System

ECS Electronic Control Suspension

ACC Adaptive Cruise Control

ECU Electronic Control Unit

OS Operating System

I/O Input/Output

GHG Greenhouse Gas

EPA Environmental Protection Agency

NHTSA National Highway Traffic Safety Administration

EV Electric Vehicle

ET Event-Triggered

TT Time-Triggered

CAN Controller Area Network

TDMA Timing Division Multiple Access

WCET Worst-Case Execution Time

FCC Full Charge Capacity

EWB Electronic Wedge Brake

PSO Particle Swarm Optimization

SQP Sequential Quadratic Programming

SISO Single-Input Single-Output

ZOH Zero-Order Hold

KKT Karush-Kuhn-Tucker

NSGA Non-Dominated Sorting Genetic Algorithm

CPS Cyber-Physical System

CQLF Common Quadratic Lyapunov Function

SLF Switched Lyapunov Function

LQR Linear Quadratic Regulator

CFG Control Flow Graph

RCS Reaching Cache States

LCS Live Cache States

DAC Digital-to-Analog Converter

EMF Back Electromotive Force

IGBT Insulated Gate Bipolar Transistor

PWM Pulse-Width Modulation

EMB Electro-Mechanical Braking

HCI Hot Carrier Injection

NBTI Negative Bias Temperature Instability

CPR Critical Path Replicas

EES Electrical Energy Storage

Nomenclature

Control

- x system state
- u control input
- y system output
- A system (state) matrix
- B input matrix
- C output matrix
- h sampling period
- A_d discretized system matrix
- B_d discretized input matrix
- $\mathcal{CO}$ controllability matrix
- t_s settling time
- r reference
- J quadratic cost
- Q weight matrix of the transient state
- R weight matrix of the control input
- S weight matrix of the final state
- U_{max} input saturation
- y_{max} maximum system output
- ϕ_0 overshoot threshold
- ϕ_e steady-state error tolerance

- t_s^0 settling time requirement
- K feedback gain
- F feedforward gain

Memory

- $\mathcal{C}$ control application
- N_c number of cache lines
- CL set of cache lines
- N_m number of main memory blocks
- M set of main memory blocks
- B set of basic blocks
- $\perp$ empty cache line
- $\top$ unknown cache line
- cs cache state
- gen cache state describing the first/last executed memory block in every cache line for a basic block in LCS/RCS computation
- CS set of cache states
- t_m main memory access time
- t_c cache access time
- τ^{sa} sensor-to-actuator delay
- E^{wc} WCET of a control program
- $\bar{E}^{wc}$ effective WCET of a control program
- $\bar{E}^{g}$ guaranteed WCET reduction

Computation

- ϕ set of available periods offered by OS
- L processor utilization
- p processor
- S schedule

Battery

- r number of times the control system can reach a steady state after a disturbance occurs with a FCC battery
- L_t total duration of battery use before next charging
- Q_{nom} nominal battery capacity
- I_c^{max} maximum current of a battery cell
- n_{sp} number of sampling periods in an invocation