Title
Closing the Gap in Control System Implementations

Author
Saha, Indranil

Publication Date
2013

Peer reviewed|Thesis/dissertation

University of California
Los Angeles

Closing the Gap
in
Control System Implementations

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Indranil Saha

2013

Abstract of the Dissertation

Closing the Gap in Control System Implementations

by

Indranil Saha

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2013

Professor Jens Palsberg, Chair

A cyber-physical system tightly coordinates discrete computation and continuous control of physical resources. Most safety-critical cyber-physical systems run sophisticated control algorithms at their core. While these control algorithms have been studied thoroughly from a theoretical standpoint, their implementation on real platforms raises many issues that need to be addressed to ensure the reliability of cyber-physical systems. While control theory studies the properties of a control system based on real analysis and mostly ignores the effect of implementing the control laws in software, these effects can have a significant impact on the performance or stability of the implementation of a control law.

Consider the following mismatches between assumptions made by control theory and guarantees provided by implementation platforms. First, control theoretic methods are based on real analysis, which assume real numbers are infinite-precision. However, in software implementation, mathematical real numbers are approximated using floating-point or fixed-point arithmetic. Second, control computation time is often assumed to be negligible, which may not often be the case in the real implementations. Third, when multiple control applications run on a

shared platform, schedulability requirements of the corresponding tasks impose additional constraints on the behavior of the systems, that traditional mathematical control design algorithms do not take into account.

In this thesis, we address these three problems related to implementation of controller software for cyber-physical systems. First, we show how the stability property can be verified for a physical system under the action of controller software and how to synthesize controller software to minimize the effect of quantization error on the stability quality. Second, we show that the naive implementation of some control algorithms may be infeasible due the computation time required for the control tasks on real platforms and provide a memoization based implementation scheme that guarantees the feasibility of the implementation along with maintaining expected control performance. Third, we address the problem of scheduling control tasks from multiple control systems on a single processor and provide static and dynamic scheduler synthesis strategies to maintain stability and achieve optimal performance in the control systems. Solving these problems takes an important step towards closing the gap between control theory and the implementation of control systems.

The dissertation of Indranil Saha is approved.

Milos D. Ercegovac

Todd Millstein

Paulo Tabuada

Rupak Majumdar

Jens Palsberg, Committee Chair

University of California, Los Angeles

2013

To my parents . . .

Table of Contents

List of Figures

List of Tables

Acknowledgments

First I would like to express my sincere gratitude to my advisor Prof. Rupak Majumdar for his continual guidance and exceptional support throughout the duration of my dissertation work. He took utmost care to teach me different skills to do effective research and guided me to grow up as an independent researcher. I would also like to thank Prof. Jens Palsberg, Prof. Todd Millstein, Prof. Milos Ercegovac and Prof. Paulo Tabuada for serving as my dissertation committee members and for their insightful comments and feedback during the oral qualifying examination and the final defense presentation. The courses taught by Prof. Rupak Majumdar, Prof. Todd Millstein, Prof. Jens Palsberg, Prof. Paulo Tabuada and Prof. Lieven Vandenberghe at UCLA have been of great help to prepare myself for my dissertation research.

This dissertation is based on the research I have done with my advisor Prof. Rupak Majumdar and a number of other collaborators. I would like to thank Dr. Adolfo Anta, Eva Darulova, Prof. Viktor Kuncak, Prof. Rupak Majumdar, Prof. Paulo Tabuada and Dr. Majid Zamani for the enjoyable collaborations. Special thanks goes to Dr. Majid Zamani for many insightful discussions that helped hone my knowledge in control theory. Many of his ideas found their way in this thesis. I would also like to thank Dr. Manuel Mazo and Dr. Matthias Rungger for many helpful discussions on different topics in control theory.

I am greatly thankful to Dr. Natarajan Shankar for hosting my three internships at SRI International and for his continual guidance and encouragements during my dissertation work. I am grateful to Dr. Ashish Gehani, Dr. Shalini Ghosh, Sam Owre, Dr. John Rushby and Dr. Ashish Tiwari for three learning-filled summer internships at SRI International. I would also like to thank Dr. Jyotirmoy Deshmukh, Dr. Jim Kapinski, Koichi Ueda and Hakan Yazarel of Toyota Technical Center for their encouragement and helpful feedback on my dissertation

work. I sincerely thank my advisor Prof. Rupak Majumdar for hosting my research visits at MPI-SWS. I am also very much thankful to all administrative staffs at UCLA Computer Science department, SRI International and MPI-SWS for there exceptional support that made my stay in these institutes very comfortable and productive.

I am in debt to my professors at Kalyani Government Engineering College, India for building my basic knowledge in Electronics and Communication Engineering, and my professors at Indian Statistical Institute, Kolkata for building my basic knowledge in Computer Science. I specially thank Prof. Bhargab Bhattacharya, Prof. Bhabani P. Sinha and Prof. Susmita Sur-Kolay of Indian Statistical Institute Kolkata for encouraging me and helping me prepare for a doctoral study. Thanks are due to my colleagues at Honeywell, Bangalore, especially Subhas Kumar Ghosh, Janardan Misra, Debapriyay Mukhopadhyay and Dr. Suman Roy, for encouraging me to pursue a doctoral study. I would also like to thank Dr. S. Ramesh of General Motors R&D for his encouragement in pursuing research in the area of safety-critical embedded systems.

I feel extremely lucky to have lab mates and colleagues like Elias Bareinboim, Sharath Gopal, Manu Jose, Hesam Samimi, Dr. K. C. Shashidhar, Sai Deep Tetali and Zilong Wang, who always took time to ask me about the progress on my research and gave good advice whenever I needed. The light moments spent with them always helped me come back with a refreshed mood to work harder on my dissertation research.

I would like to thank my friends and relatives for their continual encouragement. Thanks are due to Tilak Adhya, Indranil Basu, Dr. Saurav Basu, Kuntal Chakraborty, Kousik Debnath, Dr. Pavel Ghosh, Satadal Ghosh, Rudra Narayan Hota, Sucheta Roy, Lokesh Kumar Sambasivan, Aditya Zutshi and many others for their encouragement during my doctoral study. I would like to thank my brother Abhranil Saha and my extended family in India for their love and encouragement.

Finally, I express my sincere gratitude to my parents for cheering me up at every stage during my doctoral study. Starting from my childhood days, they have spent so much of their time and energy for my education. Without their love, endless support and encouragement, this thesis might not be possible. This thesis is dedicated to them.

Vita

2003	B.Tech.(Electronics and Communication Engineering), Kalyani Govt. Engineering College, India
2005	M.Tech.(Computer Science), Indian Statistical Institute, India
2005 – 2008	Research Scientist, Honeywell, India

Publications

Adolfo Anta, Rupak Majumdar, Indranil Saha, Paulo Tabuada. Automatic Verification of Control System Implementations. EMSOFT 2010. (Best Paper Award)

Rupak Majumdar, Indranil Saha, Majid Zamani. Performance-Aware Scheduler Synthesis for Control Systems. EMSOFT 2011.

Rupak Majumdar, Indranil Saha, Majid Zamani. Synthesis of Minimal Error Control Software. EMSOFT 2012. (Best Paper Nomination)

Indranil Saha and Rupak Majumdar. Trigger Memoization in Self-Triggered Control. EMSOFT 2012.

Eva Darulova, Viktor Kuncak, Rupak Majumdar and Indranil Saha. Synthesis of fixed-point Programs. Manuscript under submission.

Indranil Saha and Rupak Majumdar. Performance Aware Dynamic Scheduling in Networked Control Systems. Manuscript under submission.

CHAPTER 1

Introduction

A cyber-physical system tightly coordinates discrete computation and continuous control of physical resources. Applications of cyber-physical systems are numerous – ranging from simple home appliances, for example, a thermostat, to complex transportation systems, for example, autonomous vehicles. Most safety-critical cyber-physical systems run sophisticated control algorithms at their core. In a cyber-physical system, the subsystem which runs the control algorithms is referred as a control system. The control system is often implemented on a hardware platform. The control system obtains the state of the plant through a set of sensors at certain discrete time instances. The state of the plant represents the physical quantities such as position, velocity, temperature, pressure and so on. Based on the current state of the plant, the control system computes the control inputs to be applied to the plant. The control inputs are applied to the plant through a set of actuators. Often the sensors communicate with the controller through a network.

While the control algorithms have been studied thoroughly from a theoretical standpoint, their implementation as software raises many issues that need to be addressed to ensure the reliability of cyber-physical systems. Consider the common practice of designing a control system using continuous mathematics and implementing it in software. At one end of the spectrum we have control engineering that studies the evolution of continuous dynamical systems while mostly ignoring the effect of implementing feedback control laws in software. At the other end, we

have software engineering, which abstracts away physical components when modeling computation. Thus, mathematical real numbers are approximated using floating-point or fixed-point arithmetic, and the continuous world is discretized through sampling. In designing a controller mathematically, the control computation time is often assumed to be negligible, which may not often be the case in the real implementations. Moreover, the automotive and the aerospace industries are moving from using federated architectures, where a single control application is implemented on one processor, to integrated architectures, where the objective is to implement a number of control loops on a single processor. When multiple control applications run on a shared platform, schedulability requirements of the corresponding tasks impose additional constraints on the behavior of the systems. Quantifying the effect of using finite precision arithmetic in the implementation of controller program requires the application of program analysis techniques. The effect of computation time and schedulability analysis are studied in the context of implementation of real-time systems. Thus we see that building a reliable cyber-physical system requires combining knowledge from at least three areas: control engineering, software engineering and real-time systems. Unfortunately, there has been very little interaction among these fields to date. The current practice is to design a controller without taking into account the implementation constraints, and then simulate the implemented control system thoroughly to make sure that the system behaves reasonably. However, for safety-critical systems, we need formal guarantees on the behavior. In this thesis, we study how we can combine techniques from control theory, software engineering and real-time systems theory to build cyber-physical systems that are provably correct with respect to some desirable properties.

1.1 Contributions

Implementation of controller software using finite precision arithmetic. The controller for a cyber-physical system is designed based on real arithmetic, considering the dynamics of the system to be continuous. However, when the controller is realized as software, the dynamics of the system is discretized based on some chosen sampling time, and finite precision arithmetic is used to represent the variables. Now one should ensure that the implementation still ensures the desired properties of the system. This is a verification question that requires both control theoretic analysis and program analysis of control software. We show through control theoretic analysis that the verification question regarding the desired property of the implemented control system can be reduced to the question of verifying the bound on the error due to quantization effect introduced at the implementation of the controller program. We employ program analysis techniques to verify the bound on the implementation error.

There is a relevant control software synthesis challenge: For a chosen finite precision arithmetic, design a controller for which the implemented software has the least error among all possible controllers. Generally, controllers are designed to minimize the control cost (the power of the control signal) and the state cost (the deviation of the state from the desired value). We show that the controller designed based on optimization of such costs may produce a controller whose fixed point implementation may have significant error in the output. We provide a stochastic optimization-based methodology to synthesize a controller that minimizes both the state and control cost along with the error in the fixed point implementation of the controller software.

The synthesized controllers generally have the form of a linear expression (for linear control systems) or a polynomial (for nonlinear control systems). A naive compilation of a controller expression may produce a program that significantly

deviates from the controller designed using real arithmetic, whereas a different order of evaluation can result in a program that is close to the real value on all inputs in its domain. We have developed a compilation scheme that compiles real-valued arithmetic expressions to fixed point arithmetic programs, which minimizes the error in the fixed-point program with respect to the real-valued expression. Our technique is based on genetic programming, where the fitness of each candidate program is determined based on the bound on the error at the output of the program. To compute the bound on the error, we employ a static analysis technique based on affine arithmetic. This compilation scheme enables us to further improve the precision of the controller implementation.

Implementation of self-triggered control systems. The second contribution of this thesis is the introduction of a novel software implementation scheme for self-triggered control systems. In a self-triggered implementation, the control task computes the actuator signal as well as a triggering time that specifies the next time instance at which the control task should be run. Self-triggered implementations have the potential to decrease communication costs and CPU requirements over time-triggered ones by running the steady-state plant in open loop for long intervals if there is no disturbance. Thus, a self-triggered implementation is an attractive technique to be used for integrated architectures of cyber-physical systems, in which multiple control loops and non-critical applications share the same resources (CPU or communication network). However, we show that commonly claimed gains for self-triggered implementations are too optimistic. The analysis of most self-triggering algorithms ignore the execution times for computing the trigger times. We show, using implementations of several self-triggering algorithms proposed in the literature on common embedded platforms, that the execution time to compute the trigger time can be non-negligible compared to the trigger times, and may even be higher than a trigger time itself, rendering a naive implementation infeasible. We propose an implementation scheme for self-

triggered control using state quantization and memoization of trigger times in a cache, and provide guarantees on the steady state behavior of the control systems. Our implementation scheme is always feasible, and the performance of the implemented controller meets the expectations stated in the literature on self-triggered control systems.

Scheduler synthesis for optimal performance. The third contribution of this thesis is to understand the end-to-end behavior of control systems implemented on shared platforms. In integrated architectures, where the objective is to implement a number of control loops on a single processor, the most important issue is scheduling—how to schedule the control tasks with fairness so that all the control systems attain desired performances. We provide a constraint solving based static scheduler synthesis framework for scheduling control tasks from multiple control systems on a single processor. The scheduler judiciously drops control task computation to maintain schedulability and ensures not only stability but also optimal performance of the individual control systems.

In designing a static scheduler, we assume that the communication medium between the plant and controller is perfect. However, in real scenario, the communication between the sensors attached to the plant to measure the plant's state and the controller often happens through a network which introduces communication delay and packet dropout. The design of a scheduling algorithm in a networked control setting would require that the scheduler also takes into account the delay and any packet dropped due to network failure. As packet drop by the network is unpredictable, we cannot design a static scheduler to solve this problem. We show how we can synthesize a controller and a dynamic scheduling strategy that together ensure optimal performance of the control system even in the presence of delay and arbitrary packet dropout (with an upper limit on the rate of packet dropout) by the network.

1.2 Organization

The rest of this thesis is divided into eight chapters. In Chapter 2, we provide the necessary background of control theory. In Chapter 3, we show how to verify stability properties of the implementation of a control system. In Chapter 4, we provide a framework for controller synthesis, that can take into account the traditional control performance criteria together with the effect of the implementation constraints while synthesizing a controller. In Chapter 5, we show that the synthesized controller programs can be further optimized by properly choosing the order of the arithmetic operations, and present a genetic programming based tool that synthesizes a controller program taking into account all possible ordering of the arithmetic operations. In Chapter 6, we present an implementation scheme for self-triggered control using state quantization and memoization of trigger times in a cache, and provide guarantees on the steady state behavior of the control systems. In Chapter 7, we present a constraint solving based static scheduler synthesis scheme to maintain stability and achieve optimal performance in the control systems in the absence of any network induced packet drop. In Chapter 8, we present a dynamic scheduler synthesis scheme that taking into account packet drops by the network maintains stability and achieves optimal performance in the control systems. We conclude the thesis in Chapter 9, outlining a number of possible directions of future research.

CHAPTER 2

Basics of Control Theory

In this chapter, we introduce the mathematical notations used throughout the thesis. We also provide the required background of control theory.

2.1 Mathematical Notation

We denote by $\mathbb{R}$, $\mathbb{R}_0^+$, and $\mathbb{R}^+$ the real, non-negative real, and positive real numbers, respectively. The natural numbers, including zero, are denoted by $\mathbb{N}_0$. The function e^{At}, for $t \in \mathbb{R}$ denotes the matrix function defined by the convergent series:

$$\mathsf{e}^{At} = 1_{n\times n} + At + \frac{1}{2!}A^2t^2 + \frac{1}{3!}A^3t^3 + \dots$$

where $1_{n\times n}$ is the identity matrix with n rows and n columns and e is Euler's constant. The zero matrix with n rows and m columns is denoted by $0_{n\times m}$. The j-th entry of a vector $x \in \mathbb{R}^n$ is denoted by x_j and the Euclidean norm of x is given by $\|x\| = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$. We will also use the 1-norm, denoted by $\|x\|_1$ and defined by $\|x\|_1 = |x_1| + |x_2| + \ldots + |x_n|$, where $|x_j|$ is the absolute value of $x_j \in \mathbb{R}$. These two norms are related by $\|x\| \leq \|x\|_1 \leq \sqrt{n}\|x\|$. The induced 2 norm of a matrix $A \in \mathbb{R}^{n\times m}$ is given by: $\|A\| = \sqrt{\lambda_{\max}(A^TA)}$. We call the matrix A Hurwitz if its eigenvalues are inside the unit circle, centered at the origin.

A continuous function $\gamma : \mathbb{R}_0^+ \to \mathbb{R}_0^+$, is said to belong to class $\mathcal{K}$ if it is strictly increasing and $\gamma(0) = 0$. A $\mathcal{K}$-function γ is said to belong to class $\mathcal{K}_\infty$ if $\gamma(s) \to \infty$ as $s \to \infty$. A continuous function $\beta : \mathbb{R}_0^+ \times \mathbb{R}_0^+ \to \mathbb{R}_0^+$ is said to belong to class

$\mathcal{KL}$ if $\beta(r,t)$ belongs to class $\mathcal{K}_\infty$ for each fixed $t \geq 0$ and $\beta(r,t)$ is decreasing with respect to t and $\beta(r,t) \to 0$ as $t \to \infty$ for each fixed $r \geq 0$. A function $F : \mathbb{R}^n \to \mathbb{R}$ is called *radially unbounded* if $F(x) \to \infty$ as $\|x\| \to \infty$.

2.2 Mathematical Model of a Control System

2.2.1 Continuous-Time Systems

Physical systems are typically modeled by differential equations:

$$\frac{d}{dt}\xi = f(\xi, \upsilon, \omega) \tag{2.1}$$

in which the curve $\xi : \mathbb{R} \to \mathbb{R}^n$ describes how the physical quantities of interest change over time. At each time instant $t \in \mathbb{R}$, $\xi(t)$ is a vector in $\mathbb{R}^n$ containing the values of physical quantities such as positions, velocities, temperatures, pressures, etc. The vector $\upsilon(t)$ denotes an input signal acting on the system. This input can be manipulated by a controller to control the behavior of the system. The vector $\omega(t)$ denotes an uncontrollable disturbance signal acting on the system. We assume that the curves υ and ω follow some regularity assumptions, so that the existence and uniqueness of the solutions of (2.1) can be guaranteed. The curve ξ is said to be a *solution* or *trajectory* of (2.1) when there exists an input curve $\upsilon : \mathbb{R} \to \mathbb{R}^m$ such that the time derivative of ξ at time t equals $f(\xi(t), \upsilon(t))$. We denote by $\xi_{x\upsilon}(t)$ the point reached by the solution ξ of (2.1) at time $t \in \mathbb{R}$, starting at the state $x \in \mathbb{R}^n$ and under the input $\upsilon : \mathbb{R} \to \mathbb{R}^m$ and the disturbance $\omega : \mathbb{R} \to \mathbb{R}^q$. Note that different input curves υ give rise to different trajectories ξ. The objective of the control engineer is to design a controller $\upsilon = k(\xi, c)$ computing υ based on the evolution of the physical variables and the desired commands c given by the user.

The resulting input curves υ force the corresponding trajectories ξ to have desirable properties, most notably *asymptotic stability*. Intuitively, given a desired

operating point $x_0 \in \mathbb{R}^n$, we would like to design input curves υ so that trajectories ξ of (2.1) converge asymptotically to x_0, *i*.e., $\lim_{t\to\infty} \xi(t) = x_0$. Moreover, stability also requires that if $\xi(0)$ is close to x_0, then $\xi(t)$ should remain close to x_0 for $t > 0$. Whenever the controller $\upsilon = k(\xi, c)$ gives rise to trajectories with the preceding properties, we say that x_0 is a global asymptotically stable equilibrium point for

$$\frac{d}{dt}\xi = f(\xi, k(\xi, c)).$$

In the next subsection, we provide formal definitions of the notion of stability.

2.2.2 Stability Properties

Here we provide formal definitions of different stability criteria that are widely used as desirable properties of a closed loop control system. For more detailed analysis, we refer the readers to the book by Khalil [Kha02].

The closed loop system is *stable* with respect to the origin $\xi = 0$, if for every $b > 0$ there exists a $a > 0$ such that

$$\|\xi(0)\| \leq a \quad \Rightarrow \quad \|\xi(t)\| \leq b \qquad \text{for all } t \geq 0$$

The closed loop system is called *asymptotically stable* if it is stable and a can be chosen so that

$$\|\xi(0)\| \leq a \quad \Rightarrow \quad \xi(t) \to 0 \quad as \quad t \to \infty.$$

If the above condition holds for any $a > 0$, the closed loop system is called *globally asymptotically stable* with respect to the origin. The closed loop system is called *exponentially stable* if there exist positive constants a, c and λ such that for all $\|\xi(0)\| \leq a$, $\xi(t)$ satisfies the inequality

$$\|\xi(t)\| \leq c\|\xi(0)\|e^{-\lambda t}, \qquad \forall t \geq 0.$$

If the above inequality holds for any a, the closed loop system is called *globally exponentially stable*.

The stability of a closed loop control system in the presence of a piecewise continuous disturbance input $d(t)$ is given by the concept of *input-to-state stability*. A closed-loop dynamical system with a *disturbance input* is represented as

$$\dot{\xi} = f(\xi, k(\xi(t)), d(t)).$$

The system is called *input-to-state stable* (ISS) if there exists a $\mathcal{K}_\infty$ function α, and a $\mathcal{KL}$ function β, such that for any initial state $\xi(0)$, the state $\xi(t)$ satisfies the following inequality:

$$\|\xi(t)\| \leq \beta(\|\xi(0)\|, t) + \alpha(\|d\|_\infty), \qquad \forall t \geq 0$$

Sufficient conditions on different stability criteria are given using Lyapunov functions. A *Lyapunov function* is a continuously differentiable function $V : D \to \mathbb{R}$, $D \subset \mathbb{R}^n$, for which the following conditions hold:

$$V(0) = 0, \qquad V(\xi) > 0 \quad \text{for all } \xi \neq 0.$$

The derivative of V along the solution of the closed loop system is given by

$$\dot{V}(\xi) = \frac{\partial V}{\partial \xi} f(\xi, k(\xi(t)))$$

The closed loop system is stable with respect to $\xi = 0$ if there exists a Lyapunov function V such that $\dot{V}(\xi) \leq 0$ for all $\xi \neq 0$. The closed loop system is asymptotically stable with respect to origin if there is a Lyapunov function V such that $\dot{V}(\xi) < 0 \quad$ for all $\xi \neq 0$. The closed loop system is globally asymptotically stable if there exists a radially unbounded Lyapunov function with $\dot{V}(\xi) < 0$. The closed loop system is exponentially stable if there exists a Lyapunov function V and a positive constant λ such that

$$V(\xi(t)) \leq V(0)e^{-\lambda t}, \qquad \text{for all } t \geq 0.$$

The largest constant that can be used for λ in the above inequality is called the *rate of decay*. The closed loop system is ISS if there exists a radially unbounded Lyapunov function V and two $\mathcal{K}_\infty$ functions α_1 and α_2 such that

$$\dot{V}(\xi, d) \leq -\alpha_1(\|\xi\|) + \alpha_2(\|d\|) \qquad \text{for all } \xi,\ d.$$

2.2.3 Continuous-Time Linear Control Systems

In this thesis, we have mostly dealt with linear time-invariant systems. A linear time-invariant control system is given by the following differential equations:

$$\begin{cases} \dot{\xi} = A\xi + B\upsilon + \overline{B}\omega, \\ \eta = C\xi + \nu, \end{cases} \tag{2.2}$$

where, for any $t \in \mathbb{R}$, $\xi(t) \in \mathbb{R}^n$, $\upsilon(t) \in \mathbb{R}^m$, $\omega(t) \in \mathbb{R}^q$, $\eta(t) \in \mathbb{R}^p$, and A, B, $\overline{B}$, and C are matrices of appropriate dimensions. Note that υ, ω, η, and ν denote control input, disturbance, output of the system and measurement noise, respectively. We assume that $\omega(t)$ and $\nu(t)$, for any $t \in \mathbb{R}$, are zero-mean Gaussian noise processes (uncorrelated from each other). For all curves ω, we also write $\xi_{x\upsilon}(t)$ to denote the points reached at time t under the input υ from initial condition $x = \xi_{x\upsilon}(0)$.

2.2.4 Discrete-Time Linear Control Systems

To describe the mismatch between the controller specifications and its software implementations such as digital sampling and finite precision arithmetic, which is the focus of this paper, we consider the discrete-time version of (2.2), as follows:

$$\begin{cases} x[r+1] = A_\tau x[r] + B_\tau u[r] + \overline{B}_\tau d[r] + e_s, \\ y[r] = Cx[r] + v[r], \end{cases} \tag{2.3}$$

where the matrices A_τ, B_τ, and $\overline{B}_\tau$ are given by:

$$A_\tau = \mathsf{e}^{A\tau}, \quad B_\tau = \int_{r\tau}^{(r+1)\tau} \mathsf{e}^{A(\tau-t)} B dt,$$
$$\overline{B}_\tau = \int_{r\tau}^{(r+1)\tau} \mathsf{e}^{A(\tau-t)} \overline{B} dt,$$

and τ is the sampling time. The signals x, u, d, y, and v describe the exact value of the signals ξ, υ, ω, η, and ν, respectively, at the sampling instants $0, \tau, 2\tau, 3\tau, \ldots$.

Mathematically, we have:

$$\begin{aligned} x[r] =& \xi(r\tau),\ u[r] = \upsilon(r\tau),\ d[r] = \omega(r\tau), \\ y[r] =& \eta(r\tau),\ v[r] = \nu(r\tau),\quad \forall r \in \mathbb{N}_0. \end{aligned}$$

The term e_s in (2.3) is the sampling error. It can be shown that by sampling sufficiently fast, the error e_s can be made arbitrarily small [CF95a]. Since typical embedded controller implementations use sampling time in the range of milliseconds to microseconds, we will make the assumption that quantization errors dominate the sampling errors, and assume that $e_s = 0$.

2.2.5 Observer

We assume that only output y of the system is measurable and not the full state x. Hence, a (proportional) *feedback* $K : \mathbb{R}^n \to \mathbb{R}^m$ defines the input

$$u[r] = -K\widetilde{x}[r] \tag{2.4}$$

based on an estimation $\widetilde{x}$ of the state x. As explained in [Hes09], the estimation $\widetilde{x}$ can be constructed using the observer dynamic:

$$\begin{cases} \widetilde{x}[r+1] = A_\tau \widetilde{x}[r] + B_\tau u[r] + L\left(y[r] - \widetilde{y}[r]\right), \\ \quad \widetilde{y}[r] = C\widetilde{x}[r], \end{cases} \tag{2.5}$$

where $\widetilde{y}$ should be viewed as an estimate of y and the linear map $L : \mathbb{R}^p \to \mathbb{R}^n$ is called an observer gain. Here we ignore the effect of the external disturbance and the measurement noise.

Thus a controller implementation with an observer is given by:

$$\begin{cases} \widetilde{x}[r+1] = D_\tau \widetilde{x}[r] + E_\tau y[r], \\ u[r+1] = K\widetilde{x}[r+1], \end{cases} \tag{2.6}$$

where, $D_\tau = (A_\tau - B_\tau K - LC)$ and $E_\tau = L$.

By applying the feedback $u[r] = -K\widetilde{x}[r]$ and combining the dynamics of control system in (2.3) and observer in (2.5), one obtains:

$$\begin{cases} x[r+1] = A_\tau x[r] - B_\tau K\widetilde{x}[r] + \overline{B}_\tau d[r], \\ \widetilde{x}[r+1] = D_\tau \widetilde{x}[r] + E_\tau C x[r] + E_\tau v[r], \end{cases} \tag{2.7}$$

which gives the dynamics of the closed loop control system.

2.3 Performance Criteria

Apart from the stability properties, the designed control systems should also satisfy some optimality criteria. Here we present three performance criteria that we have dealt with in this thesis.

2.3.1 Region of Practical Stability

Definition 1. *A bounded set $S \subset \mathbb{R}^n$ is said to be globally asymptotically stable for a differential equation (2.1) if the following two properties hold:*

- $\forall \alpha \in \mathbb{R}^+ \; \exists t^* \in \mathbb{R}^+ \; \forall t \geq t^* \; d(\xi_x(t), S) \leq \alpha$;
- $\forall \alpha \in \mathbb{R}^+ \; \exists \beta \in \mathbb{R}_0^+ \; \forall t \in \mathbb{R}_0^+$
  $d(x, S) \leq \beta \implies d(\xi_x(t), S) \leq \alpha$.

where the distance from x to S, denoted by $d(x, S)$, is defined by:

$$d(x, S) = \inf_{y \in S} \|x - y\|.$$

The set S is called the *region of practical stability*. Whenever the set S contains a single point x_0, we speak of an asymptotically stable equilibrium point x_0 rather than of an asymptotically stable set. The first requirement in Definition 1 asks that for any initial state $x \in \mathbb{R}^n$, the corresponding trajectory ξ_x asymptotically converges to the set S in the sense that after $t^*(\alpha)$ units of time ξ_x will be α-close

to S. The second condition prevents ξ_x from deviating too much from S when x is close to S. The notion of asymptotic stability for discrete-time difference equations is obtained from Definition 1 by replacing $t, t^* \in \mathbb{R}_0^+$ with $r, r^* \in \mathbb{N}_0$.

The following result describes how stability properties are affected by additive perturbations. It summarizes several results from [CF95b] in a convenient form.

Proposition 1. *Consider the discrete-time linear system:*

$$x[r+1] = Ax[r]$$

and assume that the origin is an asymptotically stable equilibrium point. Then, for any signal $d : \mathbb{N} \to \mathbb{R}^n$ satisfying $\|d\| \leq b(d)$ for some constant $b(d) \in \mathbb{R}_0^+$, the set:

$$S = \{x \in \mathbb{R}^n \mid \|x\| \leq \gamma b(d)\}$$

is globally asymptotically stable for the discrete-time system:

$$x[r+1] \quad = \quad Ax[r] + Bd[r] \tag{2.8}$$

where γ is given by:

$$\gamma = \max_{\psi \in [0,2\pi[} \left\| (\mathsf{e}^{i\psi} 1_{n\times n} - A)^{-1} B \right\|.$$

with $i = \sqrt{-1}$. Moreover, the output $o = Cx$ is guaranteed to converge to the set:

$$S_C = \{o \in \mathbb{R}^p \mid \|o\| \leq \gamma_C b(d)\} \tag{2.9}$$

with:

$$\gamma_C = \max_{\psi \in [0,2\pi[} \left\| C(\mathsf{e}^{i\psi} 1_{n\times n} - A)^{-1} B \right\|.$$

In the control literature, γ_C is known as the $\mathcal{L}_2$ *gain* of the control system. It describes how much the disturbance signal d is amplified by the discrete-time linear system. When a disturbance d of magnitude $b(d)$ is injected into the dynamical system, as described by (2.8), the state evolution will deviate from the nominal behavior by at most $\gamma_C b(d)$ as described by the set (2.9).

2.3.2 LQR-LQG Performance

In addition to asymptotic stability, controller designers also consider the *performance* of the controller, that is, of the controllers ensuring asymptotic stability of the origin, one desires the controller that minimizes a given cost function. A common approach for optimal output feedback controller are the *Linear Quadratic Regulator* (LQR) and *Linear Quadratic Gaussian* (LQG). The LQR cost function to be minimized is given by:

$$J_{LQR} = \sum_{r=0}^{+\infty} \left\{ x[r]^T Q x[r] + u[r]^T R u[r] \right\}, \tag{2.10}$$

for some chosen weight matrices Q and R that are positive definite and of appropriate dimensions.

The LQG cost function to be minimized is given by:

$$J_{LQG} = \lim_{r \to +\infty} \mathrm{E}\left[\|e[r]\|^2 \right], \tag{2.11}$$

where E stands for expected value and e is the estimation error for the control system in (2.7) whose dynamic is given by:

$$e[r+1] = x[r+1] - \widetilde{x}[r+1] = (A_\tau - LC)e[r] + \overline{B}_\tau d[r] - Lv[r]. \tag{2.12}$$

As mentioned before, d and v are assumed to be zero-mean Gaussian noise process (uncorrelated from each other) with covariance matrices:

$$\mathrm{E}\left(d[r] d[r]^T \right) = \widehat{Q}, \ \mathrm{E}\left(v[r] v[r]^T \right) = \widehat{R}, \ \forall r \in \mathbb{N}_0, \tag{2.13}$$

where $\widehat{Q}$ and $\widehat{R}$ are some positive semi-definite matrices of appropriate dimensions.

A standard control-theoretic construction rewrites the cost function (2.10) as $J_{LQR} = x[0]^T S(K) x[0]$, where $u = -Kx$, and $S(K) \in \mathbb{R}^{n \times n}$ is a positive definite matrix that is the unique solution for S to the Lyapunov equation:

$$(A_\tau - B_\tau K)^T S (A_\tau - B_\tau K) - S + Q + K^T R K = 0, \tag{2.14}$$

where K is a controller making $A_\tau - B_\tau K$ Hurwitz. See [Hes09] for detailed information. Additionally, we have

$$\lambda_{\min}(S(K))\|x[0]\|^2 \leq J_{LQR} \leq \lambda_{\max}(S(K))\|x[0]\|^2, \tag{2.15}$$

where $\lambda_{\min}(S(K)) \in \mathbb{R}^+$ and $\lambda_{\max}(S(K)) \in \mathbb{R}^+$ are minimum and maximum eigenvalues of $S(K)$, respectively. Therefore, J_{LQR} can be minimized for all possible choices of initial conditions by just minimizing the maximum eigenvalue of $S(K)$. Note that since S is a positive definite and symmetric matrix, its maximum eigenvalue is equal to its induced 2 norm $\|S\|$.

Similarly, the cost function (2.11) can be rewritten as $J_{LQG} = \|P(L)\|$, where $P(L) \in \mathbb{R}^{n\times n}$ is a positive definite matrix that is the unique solution for P to the Lyapunov equation:

$$(A_\tau - LC)\, P\, (A_\tau - LC)^T - P + \overline{B}_\tau \widehat{Q} \overline{B}_\tau^T + L\widehat{R}L^T = 0, \tag{2.16}$$

where L is an observer gain making $A_\tau - LC$ Hurwitz. See [Hes09] for more detailed information. Therefore, J_{LQG} can be minimized by just minimizing $\|P(L)\|$.

Note that the optimal feedback $u = -Kx$ minimizing the LQR cost in (2.10) is computed using the deterministic dynamic:

$$x[r+1] = A_\tau x[r] + B_\tau u[r].$$

On the other hand, the optimal gain L minimizing the LQG cost in (2.11) is computed using the stochastic dynamic in (2.12). Thanks to the separation principle for linear control systems [Hes09], one concludes that the overall closed loop system in (2.7) is $UGAS$ even though the gains K and L are designed separately.

2.3.3 $\mathcal{L}_\infty$ to RMS Gain

We now consider the $\mathcal{L}_\infty$ to RMS gain as a performance criterion for LTI control systems and consider the effect of dropouts.

For the discrete-time LTI control system in (7.3), the $\mathcal{L}_\infty$ to RMS induced gain from w to y is defined as follows:

$$\sup_{\|w\|_\infty \neq 0, X(0)=0} \frac{\left(\limsup_{l\to\infty} \frac{1}{l} \sum_{j=0}^{l} y^T(j)y(j)\right)^{\frac{1}{2}}}{\|w\|_\infty}, \tag{2.17}$$

where $\|w\|_\infty := \sup\{\|w(k)\|_2,\ k \geq 0\}$, and $\|w(k)\|_2 = \sqrt{w^T(k)w(k)}$.

The $\mathcal{L}_\infty$ to RMS induced gain is a performance criterion showing the effect of the disturbance on the output of the plants [HBH99]. Having smaller $\mathcal{L}_\infty$ to RMS induced gain implies better performance in the sense that the effect of disturbance on the output of the plant is smaller.

CHAPTER 3

Verification

In this chapter, we introduce a methodology for the design and automatic verification of control system implementations for stability properties. Our methodology has two components. First, we provide a mathematical analysis of stability that incorporates implementation errors arising out of the mismatch between the ideal mathematical controller and its software implementation. Our analysis provides a relationship between the region where the continuous variables can be steered to under the action of the controller and the error between the ideal controller and its implementation.

Second, we perform static program analysis on the software code implementing the controller to compute an upper bound on the error between the mathematical control signal and the software output. The static analysis is based on verification condition generation [Win93], and reduces the error bound question to a validity problem for a formula in the combination theory of reals and bitvectors, for which off-the-shelf, efficient decision procedures are available [FHR07]. Together with the first part, this enables us to compute an upper bound on the region to which the *implemented* controller is guaranteed to steer the system.

We have implemented Costan (for Controller Stability Analyzer), a tool that reads in controller implementations in C and checks the maximum possible error between the outputs of the C implementation and the Simulink function implementing the mathematical controller. The current version of Costan targets *fixed-point* implementations of controllers. Fixed-point implementations use fixed

width (e.g., 16- or 32-bit) integers to implement numerical operations over real numbers. They are useful for implementing controllers on hardware platforms without floating point support. (Our methodology carries over to floating point implementations, given the semantics of floating point operations in the verification condition generation [BKW09, FSI09].) Using Costan, we have analyzed error bounds for a number of linear and non-linear control systems implementations, including a realistic train car controller [MPS76], and we have characterized the regions to which the controllers can steer the physical variables.

3.1 Motivating Example

We start with a simple example that shows the controller design process and describes the semantic gap between a mathematical controller and its implementation. We consider a simple physical model of a locomotive pulling a train car where the connection between the locomotive and the car is modeled by a spring in parallel with a damper. The model has been borrowed from [MPS76]. The dynamics of this system is given by:

$$\frac{d}{dt}\begin{bmatrix}\xi_1\\ \xi_2\\ \xi_3\end{bmatrix} = \begin{bmatrix} 0 & 1 & -1 \\ -\frac{k}{m_1} & -\frac{b+c}{m_1} & \frac{b}{m_1} \\ \frac{k}{m_2} & \frac{b}{m_2} & -\frac{b}{m_2}\end{bmatrix}\begin{bmatrix}\xi_1\\ \xi_2\\ \xi_3\end{bmatrix} + \begin{bmatrix}0\\ \frac{1}{m_1}\\ 0\end{bmatrix}\upsilon \tag{3.1}$$

$$\omega = \begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\xi_1\\ \xi_2\\ \xi_3\end{bmatrix} \tag{3.2}$$

where ξ_1 represents the distance between the locomotive and the car, ξ_2 and ξ_3 represent the velocities of the locomotive and the car, $m_1 = 176580$Kg is the mass of the locomotive, $m_2 = 100698$Kg is the mass of the car, $b = 1100$Ns/m is the damping coefficient, $k = 7874$N/m is the restitution coefficient of the spring, $c = 160$Ns/m is the aerodynamic friction coefficient, and υ represents the force

applied by the locomotive.

The control objective is to quickly reduce the possible oscillations between the locomotive and the car. Through a simple change of coordinates this problem can be reformulated to the design of a controller that forces ξ_1 to converge to zero. One possible state feedback controller for the system is given by:

$$\upsilon = -K\xi, \qquad K = \begin{bmatrix} 0.1763 & 0.3494 & -0.0602 \end{bmatrix}. \tag{3.3}$$

However, if only the velocities of the locomotive and car can be measured, (3.3) cannot be directly implemented. Hence, the feedback controller (3.3) is extended with an observer estimating the distance between the locomotive and the car as follows:

$$\begin{aligned} \frac{d}{dt}\begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \end{bmatrix} &= \begin{bmatrix} 0 & 79.58 & -127.95 \\ -1.04 & -10.68 & -1.19 \\ 0.07 & 1.49 & -11.26 \end{bmatrix}\begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \end{bmatrix} \\ &\quad + \begin{bmatrix} -78.58 & 126.95 \\ 8.69 & 1.54 \\ -1.48 & 11.25 \end{bmatrix}\omega. \end{aligned} \tag{3.4}$$

The resulting controller consists of the estimator and the feedback control law (3.3) evaluated at the estimate ζ, *i.e.*:

$$\upsilon = -K\zeta.$$

Figure 3.1 shows (part of the) fixed-point implementation of the controller. The code was generated automatically by Real Time Workshop. We point out two observations about the implementation. First, the controller is implemented using finite precision, fixed-point numbers, and "looks" very different from the mathematical controller. For example, the last five lines of the controller code implement the matrix multiplication (3.3), but the correspondence is not obvious from the code. Second, the output of the implementation need not exactly match

```
int DelayStateX_DSTATE[3];   // fixed point type: signed, 32bits, 28 fractional bits
int Xk[3], CXk[3];           // fixed point type: signed, 32bits, 28 fractional bits
int Gain1;                   // fixed point type: signed, 32bits, 29 fractional bits
int Gain2;                   // fixed point type: signed, 32bits, 28 fractional bits
int u1;                      // fixed point type: signed, 32bits, 28 fractional bits

static void controller(void)
{
  int tmp, tmp0;
  Xk[0] = DelayStateX_DSTATE[0];
  Xk[1] = DelayStateX_DSTATE[1];
  Xk[2] = DelayStateX_DSTATE[2];
  Gain2 = 310689525;  // Reference input

  ... // update observer state

  // compute control output based on state and observation
  for (tmp = 0; tmp < 3; tmp++) {
    tmp0 = (int)((long int)OutputMatrixC_Gain[tmp] * (longint)Xk[0] >> 30U);
    tmp0 += (int)((long int)OutputMatrixC_Gain[tmp + 3] * (long int)Xk[1] >> 30U);
    tmp0 += (int)((long int)OutputMatrixC_Gain[tmp + 6] * (long int)Xk[2] >> 30U);
    CXk[tmp] = tmp0;
  }
  tmp = (int)((long int)(-757257017) * (long int)CXk[0] >> 31U);
  tmp += (int)((long int)(-1500825839) * (long int)CXk[1] >> 31U);
  tmp += (int)((long int)258754934 * (long int)CXk[2] >> 31U);
  Gain1 = tmp;
  u1 = (Gain1 >> 1) + Gain2;
}
```

Figure 3.1: Fixed point controller implementation for locomotive and train car (generated by Fixed Point Advisor and Real Time Workshop)

the mathematical control function, due to finite precision arithmetic. In particular, it is not clear if this C implementation maintains the stability properties that were guaranteed by the mathematical design.

In the rest of the chapter, we describe our methodology that enables us to reason about the mathematical control system in conjunction with the implemented code.

3.2 Effect of Implementation Errors

In this section, we consider quantization errors due to the fixed-point implementation of the controller code.

3.2.1 Implementation Error in Fixed-Point Implementation

We use the notation $Q(x)$ and $Q(P)$ to denote the fixed-point representation of a vector $x \in \mathbb{R}^n$ and a matrix $P \in \mathbb{R}^{n\times m}$, respectively. A possible software implementation of the controller (2.6) can then be described by:

$$\begin{aligned}
\widehat{x}[r+1] &= Q\left(Q(Q(D_\tau)\widehat{x}[r]) + Q(Q(E_\tau)Q(y[r]))\right) \\
&= D_\tau\widehat{x}[r] + E_\tau y[r] + e_{q1} \\
\widehat{u}[r+1] &= Q\left(Q(-K)\widehat{x}[r+1]\right) \\
&= -K\widehat{x}[r+1] + e_{q2}.
\end{aligned} \tag{3.5}$$

where

$$\begin{aligned}
e_{q1} &= Q\left(Q(Q(D_\tau)\widehat{x}) + Q(Q(E_\tau)Q(y))\right) - D_\tau\widehat{x} - E_\tau y \\
e_{q2} &= Q\left(Q(-K)\widehat{x}\right) + K\widehat{x}
\end{aligned}$$

denote quantization errors. Note that the expressions for $\widehat{x}[r+1]$ and $\widehat{u}[r+1]$ were obtained by performing a quantization after each algebraic operation. A different software implementation can give rise to a different expression for the

error. Although different software implementations lead to different quantization errors, e_{q1} and e_{q2} will always denote the difference between the quantized version in a particular implementation and the un-quantized version of the algebraic expressions.

In the following discussion, we ignore the effect of external disturbance and measurement noise. The evolution, at the discrete-time instants $0, \tau, 2\tau, 3\tau, \ldots$, of the physical system (2.2) controlled by the implementation (3.5) can be obtained by combining the exact discretization of (2.2):

$$x[r+1] = A_\tau x[r] + B_\tau u[r] \tag{3.6}$$

$$y[r+1] = Cx[r+1] \tag{3.7}$$

with (3.5). The result is the discrete-time system:

$$x[r+1] = A_\tau x[r] - B_\tau K\widehat{x}[r] + B_\tau e_{q2} \tag{3.8}$$

$$\widehat{x}[r+1] = D_\tau \widehat{x}[r] + E_\tau C x[r] + e_{q1} \tag{3.9}$$

Two new signals appear in (3.6) and (3.7). They are defined as follows:

$$x[r] = \xi(r\tau), \quad u[r] = \upsilon(r\tau), \quad \forall r \in \mathbb{N}_0.$$

3.2.2 Stability analysis of controller implementations

The equations (3.8) and (3.9) describing the plant and the controller can be rewritten in matrix form:

$$w[r+1] = Gw[r] + He[r] \tag{3.10}$$

with:

$$w = \begin{bmatrix} x \\ \widehat{x} \end{bmatrix}, \quad e = \begin{bmatrix} e_{q_2} \\ e_{q_1} \end{bmatrix},$$

and:

$$G = \begin{bmatrix} A_\tau & -B_\tau K \\ E_\tau C & D_\tau \end{bmatrix}, \quad H = \begin{bmatrix} 0_{n\times n} & B_\tau \\ 1_{n\times n} & 0_{n\times n} \end{bmatrix}.$$

Under the working assumption that τ is sufficiently small, it can be shown[1] that $w_0 = 0$ is an asymptotically stable equilibrium point for:

$$w[r+1] = Gw[r].$$

Using the state space representation in (3.10), it is straightforward to compute the set where the state trajectories asymptotically converge, as defined in the following proposition. The proposition follows from Proposition 1 in Chapter 2.

Proposition 2. *Consider the discrete-time linear system in (3.10). For any signals e_{q1} and e_{q2} satisfying $\|e_{q1}\| \leq b(e_{q1})$ and $\|e_{q2}\| \leq b(e_{q2})$, $b(e_{q1}), b(e_{q2}) \in \mathbb{R}_0^+$, the output $o = Mw$ is guaranteed to converge to the set:*

$$S_M = \{o \in \mathbb{R}^q \mid \|o\| \leq \gamma_M \left(b(e_{q1}) + b(e_{q2})\right)\}$$

with:

$$\gamma_M = \max_{\psi \in [0, 2\pi[} \left\| M(\mathsf{e}^{i\psi} 1_{2n \times 2n} - G)^{-1} H \right\|$$

The constants $b(e_{q1})$ and $b(e_{q2})$ are bounds for $\|e_{q_1}\|$ and $\|e_{q_2}\|$, respectively. Note that the smaller the errors e_{q1} and e_{q2}, the smaller is the set where o converges to. In the limiting case where the quantization errors are non-existent, *i*.e., $b(e_{q1}) = 0 = b(e_{q2})$, the set S becomes the point $o = 0$. Hence, if we seek an implementation guaranteeing that the set of outputs $o \in \mathbb{R}^q$ satisfying $\|o\| \leq \rho$ is asymptotically stable, the quantization errors e_{q1} and e_{q2} in the software implementation need to satisfy:

$$b(e_{q1}) + b(e_{q2}) \leq \frac{\rho}{\gamma_M}. \tag{3.11}$$

In the preceding equation, ρ describes how much the continuous variables are allowed to deviate from the nominal behavior due to implementation errors. For

[1] Under a small sampling time τ, matrix G can be seen as a small perturbation of the matrix $\mathsf{e}^{J\tau}$ with $J = \begin{bmatrix} A & BK \\ EC & D \end{bmatrix}$. Since the latter matrix has eigenvalues with negative real part, the eigenvalues of G are inside the unit disk.

the example in Section 3.1 we have $T = 0.001$ and:

$$
\begin{aligned}
A_\tau &= \begin{bmatrix} 1 & 0.0010000 & -0.0010000 \\ -0.0000044 & 0.9999992 & 0.0000006 \\ 0.0000078 & 0.0000011 & 0.9999989 \end{bmatrix} \\
B_\tau &= \begin{bmatrix} 0.00000283 \\ 0.00566293 \\ 0.00000003 \end{bmatrix} \\
D_\tau &= \begin{bmatrix} 0.9999536 & 0.0790688 & -0.1272843 \\ -0.0010375 & 0.9893293 & -0.0011182 \\ 0.0000769 & 0.0014819 & 0.9887948 \end{bmatrix} \\
E_\tau &= \begin{bmatrix} -0.0781482 & 0.1262988 \\ 0.0086947 & 0.0014643 \\ -0.0014725 & 0.0111945 \end{bmatrix}.
\end{aligned}
$$

Since the objective is to regulate the distance between the locomotive and the car, described by x_1, the relationship between the output o and the state w is given by:

$$o = x_1 = \begin{bmatrix} 1_{1\times 1} & 0_{1\times 5} \end{bmatrix} \begin{bmatrix} x \\ \widetilde{x} \end{bmatrix}$$

and the resulting value for γ_M is 36.11. Hence, for a desired value of $\rho = 2$cm, the quantization errors need to satisfy:

$$b(e_{q1}) + b(e_{q_2}) \leq \frac{0.02}{36.11} = 5.53 \cdot 10^{-4}$$

In summary, our analysis has produced a quantitative relation given by Equation (3.11) between the implementation errors and the asymptotically stable set. In the following section, we show how bounds on the implementation errors $b(e_{q1})$ and $b(e_{q2})$ can be computed statically from the controller source code.

3.3 Symbolic Error Analysis

Intuition. Given a real-valued polynomial function $y = f(x)$, a program F implementing f using finite precision arithmetic, and a range $[l, u]$ for x, the goal of symbolic error analysis is to find how far the value $f(x)$ can be from the output of $F(\hat{x})$ when x is chosen from the range $[l, u]$ and $\hat{x}$ is the closest representation of x using the finite precision implementation of real numbers. To solve this problem, we first construct the strongest post-condition $\mathsf{SP}(F)(\hat{x}, \hat{y})$ [Win93] for the function F which is a symbolic formula relating the input $\hat{x}$ to F with its output $\hat{y}$. Then, we set up a set of constraints that is the conjunction of: (a) "x is chosen in its range" $x \in [l, u]$, (b) "x differs from $\hat{x}$ in at most the precision δ of the finite precision representation" $|x - \hat{x}| \leq \delta$, (c) $y = f(x)$, (d) "the program F transforms $\hat{x}$ to $\hat{y}$" $\mathsf{SP}(F)(\hat{x}, \hat{y})$, and finally, ask (e) what is the maximum difference between y and $\hat{y}$ under the constraints (a)-(d)?

The rest of this section formalizes this intuition.

3.3.1 Real-Valued Programs

We represent (mathematical) control functions as *real-valued programs*. A real-valued program $P = (I, L, O, \mathsf{rng}, s)$ consists of a set of *input variables* I, a set of *local variables* L, a set of *output variables* O, a function rng mapping each input variable i in I to an interval $[l_i, h_i]$ of the reals, and a body s defined by the grammar:

$$s \ ::= \ x := c \mid x := y \oplus z \mid s_1; s_2 \mid \texttt{assume}(e) \mid s_1 [\!] s_2 \tag{3.12}$$

where x ranges over variables in $L \cup O$, y, z range over variables in $I \cup L$, c ranges over arbitrary rational constants, e ranges over Boolean expressions over the variables in $I \cup L$, and $\oplus \in \{+, -, *\}$ ranges over arithmetic operations. The body s consists of assignment statements, sequential composition $s_1; s_2$, conditionals (`assume`), and non-deterministic choice $s_1 [\!] s_2$. For x, z variables and c a rational

s	$\mathsf{SP}_{\mathbb{R}}(s, \theta)$
$x := c$	$\exists x'.\theta[x'/x] \wedge x = c$
$x := y \oplus z$	$\exists x'.\theta[x'/x] \wedge x = y \oplus z$
$\texttt{assume}(e)$	$\theta \wedge e$
$s_1; s_2$	$\mathsf{SP}_{\mathbb{R}}(s_2, \mathsf{SP}_{\mathbb{R}}(s_1, \theta))$
$s_1 [\!] s_2$	$\mathsf{SP}_{\mathbb{R}}(s_1, \theta) \vee \mathsf{SP}_{\mathbb{R}}(s_2, \theta)$

Table 3.1: Strongest postconditions for real-valued programs

constant, we write $x := c * z$ as shorthand for $y := c; x := y * z$.

Note that there are no loops in the language. This is because most programs in this domain come with *static* bounds on the number of iterations of a loop, and so loops can be unrolled statically. We keep the non-deterministic choice operation to model external conditions that choose between different implementations. For simplicity of exposition, we omit arrays and pointers as well as non-recursive function calls from our language, although our implementation handles these features.

For a set X of variables, an X-valuation is a mapping from X to the reals. We write $\{\{X\}\}_{\mathbb{R}}$ for the set of all X-valuations. For an X-valuation ν and variable $x \in X$, we write $\nu(x)$ for the value of x under the valuation ν. A *program state* is a $I \cup L \cup O$-valuation. In the following, we use first-order formulas with free variables in $I \cup L \cup O$ to denote sets of program states: a formula θ represents the set of program states that satisfy the formula θ.

The semantics of a real-valued program is given using the strongest postcondition operation $\mathsf{SP}_{\mathbb{R}}$. The function $\mathsf{SP}_{\mathbb{R}}$ maps a body s and a first-order formula θ to a first-order formula $\mathsf{SP}_{\mathbb{R}}(s, \theta)$. We use the notation $\theta[x'/x]$ to denote the formula θ in which each occurrence of x is substituted by x'. For a first-order formula θ with free variables in X, and an X-valuation ν, we say ν *satisfies* θ if the

formula obtained by substituting each occurrence of $x \in X$ with $\nu(x)$ evaluates to true. In the following, we identify an X-valuation ν with a $|X|$-dimensional vector. For a first-order formula θ and a set $X = \{x_1, \ldots, x_k\}$ of variables, we write $\exists X.\theta$ as shorthand for $\exists x_1.\exists x_2 \ldots \exists x_k.\theta$.

The transformer $\mathsf{SP}_{\mathbb{R}}$ is shown in Table 3.1. The operations have the usual semantics [Win93], and $\mathsf{SP}_{\mathbb{R}}(s, \theta)$ returns a formula that characterizes the set of program states that can be reached by executing the statement s from a program state satisfying the formula θ. An I-valuation $\nu \in \{\{I\}\}_{\mathbb{R}}$ satisfies the rng constraints if it maps every $v \in I$ to a value in $\mathsf{rng}(v)$, i.e., if $\nu(v) \in \mathsf{rng}(v)$ for each $v \in I$. The semantics of a real-valued program defines a mathematical relation, written $[\![P]\!]_{\mathbb{R}}$, between I-valuations and O-valuations in the following way: the pair of valuations $(\nu, \mu) \in [\![P]\!]_{\mathbb{R}}$ is in the relation if ν is an I-valuation, μ is an O-valuation, and μ satisfies $\exists L.\mathsf{SP}_{\mathbb{R}}(s, \bigwedge_{v \in I} v = \nu(v))$.

3.3.2 Fixed-Point Representation

A *fixed-point type* is a triple $\langle s, n, m \rangle$ consisting of a *sign bit* $s \in \{\mathtt{s}, \mathtt{u}\}$ (for *signed* and *unsigned*), a *length* $n \in \mathbb{N}$, and a *fractional part* $m \in \mathbb{N}$. A fixed-point type $\langle s, n, m \rangle$ interprets a bitvector of n bits as a rational number in the following way. A bitvector $b = b_{n-1} \ldots b_0$ of type $\langle \mathtt{u}, n, m \rangle$ represents the rational number

$$\frac{1}{2^m} \sum_{i=0}^{n-1} 2^i b_i$$

i.e., the bitvector represents a rational number where the last m bits are used for the fractional part, and the top $n-1-m$ bits for the integer part. The *range* of possible values for the type $\langle \mathtt{u}, n, m \rangle$ is the set $U_{n,m} = \{\frac{p}{2^m} \mid p \in \mathbb{N}, 0 \leq p \leq 2^n - 1\}$.

The bitvector $b_{n-1} \ldots b_0$ of type $\langle \mathtt{s}, n, m \rangle$ represents the rational number

$$\frac{1}{2^m} \left[-2^{n-1} b_{n-1} + \sum_{i=0}^{n-2} 2^i b_i \right]$$

Thus, a signed fixed-point number represents a positive, zero, or negative rational

number. The *range* of possible values for the type $\langle \mathtt{s}, n.m \rangle$ is the set $S_{n,m} = \{\frac{p}{2^m} \mid p \in \mathbb{Z}, -2^{n-1} \leq p \leq 2^{n-1} - 1\}$. The most significant bit b_{n-1} of a fixed-point number of type $\langle \mathtt{s}, n, m \rangle$ is called the *sign* bit.

Representation of constants. Since not all real numbers can be represented using fixed-point types, we represent fixed-point *approximations* to real numbers. Let $\langle \mathtt{u}, n, m \rangle$ be a fixed-point type, and let $0 \leq c < 2^{n-m}$ be a real number. The *representation* $\mathsf{repr}(c) : \langle \mathtt{u}, n, m \rangle$ of c is the number $\frac{p}{2^m}$ for $0 \leq p \leq 2^n - 1$ for which $c - \frac{p}{2^m}$ is minimized. Clearly, $(c - r) \leq \frac{1}{2^m}$ for the representation r of c. Similarly, for the fixed-point type $\langle \mathtt{s}, n, m \rangle$ and a rational $-2^{n-1-m} \leq c < 2^{n-m-1}$, the *representation* $\mathsf{repr}(c) : \langle \mathtt{s}, n, m \rangle$ of c is the number $\frac{p}{2^m}$ for $-2^{n-1} \leq p \leq 2^{n-1} - 1$ for which $c - \frac{p}{2^m}$ is minimized. Again, $(c - r) \leq \frac{1}{2^m}$ for the representation r of c.

Let X be a set of variables and let typ be a function mapping each $x \in X$ to a fixed-point type. For a valuation $\nu \in \{\{X\}\}_{\mathbb{R}}$ mapping each variable in X to a real value, we write $\mathsf{repr}(\nu)$ for the fixed-point valuation that maps each variable x to $\mathsf{repr}(\nu(x))$ of type $\mathsf{typ}(x)$.

Arithmetic operations and assignments. Consider the typed variables $x : \langle \mathtt{u}, n_x, m_x \rangle$, $y : \langle \mathtt{u}, n_y, m_y \rangle$, and $z : \langle \mathtt{u}, n_z, m_z \rangle$, and the assignment $x := y + z$. In fixed-point arithmetic, the addition must be performed by first scaling the bitvectors y and z so that their binary points align. Moreover, the addition can result in a number with $\max\{n_y, n_z\} + 1$ bits, and this number must be scaled to fit into the bitvector x. Table 3.2 shows the sequence of bitvector operations to perform these steps.

We consider the case $m_y \leq m_z$ and $n_y \leq n_z$, the other cases are similar. First, we multiply y by $2^{m_z - m_y}$ to align the binary points of the two operands (variable t_1 in Table 3.2). Second, the variables t_1 and z are added as bitvectors. This creates a number t_2 with m_z fractional bits, and $n_z - m_z + 1$ integer bits. These bits must be "fitted" into the n_x bits of x. Assuming $m_x \leq m_z$ (the other cases

Command	Semantics	Assumptions
$x : \langle \mathtt{u}, n_x, m_x \rangle := c$	$x := \mathsf{repr}(c) : \langle \mathtt{u}, n_x, m_x \rangle$	$0 \leq c < 2^{n_x - m_x}$
$x : \langle \mathtt{s}, n_x, m_x \rangle := c$	$x := \mathsf{repr}(c) : \langle \mathtt{s}, n_x, m_x \rangle$	$-2^{n_x - m_x - 1} \leq c < 2^{n_x - m_x - 1}$
$x : \langle \mathtt{u}, n_x, m_x \rangle :=$	$t_1 : \langle \mathtt{u}, n_z + 1, m_z \rangle := \mathsf{shl}(y, m_z - m_y);$	$m_y \leq m_z$, $n_y \leq n_z$ $m_x \leq m_z$
$\quad y : \langle \mathtt{u}, n_y, m_y \rangle \pm z : \langle \mathtt{u}, n_z, m_z \rangle$	$t_2 : \langle \mathtt{u}, n_z + 1, m_z \rangle := t_1 \pm z;$	
	$t_3 : \langle \mathtt{u}, n_z + 1, m_x \rangle := \mathsf{shr}(t_2, m_z - m_x);$	
	$x : \langle \mathtt{u}, n_x, m_x \rangle := \mathsf{lsb}(t_3, n_x);$	
$x : \langle \mathtt{u}, n_x, m_x \rangle :=$	$t_1 : \langle \mathtt{u}, n_y + n_z, m_y + m_z \rangle = y * z;$	$m_x \leq m_y + m_z$
$\quad y : \langle \mathtt{u}, n_y, m_y \rangle * z : \langle \mathtt{u}, n_z, m_z \rangle;$	$t_2 : \langle \mathtt{u}, n_y + n_z, m_x \rangle = \mathsf{shr}(t_1, m_y + m_z - m_x);$	
	$x : \langle \mathtt{u}, n_x, m_x \rangle = \mathsf{lsb}(t_2, n_x)$	

Table 3.2: Semantics of unsigned fixed-point operations in terms of bitvector operations. The other cases are symmetric. The shr and shl operators are bitvector shift-right and shift-left operations, and $\mathsf{lsb}(x, k)$ picks the lower order k bits of a bitvector x.

are similar), we truncate the m_z fractional bits by right-shifting the bitvector $m_z - m_x$ bits and storing the result in t_3 (thus keeping m_x bits of precision for the fractional part). Finally, we copy the lower n_x bits of t_3 into the bitvector x. The subtraction operation has to consider two cases. If x is greater than z, in which case the operation proceeds similar to addition. If x is less than z, then by 2's complement arithmetic, a 2^n term is added to the result.

While we can give a direct semantics for signed arithmetic operations using signed bitvector operations, it is conceptually easier to give the semantics for signed numbers by reduction to unsigned ones. We do this by separately tracking the sign and the magnitude of a signed number, performing the operations on the magnitudes using unsigned arithmetic, and finally putting the appropriate sign bits back.

To perform fixed-point multiplication of y and z, we multiply y and z as signed or unsigned integers to get a bitvector with $n_y + n_z$ bits in the unsigned case (and $n_y + n_z + 1$ bits in the signed case) of which $m_y + m_z$ bits represent the fractional

part. These bits are "fitted" into x by first truncating the fractional part to keep m_x bits of precision, and then copying the lower n_x bits into x (and copying the sign bit into the sign bit of x). The corresponding bitvector operations are shown in Table 3.2.

3.3.3 Fixed-point Semantics

A fixed-point program $P = (I, L, O, \mathsf{typ}, s)$ consists of sets I, L, O of *input*, *local*, and *output* variables respectively, a type-map typ from $I \cup L \cup O$ to fixed-point types, and a body s defined by the grammar (3.12). That is, a fixed-point program is syntactically identical to a real-valued program, but each variable is interpreted as a fixed-point type rather than a rational number. We assume that programs are well-typed in that arithmetic operations do not mix signed and unsigned types.

For a set X of fixed-point variables, an X-fixed-point valuation is a function that maps each variable $v \in X$ with $\mathsf{typ}(v) = \langle \mathtt{u}, n, m \rangle$ (respectively, $\mathsf{typ}(v) = \langle \mathtt{s}, n, m \rangle$) to a value in $U_{n,m}$ (respectively, $S_{n,m}$). We write $\{\{X\}\}$ for the set of all X-fixed-point valuations. When the type of variables (rational or fixed-point) is clear from the context, we omit "fixed-point" and simply write valuation. A fixed-point program state is an $I \cup L \cup O$-fixed-point valuation. Notice that fixed-point programs are finite-state, since the possible set of fixed-point valuations to all the fixed-point variables in a program is finite.

The semantics of a fixed-point program is given using the strongest post-condition operation SP mapping a body s and a first-order formula θ to a first-order formula $\mathsf{SP}(s, \theta)$. The definition of SP is identical to $\mathsf{SP}_{\mathbb{R}}$ for the constructs ;, $[\![]\!]$, and $\mathtt{assume}(e)$. The strongest postcondition operation for assignments is defined using the semantics of arithmetic operations and assignments given in Table 3.2.

A fixed-point program P defines a relation $[\![P]\!]$ between I-valuations and O-

valuations: the pair $(\nu, \mu) \in [\![P]\!]$ if $\nu \in \{\{I\}\}$ is an I-valuation, $\mu \in \{\{O\}\}$ is an O-valuation, and μ satisfies $\exists w \in L.\mathsf{SP}(s, \bigwedge_{v \in I} v = \nu(v))$.

3.3.4 Symbolic Error Analysis

We now formally define the error between a real-valued program and its fixed-point implementation.

Definition 2. *The* implementation-error decision problem *asks, given a real-valued program $P = (I, L, O, \mathsf{rng}, s)$, a fixed-point program $P' = (I, L', O, \mathsf{typ}, s')$, and an error bound $\epsilon > 0$, is it true that for every $\mu \in \{\{I\}\}_{\mathbb{R}}$ satisfying the rng constraints, we have $\|\nu - \nu'\| < \epsilon$ for every pair $\nu \in \{\{O\}\}_{\mathbb{R}}$, $\nu' \in \{\{O\}\}$ satisfying $[\![P]\!]_{\mathbb{R}}(\mu, \nu)$ and $[\![P']\!](\mathsf{repr}(\mu), \nu')$? (Note that P and P' have the same set of input and output variables, but can have different local variables and different bodies.)*

Intuitively, for an input valuation μ satisfying the rng constraints, we run P on μ and P' on $\mathsf{repr}(\mu)$ and compare the results. The answer to the implementation-error decision problem is "yes" if the norm of the difference of the outputs between the programs is less than ϵ.

We now show that the implementation-error decision problem reduces to the satisfiability problem for the combination theory of bitvectors and reals with addition and multiplication. For a variable $v \in I \cup O$, we write v as the real-valued variable in P and $\mathsf{fp}(v)$ for the fixed-point version of v in P'. For a formula φ and a set X of variables, we write $\varphi[\mathsf{fp}(X)/X]$ for the formula obtained by replacing

each variable $x \in X$ appearing φ with $\mathsf{fp}(x)$. Consider the following formula:

$$\begin{array}{ll}
\bigwedge_{v \in I} v \in \mathsf{rng}(v) \wedge & \text{(a)} \\
\bigwedge_{v \in I : \mathsf{typ}(v) = \langle \cdot, n, m \rangle} |v - \mathsf{fp}(v)| \leq \frac{1}{2^m} \wedge & \text{(b)} \\
\mathsf{SP}_{\mathbb{R}}(s, \mathit{true}) \wedge & \text{(c)} \\
\mathsf{SP}(s', \mathit{true})[\mathsf{fp}(I \cup L' \cup O)/(I \cup L' \cup O)] & \text{(d)} \\
\Rightarrow & \\
\quad \|\mathsf{o} - \mathsf{fp}(\mathsf{o})\| \leq \epsilon & \text{(e)}
\end{array} \tag{3.13}$$

where o denotes the vector of all elements of O. In the formula, (a) encodes the rng constraints on the variables in I, (b) encodes the quantization error in the inputs, (c) and (d) encode the semantics of the programs P and P' respectively, and (e) encodes that the output differences are bounded by ϵ. The answer to the implementation-error decision problem is "yes" iff the above formula is valid.

By naive enumeration, the bitvector constraints in the above formulas can be reduced to Boolean operations. Thus, the implementation-error decision problem reduces to a satisfiability question in the theory of reals with multiplication (naively, by enumerating all of the finitely many bits). The decidability of this latter theory [Tar51, Can88] implies the following theorem.

Theorem 1. *The implementation-error decision problem is in PSPACE.*

Note that in order to prove the error bound is *not* ϵ, we can (1) guess the bits in polynomial time, and (2) solve the resulting problem in the existential theory of reals with addition and multiplication [Can88]. This gives a PSPACE bound for the problems, using Savitch's theorem and closure of PSPACE under complementation.

In practice, instead of enumeration, in our implementation, we use efficient decision procedures for the combination theory.

While we assume that ϵ is given as an input, notice that we can approximate the minimal ϵ that bounds the error to any desired precision by using binary

search in a range.

Linear Approximation. The solution to the absolute-error problem uses a reduction to a decision procedure for *non-linear* arithmetic. In practice, these decision procedures are not as scalable as decision procedures for *linear* arithmetic. We now consider a special case of the problem for which we can reduce the error problems to problems in linear arithmetic.

A *linear* constraint is of the form $\sum_i c_i x_i \sim b$ where c_i and b are real numbers and $\sim \in \{\geq, \leq, =\}$. A linear formula is a Boolean combination of linear constraints. As *linear* real-valued program $P = (I, L, O, \mathsf{rng}, s)$, every assignment statement in s is one of the forms $x := c$, $x := y \pm z$, or $x := c * y$ for variables x, y, z and real constant c, and in every assume statement $\texttt{assume}(e)$, we have e is a linear constraint.

Linear real-valued programs are an important special case: the implementation of a linear controller or a non-linear controller through lookup tables and linear interpolation is a linear real-valued program.

For linear real-valued programs, the constraints $\mathsf{SP}_{\mathbb{R}}(s, \mathit{true})$ is a formula in linear arithmetic. Thus, in the constraints (3.13), the antecedent is a formula in the combination theory of linear arithmetic and bitvectors. The problem is that we use the Euclidean norm, which introduces non-linear terms.

We define the *linear approximation* to the implementation-error problem in which we ask if the 1-norm of the output difference in less than ϵ, i.e., if $\|\nu - \nu'\|_1 < \epsilon$ in Definition 2 instead of the Euclidean norm. For linear real-valued programs, the linear approximation to the problem can be reduced to a decision problem in the combination theory of linear arithmetic and bitvectors. This gives the following result.

Theorem 2. *For a linear real-valued program P and a fixed-point program P', the linear approximation to the implementation-error problem is coNP-complete.*

To show that an error bound is greater than ϵ, we can guess the bits and reduce the problem to the satisfiability question for linear arithmetic. Thus, the complement of the problem is in NP. coNP-completeness follows from the hardness of Boolean reasoning.

Notice that the linear approximation can be used to provide a conservative upper bound on the Euclidean norm, using the fact that $\|x\| \leq \|x\|_1$ for any vector x.

3.3.5 Arithmetic Encoding

While the bitvector semantics provides an "execution semantics" for fixed-point programs, we now give an alternate semantics to fixed-point programs by reduction to constraints over integers that is more suited to symbolic analysis. In the arithmetic semantics, we associate an integer variable $\hat{x}$ with each fixed-point variable x. The key idea is to simulate the bitvector operations on integers.

The fixed-point implementation of an arithmetic operation in $\{+, -, *\}$ involves the arithmetic operation, and optionally one or more shift operations and an lsb operation in the fixed-point code. In the integer encoding, after each core arithmetic operation, we separate the sign and magnitude of the result before applying shift and lsb (least significant bits) operations on it. Let t be the temporary variable representing the result of the arithmetic operation. We represent by $\mathsf{sgn}(t)$ the sign of t and by $\mathsf{mgn}(t)$ the magnitude of t. The magnitude $\mathsf{mgn}(t)$ can be considered an unsigned fixed-point number with the same length and the same number of fraction bits as t. Thus, we only need to simulate the bitvector operations on unsigned bitvectors using arithmetic operations (and using the Boolean structure to encode the different possibilities on the sign bits).

Now let x be an unsigned number. We simulate $\mathsf{shl}(x, \ell)$ by $\hat{x} * 2^{\ell}$ (note that ℓ is a constant, so this is multiplication by a constant). We simulate $\mathsf{shr}(x, \ell)$ by

$(\hat{x} - (\hat{x} \mod 2^\ell))/2^\ell$. We simulate $\mathsf{lsb}(x, \ell)$ by $\hat{x} \mod 2^\ell$. Note that the shift operations and the lsb operation does not modify the sign of the operand. So now we can determine the sign of the result of the whole arithmetic operation to be $\mathsf{sgn}(t)$. While $x \mod k$ is not directly implemented as an operation in a linear arithmetic decision procedure, the predicate $y = x \mod k$ is easily encoded as $\exists z.x = kz + y \wedge 0 \leq y < k$.

3.4 Extensions

We now indicate two extensions to the analysis in Section 3.2.

Sensor and Actuator Errors. The first extension deals with errors arising from sensors and actuators in the system.

Sensing errors e_{sense} can be defined as the mismatch $e_{sense} = \widehat{y}[n] - y[n]$ between the measured output $\widehat{y}[n]$ of the sensor and the true value $y[n]$ of the signal. Similarly, actuation errors are defined by the mismatch $e_{act} = \widehat{u}[n] - u[n]$ between the desired actuator value $u[n]$ and the actual value $\widehat{u}[n]$ enforced by the actuator. Bounds on these errors are readily available from the sensor and actuator specifications. Using e_{sense}, we can redefine the quantization error e_{q1} as:

$$e_{q1} = Q(Q(D_\tau)\widehat{x} + Q(E_\tau)Q(y + e_{sense})) - D_\tau\widehat{x} + E_\tau y.$$

Similarly, we can redefine the quantization error e_{q2} as:

$$e_{q2} = Q(Q(K)\widehat{x}) + e_{act} - K\widehat{x}.$$

With these new definitions for e_{q1} and e_{q2}, the analysis in Section 3.2.2 remains valid and the implemented controller is guaranteed to steer the state of the plant to the set of states x satisfying (3.11).

Nonlinear Systems. The analysis of Section 3.2 can be extended to nonlinear control systems by performing an analysis based on *Lyapunov functions*.

A Lyapunov function $V : \mathbb{R}^n \to \mathbb{R}_0^+$ satisfies $V(x) = 0 \implies x = 0$ and $\frac{\partial V}{\partial x} f(x, k(x)) \leq -\lambda V(x)$. It is known that the existence of a Lyapunov function implies that $x_0 = 0$ is an asymptotically stable equilibrium point.

We now illustrate how our analysis for implementation errors can be done for feedback controllers of the form $\upsilon = k(\xi)$ designed for a nonlinear control system $\frac{d}{dt}\xi = f(\xi, \upsilon)$. When designing controller k with the objective of rendering $x_0 = 0$ a globally asymptotically stable equilibrium point, a Lyapunov function for $\frac{d}{dt}\xi = f(\xi, k(\xi))$ is also designed. Moreover, if the controller is *robust*, a typical requirement for controller design, the following stronger inequality also holds:

$$\frac{\partial V}{\partial x} f(x, k(x) + e) \leq -\lambda V(x) + \sigma \|e\|^2.$$

Due to the implementation errors, the actuators receive the value:

$$u = Q(k(Q(x))) = k(x) + \varepsilon_{q3}$$

where the quantization error ε_{q3} is defined by:

$$\varepsilon_{q3} = Q(k(Q(x))) - k(x).$$

Hence, the time derivative of $V \circ \xi$ will be given by:

$$\frac{d}{dt} V \circ \xi = \frac{\partial V}{\partial x}\Big|_{x=\xi} f(\xi, k(\xi) + \varepsilon_{q3}) \leq -\lambda V \circ \xi + \sigma \|\varepsilon_{q3}\|.$$

Through integration we arrive at:

$$\begin{aligned} V \circ \xi(t) &\leq e^{-\lambda t} V \circ \xi(0) + \int_0^t e^{-\lambda(t-\tau)} \sigma \|\varepsilon_{q3}(\tau)\| d\tau \\ &\leq e^{-\lambda t} V \circ \xi(0) + \frac{\sigma b(\varepsilon_{q3})}{\lambda} \end{aligned}$$

where $b(\varepsilon_{q3})$ is an upper bound for $\|\varepsilon_{q3}\|$. Since:

$$\lim_{t \to \infty} V \circ \xi(t) \leq \sigma \frac{b(\varepsilon_{q3})}{\lambda}$$

the trajectories of the controlled system are guaranteed to converge to the set of states x defined by $V(x) \leq \sigma b(\varepsilon_{q3})/\lambda$. As expected, the implemented controller

still enforces asymptotic stability. However, in the presence of implementation errors, we can only guarantee that trajectories converge to a set containing $x_0 = 0$ and the size of this set decreases when the error ε_{q3} is reduced.

3.5 Evaluation

3.5.1 Implementation

We have implemented Costan, an automatic tool to compute the error bound between a mathematical control law and a fixed-point implementation for the control law. We model the mathematical control system in Simulink, and generate fixed-point controller code using Simulink's Fixed-Point Advisor and Real-Time Workshop. The real-valued program for the controller is extracted from the Simulink model of the controller as an imperative program.

The inputs to Costan consist of the mathematical model and the fixed-point implementation of the controller, a mapping between the input and output variable names used in the two descriptions, the ranges of values for the input variables, and the fixed-point types for each variable in the fixed-point implementation. Currently, the ranges for the input variables are determined by Matlab simulations. Ranges for the variables measured and computed by the controller can also be obtained by analyzing the mathematical models. Given a desired operating region for the physical system, the control designer can compute how much the physical variables will deviate from this region while converging to the desired equilibrium point. The possibility of computing such value is essentially a consequence of the second requirement in Definition 1 in Chapter 2. A similar analysis can then be performed for the observer in order to determine the range for all the variables in the control code. The fixed-point types can be obtained from Simulink Fixed-point advisor, or given as inputs to the Fixed-point code generator.

The core of Costan generates verification conditions as well as the constraints for the implementation-error decision problem shown in Equation (3.13). Loops are statically unrolled. In most cases, there is an explicit constant bound on the number of iterations. A few loops, e.g., those implementing binary search over a lookup-table to look up precomputed values, do not have an explicit constant bound in the code, but these bounds can be inserted from the (static) knowledge of the size of the lookup table.

Our tool uses the decision procedures Yices [DM06] and HySat [FHR07]. We use Yices to solve the linear approximation to the implementation-error problem for linear controllers. For nonlinear controllers, we use HySat. For a given ϵ, if the constraints from Equation (3.13) are not valid, we get a concrete test input that indicates where the mathematical controller and the implementation diverges. We use the integer-based encoding for fixed-point arithmetic over the bitvector implementation. This is because in our experiments (described below), the bitvector encoding scaled poorly.

Starting with a range for the error bound, Costan performs a binary search over the range to find out the smallest ϵ for which the difference between the outputs is less than ϵ. As a terminating condition we specify a margin on the range. If at any iteration the size of the range for ϵ goes below this margin then we terminate the algorithm deciding the upper bound of current range to be the error bound.

In most examples, the mathematical controller and the fixed-point implementation had identical syntactic structure. This allowed us to perform compositional analysis, in which we computed the best error bounds on all live variables going out of a block of code based on computed error bounds on all variables reaching the block of code. However, our analysis cannot be fully compositional, as the mathematical controller and the implementation can take different branches at a conditional. So, at conditionals, our analysis "fell back" to verification condition

Example	Error bound	Set size (ρ)	Run time
vehicle steering (16bit)	0.0163	0.0375	1m14.313s
pendulum (16bit)	0.0508	0.1806	2m36.409s
dc motor (16bit)	0.0473	1.0889	2m15.110s
train car - 1 car (32bit)	5e-7	2.6080e-5	3m25.478s
train car - 2 cars (32bit)	1.5e-6	9.4000e-5	5m39.607s
train car - 3 cars (32bit)	8.5e-6	0.0010	9m34.485s
train car - 4 cars (32bit)	3.351e-5	0.0080	10m9.179s
train car - 5 cars (32bit)	1.655e-4	0.0627	20m28.822s
jet engine[poly] (16bit)	4e-3	0.0230	0m0.551s
jet engine[3×8]	6.40	37.0431	0m34.636s
jet engine[5×10]	4.48	25.9296	0m34.293s
jet engine[7×14]	2.73	15.8009	1m6.981s
jet engine[21×21]	1.25	7.2348	18m15.794s
jet engine[21×101]	0.88	5.0933	50m23.127s
jet engine[100×100]	0.33	1.9100	103m19.977s

Table 3.3: Experimental Results

generation.

3.5.2 Experiments

We have applied Costan to a set of linear and nonlinear controller implementations to estimate the implementation errors and hence, using the analysis of Section 3.2, the region of asymptotic stability for the systems. Table 3.3 shows our experimental results on linear and nonlinear controllers, including the set $S = \{||x|| \leq \rho\}$ to where the state trajectories converge.

Linear Controllers. As the first example we choose the vehicle steering model from [AM09]. This model describes how to control the trajectory of a vehicle

through an actuator that causes a change in the orientation. It is possible to design a linear feedback controller that stabilizes the dynamics and tracks a given reference value r of the lateral position of the vehicle. The inputs to the controller are the reference r and two state inputs coming from the controlled system. The output of the controller is the control signal that goes as an input to the vehicle steering system. For the reference input we choose the range to be $[0, 100]$. We considered a 16-bit implementation. Our analysis finds that the absolute error for this fixed-point C code is bounded above by 0.0163.

We also demonstrate the scalability of the integer encoding over the direct bitvector encoding. Figure 3.2 shows how the time due to decision procedure varies with ϵ for bitvector-based and the integer-based encodings. The SAT-solving time grows rapidly near the "best" ϵ, and the bitvector analysis timed out in the range $[0.012, 0.8]$ while the integer encoding was stable.

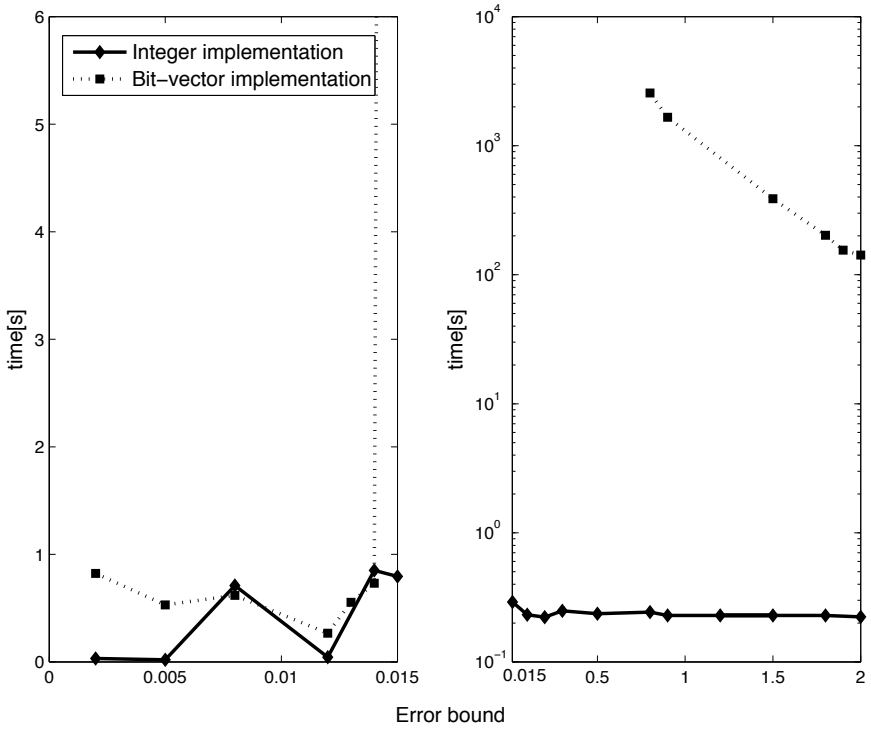


Figure 3.2: Decision procedure runtime for bitvector and integer implementation

We considered two other "textbook" linear control systems with feedback: the simple inverted pendulum [KA02] and the armature-controlled DC motor system [Zak03]. For the simple inverted pendulum, the controller has three inputs: one is the reference input and the other two are the state inputs. Our analysis reveals that the absolute error bound for a 16-bit implementation of the controller is 0.0508. The controller for the DC motor system has four inputs: the reference input and three state inputs. The error bound obtained for the DC motor controller output for a 16-bit implementation is 0.0473.

In the three examples described above the states of the system were directly measured. To consider a system where only some of the states can be measured, we revisit the example in Section 3.1. We scale up the example gradually by adding up to 5 cars. The measured states correspond to the velocities of the locomotive and the train cars. The states that cannot be measured correspond to the distance between the locomotive and the first car and the distances between any two cars. Hence, for n train cars we have $n + 1$ measured states and n states that are estimated using the measured states. The inputs to the controller are the measured states, while the control output, the force applied to the engine, is computed from the measured inputs and the estimated states. We consider 32-bit fixed-point representation for all the variables. The length of the fraction part of each variable is decided in such a way that no overflow occurs and the precision of the variable is maximized. Table 3.3 shows the error bounds for train-car controllers with up to 5 cars. The model with 5 cars (with 11 states) can be analyzed in about 20 minutes. The 32-bit implementation of the controller helps us achieve significantly small error bound.

While generating fixed-point code for a controller, a few times we have erroneously introduced overflow in some variables. Costan has been able to detect the overflow by detecting that the desired error bound is impossible to achieve. From the counterexample obtained from the decision procedure we were able to detect

the overflow in the implementation.

Nonlinear Controllers. We analyzed two implementations of a jet-engine controller from [KK95a]. There are two inputs, and the control law is a polynomial function of the inputs. One implementation directly evaluates the polynomial control law using fixed-point arithmetic. The other pre-computes the control law for various values of the input, and looks up the control action for an input from the "closest" point in the table. In both the cases we consider 16-bit implementation. When the controller is implemented as a lookup table, apart from the quantization error, error is also introduced due to the approximation of the output values in the lookup table. The error bound on the output of the controller based on lookup tables is dominated by the error in the lookup table, which depends on the size of the table. We used Costan to compute the error bounds on the control output for lookup tables of different size. Table 3.3 shows error bound obtained for different dimensions of the lookup table (e.g., the example "jet engine[3×8]" denotes a controller with a lookup table with $3 \times 8 = 24$ entries). For larger lookup tables, our implementation adopts a compositional strategy of dividing up the input range, computing output errors in each range, and then taking the maximum of the results. Calculation of the error bound on the single lookup table takes more than 2 hours for the lookup table with 2121 entries, while we can get the same error bound in less than one hour by dividing the lookup table into four smaller tables. The largest lookup table that we analyze has 10000 entries, we break it into 16 lookup table of equal size and successfully calculate the error bound in less than 2 hours. As might be expected, the direct polynomial evaluation with 16-bit arithmetic is more precise than even our largest lookup table implementation.

3.6 Related Work

While there has been a lot of work on static analysis of safety-critical control system implementation code [BCC03, Fer04, Cou05, GPB07, BGP09, FSI09, DGP09], the main emphasis in previous work has been on low-level properties such as arithmetic overflows or buffer overruns. By incorporating mathematical analysis of control design into our methodology, we can focus on *application level* properties such as stability. By putting the control designer "in the loop" we can move the complexity of some of this analysis from the static analysis tool to the mathematics of control. While we describe a program analysis based on verification-condition generation, analyses based on abstract interpretation, such as Astree [BCC03] and Fluctuat [GPB07, BGP09] will also be applicable to the analysis.

An alternate approach outlined in [FA08a, FA08b, AFP09] generates mathematical stability proofs and *compiles* such proofs into the implementation. We believe our methodology holds the advantage of *separation of concerns*. By forcing the implementation to conform to one of several possible stability proofs, we limit the space of implementation and optimization options. Instead, by producing a —more abstract— relationship between implementation errors and the regions where the physical variables can be steered to, we are free to explore different implementations. For example, in our experiments, we consider a non-linear control system with two different implementations: one based on a direct evaluation of a polynomial control law and a second based on the evaluation of the same polynomial using a lookup table and interpolation. Compiling the stability proof for the second option (which looks very different from the mathematical polynomial function) would be hard. Further, the compilation of stability proofs requires extensive changes to already complex auto-code generators to produce the proof of stability along with the implementation. Moreover, if controller implementations are written from scratch, the compilation strategy does not work, but our

methodology, being agnostic to the source of the controller, does.

CHAPTER 4

Synthesis

In this chapter, we shift our focus from verification to synthesis. For linear systems, a standard optimal control design approach uses the *linear quadratic regulator* (LQR) and *linear quadratic Gaussian* (LQG) algorithms [Hes09], which find a feedback controller stabilizing the plant while minimizing quadratic cost functions. The LQR cost function takes into account the deviations of the state and control inputs from ideal values and the LQG cost function takes into account the deviation of the state from its estimation. However, they usually do not take implementation errors arising from fixed-precision arithmetic into account. Thus, a controller optimizing only the LQR-LQG cost may have a large implementation error because its implementation on a fixed-precision platform has large numerical errors, but a controller "close" to the optimal performance may have much lower numerical errors when implemented on the same platform.

We present a methodology to modify the performance criterion of LQR-LQG to additionally minimize the error due to quantization in the implementation. Technically, we answer the following two challenges. First, how can we estimate the error due to quantization in a given implementation? Second, how can we find Pareto-optimal points for the two objectives given by the LQR-LQG and quantization error cost functions? We proceed as follows.

For the first step, for a given linear feedback controller and the operating intervals of the states of the plant and the controller, we first perform a precise range analysis of the controller variables, and use the computed ranges to allocate

bitwidths to each controller variable. We implement our range analysis based on linear programming. Using the allocated bitwidths, we generate code for a fixed-precision program implementing the control law. Finally, we use an algorithm based on mixed-integer linear programming to find a bound on the maximum difference between the ideal control law and the output of the fixed-precision program.

For the second step, we optimize a weighted linear combination of the two cost functions using a stochastic local search technique. LQR-LQG is attractive because it gives rise to a convex optimization problem, for which efficient solutions are known. Unfortunately, additionally tracking the quantization error results in a non-convex optimization problem. We solve the non-convex optimization problem using *particle swarm optimization* (PSO), a population-based stochastic optimization approach [KE95, LAS09, JLY07]. PSO iteratively solves an optimization problem by maintaining a population (or *swarm*) of candidate controllers, called *particles*, and moving them around in the search-space of possible controllers, trying to minimize the objective function. In our setting, a particle represents gain parameters for a controller.

In more detail, our algorithm proceeds as follows. Given a linear control design problem, we set up a non-convex optimization problem to minimize a weighted combination of the LQR-LQG cost function and the implementation error. We minimize this cost function using PSO. In each step of PSO, given a new controller, we perform the following checks. First, we check if the controller is stabilizing (by examining the eigenvalues of the controlled system). If not, we assign the controller an infinite cost. If it is stabilizing, we generate the best possible fixed-point code for this controller under a hardware budget and perform static analysis to estimate a bound on the implementation error. We compute the value of the objective function by taking the weighted sum of the LQR-LQG cost and this bound. We continue PSO until convergence or until some iteration bound is met.

At this point, we output the controller that minimized the objective function.

We have implemented this methodology on top of Matlab's Control Theory Toolbox, using an implementation of PSO proposed in [EKG12], and a custom static analysis using the lp_solve linear programming tool. In our experiments, we compare the LQR-LQG cost and implementation errors of controllers generated by conventional LQR-LQG optimization (implemented in Matlab) with controllers generated by PSO using our methodology. In most cases, our controllers have LQR-LQG costs close to the optimal LQR-LQG controllers, but have implementation errors that are reduced by a factor of 4 or more. Thus, we generate controllers with guaranteed bounds on practical stability regions that are 4 times or more smaller than the pure LQR-LQG optimal controllers. Our work provides an integrated analysis to take quantization errors into account in model-based design and implementation of controllers. While we have instantiated the methodology using the LQR and LQG costs and quantization errors, our algorithm is more generally applicable to other performance criteria and other sources of modeling or implementation error.

4.1 Stability of Perturbed Systems

4.1.1 The Effect of Errors

In this subsection we will show the effect of both external disturbance and measurement error, and the error due to implementation using fixed-precision arithmetic on the stability of the closed loop control systems. We extend the analysis in Section 3.2 to take into account the external disturbance and the measurement noise. Using a fixed-point implementation of the feedback gain as well as the

observer dynamic, one gets the following overall dynamics:

$$\begin{cases} x[r+1] = A_\tau x[r] - B_\tau K\widehat{x}[r] + \overline{B}_\tau d[r] + B_\tau e_{q2}, \\ \widehat{x}[r+1] = D_\tau \widehat{x}[r] + E_\tau C x[r] + E_\tau v[r] + e_{q1}, \end{cases} \tag{4.1}$$

where e_{q1} and e_{q2} are quantization errors in observer dynamic and feedback gain codes, respectively. Now, one can rewrite the control system in (4.1) as follows:

$$w[r+1] = Gw[r] + H_1 e_1[r] + H_2 e_2[r], \tag{4.2}$$

with:

$$w = \begin{bmatrix} x \\ \widehat{x} \end{bmatrix}, \quad e_1 = \begin{bmatrix} d \\ v \end{bmatrix}, \quad e_2 = \begin{bmatrix} e_{q2} \\ e_{q1} \end{bmatrix},$$

and:

$$G = \begin{bmatrix} A_\tau & -B_\tau K \\ LC & A_\tau - B_\tau K - LC \end{bmatrix}, \; H_1 = \begin{bmatrix} \overline{B}_\tau & 0_{n\times p} \\ 0_{n\times q} & L \end{bmatrix},$$
$$H_2 = \begin{bmatrix} 0_{n\times n} & B_\tau \\ I_n & 0_{n\times m} \end{bmatrix}.$$

The following proposition follows from Proposition 1 in Chapter 2 and describes the stability properties of linear control systems in (4.2) with respect to disturbance, measurement noise, and implementation errors in the feedback gain and observer dynamic.

Proposition 3. *Consider the discrete-time linear system in (4.2). For any input e_1 and e_2 satisfying $\|e_1[r]\| \leq b(e_1)$ and $\|e_2[r]\| \leq b(e_2)$ for any $r \in \mathbb{N}_0$ and some constants $b(e_1), b(e_2) \in \mathbb{R}_0^+$, the system is globally asymptotically stable with respect to the set:*

$$\mathcal{A} = \{x \in \mathbb{R}^n \mid \|x\| \leq \gamma_1 b(e_1) + \gamma_2 b(e_2)\},$$

where γ_1 and γ_2 are given by:

$$\gamma_j = \max_{\theta\in[0,\ 2\pi[} \left\| \left(\mathsf{e}^{i\theta} I_{2n} - G\right)^{-1} H_j \right\|, \quad \textit{for } j = 1, 2,$$

with $i = \sqrt{-1}$. *Moreover, the output* $y = [C \;\; 0_{p\times n}]\, w \in \mathbb{R}^p$ *is guaranteed to converge to the set:*

$$\mathcal{A}_y = \{y \in \mathbb{R}^p \mid \|y\| \leq \gamma_{1y} b(e_1) + \gamma_{2y} b(e_2)\}, \tag{4.3}$$

where γ_{1y} *and* γ_{2y} *are given by:*

$$\gamma_{jy} = \max_{\theta\in[0,\ 2\pi[} \left\| [C \;\; 0_{p\times n}] \left(\mathsf{e}^{i\theta} I_{2n} - G\right)^{-1} H_j \right\|, \quad \textit{for } j = 1, 2. \tag{4.4}$$

The error vector e_1 includes disturbance and measurement noise, depending for example on the environment and the quality of the sensors collecting measurements. Hence, the controller designer does not have any control on the value of $b(e_1)$. However, one can reduce the amount of γ_{1y} by appropriately choosing gains K and L. On the other hand, one can reduce the amount of not only γ_{2y} but also $b(e_2)$ by appropriately choosing gains K and L. We use Proposition 3 in the following way. Given a feedback gain K and an observer gain L, we compute $\mathcal{L}_2$ gains γ_{1y} and γ_{2y} and an upper bound $b(e_2)$ on the implementation error e_2. Then the output of the controlled system (with implementation error) must converge to set $\mathcal{A}_y$ in (4.3). We show later that appropriate choices of gains K and L can shrink the size of the set $\mathcal{A}_y$ and hence, provide a tighter bound on the set to which the output of the system converges.

4.1.2 Example

We now present a simple motivating example showing how different choices of controllers may result in different steady state errors due to their fixed-point implementations, yet providing approximately the same LQR-LQG performance.

Consider the following simple physical model of a bicycle, borrowed from [AM09]:

$$\begin{cases} \begin{bmatrix} \dot{\xi}_1 \\ \dot{\xi}_2 \end{bmatrix} = \begin{bmatrix} 0 & \frac{g}{h} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \xi_1 \\ \xi_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} (\upsilon + \omega), \\ \eta = \begin{bmatrix} \frac{av_0}{bh} & \frac{v_0^2}{bh} \end{bmatrix} \begin{bmatrix} \xi_1 \\ \xi_2 \end{bmatrix} + \nu, \end{cases} \tag{4.5}$$

where ξ_1 is the steering angular velocity, ξ_2 is the steering angle, η is the role angle, υ is the torque applied to the handle bars, $g = 9.8m/s^2$ is the acceleration due to gravity, $h = 1.5m$ is the height of the center of mass, $v_0 = 2m/s$ is the velocity of the bicycle at the rear wheel, $a = 0.5m$ is the distance of the center of mass from a vertical line through the contact point of the rear wheel and $b = 1m$ is the wheel base.

The control objective is to design a feedback gain $K \in \mathbb{R}^{1\times 2}$ and an observer gain $L \in \mathbb{R}^{2\times 1}$ such that the feedback control law $u = -K\widehat{x}$, where $\widehat{x} = [\widehat{x}_1,\ \widehat{x}_2]^T$ is the state of the observer in (2.5), makes the closed loop system globally asymptotically stable. By choosing the matrices $Q = I_2$ and $R = 1$ inside the LQR cost function and $\widehat{Q} = 1$ and $\widehat{R} = 1$ in (2.13), the feedback and observer gains minimizing the LQR and LQG costs are given by $K_1 = [5.1538,\ 12.9724]$, and $L_1 = [0.0317,\ 0.0118]^T$, respectively. Consider a second pair of feedback and observer gains given by $K_2 = [3.0253,\ 12.6089]$ and $L_2 = [0.0132,\ 0.1021]^T$. For the initial condition $x = (0.2,\ 0.2)^T$, the value of the LQR cost function is 264.1908 for feedback gain K_1 and 284.1578 for K_2. Moreover, the value of the LQG cost function is 0.0229 for observer gain L_1 and 0.0246 for L_2. So, the gains K_2 and L_2 give cost functions about 7% greater than the optimal gains K_1 and L_1.

We now show how different choice of feedback and observer gains result in different fixed-point implementation errors. For now, let us assume that $\omega(t) = 0$ and $\nu(t) = 0$, for any $t \in \mathbb{R}_0^+$. In Figure 4.1, we show the output of the closed-loop system starting from the initial condition $x = (0.2,\ 0.2)^T$, when the feedback gain and observer dynamic are implemented using 16-bit fixed-point representation.

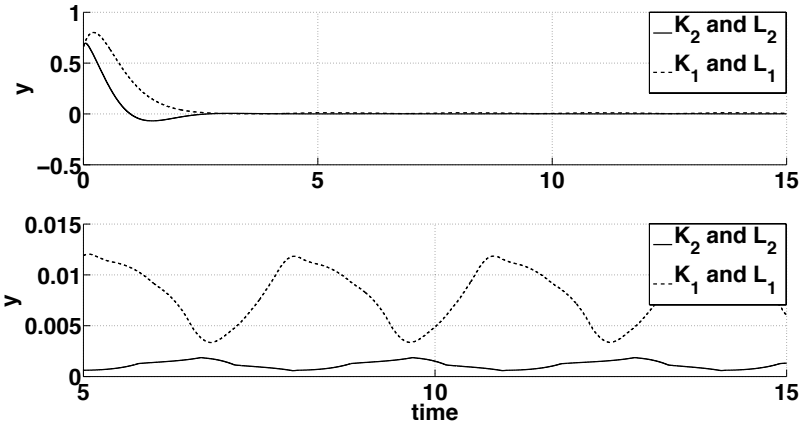


Figure 4.1: Evolution of the output y with initial state $(0.2,\ 0.2)^T$ for the pair of gains $(K_1,\ L_1)$ and $(K_2,\ L_2)$ using 16-bit implementation. Upper panel: evolution of y from 0 to 15 seconds. Lower panel: evolution of y from 5 to 15 seconds (magnified version).

As can be observed from Figure 4.1, the output of the controlled system does not converge to the equilibrium point at the origin because of the fixed-point implementation error in the controllers. Furthermore, the practical stability region using gains K_2 and L_2 is much smaller than the one using gains K_1 and L_1.

Using bounds on implementation errors for the two controllers (described in Section 4.2) and Proposition 3, we can prove that the output of the system with feedback and observer gains K_1 and L_1 (resp. K_2 and L_2) converges to a ball centered at the origin with radius 0.5486 (resp. 0.0513), whenever the output of the system and the state of the observer take values in the interval $[-1,\ 1]$ and the feedback gain and observer dynamic are implemented using 16-bit fixed-point implementation. As can be seen, given a 16-bit implementation, feedback and observer gains K_2 and L_2 may be preferred to gains K_1 and L_1 because they have guaranteed bounds on practical stability region that is 10 times smaller than gains K_1 and L_1 and provide approximately similar LQR/LQG performance. If one considers the effect of disturbance and measurement noise, it can be proved that

the output of the system with feedback and observer gains K_1 and L_1 (resp. K_2 and L_2) converges to a ball centered at the origin with radius $5.0489b(e_1)+0.5486$ (resp. $2.5341b(e_1) + 0.0513$), where $b(e_1)$ is an upper bound on the size of the vector e_1 introduced in (4.2).

4.2 Computing Quantization Error

In this section we show how to compute a bound on the fixed-point implementation error for given feedback and observer gains K and L. We assume that the outputs of the controlled system and the state of the observer are restricted to compact subsets $Y \subset \mathbb{R}^p$ and $\widehat{X} \subset \mathbb{R}^n$, respectively.

4.2.1 Best fixed-point implementation

An operation using real arithmetic may have different fixed-point implementations depending on how many bits are allocated to hold the integer part and the fraction part of the variables and intermediate results. Allocating fewer bits than required to hold the integer part may lead to overflow. On the other hand, if more than the required number of bits are allocated to the integer part, the quantization error increases due to assigning fewer bits to the fractional part. When we compare the fixed-point implementations of different expressions, we consider their best possible implementation, which we define next. Let us fix the number of bits to be v for the representation of every variable in the implementation of an expression. Then we define the best fixed-point implementation as follows:

Definition: **Best fixed-point implementation for given intervals.** For a given v and given intervals for the variables and intermediate results, an implementation I is called the *best fixed-point implementation*, if for every input variable or intermediate result that takes values from an interval $[r^{min}, r^{max}]$, the fixed-point representation is given by $\langle 1, v, w\rangle$, where $w = v - 1 - z$ and z, the

number of integral bits, is given by

$$z = \lceil \log_2(\max(\mathsf{abs}(r^{min}), \mathsf{abs}(r^{max}))) \rceil \tag{4.6}$$

For example, if the interval for a variable is [-35.55, 48.72], the representation for the variable in the best 16-bit fixed-point representation has $z = 6$ bits for the integer part, so it is given by $\langle 1, 16, 9 \rangle$. For a constant $C = 0.0864$ where $z = -3$, the representation is given by $\langle 1, 16, 18 \rangle$.

Note that the best fixed-point implementation depends on intervals assigned to intermediate variables. We say that intervals are tight if the intervals for internal nodes are as small as possible, which we can define by doing interval computation and fixed point allocation in parallel, as follows. Suppose we have an arithmetic operation $x = x_1 * x_2$, and we have tight intervals S_1 and S_2 associated with the variables x_1 and x_2. Let $S = \{x_1 * x_2 \mid x_1 \in S_1, x_2 \in S_2\}$ and let z and w be given by (4.6) taking $r^{min} = \inf(S)$ and $r^{max} = \sup(S)$. We then require the interval assigned to the node to be $[\mathsf{roundF}(r^{min}, v, w),\ \mathsf{roundF}(r^{max}, v, w)]$ where roundF denotes the rounding used in fixed-point computations when the representation is $(1, v, w)$.

A property of the above definitions is that, in the special case when the input intervals have lower bound equal to upper bound and are representable as fixed point numbers, then the tight intervals for intermediate nodes also have their lower bounds equal to their upper bounds, and are equal to the values of the sub-expression when evaluated in fixed-point arithmetic.

4.2.2 Error Bound Computation

In Chapter 3, we used a combination of decision procedures and binary search technique to compute the error bound. While decision procedures work for both linear and nonlinear control systems, the amount of time required by the decision procedures to compute the error bound is not small enough to be used for

controller synthesis. In this chapter we concentrate on synthesizing controllers for linear control systems, and apply a mixed-integer linear-programming-based optimization technique to find out the error bound between a computation in real arithmetic and its best fixed-point implementation. This technique is both precise and scalable for linear control systems.

Suppose we have an arithmetic operation $s : a = b\ op\ c$, where $op \in \{+, -, *\}$, where we assume that if $op = *$, then either b or c is a constant. If $op = +$ or $op = -$, then b and c can both be variables. We associate an integer variable $\widehat{x}$ with the fixed-point representation of a real variable x. Let the range of the values for a and b and c are $[l_a, u_a]$, $[l_b, u_b]$, and $[l_c, u_c]$, respectively. Let the fixed-point representation of a, b and c be $\langle \mathtt{1}, n_a, m_a \rangle$, $\langle \mathtt{1}, n_b, m_b \rangle$, and $\langle \mathtt{1}, n_c, m_c \rangle$, respectively. Let $b(e_b)$ and $b(e_c)$ be bounds on the quantization errors of b and c, respectively. The optimization problem to find the bound on the error is given by:

$$\begin{array}{rl} \text{maximize} & |a - 2^{-m_a}\widehat{a}| \\ \text{subject to} & l_a \leq a \leq u_a, \quad l_b \leq b \leq u_b \\ & \left|b - 2^{-m_b} * \widehat{b}\right| \leq b(e_b) \\ & |c - 2^{-m_c} * \widehat{c}| \leq b(e_c) \\ & a = b\ op\ c \\ & \Phi(\mathsf{fp}(s)) \end{array} \tag{4.7}$$

where $\mathsf{fp}(s)$ is the fixed-point representation of the statement s and $\Phi(\mathsf{s})$ denotes a logical formula that relates the inputs and outputs of the fixed-point representation s. Technically, Φ is the *strongest postcondition* [Win93] of s with respect to *true*. We compute Φ using an arithmetic encoding of a fixed-point computation as shown in Chapter 3. Here we illustrate the computation of the strongest postcondition Φ using an example.

Example. Suppose we have the following arithmetic operation

$$s :\ y = -7.2479\ *\ x\ .$$

Assume the compact set for x is [-1, 1]. The fixed-point expression corresponding to s in the best fixed-point implementation is

$$\mathsf{fp}(s) : -\widehat{y} = (-115 * \widehat{x}) \gg 6 \ .$$

The strongest postcondition $\Phi(\mathsf{fp}(s))$ of $\mathsf{fp}(s)$ is given by:

$$\begin{aligned}
\Phi(\mathsf{fp}(s)) := \quad & tmp = -115 * \widehat{x} \wedge \\
& tmp \geq 0 \rightarrow tmp1 = tmp \wedge \\
& tmp < 0 \rightarrow tmp1 = -tmp \wedge \\
& tmp1 = 2^6 * divisor + remainder \wedge \\
& remainder \geq 0 \wedge remainder < 2^6 \wedge \\
& tmp \geq 0 \rightarrow \widehat{y} = divisor \wedge \\
& tmp < 0 \rightarrow \widehat{y} = -divisor \ ,
\end{aligned}$$

where tmp, $tmp1$, $divisor$, and $remainder$ are integer variables.

Depending on the arithmetic operation, we need to solve at most four instances of mixed integer linear programming problems to solve the optimization problem in (4.7), and the maximum among all of these instances gives the bound on the error in the fixed-point implementation.

We use the above technique to compute the bound on the error in one operation in the fixed-point implementation of a gain. The implementation of a gain involves a series of arithmetic operations. We compute the error bound for the output of one arithmetic operation at a time. Let $s : a = b \ op \ c$ is an arithmetic operation in the implementation of a gain. In the arithmetic operation, b and c may either be a constant, a state variable or a temporary variable which captures the result of some previous operation. If b (or c) represents a constant, and the fixed-point representation contains m bits for the fraction part, then the error in the fixed point representation is bounded by $\frac{1}{2^m}$. If b (or c) represents a state variable, then the fixed-point datatype can be determined from the given compact set for the state, and the fixed-point datatype can be determined accordingly. Then the error in the fixed-point representation is bounded by $\frac{1}{2^m}$, where m is the number

of bits to represent the fraction part in the fixed-point datatype of the variable. If b (or c) is a temporary variable used to hold the result of an earlier computation, then the range and error bound for the variable are already known.

4.3 Optimal Controller Synthesis

We now describe our controller synthesis algorithm that minimizes a cost function combining LQR and LQG performance, disturbance, measurement noise, and implementation errors.

4.3.1 Optimization objectives

The example in Section 4.1.2 suggests that the control design should optimize for the following objectives: the LQR and the LQG costs for performance, error caused by disturbance and measurement noise, and the implementation error given by a fixed-precision encoding. Accordingly, we define a cost function that is weighted sum of the four factors:

$$\mathcal{J}(K,L) = w_1 \frac{\|S(K)\|}{\|S^*\|} + w_2 \frac{\|P(L)\|}{\|P^*\|} + w_3 \frac{\gamma_{1y}}{\gamma_{1y}^*} + w_4 \frac{\gamma_{2y} b(e_2)}{\gamma_{2y}^* b^*(e_2)}, \tag{4.8}$$

where $w_1, \ldots, w_4$ are weighting factors, S^* and P^* are matrices, computed from Lyapunov equations in (2.14) and (2.16) using standard LQR and LQG gains (K_{LQR} and L_{LQG}), γ_{1y} and γ_{2y} (resp. γ_{1y}^* and γ_{2y}^*) are the $\mathcal{L}_2$ gains in (4.4) using feedback and observer gains K and L (resp. K_{LQR} and L_{LQG}) and $b(e_2)$ (resp. $b^*(e_2)$) is the bound on the implementation error of given feedback and observer gains K and L (resp. K_{LQR} and L_{LQG}). Minimizing the terms γ_{1y} and $\gamma_{2y} b(e_2)$ inside (4.8) results in a tighter bound on the set $\mathcal{A}_y$ in Proposition 3. Since the four factors in (4.8) have different scales, we normalized them by their values using the standard gains K_{LQR} and L_{LQG}. The designer can choose $w_1, \ldots, w_4$ based on the priorities on LQR and LQG performances and steady state error. Our

objective is to find feedback and observer gains that minimize the cost function $\mathcal{J}$.

We focus on implementation errors arising out of fixed-precision arithmetic. The bound $b(e_2)$ is computed using the strategy explained in Section 4.2. Since the cost function $\mathcal{J}$ is not necessarily convex with respect to the feedback and observer gains K and L, we cannot reduce the design problem to a convex optimization problem. We use a heuristic stochastic optimization approach to find feedback and observer gains K and L minimizing $\mathcal{J}$.

In our exposition, we consider the plant model to be precise, and only consider quantization effects as the source of error. Our methodology can consider both additive and multiplicative uncertainties in the plant model as well [GL94]. We can take those uncertainties into account by adding appropriate extra terms to the cost function in (4.8), using the results provided in [ZSK09, ZKS07]. We omit the details for simplicity.

4.3.2 Particle Swarm Optimization

Since the cost function is non-convex, we use a stochastic local search technique approach called particle swarm optimization (PSO). It maintains a set of potential solutions (called "particles") in a compact d-dimensional search space $D = \prod_{j=1}^{d}[y^j_{\min}, y^j_{\max}] \subset \mathbb{R}^d$, minimizing a given cost function. The particles move in this space according to their velocity. Each particle, indexed by $i \in \mathbb{N}$, has a position $y_i \in \mathbb{R}^d$, changing between $y_{\min}$ and $y_{\max}$, and a velocity vector $v_i \in \mathbb{R}^d$, changing between some vectors $v_{\min}$ and $v_{\max}$. The terms $v_{\min}$ and $v_{\max}$ are often set to the maximum dynamic range of the variables on each dimension [ZKS09]: $-v^j_{\min} = v^j_{\max} = |y^j_{\max} - y^j_{\min}|$. Every particle remembers its own best position (i.e., the lowest value of the cost function achieved so far by this particle) in a vector $P_i \in \mathbb{R}^d$. The best position with respect to the cost function among

all of the particles so far is stored in a vector $P_g \in \mathbb{R}^d$.

PSO updates the positions and velocities of all particles iteratively. The new velocity and position for particle i are determined as:

$$v_i^{l+1} = w^l v_i^l + c_1 r_1 \left(P_i^l - y_i^l\right) + c_2 r_2 \left(P_g^l - y_i^l\right), \tag{4.9}$$

$$y_i^{l+1} = y_i^l + v_i^{l+1}, \tag{4.10}$$

where the superscript l denotes the iteration number, the subscript $i = 1, \ldots, N$ denotes the index of the particle, and N is the number of particles. The constant w^l in (4.9) is updated using the inertia weight approach [EKG12] as the following:

$$w^l = w_{\max} - \frac{w_{\max} - w_{\min}}{l_{\max}}(l-1), \tag{4.11}$$

where $w_{\max}$ and $w_{\min}$ are adjusted to 1 and $\frac{c_1+c_2}{2} - 1$ and $l_{\max}$ is the maximum number of iterations. The constants c_1 and c_2 in (4.9) are the acceleration constants, influencing the convergence speed of particles toward its own and global best positions and set to 0.5 and 1, respectively [EKG12]. The constants r_1 and r_2 in (4.9) are uniformly distributed random numbers on the interval $[0, 1]$.

4.3.3 Overall Algorithm

The PSO algorithm is used to search for feedback and observer gains $K \in \mathbb{R}^{m\times n}$ and $L \in \mathbb{R}^{n\times p}$ for the control system (4.1), minimizing (4.8). Note that a particle in PSO represents a feedback and an observer gain K and L, respectively, moving in an $m \times n + n \times p$ dimensional search space. To discard those gains that make the controlled system unstable, we penalize unstable gains by including a penalty term $\widetilde{P}$ in the cost function such that $\widetilde{P} = 0$ if $A_\tau - B_\tau K$ and $A_\tau - LC$ are Hurwitz and $\widetilde{P} = +\infty$ otherwise. The cost function for PSO is then $F(K, L) = \mathcal{J}(K, L) + \widetilde{P}(K, L)$.

The design steps are as follows:

(1) Initialize positions of N feedback gains K_i and observer gains L_i by K_{LQR}

and L_{LQG}, respectively, and uniformly randomly initialize their velocities, for $i = 1, \ldots, N$.

(2) Given any feedback gain K_i and observer gain L_i, compute the cost function $F(K_i, L_i)$. To compute $\widetilde{P}$, check if $A_\tau - B_\tau K$ and $A_\tau - LC$ are Hurwitz. There are some steps to compute $\mathcal{J}$. First, compute $S(K_i)$ and $P(L_i)$ by solving the Lyapunov equations (2.14) and (2.16), respectively, and find their induced 2-norm. Second, compute the $\mathcal{L}_2$ gains γ_{1y} and γ_{2y}. Third, compute $b(e_2)$ by solving the optimization problems from Section 4.2.

(3) Compare $F(K_i, L_i)$ to its own best position P_i so far and the global best position P_g so far. If $F(K_i, L_i)$ is less than the previous personal best (resp. the global best), update the best position (resp. the global best) to K_i and L_i.

(4) Modify the velocity and position of each pair K_i and L_i according to (4.9) and (4.10).

(5) If the number of iterations, denoted by l, reaches the maximum, denoted by $l_{\max}$, or the value of F does not change for the global best position P_g for 50 consecutive iterations up to error 10^{-6} then go to Step (6), otherwise go to Step (2);

(6) The latest P_g is an estimate for the optimal controller.

4.4 Extension: PID Controllers

PID controllers are a common class of controllers in many industries, such as automotive, power systems, servomotors, and so on. We now extend the analysis of Section 4.1.1 to PID controllers. A PID controller generalizes a proportional feedback controller, and includes three terms: a proportional term, an integrator,

and a differentiator. For an input υ, the output η of the PID controller is computed as follows:

$$\eta(t) = K_P \upsilon(t) + K_I \int_0^t \upsilon(s)ds + K_D \frac{d\upsilon(t)}{dt}, \quad \forall t \in \mathbb{R}_0^+, \tag{4.12}$$

where K_P, K_I, and K_D are called proportional, integrator, and differentiator gains, respectively. To describe the mismatch between the PID specifications and its software implementation, we consider the discrete-time version of (4.12). An integrator term:

$$\eta(t) = \int_0^t \upsilon(s)ds, \quad \forall t \in \mathbb{R}_0^+,$$

can be discretized based on the trapezoidal approximation as follows:

$$y[r+1] = y[r] + \frac{\tau}{2}\left(u[r+1] + u[r]\right), \quad \forall r \in \mathbb{N}_0, \tag{4.13}$$

where τ is the sampling time, $y[r] = \eta(r\tau) + e_1$ and $u[r] = \upsilon(r\tau)$, for any $r \in \mathbb{N}_0$. A common way of discretizing a differentiator, is based on the backward Euler method. A differentiator term:

$$\eta(t) = \frac{d\upsilon(t)}{dt}, \quad \forall t \in \mathbb{R}_0^+,$$

can be discretized as follows:

$$y[r+1] = \frac{u[r+1] - u[r]}{\tau}, \quad \forall r \in \mathbb{N}_0, \tag{4.14}$$

where $y[r] = \eta(r\tau) + e_2$ and $u[r] = \upsilon(r\tau)$, for any $r \in \mathbb{N}_0$. By using the fast sampling time assumption, we can ignore the errors e_1 and e_2 in the discretized versions of the integrator and differentiator in comparison with quantization errors. To follow the same analysis as in Section 4.1.1, we need a state space realization of PID controller. By resorting to control theoretic results (see, e.g., [Kai80]) and using the discretization rules in (4.13) and (4.14), the state space realization of discretized PID controller with input $\widehat{u}[r]$ and output $\widehat{y}[r]$ are obtained as follows:

$$\begin{cases} \widehat{x}[r+1] = \widehat{A}\widehat{x}[r] + \widehat{B}\widehat{u}[r], \\ \widehat{y}[r] = \widehat{C}\widehat{x}[r] + \widehat{D}\widehat{u}[r], \end{cases} \tag{4.15}$$

where

$$\widehat{A} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix},\ \widehat{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix},\ \widehat{C} = \begin{bmatrix} \frac{K_D}{\tau} & K_I\tau - \frac{K_D}{\tau} \end{bmatrix},$$
$$\widehat{D} = \left(K_P + \frac{K_I\tau}{2} + \frac{K_D}{\tau} \right).$$

Without loss of generality, consider a single-input ($m = 1$) single-output ($p = 1$) discrete-time linear control system of the form:

$$\begin{cases} x[r+1] = Ax[r] + Bu[r], \\ y[r] = Cx[r]. \end{cases}$$

Since the input of the PID controller is equal to the negative of the output of the plant ($\widehat{u} = -y$) because of negative feedback and the output of the PID controller is equal to the input of the plant ($u = \widehat{y}$), one obtains:

$$\begin{cases} x[r+1] = \left(A - B\widehat{D}C \right) x[r] + B\widehat{C}\widehat{x}[r], \\ \widehat{x}[r+1] = -\widehat{B}Cx[r] + \widehat{A}\widehat{x}[r]. \end{cases} \tag{4.16}$$

Similar to what has been explained in Section 4.1.1, by fixed-point implementation of the PID controller, one gets the following overall dynamic:

$$\begin{cases} x[r+1] = \left(A - B\widehat{D}C \right) x[r] + B\widehat{C}\widehat{x}[r] + Be_{q2}, \\ \widehat{x}[r+1] = -\widehat{B}Cx[r] + \widehat{A}\widehat{x}[r] + e_{q1}, \end{cases} \tag{4.17}$$

where e_{q1} and e_{q2} are quantization errors in computing the PID controller. Now, we can use the same strategy, as explained in Subsection 4.3.3, to design parameters K_P, K_I, and K_D of PID controllers minimizing a performance-based cost function as well as the effect of quantization error. For example, one can consider:

$$\mathcal{J}(K_P, K_I, K_D) = \frac{w_1}{\mathrm{PM}} + \frac{w_2}{\mathrm{GM}} + w_3\gamma(b(e_{q1}) + b(e_{q2})), \tag{4.18}$$

where PM and GM are phase and gain margins, w_1, w_2, w_3 are weighting factors, γ is the $\mathcal{L}_2$ gain of the linear control system (4.17) and $b(e_{q1})$ and $b(e_{q2})$ are the bounds on the implementation errors e_{q1} and e_{q2}. Note that control over PM and

Control systems	# bits	Synthesized gains		Time cost
		K	L	
Bicycle	16	[3.0253 12.6089]	$[0.0132 \;\; 0.1021]^T$	1h36m41s
DC motor position	16	[0.1129 0.0211 0.0093]	$[0.0390 \;\; 0.3700 \;\; -0.0175]^T$	1h39m06s
Pitch angle control	32	[-0.1202 42.5655 1.0001]	$[0.0001 \;\; 0.0000 \;\; 0.0017]^T$	8h31m53s
Inverted pendulum	32	[-1.5362 -2.0254 16.5192 2.7358]	$\begin{bmatrix} 0.0017 & 0.0021 & 0.0012 & 0.0000 \\ 0.0001 & 0.0018 & 0.0122 & 0.0770 \end{bmatrix}^T$	9h54m17s
Batch reactor process	16	$\begin{bmatrix} 0.0583 & 0.9093 & 0.3258 & 0.8721 \\ -2.4638 & -0.0504 & -1.7099 & 1.1653 \end{bmatrix}$	$\begin{bmatrix} 0.0774 & -0.0022 & 0.0267 & 0.0356 \\ -0.0103 & 0.0227 & 0.0398 & 0.0001 \end{bmatrix}^T$	3h08m29s

Table 4.1: Synthesized gains and required time for synthesizing them.

Control systems	*lub* of LQR cost		LQG cost		Steady state error	
	LQR	Synthesized K	LQG	Synthesized L	LQR-LQG	Synthesized gains
Bicycle	$3956.3\|x\|^2$	$4331.7\|x\|^2$	0.0229	0.0246	$5.0489b(e_1)$+0.5486	$2.5341b(e_1)$+0.0513
DC motor position	$1001.6\|x\|^2$	$1376.7\|x\|^2$	36.6315	36.6731	$30.566b(e_1)$+0.16	$15.421b(e_1)$+0.011
Pitch angle control	$2.9732 \times 10^6\|x\|^2$	$2.9887 \times 10^6\|x\|^2$	0.0013	0.0018	$2.6781b(e_1)$+0.4746	$1.4453b(e_1)$+0.0807
Inverted pendulum	$4.2988 \times 10^4\|x\|^2$	$5.3471 \times 10^4\|x\|^2$	0.3600	0.3897	$83.4217b(e_1)$+0.0432	$30.3801b(e_1)$+0.0086
Batch reactor process	$223.1773\|x\|^2$	$223.1825\|x\|^2$	0.0731	0.0949	$2.9309b(e_1)$+0.4194	$2.1216b(e_1)$+0.1642

Table 4.2: Least upper bound (*lub*) on the LQR cost (2.10), for a given initial condition x, the LQG cost (2.11), and the Euclidean norm of the steady state error for the LQR-LQG and the synthesized gains.

GM guarantees robust stability of the closed-loop systems [Hes09]. The phase and gain margins measure the system's tolerance to the time delay and the steady state gain, respectively.

4.5 Evaluation

4.5.1 Implementation

We have developed Ocosyn, a tool that implements the algorithm presented in Section 4.3.3 in Matlab. We use a PSO function in Matlab from [EKG12]. We implemented a static analyzer in OCaml that synthesizes the best fixed-point program and computes the bound on the fixed-point implementation error for

given feedback and observer gains K and L, respectively. The tool gets the number of bits in the fixed-point datatype, compact subsets $Y \subset \mathbb{R}^p$ and $\widehat{X} \subset \mathbb{R}^n$, and feedback and observer gains K and L, respectively, as inputs. The optimization problems in computing the error bound are solved using the mixed-integer linear programming tool lp_solve [LP]. All the experiments were done on a laptop with CPU Intel Core 2 Duo at 2.4 GHz.

4.5.2 Experiments

Linear Control Systems We have applied Ocosyn to a number of linear control systems. In all of the experiments, the number of particles in PSO is $N = 24$, the maximum number of iterations is $l_{\max} = 100$, and we choose the matrices $Q = I_n$ and $R = I_m$ in (2.10) and $\widehat{Q} = I_q$, and $\widehat{R} = I_p$ in (2.13). The value of $l_{\max}$ was chosen in such a way that appropriate gains are obtained in terms of the cost function (4.8) (or (4.18)) for all control systems. Moreover, we assume that the search space is $D = \prod_{i=1}^{n\times m+n\times p}[-150,\ 150] \subset \mathbb{R}^{n\times m+n\times p}$, which contains the standard LQR and LQG gains for all the examples. Further, we work on the compact subsets $Y = \prod_{i=1}^{p}[-1, 1] \subset \mathbb{R}^p$ and $\widehat{X} = \prod_{i=1}^{n}[-1, 1] \subset \mathbb{R}^n$. All constants and variables are expressed in SI units.

Our unstable examples include a bicycle [AM09], a DC motor position control [CMU], a pitch angle control [CMU], an inverted pendulum [CMU], a batch reactor process [GL94], and another inverted pendulum for PID synthesis [CMU]. See Table 5.3 and 5.5 for experimental results. Note that for those examples for which a 32-bit implementation is chosen, the 16-bit one provides a stability region which is even larger than the range of the variables inside the controller. For all the examples except the batch reactor process, the weighting factors in (4.8) are chosen as $w_1 = w_2 = w_3 = 1$ and $w_4 = 5$. For the batch reactor process example, the weighting factors are chosen as $w_1 = w_3 = 1$, $w_2 = 2$, and $w_4 = 5$.

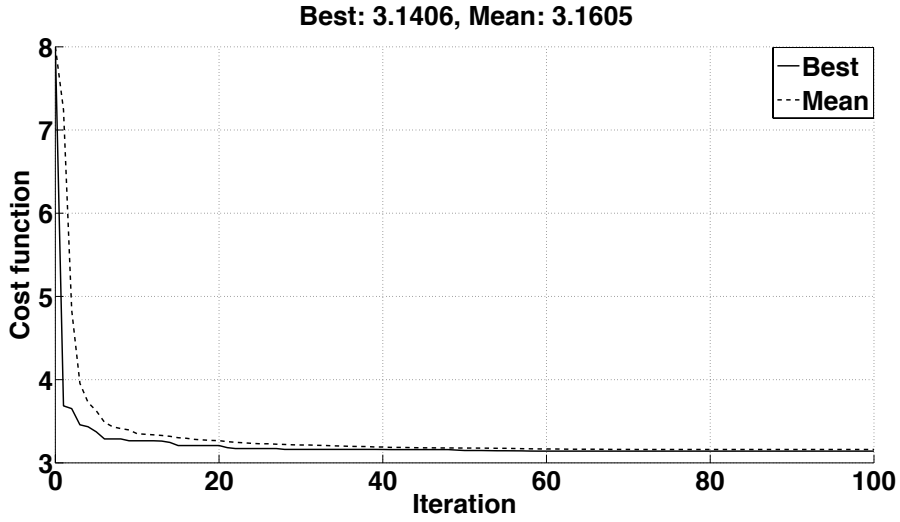


Figure 4.2: Cost of the best particle and average cost of all population vs iteration.

As can be seen from Table 5.5, in comparison with the conventional LQR-LQG approach, the synthesis approach proposed here worsens the LQR and LQG performances by at most 1.37 times (for DC motor position) and 1.38 times (for Pitch angle control), respectively. However, the proposed synthesis approach improves the size of the region of practical stability due to quantization error by at least 2.55 times. For certain examples, the improvement goes beyond the factor of 10. For the bicycle and DC motor position control, the region of practical stability due to quantization error improves by a factor of 10.69 and 14.55, respectively.

To assess the quality of the proposed stochastic search method, we run the algorithms 10 times for the bicycle model. The resulted standard deviation of the cost function $\mathcal{J}$ in (4.8) of all runs was 0.2806 which is around 9% of the best cost 3.1406. Figure 4.2 shows how the value of the cost function improves monotonically with the number of iteration for the best run. The fixed-point C code for the synthesized controller is shown in Figure 4.3.

PID controller In this example, we provide a PID controller for an inverted pendulum whose dynamic is given by a transfer function. Consider the transfer

```
float output(float yin)
{
    static int x1 = x1_0; // fixdt(1,16,14)
    static int x2 = x2_0; // fixdt(1,16,14)
    int x1_new;           // fixdt(1,16,14)
    int x2_new;           // fixdt(1,16,14)
    int u;                // fixdt(1,16,11)

    // Intermediate variables
    int Gain1;            // fixdt(1,16,15)
    int Gain2;            // fixdt(1,16,15)
    int Gain3;            // fixdt(1,16,15)
    int Add1;             // fixdt(1,16,14)
    int Gain4;            // fixdt(1,16,15)
    int Gain5;            // fixdt(1,16,15)
    int Gain6;            // fixdt(1,16,15)
    int Add2;             // fixdt(1,16,15)
    int Gain7;            // fixdt(1,16,13)
    int Gain8;            // fixdt(1,16,11)

    y = convert_to_fixedpoint(yin);
    Gain1 = (31499 * x1) >> 14;  Gain2 = (-3145 * x2) >> 14;  Add1 = (Gain1 + Gain2) >> 1;
    Gain3 = (432 * y) >> 14;  x1_new = ((Add1 << 1) + Gain3) >> 1;
    Gain4 = (-1907 * x1) >> 14;  Gain5 = (23835 * x2) >> 14;  Add2 = Gain4 + Gain5;
    Gain6 = (3345 * y) >> 14;  x2_new = (Add1 + Gain6) >> 1;
    Gain7 = (24783 * x1_new) >> 14;  Gain8 = (25823 * x2_new) >> 14;
    u = (Gain7 + (Gain8 << 2)) >> 2;
    return(float(u));
}
```

Figure 4.3: synthesized fixed-point controller C code for Bicycle.

function of an inverted pendulum, borrowed from [CMU], given by:

$$\frac{\Phi(s)}{U(s)} = \frac{\frac{ml}{q}s}{s^3 + \frac{b(I+ml^2)}{q}s^2 - \frac{(M+m)mgl}{q}s - \frac{bmgl}{q}}, \tag{4.19}$$

where $q = (M+m)(I+ml^2) - (ml)^2$, output ϕ is the angular position of the mass to be balanced, input v is the force applied to the cart, $g = 9.8$ is the acceleration due to gravity, $l = 0.3$ is the length of the rod, $m = 0.2$ is the mass of the system to be balanced, $M = 0.5$ is the mass of the cart, $b = 0.1$ is the coefficient of friction of the cart, and $I = 0.006$ is the moment of inertia of the pendulum. Using standard results in control theory [Kai80], one obtains the following state space realization for the inverted pendulum:

$$\begin{cases} \begin{bmatrix} \dot{\xi}_1 \\ \dot{\xi}_2 \\ \dot{\xi}_3 \end{bmatrix} = \begin{bmatrix} -0.1818 & 3.8977 & 0.5568 \\ 8.000 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} v \\ \phi = \begin{bmatrix} 0 & 0.5682 & 1 \end{bmatrix} \begin{bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{bmatrix}. \end{cases}$$

Our objective is to design PID gains K_P, K_I, and K_D minimizing the cost function (4.18) with weighting factors $w_1 = w_2 = w_3 = 1$ and such that the closed loop system has a settling time (t_s) of less than 5 seconds and such that the pendulum does not move more than 0.05 radians away from the vertical axis. The latter two constraints are treated the same as the stability constraint in Subsection 8.3.2 by penalizing the cost function (4.18). The synthesized gains are $K_P = 109.032$, $K_I = 1.2268$, and $K_D = 13.9945$. The closed loop system has $PM = +\infty$, $GM = 26237$, $\gamma(b(e_{q1}) + b(e_{q2})) = 4.1705 \times 10^{-4}$, settling time $t_s = 0.4790$, and ensures that the pendulum does not move more than 0.0098 radians away from the vertical axis.

4.6 Related Work

The results in [Wil85, Wil89, LSG92] provide controller synthesis approaches minimizing some performance criteria where controllers are implemented using fixed-point arithmetic. The results in [Wil85, Wil89, LSG92] assume some excitation conditions under which the quantization error can be modeled as a zero mean uniform white noise. Furthermore, they do not provide any bounds on regions of practical stability. Our results do not make any assumptions on the quantization error and provide an explicit bound on the region of practical stability.

Static analysis for range analysis has been studied extensively in the context of optimum bitwidth allocation to intermediate variables in a fixed-point program, mostly in the DSP domain [LGC06b, LCN07a, OCC07a]. These approaches employ abstractions based on interval arithmetic [Moo66] or affine arithmetic [SF97]. Jha [Jha11] gives an algorithm for optimal fixed-point program synthesis based on inductive synthesis. Jha's algorithm is general, but takes several minutes for each synthesis step. We found our mixed-integer linear programming approach to be both precise and reasonably fast for our application.

CHAPTER 5

Optimization

The precision of a fixed-point computation can depend on the order of evaluation of arithmetic operations. Since fixed-point arithmetic is not associative, and multiplication does not distribute over addition, the order in which a real polynomial is evaluated can cause differences in the error of the computation. Since the performance of controllers depends on the error introduced in the controller output, this difference can have significant impact on the performance of the controller. However, optimizing the error in the evaluation has received much less attention in fixed-point compilation, and has been limited to peephole optimizations (such as removing redundant shift operations locally) [AC00].

We present a technique to synthesize a fixed-point implementation for a given real-valued specification. Our synthesis method chooses the evaluation order of arithmetic operations to minimize the computation error. Given a real-valued arithmetic expression t, we aim to find a fixed-point implementation t', such that (1) the expressions t and t' are equivalent when interpreted over reals, and (2) the error between the real value and the fixed-point value computed by t' is minimal over all other fixed-point implementations equivalent to t. We show that the decision problem of finding an evaluation order that minimizes the error bound between the specification and the implementation is NP-hard, so a tractable *complete* search algorithm is unlikely.

Our technique is therefore based on a heuristic search implemented through genetic programming (GP) [PLM08]. We use the mutation and crossover opera-

tions of genetic programming to generate new sub-expressions. To evaluate the fitness of a proposed solution, we use a static analysis based on affine arithmetic to compute an upper bound on the error. The objective of the search is to minimize the upper bound computed by the static analysis. While our static analysis only computes an upper bound, we show, through extensive simulations, that the statically-computed upper bounds are proportional to the actual errors observed by simulations. We can thus use the less expensive upper bounds to compare two expressions with respect to precision.

We have implemented our technique on top of Ocosyn and we have evaluated it on a set of control application benchmarks. Our experiments demonstrate that our technique is adequate in finding good fixed-point implementations for linear controllers. For non-linear computations we encounter limitations in using static analysis based on affine-arithmetic, but our search method works with any technique to estimate variable ranges, so further improvements in this area can be incorporated into our approach.

5.1 Motivating Example

We motivate the problem using a controller for a batch reactor processor [Ros74]. The computation of a state of the controller is given by the following expression:

$$\begin{aligned}&(-0.0078) * st_1 + 0.9052 * st_2 \ + (-0.0181) * st_3 \ + \\ &(-0.0392) * st_4 \ + (-0.0003) * y_1 + 0.0020 * y_2\end{aligned} \tag{5.1}$$

where st_i is an internal state of the controller and y_i is an input to the controller. There are additional similar expressions to compute the other states and the outputs of the controller.

Consider a fixed-point implementation of this controller. If we assume an input range of $[-10, 10]$ for all input variables and a uniform bit length of 16, each input variable gets assigned the fixed-point format $\langle 1, 16, 11\rangle$. This means that of the

16 bits we use 1 bit to represent the sign of the number, 4 bits for the integer part ($10 < 2^4 = 16$), and the remaining 11 bits for the fractional part. The constant -0.0078 gets the format $\langle 1, 16, 22\rangle$ ($0.0078 < 2^{16-1-22} = 2^{-7} = 0.0078125$). If we multiply st_1 now by -0.0078, the result will have 33 bits, which we fit into 16 bits by performing a right shift. Following the order of arithmetic operations in (5.1) gives a fixed-point arithmetic program shown in Figure 5.1.

The fixed-point arithmetic implementation of the controller can have a large roundoff error. For example, because of the representation, the input values can already have an error as large as 0.00049. These errors then propagate throughout the computation. For a specific implementation, such as the one above, we can compute an upper bound on the error using an affine arithmetic-based static analysis. For our example, the maximum absolute error bound is 3.9e-3. We can bound the error from below using simulation, where we run the floating-point and the fixed-point programs side-by-side on a large number of random inputs and compare the results. Note that this technique only gives us an under-approximation. Using this approach, we get a lower bound on the error of 3.06e-3.

One way to reduce the error is to increase the bit length. If we add one bit to each variable, we get a simulated maximum error of 1.51e-3, which is an improvement by about 50%. However, increasing bit-widths means incurring more cost and may not be desirable.

A different possibility is to use a different order of evaluation for the expression. As fixed-point arithmetic operations are not associative, two different evaluation orders for the same implementation can have significantly different absolute errors. Consider the following reordering of Equation (5.1):

$$\begin{aligned}((0.9052 * st_2) + (((st_3 * -0.0181) \ + (-0.0078 * st_1)) + \\ (((-0.0392 * st_4) \ + (-0.0003 * y_1)) + (0.002 * y_2))))\end{aligned} \tag{5.2}$$

When implemented using 16-bit fixed-point arithmetic, we find, using our static analysis, that the maximum error bound is 1.39e-03, which is an even larger

```
tmp0 = ((-32716l * st1) >> 18)
tmp1 = ((29662l * st2) >> 15)
tmp2 = ((tmp0 + (tmp1 << 4)) >> 4)
tmp3 = ((-18979l * st3) >> 16)
tmp4 = (((tmp2 << 4) + tmp3) >> 4)
tmp5 = ((-20552l * st4) >> 15)
tmp6 = (((tmp4 << 4) + tmp5) >> 4)
tmp7 = ((-20133l * y1) >> 22)
tmp8 = (((tmp6 << 4) + tmp7) >> 4)
tmp9 = ((16777l * y2) >> 19)
tmp10 = (((tmp8 << 4) + tmp9) >> 4)
return tmp10
```

Figure 5.1: A possible fixed-point implementation for the example expression.

improvement of 55%, without requiring any extra hardware.

Our approach is to search the space of possible implementations of an arithmetic expression to find one that has the minimum fixed-point implementation error bound. We search the space using *genetic programming* (GP). GP finds this expression by evolving a population of expressions through selection of the best expressions with respect to the error, mutation and crossover. Figure 5.2 summarizes the worst-case error bounds for the different formulations of the expression. By exhaustively enumerating all possible rewrites, we see that the maximum error bounds can vary between approximately 1.39e-3 and 3.11e-3. That is, even for a relatively short example, the worst error bound can be over a factor of 2 larger than the best possible one. GP can find the optimal expression without an exhaustive enumeration. This does not cost any additional hardware, thus we get the additional precision "for free".

expression	simulated error
original	3.06e-3
worst rewrite	3.11e-3
additional bit	1.52e-3
best rewrite	1.39e-3
best found by GP	1.39e-3

Figure 5.2: Summary of absolute errors for different implementations

5.2 Generating Minimal-Error Fixed-Point Expression

In this section we describe our algorithm to solve the problem of synthesizing minimal-error fixed-point program for a given expression. An *expression* is generated by the following grammar:

$$t ::= v \mid x \mid t_1 + t_2 \mid t_1 - t_2 \mid t_1 * e_2 \mid t_1/t_2$$

where v and x are rational constants or variables, respectively. The fixed-point implementation of an expression then consists of assigning fixed-point representations to all input variables and intermediate results, i.e. to each node in the expression abstract syntax tree (AST).

Definition: **Worst-case error.** Assume a fixed-point implementation of an expression t. Given the values of variables, we define the expression error as $|t_r - t_f|$ where t_r is the value of t computed in real numbers, and t_f the value computed by the fixed-point implementation. Given the intervals for input variables of t, the *worst-case error* for a fixed-point implementation of t is the maximum over all expression errors where the values of variables range over the fixed-point representable values from the given intervals.

Given a real valued expression t we aim to find an expression t' that is mathematically equivalent to t and whose implementation in fixed-point arithmetic minimizes, among all equivalent expressions, the worst-case absolute error over

all inputs in given ranges I:

$$\min_{\text{equivalent } t'} \quad \max_{x \in I} \left| t_r(x) - t'_f(x) \right|$$

Our best expression search algorithm is based on *Genetic Programming*.

5.2.1 Genetic Programming

Genetic algorithms are heuristic search algorithms inspired by natural evolution. The algorithm evolves a population of candidate solutions by repeating the following steps for each new generation of solutions: from two candidates selected from the current generation new solutions are created by mutation and crossover whose quality is evaluated by a user-defined fitness function.

The candidate solutions are usually represented by strings so that these operations mimic closely natural evolution. Candidates for *mutation* and *crossover* are selected by *tournament selection* where a fixed number of candidates is chosen at random and the one with the highest fitness is selected as the final candidate.

Note that the problem domain can be very complex and that it does not need to have a gradient for guiding the search. Genetic programming [PLM08] is a variant of a genetic algorithm that performs the search over computer programs instead of strings. Mutation and crossover operators are thus defined on abstract syntax trees (ASTs).

Algorithm 5.2.1 gives an overview of our search procedure based on genetic programming. The input to our algorithm is a real valued expression and ranges for its variables. Our tool initializes the initial population with 30 copies of this expression and the search is repeated for 30 generations. We explain the steps of the algorithm in the following subsection.

Algorithm 5.2.1: Overview of the search procedure

```
1:Input: expression, input ranges
2:initialize population of 30 expressions

3: repeat for 30 generations
4:   generate 30 new expressions:
5:     select 2 expressions with tournament selection
6:     do equivalence-preserving crossover
7:     do equivalence-preserving mutation
7:     evaluate fitness (worst-case error bound)

8: Output: best expression found during entire run
```

5.2.2 Instantiating Genetic Programming

We start with (mathematically) correct program and instantiate the general genetic programming algorithm to find a program that is *numerically* as correct as possible (correctness is defined with respect to an evaluation of the expression in mathematical reals). Thus, our mutation and crossover operators need to generate expressions that are mathematically equivalent to the initial expression and the fitness function needs to quantify the numerical precision.

Mutation The mutation operator selects a random node in the expression AST and applies one of the applicable rewrite rules from Figure 5.3. The rules capture the usual commutativity, distributivity and associativity of real arithmetic. Some of these rules do not have an effect on the numerical precision by themselves, but are necessary to generate other rewrites of an expression. To keep the operations simple, we rewrite subtractions ($a - b \rightarrow a + (-b)$) and divisions ($a/b \rightarrow a * (1/b)$) before the GP run.

(1) (a + b) + c = a + (b + c)
(2) a + b = b + a
(3) (-a) + (-b) = -(a + b)
(4) (a * b) * c = a * (b * c)
(5) a * b = b *a
(6) (-a) * b = - (a * b)
(7) a * (-b) = - (a * b)
(8) 1/a * 1/b = 1/(ab)
(9) - (1/a) = 1/(-a)
(10) (a * b) + (a * c) = a * (b + c)
(11) (a * c) + (b * c) = (a + b) * c
(12) (a * b) + (c * a) = a * (b + c)
(13) (b * a) + (a * c) = (b + c) * a

Figure 5.3: Rewrite rules.

Crossover While maintaining mathematical equivalence is easy for the mutation operation, in the case of crossover it is not evident how to perform it efficiently in general. Given two trees t_1 and t_2 as candidates for the crossover, the genetic algorithm picks a random node in t_1, which is the root of the subtree s_1. The problem is then the following: find in an efficient way a subtree s_2 in t_2 that is mathematically equivalent to s_1. Instead of implementing a general decision procedure, we chose to do the following. At initialization, each subtree is annotated with a label that is the string representation of the expression at that subtree. During mutation, labels are preserved in the new generation as much as possible. For example, suppose we have the node $(a + b) + c$, with label `(a + b) + c`. We can apply mutation rule 1 to obtain $a + (b + c)$ but the label will remain `(a + b) + c`. Note that some of the mutation rules break equivalences (e.g. mutation rule 10), hence not all labels can be preserved. In that case we add a new label. During crossover, we then only need to check for identical labels. If labels match, it means that the subtrees come from the same initial subtree and hence are mathematically equivalent and we can exchange them in a crossover operation.

Parameters Our genetic programming pipeline has several parameters that can influence the results: the number of best individuals passed on to the next generation unchanged (elitism) (0, 2 or 6), the number of individuals considered during tournament selection (2, 4 or 6), and the probability of crossover (0.0, 0.5, 0.75 or

1.0). The most successful setting we found is with a tournament selection among 4 and an elitism of 2 while performing crossover every time, i.e. with probability of 1.0. Note, however, that even in the case of other settings, the improvements are still significant (on the order of 50%).

5.2.3 Fitness Evaluation

We use a static analysis based approach to compute the fitness of an expression. Our static analysis tool computes sound over approximations of the ranges of all variables and of the maximum absolute error of the corresponding best fixed-point implementation as introduced in Section 4.2.1.

Our tool uses affine arithmetic to compute the ranges of all intermediate values. From this we can determine the best possible fixed-point format and the quantization error at each computation step. The error bounds we compute are sound with respect to real arithmetic.

If we are interested in proving that the roundoff errors stay within certain bounds, the computed bounds on the absolute errors need to be as tight as possible. Note that the main requirement on the analysis in our problem is slightly different. While tight bounds on errors are an advantage, what we need to know is the *relative precision* of our analysis tool. That is, we need to know whether the analysis tool is able to distinguish a better implementation from a less precise one. To see why this is different from the usual case, note that the analysis tool assumes worst-case errors at each computation step. It general, however, the worst-case errors will not be attained at all computation steps.

Thus, before using our analysis tool in a GP framework, we evaluate this property experimentally. We generate a number of random rewrites for an expression, for which we then obtain the actual errors by simulation. We present here the results for one linear and one nonlinear benchmark (batch controller [GL94], state

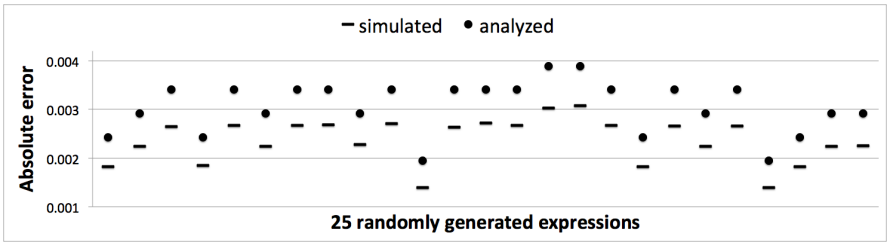


Figure 5.4: Comparison of analyzed upper bound and simulated lower bound on maximum errors for the linear benchmark batch processor (state 2).

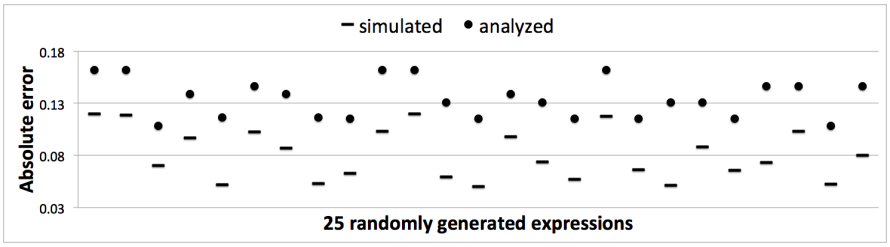


Figure 5.5: Comparison of analyzed upper bound and simulated lower bound on maximum errors for the nonlinear benchmark rigid body (out1).

2 and rigid body [AT10], output 1 respectively). For 100 random different expression formulations, the ratio between the analyzed upper bound on the error and the simulated lower bound on the error has a mean of 1.29387 and a variance of 0.00082 for the batch controller, state 2 benchmark and a mean of 1.66697 and a variance of 0.08315 for the rigid body, output 1 benchmark. Figures 5.4 and Figure 5.5 show a direct comparison between the analyzed and simulated errors. In the linear case, the computed bounds on the errors are proportional to the actual errors, thus indicating a good relative precision. In the nonlinear case the correspondence is not as precise, however we expect it to be still sufficient for our purpose. The "more nonlinear" a computation becomes, the less precise we expect affine arithmetic to be.

5.2.4 Why Genetic Programming?

It is in general not evident from an expression whether it is in a good form with respect to precision and exhaustively enumerating all possible formulations of expressions becomes impossible very quickly. For only linear expressions the number of possible orders of adding n terms modulo commutativity, which does not affect precision, is $(2n-3)!!$[1]. For our example from Section 5.1 with 6 terms there are already 945 expressions. For our largest benchmark with 15 terms there are too many possibilities to enumerate.

We thus need a suitable search strategy to find a good formulation of an expression among all the possibilities. We show in Section 5.4 that the problem of finding an expression whose worst-case error bound is minimal is NP-hard and that it amounts to minimizing the ranges of intermediate variables. Since the inputs for the expressions can, in general, be positive and negative, optimizing one subcomputation may lead to a very large intermediate sum in a different part of the expression. An algorithm that tries to find the optimal solution in a systematic way (e.g. dynamic programming) is thus unlikely to succeed. Our problem also does not have a notion of a gradient and it cannot be easily formulated in terms of inputs and outputs or constraints.

Genetic programming does not rely on any of these features, and its formulation as a search over program AST fits our problem very nicely.

5.3 Optimal Controller Synthesis

The controller for a discrete-time linear control system is given by Equation (2.5) and Equation (2.4). If we implement the controller using fixed-point arithmetic, we introduce additive error to the output of the controller. Thus the fixed-point

[1]The number of full binary trees with n leaves is C_{n-1}, where C_n are the Catalan numbers. We can label each of the trees in n! ways. Taking into account commutativity gives: $\frac{C_{n-1} \cdot n!}{2^{n-1}}$.

implementation of the controller is given by:

$$\begin{cases} \widehat{x}[r+1] = (A_\tau - B_\tau K - LC)\widehat{x}[r] + Ly[r] + e_{q1}, \\ \widehat{u}[r+1] = -K\widehat{x}[r+1] + e_{q2}, \end{cases} \tag{5.3}$$

where $e_{q1} \in \mathbb{R}^n$ and $e_{q2} \in \mathbb{R}^m$. The vector $e = \begin{bmatrix} e_{q1} \\ e_{q2} \end{bmatrix}$ captures the implementation error of the controller. As shown in Proposition 3 in Chapter 4, in the presence of implementation error the state of plant can only be shown to converge asymptotically in a set around the origin. The set is called the *region of practical stability*.

Controller synthesis has been traditionally performed by minimizing LQR and LQG costs [Hes09]. In Chapter 4, we showed how to synthesize a controller co-optimizing both the LQR/LQG cost and the region of practical stability. There we optimize a weighted linear combination of the two cost functions using a stochastic local search technique. The optimization problem is non-convex, and is solved using *particle swarm optimization* (PSO), a population-based stochastic optimization approach [KE95]. PSO iteratively solves an optimization problem by maintaining a population (or *swarm*) of candidate controllers, called *particles*, and moving them around in the search-space of possible controllers, trying to minimize the objective function.

In each step of PSO, given a new controller, a bound on the implementation error is estimated for a naive implementation. The value of the objective function is computed by taking the weighted sum of the LQR-LQG cost and this bound. Note that the implementation does not consider all possible expressions for a controller. If the controller is given by $K = [k_1,\ k_2, \ldots, k_n]$ and the state of the plant is denoted by $x = [x_1,\ x_2, \ldots, x_n]$, then the expression that is considered for the implementation of the fixed-point program is of the form $f = (((k_1x_1 + k_2x_2) + k_3x_3) + \ldots + k_nx_n)$. We refer to this expression as the *naive expression*. Note that the implementation of another expression may give a better bound on

the error. The expression giving the least bound for the error is referred to as the *best expression*.

We call the implemented controller corresponding to the naive expression the *baseline implementation*. The naive expression can be passed through the genetic algorithm based rewriting method to get the best expression, and the corresponding controller implementation is called the *improved implementation*. Let us suppose that the controller gains synthesized by the PSO method in Chapter 4 is denoted by K. Let us assume that the bound on the error in its baseline implementation is b^b_K, and that for its improved implementation is b^i_K. Now there may exist another controller K' with the bounds on the error in its baseline implementation and improved implementation to be $b^b_{K'}$ and $b^i_{K'}$ respectively such that $b^b_K < b^b_{K'}$, and $b^i_{K'} < b^i_K$. This implies that if we use the method in Chapter 4 to implement the baseline controller and then employ the genetic programming based best expression search strategy presented in Section 5.2 to get the improved controller, we may not obtain the best possible controller implementation.

To achieve the best possible implementation for a controller, we take the following strategy. In every step of PSO, for a given controller, we start with the naive expression and apply our genetic-programming-based rewriting technique to find the best expression. We use this bound on the implementation error of the best expression in the objective function. The controller implemented using this new objective function is referred as the *optimal implementation*. In the Evaluation section, we show that such combination of search strategies substantially improves the guaranteed properties of the synthesized controllers, showing that techniques for synthesis of fixed-point programs have concrete benefits for practical controller synthesis.

5.4 Optimal Fixed-Point Program Synthesis Problem is NP-hard

This section shows that given an arithmetic expression, the problem of finding a mathematically equivalent expression for which the computed worst-case error bound of the output of the fixed-point implementation is least is NP-hard. This justifies our use of a heuristic search method such as genetic programming. Though our algorithm supports operators '+', '-' , '*' or '/', we show the NP-hardness proof already for a problem that deals with the operator '+' alone.

For such an expression T, we define $E(T)$ as the *worst-case error bound of the best fixed-point implementation*, as follows.

Let the set of internal nodes of T be denoted by n_i and consider the best fixed-point implementation of T with tight intervals, as introduced in Subsection 4.2.1. As the worst case error e_i at node n_i we use

$$e_i = R(\max(\mathsf{abs}(r_i^{min}), \mathsf{abs}(r_i^{max})))/2^{v-1}$$

where R is a function used to make the bound uniform. A possible sound choice for R include s $R(x) = 2^{\lceil \log_2 x \rceil}$, which, according to the Definition (4.6) makes the error equal to the value of the least significant bit 2^{-w}. Another, slightly more conservative choice, is $R(x) = 2^{1+\log_2 x} = 2x$, which we adopt here.

Minimum-Error Fixed-point Set Range Sum (MEFxRS): Let $X = \{x_1, \ldots, x_p\}$ denote a set of variables, $x_i \in \mathbb{R}$. Given an expression of the form $\sum_{i=1}^{p} x_i$, where each variable x_i can take value from a range $[r_i^{min}, r_i^{max}]$, find the ordering of the addition operations that yields the minimal worst-case error bound of the best fixed-point implementation of the expression.

Our objective is to show that the problem MEFxRS is NP-hard. Towards that end, we first define a simplified problem where we compute the sum of a set of integers (instead of intervals). Note that the numbers may be both positive and

negative.

Minimum-Error Fixed-point Set Value Sum (MEFxVS): Let $X = \{v_1, \ldots, v_p\}$ denote a set of integers. Given an expression of the form $\sum_{i=1}^{p} v_i$, find out the ordering of the addition operations, that yields the minimal worst-case error bound at the output of the best fixed-point implementation of the expression.

It is straight-forward to show that an instance of a MEFxVS problem with values v_i can be reduced to a MEFxRS problem, for example, by letting $r_i^{min} = r_i^{max} = v_i$. In what follows, we show that MEFxVS is NP-hard.

To derive an expression for the worst-case error bound, we consider the AST of the expression, and in particular one internal node n_i representing a partial sum. Let the fixed-point value at n_i be c_i. To use our definition for the worst-case error bound, note that $r_i^{min} = r_i^{max} = c_i$. Then,

$$e_i = \frac{1}{2^{v-1}} R(|c_i|) = \frac{1}{2^{v-2}} |c_i|$$

Thus, at any internal node n_i, the error e_i is $\alpha|c_i|$ for $\alpha = 2^{-(v-2)}$. The errors at the leaf nodes are constant and we denote their sum by e_0. The worst-case error bound for the implementation tree T is thus given by $E(T) = e_0 + \alpha \sum_{i=1}^{p-1} |c_i|$ For a fixed number of overall bits v , e_0 and α is constant. The implementation error can thus be minimized by minimizing the partial sum at the internal nodes of an implementation tree. This sum is called the *cost of* T and is given by $C(T) = \sum_{i=1}^{p-1} |c_i|$ in [KW98].

Kao and Wang show in [KW98, Section 2] that the decision problem $C(T) \leq K$, where $C(T)$ is the cost of the implementation tree T and K is a positive integer, is NP-hard. Using that result we have the following theorem.

Theorem 3. *The Minimum-Error Fixed-point Set Range Sum (MEFxRS) problem is NP-hard.*

5.5 Evaluation

5.5.1 Implementation

We have implemented the algorithm to synthesize the optimal implementation of a controller presented in Section 5.3 on top of Ocosyn and we call the resulting tool Ocosyn+. Ocosyn+ incorporates genetic algorithm based expression rewriting in the search for the optimal controller using PSO. For implementing our genetic programming based expression search algorithm, we use ECJ [ECJ], an evolutionary computation framework written in Java. We use a PSO function in Matlab from [EKG12]. We have used the same setup for PSO as used in Chapter 4. The synthesis experiments were done on a laptop running Mac OS X version 10.7.4 with 2 GHz Intel Core i7 CPU and 8GB 1600MHz DDR3 Memory.

5.5.2 Experiments

We evaluate our technique on a number of benchmarks. Our benchmarks include a bicycle model [AM09], a DC motor position control [CMU], a pitch angle control [CMU], an inverted pendulum [CMU] and a batch reactor process [GL94]. The controllers for these system were taken from Chapter 4, which attempted to minimize the size of the region of practical stability by choosing a controller whose baseline fixed-point implementation has the best possible bound on the error among all controllers that stabilize the plant. To show the scalability of our tool we choose the example of a locomotive pulling a train car where the connection between the locomotive and the car is modeled by a spring in parallel with a damper [MPS76]. By increasing the number of cars, we can increase the dimension of the system. We also consider a nonlinear controller for a rigid body [AT10].

Each benchmark consists of one expression and ranges for its input parameters. We wish to minimize the error on the one output value it computes over all possible

Benchmark	err	$\frac{\text{orig.- no-cross}}{\text{orig.}}$	$\frac{\text{orig.- best}}{\text{orig.}}$	g
bicycle (out1)	2.66e-3	0.00	0.00	-
bicycle (state1)	2.53e-4	0.19	0.19	1
bicycle (state2)	1.82e-4	0.00	0.00	-
dc motor (out1)	1.06e-4	0.00	0.00	-
dc motor (state1)	2.77e-4	0.00	0.00	-
dc motor (state2)	3.75e-4	0.25	0.25	4
dc motor (state3)	1.27e-4	0.00	0.00	-
pendulum (out1)	8.09e-8	0.03	0.03	5
pendulum (state1)	5.13e-9	0.17	0.17	1
pendulum (state2)	6.11e-9	0.38	0.38	16
pendulum (state3)	5.14e-9	0.00	0.00	-
pendulum (state4)	4.97e-9	0.27	0.27	7
pitch angle (out1)	1.33e-7	0.18	0.18	4
pitch angle (state1)	4.26e-9	0.30	0.30	2
pitch angle (state2)	2.79e-9	0.00	0.00	-
pitch angle (state3)	3.81e-9	0.20	0.20	2
batch reactor (out1)	5.15e-4	0.00	0.00	-
batch reactor (out2)	1.28e-3	0.12	0.12	2
batch reactor (state1)	3.46e-4	0.15	0.15	1
batch reactor (state2)	2.77e-4	0.00	0.00	-
batch reactor (state3)	3.55e-4	0.26	0.26	2
batch reactor (state4)	4.11e-4	0.23	0.23	7
batch (out1)	4.53e-3	0.07	0.07	2
batch (out2)	1.10e-2	0.12	0.12	7
batch (state1)	1.95e-3	0.50	0.50	3
batch (state2)	1.94e-3	0.50	0.50	6
batch (state3)	2.25e-3	0.36	0.37	18
batch (state4)	1.97e-3	0.33	0.33	7

Table 5.1: Continued in the next page

Benchmark	err	$\frac{\text{orig.- no-cross}}{\text{orig.}}$	$\frac{\text{orig.- best}}{\text{orig.}}$	g
traincar 1 (out)	1.11e-4	0.09	0.09	2
traincar 1 (state 1)	1.98e-6	0.03	0.03	6
traincar 1 (state 2)	3.57e-7	0.25	0.25	16
traincar 1 (state 3)	2.79e-7	0.24	0.24	7
traincar 2 (out)	7.40e-5	0.09	0.09	19
traincar 2 (state 3)	1.23e-7	0.49	0.59	21
traincar 3 (out)	1.26e-3	0.13	0.13	7
traincar 3 (state 6)	1.32e-7	0.48	0.58	21
traincar 3 (state 7)	1.31e-7	0.43	0.53	17
traincar 4 (out)	9.34e-3	0.26	0.29	27
traincar 4 (state 1)	7.29e-8	0.73	0.73	19
traincar 4 (state 2)	7.34e-8	0.67	0.73	25
traincar 4 (state 3)	1.01e-7	0.66	0.60	14
traincar 4 (state 4)	6.96e-8	0.64	0.70	26
traincar 4 (state 5)	1.42e-7	0.61	0.68	26
traincar 4 (state 6)	1.67e-7	0.59	0.59	16
traincar 4 (state 7)	1.67e-7	0.56	0.56	13
traincar 4 (state 8)	1.38e-7	0.60	0.60	19
traincar 4 (state 9)	1.67e-7	0.47	0.47	7
rigid-body (out1)	1.08e-1	0.33	0.33	5
rigid-body (out2)	9.92e-1	0.20	0.20	15

Table 5.2: Maximum absolute errors for the best expression found by GP with the settings elitism: 2, tournament selection: 4, with and without crossover (seed used: 4357). err is the analyzed error. g denotes the generation in which the solution is found.

inputs. Some of the benchmarks compute internal states of a controller (denoted with e.g. "state 1"). Since each state is computed with a different expression, we treat them here as separate benchmarks. For all benchmarks we consider a fixed bit length, signed fixed-point format and truncation as the rounding mode.

Synthesizing the Best Expression. Table 5.1 and Table 5.2 lists the maximum absolute errors (computed by our analysis tool) for the best expressions found by GP for all our benchmarks. The benchmarks are ordered approximately by complexity with the smaller linear benchmarks first and the nonlinear benchmarks at the end of the table. From the third column we can see that we get substantial improvements in precision of up to 70%. The tables show the results obtained for the most successful setting we have found. Most successful in this case means that the best expression among all the runs was found nearly every time. It also shows the results with crossover turned off. The comparison of these two columns suggests that crossover is helpful. We therefore expect that randomized local search techniques are not as effective as genetic programming, but they still produce useful reductions in the errors.

Performance Enhancement in Control Systems. In Table 5.3 we provide the controllers synthesized by Ocosyn+ and the time required to synthesize them. In Table 5.4, we present the size of the region of practical stability for the baseline, improved and the optimal controllers for different benchmark systems and also the percentage improvement in the size of the region for the improved and the optimal controllers. The baseline and the improved implementations are based on the controllers provided in Chapter 4, and the optimal implementations are based on the controllers synthesized using Ocosyn+. Note that the region of practical stability for the baseline implementation varies from the result provided in Chapter 4. For example, for bicycle, the size of the region was presented as 0.0513 ignoring the effect of disturbance and measurement noise, the corresponding figure is 0.0785 in our experiment. This discrepancy is due to the fact that we use a different

Control systems	# bits	Synthesized gains		Time cost
		K	L	
bicycle	16	$[\ 3.0265e+0\ \ 1.2608e+1\ ]$	$[6.9088e-3\ \ 1.1135e-1\ ]^T$	51m36s
dc motor	16	[1.1760e-1 1.7400e-2 1.3300e-2]	[4.0400e-2 3.6720e-1 -1.2400e-2]^T	43m15s
pitch angle	32	[-1.2022e-1 4.2566e+1 1.0004e+0]	[3.2131e-4 2.1565e-5 1.8907e-3]^T	1hr21m58s
pendulum	32	[-1.5362e+0 -2.0254e+0 1.6519e+1 2.7358e+0]	$\begin{bmatrix} 1.7000e\text{-}3 & 2.1000e\text{-}3 & 1.2000e\text{-}3 & 0.0000e+0 \\ 1.0000e\text{-}4 & 1.8000e\text{-}3 & 1.2200e\text{-}2 & 7.7000e\text{-}2 \end{bmatrix}^T$	38m43s
batch reactor	16	$\begin{bmatrix} 5.9434e\text{-}2 & 9.0617e\text{-}1 & 3.2788e\text{-}1 & 8.7115e\text{-}1 \\ -2.4646e+0 & -4.4966e\text{-}2 & -1.7086e+0 & 1.1691e+0 \end{bmatrix}$	$\begin{bmatrix} 7.6055e\text{-}2 & -1.8342e\text{-}3 & 2.9025e\text{-}2 & 3.0801e\text{-}2 \\ -1.1106e\text{-}2 & 2.2255e\text{-}2 & 3.9666e\text{-}2 & -9.2832e\text{-}4 \end{bmatrix}^T$	2hr45m35s

Table 5.3: Synthesized gains and required time for synthesizing them.

method to estimate the bound on the error in the fixed-point implementation. The abstract interpretation based error estimation method used in this chapter is an order of magnitude faster than the mixed-integer linear programming approach in Chapter 4., which is apparent from the "time cost" column in Table 5.4. Even after incorporating the genetic programming based expression evaluation method in the synthesis process, our tool takes less time to synthesize a controller for all the benchmarks. Moreover, though our error estimation is less precise in comparison to that of Chapter 4 for 16 bit implementations, for 32 bit implementations (pitch angle and inverted pendulum) our estimation is significantly more precise.

The results in Table 5.4 show that we can improve the size of the region by rewriting the expression used in the baseline implementation. However, the improvement becomes significant when we incorporate the rewriting technique in the search method. Our results show that it is even possible to achieve more that 50% improvement in the synthesized controller with respect to the baseline controller. Table 5.5 shows the LQR/LQG cost of the baseline and the optimal controllers. The results show that in most of the examples, LQR and LQG costs do not degrade in the optimal controller with respect to the baseline controller. Only for the DC motor position example, the degradation in LQR cost is 8%. In a few instances, the LQG cost even improves.

Control systems	Region of Practical Stability			Improvement (%)	
	Baseline	Improved	Optimal	Improved	Optimal
bicycle	7.85e-02	7.70e-02	6.99e-02	1.93	10.96
dc motor	1.64e-02	1.44e-02	9.80e-03	12.14	40.24
pitch angle	1.08e-02	8.87e-03	5.15e-03	18.00	52.32
pendulum	3.11e-04	2.64e-04	2.51e-04	14.76	19.26
batch reactor	2.59e-01	2.24e-01	2.07e-01	13.31	20.08

Table 5.4: Improvement in the size of region of practical stability for the improved and synthesized controllers

Control systems	*lub* of LQR cost		LQG cost	
	Baseline	Optimal	Baseline	Optimal
bicycle	$4.33+3\|x\|^2$	$4.33e+3\|x\|^2$	2.46e-2	2.47e-2
dc motor	$1.38e+3\|x\|^2$	$1.50e+3\|x\|^2$	3.67e+1	3.67e+1
pitch angle	$2.99e+6\|x\|^2$	$2.99e+6\|x\|^2$	1.80e-3	1.58e-3
pendulum	$5.35e+4\|x\|^2$	$5.35e+4\|x\|^2$	3.90e-1	3.90e-1
batch reactor	$2.23e+2\|x\|^2$	$2.23e+2\|x\|^2$	9.49e-2	9.08e-2

Table 5.5: Least upper bound (*lub*) on the LQR cost, for a given initial condition x and the LQG cost for the baseline and the optimal implementations.

5.6 Related Work

A lot of research has gone into providing semi-automated compilation support from floating-point to fixed-point implementations [FRC03, MSB07, BR05, OCC07b]. The primary concern in these works has been bitwidth allocation: finding out the number of bits to allocate for the integral and the fractional parts of each real variable, so that the resulting implementation does not lose too much precision and the fixed-point variables do not overflow.

The range analysis problem has been studied extensively in the context of optimum bit-width allocation to intermediate variables in a fixed-point program, mostly in the DSP domain. Both static [LGC06a, LCN07b, OCC07b] and simulation-based [BR05, MSB07] approaches have been used. Existing work applied genetic programming as a technique to construct asynchronous circuits [MPS11] or programs that can be model checked [KP08].

Jha [Jha11] gives an algorithm for optimal fixed-point program synthesis based on inductive synthesis. His objective is to find out the best fixed-point implementation for a given expression, and does not consider rewriting of expressions. Moreover, it takes several minutes to synthesize a fixed-point program corresponding to an expression, where our abstract interpretation based technique can synthesize a fixed-point program corresponding to an expression in seconds .

To our knowledge, the only work that considers rewriting of expressions to improve precision in the context of abstract interpretation is [IM12]. The authors develop an abstract domain for representing an under-approximation of mathematically equivalent expressions. They then use a local, greedy search to extract some expression with a more precise formulation in a floating-point implementation. While the computation of precision is similar to ours, this approach excludes many possible expressions due to the local search and due to the under approximation in the abstract domain. Also, no validation of the results obtained with a static analysis tool as compared to "true" errors is performed and thus the issue of imprecision when dealing with nonlinear expressions is missed. In previous work [Mar09] the authors also considered fixed-point arithmetic. They used, however, only the maximum number of bits required to hold the integer part as the precision measure, which is too imprecise to distinguish many expressions.

CHAPTER 6

Memoization

In this chapter, we will shift our focus to a different aspect of control system implementation — the effect of computation time on the feasibility of implementation of a control system. We will introduce the related problem in the context of self-triggered implementation of the control systems. Self-triggered implementations of digital controllers have been proposed recently as a computation- and communication-efficient technique for the software realization of a control law [VMF03, LCH07, MT08, WL09, MAT09, MT09, ASP10, AT10]. In a self-triggered implementation, the control task computes, in addition to the actuation signal, the next instant of time at which the control law should be recomputed (the *trigger time*). In between the control updates, the actuation signal is held constant and the plant evolves in open loop. The appropriate choice of trigger time ensures that the resulting system is still stable and has required performance.

Self-triggering is an attractive technique for integrated architectures of cyber-physical systems, in which multiple control loops and non-critical applications share the same resources (CPU or communication network) [OSH09]. Unlike the traditional *time-triggered* approach, where the control computation is performed periodically with a fixed period, it has the potential of making many fewer computations and lower use of the communication bandwidth. This is because the period in a time-triggered approach is chosen under a worst-case assumption, and control computations are performed at this rate even when the plant is in steady state and there are no disturbances [AW90a]. Unlike *event-triggered* approaches

[Arz99, San06, HSV08], in which the state of the plant is continuously monitored using dedicated hardware to detect an event that triggers control re-computation, self-triggered approaches do not require the computation costs or extra hardware for continuous sensing. Indeed, simulations of some benchmark control systems demonstrate that self-triggered implementations can provide significant savings in both computation and communication [MAT09, MT09, ASP10, AT10].

In this chapter, we show that savings estimates claimed for self-triggered implementations are somewhat optimistic. In most simulations of self-triggered implementations, it is assumed that the execution time required to compute the trigger time is negligible. We show that this assumption is not realistic for a number of common embedded platforms, and that the time required to compute the trigger time can indeed be more than the trigger time itself, making a direct implementation infeasible.

We propose an implementation scheme for self-triggered controllers using memoization of trigger times. We demonstrate that our implementation can provide similar savings in communication bandwidth as the naive self-triggered implementation, while keeping computation costs low. We maintain a *memoization region*: a subset of the state space around a given operating point for the controller. We quantize the memoization region into a grid of chosen precision, and compute the trigger times for states on the grid on demand. To compute the trigger time of a state ξ, we compute its quantization on the grid and see if the trigger time has been cached from a previous computation. If so, we return the pre-computed value. If not, instead of computing the trigger time with the same priority as the control task, we schedule it as a background task of lower priority, and fall back on a time-triggered implementation in case it is not computed in time. If the operating point is fixed, the entire memo table can be pre-computed and there is no runtime overhead. However, in many cases the operating points can change dynamically and may not be known statically, or the space required to

keep the entire grid may be too high. In these cases, we compute the entries of the memo table incrementally and on demand, falling back on a time-triggered implementation if the computation does not finish in time.

The quantization of states in computing trigger times introduces an error in the implementation of the controller, since the trigger time may not be computed for the given state, but for its approximation on the grid. We show how we can bound the effects of this error, and provide a bound on the practical stability of the controlled system [LL61, PW07, AMS10] based on results on self-triggered controllers for control systems with bounded disturbances [ASP10]. While we focus on trigger times, quantization and memoization can also be applied to the computation of the control law, and a similar error analysis can be performed. (In our examples, the computation of the control law takes negligible time in comparison to the computation of the trigger time, so we focus only on memoizing trigger times.)

We have developed a framework to evaluate self-triggered implementations of control systems. Our framework uses Truetime [CHL03] for the simulation of control applications and aiT [HF09] for the computation of worst case execution times. We evaluate our implementation on the example of a batch reactor process, a standard linear system benchmark used in previous papers on self-triggered control [MAT09, MT09, ASP10], and an example of a jet engine compressor, a standard nonlinear control benchmark [AT10]. Experimental results show that our hybrid implementation attains communication costs close to that of naive self-triggered implementations, and simultaneously reduces computation costs significantly.

6.1 Self-Triggered Control

The evolution of a dynamical system with time is captured by a differential equation:

$$\dot{\xi} = f(\xi, \upsilon),\ \xi(0) = \xi_0 \tag{6.1}$$

where $\xi(t) \in \mathbb{R}^n$ denotes the state of the system at time t and $\upsilon(t) \in \mathbb{R}^m$ denotes the external input to the system at time t. Equation (6.1) is obtained from Equation (2.1) ignoring the effect of external disturbance. The curve $\xi : \mathbb{R} \to \mathbb{R}^n$ is said to be a *solution* or *trajectory* of (2.1) when there exists a piecewise continuous input curve $\upsilon : \mathbb{R} \to \mathbb{R}^m$ such that the time derivative of ξ at time t equals $f(\xi(t), \upsilon(t))$. The control system is called *linear time invariant* if there are matrices $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$ such that

$$\dot{\xi}(t) = A\xi(t) + B\upsilon(t), \qquad \xi(t) \in \mathbb{R}^n, \upsilon(t) \in \mathbb{R}^m. \tag{6.2}$$

Equation (6.2) is obtained from Equation (2.2) by ignoring the effect of external disturbance, and assuming that the full state of the plant is available for the computation of the control signal. A controller

$$\upsilon(t) = k(\xi(t)) \tag{6.3}$$

computes the input $\upsilon(t)$ as a function of the state $\xi(t)$.

6.1.1 Self-Triggered Implementation

To implement a control law for the system (6.1) on a digital computer, the state of the plant is sampled at a sequence of time instants $t_0 = 0,\ t_1,\ t_2, \ldots$ The state of the plant at time instant t_i is $\xi(t_i)$, and the computed control signal is given by $u(t_i) = k(\xi(t_i))$. The time instant t_i is called the *trigger time*. The control signal is applied to the plant immediately after it is available, and is held constant until the next trigger time t_{i+1}. The control signal computation time is generally in

the microsecond range and can be considered to be negligible. Thus the control signal in the digital implementation of the system in (6.1) is given by

$$\upsilon(t) = \upsilon(t_i) \quad \text{for } t \in [t_i, t_{i+1}[\tag{6.4}$$

In a *time-triggered* implementation, the control engineer fixes a sampling period $T > 0$, and selects a periodic sequence of trigger times $t_i = iT$, for $i \in \mathbb{N}$. For *self-triggered* implementations, the sequence $\{t_i\}_{i\in\mathbb{N}}$ is computed online. At time t_i, in addition to the control signal, the next time instant t_{i+1} when the plant's state would be sampled is computed based on the current state $\xi(t_i)$. This computation is done based on a rule that is designed to ensure stability and desired performance of the control system.

We focus on linear control systems. Here we briefly recall the self-triggered implementation of a linear controller achieving exponential stability as proposed in [MAT09, MT09]. Consider the linear time-invariant control system (6.2). The system is rendered closed loop exponentially stable by using a controller

$$\upsilon(t) = -K\xi(t). \tag{6.5}$$

where $\xi(t)$ is the solution of (6.2) at time t, and matrices A, B, and K are of appropriate dimensions. The closed loop system is given by

$$\dot{\xi} = (A - BK)\xi \tag{6.6}$$

As the closed loop system is stable, there exists a Lyapunov function

$$V(\xi(t)) = \xi(t)^T P \xi(t) \tag{6.7}$$

where P is a symmetric positive definite matrix. The matrix P can be obtained by solving the following Lyapunov equation:

$$(A - BK)^T P + P(A - BK) + Q = 0 \tag{6.8}$$

where Q is a given symmetric positive definite matrix.

The Lyapunov function in (6.7) admits an exponential decay. Let V be a Lyapunov function satisfying (6.8) and let $\lambda_0 > 0$ denote its rate of decay. For self-triggered implementations, we fix $0 < \lambda < \lambda_0$ and define a function S upper-bounding the evolution of V:

$$S(t, \xi(t_k)) = V(\xi(t_k))e^{-\lambda(t-t_k)}. \tag{6.9}$$

Thus, at $t = t_k$, we have $S(t, \xi(t_k)) = V(\xi(t_k))$ and for $t_k < t < t_{k+1}$, we have $S(t, \xi(t_k)) > V(\xi(t))$. The constant λ in the function S specifies the desired performance of the control system.

The self-triggered implementation has to ensure that the value of $V(\xi(t))$ never goes beyond the value of $S(t, \xi(t_k))$. To ensure this, a *triggering function* is defined as

$$h(t, \xi(t_k)) = V(\xi(t)) - S(t, \xi(t_k)) \qquad \text{for all } t \geq t_k \tag{6.10}$$

Triggering happens when $h(t, \xi(t_k)) = 0$. At that moment, the plant's state is sampled again. The triggering scheme ensures that there exists a positive constant $\tau_{\min}$ such that for any i, we have $\tau_i = t_{i+1} - t_i \geq \tau_{\min}$. The value of $\tau_{\min}$ is effectively computable from the parameters of the control system and the Lyapunov function (see Theorem 5.1 in [MAT09]).

To implement self-triggered control, the designer fixes two design parameters: a maximum duration $\tau_{\max}$ between two triggering instants that determines how long the system is allowed to operate in open loop, and a discretization parameter $\Delta > 0$ that puts a grid on the interval $[\tau_{\min}, \tau_{\max}]$. Then, a discrete version of the triggering function h from (6.10) is defined:

$$\widetilde{h}(n, \xi(t_k)) = V(\xi(t_k + n\Delta)) - S(n\Delta, \xi(t_k)) \tag{6.11}$$

The self-triggered implementation $\Gamma_l : \mathbb{R}^n \to \mathbb{R}^+$, $\Gamma_l(\xi(t_k)) = t_{k+1}$ for linear

control systems is given by:

$$\begin{aligned} t_{k+1} &= \max\{t_k + \tau_{\min}, t_k + n_k\Delta\} \\ n_k &= \max\{s \leq \lfloor \frac{\tau_{\max}}{\Delta} \rfloor \mid \text{ for all } n \in [0,s] : \widetilde{h}(n, \xi(t_k)) \leq 0\} \end{aligned} \tag{6.12}$$

Note that the worst case execution time depends on n_k and n_k depends on $\tau_{\max}$ and Δ. The self-triggered implementation scheme is feasible only if the time required to compute the trigger time t_{k+1} is less than $t_{k+1} - t_k$.

6.1.2 Problem: Trigger Time Computation

We now illustrate the problem of implementing the self-triggering scheme defined above through a motivating example of a batch reactor process originally described in [Ros74] and used in [MAT09, MT09, ASP10] as a benchmark. The model of the plant is given by

$$\dot{\xi} = \begin{bmatrix} 1.38 & -0.20 & 6.71 & -5.67 \\ -0.58 & -4.29 & 0 & 0.67 \\ 1.06 & 4.27 & -6.65 & 5.89 \\ 0.04 & 4.27 & 1.34 & -2.10 \end{bmatrix} \xi + \begin{bmatrix} 0 & 0 \\ 5.67 & 0 \\ 1.13 & -3.14 \\ 1.13 & 0 \end{bmatrix} \upsilon.$$

The feedback controller

$$\upsilon = - \begin{bmatrix} 0.1006 & -0.2469 & -0.0952 & -0.2447 \\ 1.4099 & -0.1966 & 0.0139 & 0.0823 \end{bmatrix} \xi$$

renders the system exponentially stable. Let us denote the worst case execution time (WCET) of the computation of the trigger time t_{k+1} by τ_c. The rate of decay is $\lambda_0 = 0.41$. Setting $\lambda = 0.9\lambda_0$, we get $\tau_{\min} = 18ms$. The authors of [MT09] chose $\tau_{\max} = 358ms$ and $\Delta = 10ms$. The maximum possible value for n_k is thus 35. With these choices, we implement the trigger time computation given by (6.12) using the discrete triggering function (6.11) as a C program and cross-compile it using a GNU-based cross-compiling system. We then compute the

WCET of the trigger time computation using aiT [HF09], a state-of-the-art worst case execution time analysis tool. The WCET of the trigger time is $29.793ms$ on a Leon 2 processor, assuming the processor frequency to be $100MHz$ (Most of the commercial implementations of Leon 2 processor have clock frequency below $100MHz$ [Leo]). As we see, the WCET is greater than $\tau_{\min}$. This implies that there may be cases where the controller will produce the next trigger time at an instant when the trigger time has already passed. This clearly shows the infeasibility of the implementation of the triggering rule provided in [MT09]. (In simulating the performance of their controller, the authors of [MT09] ignore trigger computation times.)

Note that we cannot increase the value of $\tau_{\min}$ as it depends on the parameters of the control system, and increasing $\tau_{\min}$ may cause the self-triggered implementation of the control system to violate stability requirements. The self-triggered implementation may be made feasible by decreasing the value of τ_{max}, as τ_{max} is a design parameter and the worst case execution time is directly proportional to τ_{max}. Decreasing the value of τ_{max} does not have any effect on the performance of the control system. However, it causes the trigger of the controller to occur more frequently than what the designer in [MT09] expected, and thus increases communication between the controller and the actuators (negating much of the benefits of self-triggering). Thus, the performance advantages of self-triggered implementations must be evaluated taking the computation time for the trigger time into account.

6.2 Hybrid Implementation

We now present a hybrid implementation scheme for self-triggered control systems. Our implementation utilizes a time-vs-space tradeoff, and memoizes trigger times for future reuse. We assume that a fixed-size piece of memory is available to store

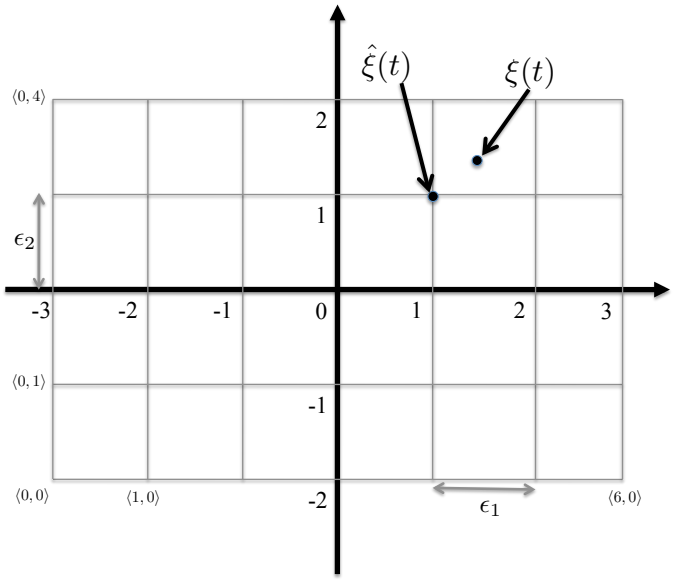


Figure 6.1: Memoization region and table

the *memoization table*, which maps a state of the system (a vector in $\mathbb{R}^n$) to a trigger time. The memoization table can be accessed using indices computed from a state as described later.

6.2.1 Memoized Trigger Time Computation

For the implementation, we fix a *memoization region* $[-w, w] \subseteq \mathbb{R}^n$ of the control system around the origin. We want to memoize the computed trigger time for any state in this region. Since the memoization table is finite, we discretize the memoization region using a *quantization factor* $\epsilon \in \mathbb{R}^n$. We describe the choice of ϵ in Section 6.2.3. The memoization region is represented as a multi-dimensional array, where each element is a trigger time. The number of entries in the table is given by

$$N = \prod_{i=1}^{n} \frac{2w_i}{\epsilon_i} \tag{6.13}$$

As an example, Figure 6.1 shows an example system with two state variables and the memoization region $[-3, 3] \times [-2, 2]$. The quantization factor $\epsilon = (1, 1)$,

that is, each state is divided into intervals of size one unit. There are 24 different entries in the memoization table, and they are indexed as shown in the figure, from $(0, 0)$ at the bottom left corner to $(5, 3)$. The quantized value of a state ξ is the state corresponding to the closest grid point to its left and bottom, e.g., the state $\xi = (1.3, 1.3)$ is quantized to $\hat{\xi} = (1.0, 1.0)$, with index $(4, 3)$. Note that the difference between $\hat{\xi}(t)$ and $\xi(t)$ is bounded by the quantization factor ϵ, thus $\hat{\xi}(t) \geq \xi(t) - \epsilon$.

Algorithm 6.2.1 shows the pseudo-code for trigger-time computation with memoization. The function `findTriggerTime` takes as input the system state $\xi(t_k)$ at the kth sample time t_k and returns t_{k+1}, the next time the plant should be sampled. The memoization table `Memo` stores trigger times. The function `findIndex` takes the state $\xi(t_k)$ and returns the index of its discretization, if within the memoization region, or a special value denoting that the current state is outside the memoization region. In case the current state is outside the memoization region, the trigger time computation is scheduled (line 5). In case the current state is within the memoization region, the memoization table is first checked (line 7). If the trigger time has been pre-computed, it is returned (line 11). Otherwise, the trigger time computation is scheduled with the discretized state (line 9). When the computation is done, the memoization table is updated with the computed value.

The statement

$$\min(\tau_{\min},\ \texttt{schedule trigger}(\hat{\xi}))$$

indicates that the task of computing the trigger time is scheduled as a background task of lower priority than the control loop. If the task finishes before $\tau_{\min}$, its value is returned, otherwise, the plant is sampled again at $t_k + \tau_{\min}$. That is, if the background task is not finished in time, the controller reverts to a time-triggered implementation with period $\tau_{\min}$. On completion of the task to compute the trigger time, the memoization table `Memo` is updated with the computed value (in case

Algorithm 6.2.1: Trigger Time Computation with Memoization.

Input: $\xi(t_k)$, the state of the system at time instant t_k

Output: t_{k+1}, the next trigger time of the controller

```
1  function findDeliveryTime(ξ)
2  begin
3      ⟨s_1 ... s_n⟩ = findIndex(ξ)
4      if then
5          outsideRange(⟨s_1 ... s_n⟩)
6      else
7          return min(τ_min, schedule trigger(ξ))
8      end
9      if then
10         Memo[s_1 ... s_n] is not found
11     else
12         ξ̂ = quantizeState(⟨s_1 ... s_n⟩)
13         return min(τ_min, schedule trigger(ξ̂))
14     end
15     return Memo[s_1 ... s_n]
16 end
```

the state was within the memoization region). Moreover, in our implementation, when one trigger time computation is running, no other `trigger` task is spawned, and the controller works in a time-triggered mode using the minimum trigger time τ_{min} as the sampling period. We call this implementation scheme *hybrid*.

6.2.2 Dynamic Choice of Memoization Region

In Section 6.2.1 we assume that in the steady state the control system operates at the origin, and we fix the memoization region to be $[-w, w] \subseteq \mathbb{R}^n$. If a control system has only one operating point, the memoization table can be precomputed

offline for all the states in the memoization region around the operating point instead of computing the table online. However, in reality, a control system may have a number of operating points, and the system can move from one operating point to another during its life cycle. All these operating points may not be known to the designer of the system a priori. For example, one of the states of the batch reactor process introduced in Section 6.1.2 is the temperature of the reactor. The desired temperature of the reactor is different during the different reaction phases. It is not possible to store the memoization table for all operating points in the memory.

We handle the change in operating point during runtime by online reconfiguration of a parameter used in Algorithm 6.2.1. Suppose the operating point changes from ξ_1 to ξ_2. We assume that the size of the memoization region remains the same for any operating point. Thus, the memoization region changes from $[\xi_1 - w, \xi_1 + w]$ to $[\xi_2 - w, \xi_2 + w]$. Even though there may be an overlap between the old and new memoization regions, the data in the overlapped region is not reused, as that data needs to be moved to a new location in the memoization table, and this operation may take many CPU cycles. The `findIndex` function in Algorithm 6.2.1 depends on an offset vector that maps a state in the memoization region to a particular position in the memoization table. When the operating point changes, we perform the following two actions. First, the memoization table is cleared. Second, the offset parameter is reconfigured to reflect the new memoization region.

6.2.3 Effect of State Quantization

As our memoization based implementation employs quantization of the state of the control system, we need to take into account the effect on the quantization on the triggering time. Since the trigger time computation does not have a closed form formula, correcting the triggering rule to take into account the quantization error

is not straightforward. Rather, we show how the quantization error influences the behavior of the closed loop system in the steady state.

To show how the quantization error on the states effect the overall behavior of a linear control system, we resort to a result proved by Almeida, Silvestre, and Pascoal [ASP10]. Consider the following linear time-invariant control system:

$$\dot{\xi}(t) = A\xi(t) + B\upsilon(t) + \delta(t), \qquad \xi(t) \in \mathbb{R}^n, \upsilon(t) \in \mathbb{R}^m, \tag{6.14}$$

where $\delta(t)$ is exogenous disturbance with bounded $\mathcal{L}_\infty$ norm δ_b. Note that the system in (6.14) is obtained by introducing an additive disturbance term $\delta(t)$ in the system in (6.2). In the presence of disturbance it is not possible to render the system (6.14) asymptotically stable to the origin. Rather, it was shown in [ASP10] that by appropriately modifying the triggering rule in (6.10), it is possible to ensure that the system is exponentially stable with respect to a region around the origin.

Consider the Lyapunov function in (6.7). It can be shown that in the presence of bounded disturbance, the feedback law in (6.5) can render the states of the system (6.2) exponentially in a region defined by

$$V(\xi(t)) \leq \max\{V(\xi(t_0))e^{-\gamma_0(t-t_0)}, V_b\}, \tag{6.15}$$

where $\xi(t_0)$ denotes the initial state of the evolution at time t_0. Note that as $t \to \infty$, $V(\xi(t))$ is inside a region defined by V_b. For the system (6.14), V_b is given by

$$V_b = \frac{4\lambda_{\max}^3(P)\delta_b^2}{(\lambda_{\min}(Q) - \gamma_0\lambda_{\max}(P))^2}, \tag{6.16}$$

where P and Q are matrices in the Lyapunov Equation (6.8).

Now the performance function in (6.9) need be modified according to the behavior of $V(\xi(t))$ in (6.15). The modified specification function is given by

$$S'(t, \xi(t_k)) = \max\{V(\xi(t_k))e^{-\gamma(t-t_k)}, V_b\}, \tag{6.17}$$

where $0 < \gamma < \gamma_0$. The triggering function in (6.10) is now modified as follows:

$$h(t, \xi(t_k)) = U(t, \xi(t)) - S'(t, \xi(t_k)), \tag{6.18}$$

where $U(t, \xi(t))$ is given by

$$U(t, \xi(t_k)) = V(\xi(t_k)) + \mu(t, \xi(t_k)), \tag{6.19}$$

with $\mu(t, \xi(t_k)) = \beta(t)(2\|P\xi(t_k)\| + \lambda_{\max(P)}\beta(t))$ and $\beta(t) = \frac{\delta_b}{\sigma}(e^{\sigma(t-t_k)} - 1)$. The value of σ can be chosen as any value between $\|A\|$ and $\frac{1}{2}\lambda_{\max}(A + A^T)$.

As shown in [ASP10], the self-triggered implementation in (6.12) with the triggering function given in (6.18) renders the states of the system (6.14) in finite time to the set $\{\xi(t) \in \mathbb{R}^n : V(\xi(t)) \leq V_b\}$. We call this set the *region of practical stability*, or in short, the *stability region*.

We model the quantization error arising from the memoization based implementation of a self-triggered linear control system as the disturbance δ in (6.14). For a state $\xi(t)$ the control signal and the trigger time is computed by its quantized value $\hat{\xi}(t)$, where $\xi(t) - \hat{\xi}(t) = \bar{\epsilon}(t)$. The computed control signal is given by

$$\hat{\upsilon}(t) = -K\hat{\xi}(t) = \upsilon(t) + K\bar{\epsilon}(t).$$

Plugging the value of the control signal in (6.2), we get

$$\dot{\xi}(t) = A\xi(t) + B\upsilon(t) + BK\bar{\epsilon}(t), \qquad \xi(t) \in \mathbb{R}^n, \upsilon(t) \in \mathbb{R}^m. \tag{6.20}$$

Comparing (6.20) with (6.14) we get that the disturbance arising due to quantization in the states is given by

$$\delta(t) = BK\bar{\epsilon}(t). \tag{6.21}$$

Suppose we have been given an upper bound $V_b^{\max}$ on V_b as the specification. To find the upper bound of the quantization factor we proceed as follows: The

upper bound on the $\mathcal{L}_\infty$ norm of $\delta(t)$ is given by

$$\delta_b^{\max} = \sqrt{\frac{(\lambda_{min}(Q) - \gamma_0 \lambda_{\max}(P))^2}{4\lambda_{\max}^3(P)} V_b^{\max}}. \tag{6.22}$$

The upper bound on the quantization factor ϵ is found by solving the following optimization problem:

$$\begin{aligned} &\text{maximize} && \epsilon_1 \\ &\text{subject to} && \|BK\epsilon_1[1\ 1\ 1\ 1]^T\| \leq \delta_b^{\max}, \end{aligned} \tag{6.23}$$

where $\bar{\epsilon}(t) = \epsilon_1[1\ 1\ 1\ 1]^T$. Here we have chosen the quantization factors in all dimensions to be equal. The optimization problem in (6.23) is a convex one. The solution of the optimization problem provides the upper bound on the quantization factor of the state.

Theorem 4. *Consider the linear control system* (6.2)*, with stabilizing controller* (6.5) *and Lyapunov function V obtained from* (6.7) *and* (6.8)*. Let $V_b^{\max}$ be a given bound on the stability region. Suppose the controller is implemented using the hybrid implementation using Algorithm 6.2.1, using a discretization ϵ given by* (6.23)*. Then, the state ξ is guaranteed to converge to the set $\{\xi \in \mathbb{R}^n \mid V(\xi) \leq V_b^{\max}\}$.*

6.2.4 Fixed-point Representation

The trigger time is computed in (6.12) as a floating point entity. A single precision floating point value needs four bytes of space in memory. To save memory, we can store the trigger time as a fixed-point number. The trigger times are always positive, thus we do not need to deal with the sign of a number. The fixed-point representation of a trigger time is given as a pair $\langle n, m\rangle$ consisting of a *length* $n \in \mathbb{N}$, and a *length of the fractional part* $m \in \mathbb{N}$. The length of the integer part is $n - m$, if $n > m$, and 0 otherwise. To achieve the highest precision for a chosen length m of the representation, the length of the fractional part m is chosen such that the maximum trigger time $\tau_{\max}$ satisfies $2^{n-m-1} \leq \tau_{\max} < 2^{n-m}$. In this

case, the maximum error in the fixed-point representation is given by 2^{-m}. For example, if the range of the trigger time to be represented is given by $[0.020, 0.400]$ and we choose $n = 8$, then $m = 9$ and the bound on the error in the representation is given by 2^{-9}.

Given a computed trigger time t, its fixed-point representation can be stored as an integer given by $\hat{t} = \lfloor (t \times 2^m) \rfloor$. From the fixed-point representation $\hat{t}$, the floating point value $\widetilde{t}$ of the trigger time is obtained by $\widetilde{t} = 2^{-m}\hat{t}$. To use fixed-point representations in Algorithm 6.2.1, line 9 and line 11 are adapted appropriately. Note that it is always safe to use $\widetilde{t}$ obtained from the memoization table instead of using t, as $\widetilde{t} \leq t$.

6.3 Nonlinear Systems

We now extend the hybrid implementation to self-triggered implementations of nonlinear control systems.

Triggering Rule. Consider the nonlinear dynamical system (2.1). We recall the setting of [AT10]. The objective of the controller is to ensure that an invariant on the state of the plant, given as the specification, is never violated. Henceforth, we will refer to this specification as *invariant specification*. The idea behind a self-triggered implementation is that at any time instant t, the difference between the current state $x(t)$ and the last measured state $x(t_i)$ for $i \in \mathbb{N}$ is bounded in such a way that the invariant specification on the closed loop system is not violated. This error is referred as the *measurement error* and is given by

$$e(t) = \xi(t_i) - \xi(t) \quad \text{for } t \in [t_i, t_{i+1}]. \tag{6.24}$$

The dynamics of the closed loop control system is given by:

$$\dot{\xi} = f(\xi, k(\xi + e)). \tag{6.25}$$

Now if the control law is designed to render the system (2.1) ISS with respect to

measurement error, there exists a Lyapunov function V for the system satisfying the following inequality:

$$\dot{V} \leq -\alpha_1(\|x\|) + \alpha_2(\|e\|). \tag{6.26}$$

where α_1 and α_2 are $\mathcal{K}_\infty$ functions. To maintain stability of the closed loop system (6.25), we have to ensure that $\dot{V} < 0$. This can be ensured if the following condition holds:

$$\alpha_2(\|e\|) \leq \kappa\alpha_1(\|x\|), \qquad \kappa > 0. \tag{6.27}$$

The triggering strategy in a nonlinear control system is designed based on the invariant specification on the states of the system around the equilibrium point. If a Lyapunov function satisfying the inequality in (6.26) can be discovered for the system, then an invariant on the admissible error can be derived from the invariant on the states using (6.27). As shown in [AT10], these two invariants can be used to derive a closed form formula z to compute the next sampling time t_{k+1} from the sampled state $\xi(t_k)$ at the current sampling time t_k.

The self-triggered implementation $\Gamma_{nl} : \mathbb{R}^n \to \mathbb{R}^+$, $\Gamma_{nl}(\xi(t_k)) = t_{k+1}$ for nonlinear control systems is given by:

$$t_{k+1} = z(\xi(t_k)). \tag{6.28}$$

Hybrid Implementation. Given the triggering rule (6.28), the hybrid implementation is similar to the implementation of linear controllers, where we discretize the state space and memoize computed trigger times.

Since there is a closed form formula to compute the triggering time, we can correct the triggering times by taking into account the maximum possible error introduced by quantizing the states. To find the maximum possible error introduced

in the trigger time, we solve the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & \tau_e \\ \text{Subject to} \quad & \tau = z(\xi) \quad \hat{\tau} = z(\hat{\xi}) \\ & \tau_e = \tau - \hat{\tau} \\ & \xi - \hat{\xi} \leq \epsilon \quad \xi \geq \hat{\xi}. \end{aligned} \tag{6.29}$$

In this optimization problem, ξ denotes the sampled state and $\hat{\xi}$ denotes the quantized state obtained from ξ. We denote by τ and $\hat{\tau}$ the trigger time computed based on ξ and $\hat{\xi}$ respectively. We maximize τ_e, the difference between τ and $\hat{\tau}$, subject to additional constraints that the difference between ξ and $\hat{\xi}$ is bounded by the quantization factor ϵ and $\xi \geq \hat{\xi}$.

For a given state $\xi(t_k)$, if the maximum value of τ_e is obtained as τ_e^{max} then the triggering time is computed by the following modified form of (6.28):

$$t_{k+1} = z(\hat{\xi}(t_k)) - \tau_e^{max}. \tag{6.30}$$

Note that unlike linear control systems, the control signal in a nonlinear control system is computed based on $\xi(t_k)$ instead of $\hat{\xi}(t_k)$.

6.4 Evaluation

6.4.1 Implementation

In our experiments we have used the Truetime simulator [CHL03] to implement the control tasks for our example control systems and simulate the systems under different conditions. We choose PowerPC MPC5xx [Pow] and Leon2 [Leo] processors as the two target platforms on top of which the embedded control applications are built. The PowerPC MPC5xx is widely used in Delphi Corporation and Robert Bosch GmbH to develop engine controllers. The Leon 2 processor was developed by European Space Research and Technology Centre to be used

for European space projects. The worst-case execution times [WEE08] for control computation and trigger time computation on the target processors have been obtained using the worst-case execution time analysis tool aiT [HF09]. The control computations and trigger time computations are implemented in C, and then cross-compiled for respective processors using GNU-based cross compilation systems. The worst-case execution times are computed by aiT based on the compiled binary code. The computation of trigger time involves computation of square root function and exponential function. We implement the square root computation using the *Babylonian method* [Kos09]. The exponential function is computed by expanding it as a Taylor series.

6.4.2 Experiments

We demonstrate our results on a benchmark linear control system: a *batch reactor processor*, and a benchmark nonlinear control system: a *jet engine compressor*. We report CPU times and communications costs. Given a time interval, CPU times are reported as the total CPU time required by all control tasks over the time interval. The communication cost is the number of control tasks run over a time interval. This captures the cost of communication, since each control task requires transmitting the plant state from the sensors to the controller and the actuation signal from the controller to the actuators.

Linear System: Batch Reactor Process. First, we present our results on the batch reactor process from Section 6.1.2. The Lyapunov function for the closed loop system has been computed using (6.8) with $Q = I_4$. With this choice of Q, we have $\lambda_{\max}(P) = 1.2185$ and $\lambda_{\min}(Q) = 1$. The other parameters are chosen in accordance with [ASP10]: $\tau_{\min} = 39.8ms$, $\tau_{\max} = 720ms$, $\Delta = 10ms$, $\gamma_0 = 0.08207$, $\gamma = 0.008207$. We compute the values of $\tau_{\max}$ on both PowerPC and Leon2 processors to ensure that the trigger computation time τ_c does not exceed the minimum trigger time $\tau_{\min}$. The values of modified $\tau_{\max}$ are $510ms$

and $270ms$ on PowerPC and Leon2, respectively.

Suppose we want the state of the system to converge in a stability region $V_b^{\max}$ of size 0.5 when the system is free from any external disturbance. The upper bound on the quantization factor can be found to be 0.04346 by first finding the value of $\delta_b^{\max}$ for $V_b^{\max} = 0.5$ using (6.22), and then solving the optimization problem in (6.23). We choose the quantization factor to be 0.04 in each dimension. We store the trigger times in the memoization table as a 8-bit fixed-point number. The fixed-point representation is given by $\langle 8, 8\rangle$, as $0.5 < \tau_{\max} < 1$. The trigger time retrieved from the memoization table may by at most 2^{-8} less than its computed value. We choose the memoization region to be $[-0.5, 0.5]$ in each dimension. With the quantization factor to be 0.04, the number of entries in the memoization table is computed by (6.13) to be 390625. As each entry of the memoization table takes 1 byte, the memoization table requires at most 381.47KB. Note that if this amount of memory is not available, we have two options. First, we may increase the value of the quantization factor which in turn increases the bound on the size of the stability region. Second, we may shrink the memoization region without changing the quantization factor. The second option ensures that the specification on the size of the stability region is met, but we memoize trigger time for the states in a narrower region, thus the computation time grows.

We compare three different implementations for the batch reactor processor: (i) a time-triggered implementation using the sampling period $\tau_{\min}$, (ii) a self-triggered implementation using the triggering rule (6.12), where the triggering function is given by (6.18). The maximum trigger time $\tau_{\max}$ is updated to ensure that the computation of the trigger time is guaranteed to be finished before the minimum trigger time $\tau_{\min}$ ($\tau_{max} = 510ms$ for PowerPC and $\tau_{max} = 270ms$ for Leon2 processor), and (iii) our hybrid implementation, without decreasing the maximum trigger time ($\tau_{max} = 720ms$). We fall back to the time-triggered implementation using trigger time $\tau_{\min}$ when the trigger time computation takes more

Implementation	Disturbance Free		Disturbance Scenario 1		Disturbance Scenario 2	
	PowerPC	Leon2	PowerPC	Leon2	PowerPC	Leon2
TT	0.164s	0.226s	0.164s	0.226s	0.164s	0.226s
ST	167.514s	342.554s	167.188s	341.117s	167.691s	342.541s
Hybrid	0.959s	1.824s	4.039s	7.336s	31.322s	57.360s

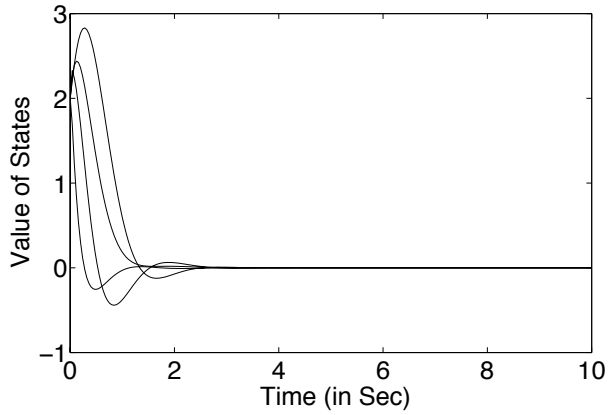
Table 6.1: CPU time required for different implementations of the controller of the batch reactor process running for 2000s

than $\tau_{\min}$ time.

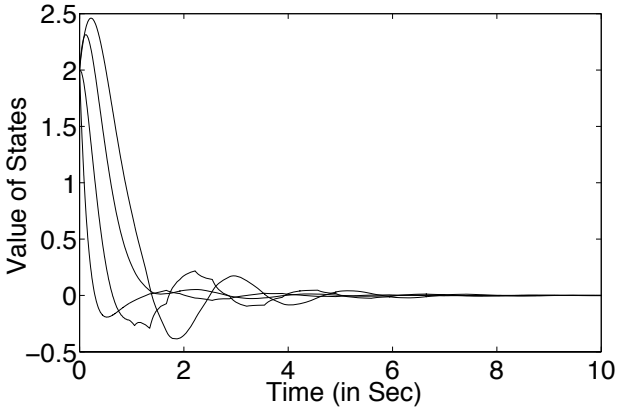
We simulate the self-triggered batch reactor process on three scenarios: (a) the system is free from any external disturbance (Disturbance free), (b) the system is subjected to a periodic external disturbance signal of pulse shape with amplitude 8, period 800 samples, pulse width 40 samples, zero phase delay, and sample time $10ms$. (Disturbance scenario 1), and (c) the system is subjected to a normally distributed random disturbance signal with mean 0, variance 1, and sample time $1ms$ (Disturbance scenario 2).

The evolution of the state of the batch reactor process with initial state $\langle 2, 2, 2, 2\rangle$ is shown in Figure 6.2 for the case when no external disturbance is present. While the state of the time-triggered implementation and the self-triggered implementation eventually go to the origin, the state of the hybrid implementation eventually enters the region defined by $V_b^{\max}$ and remains there forever.

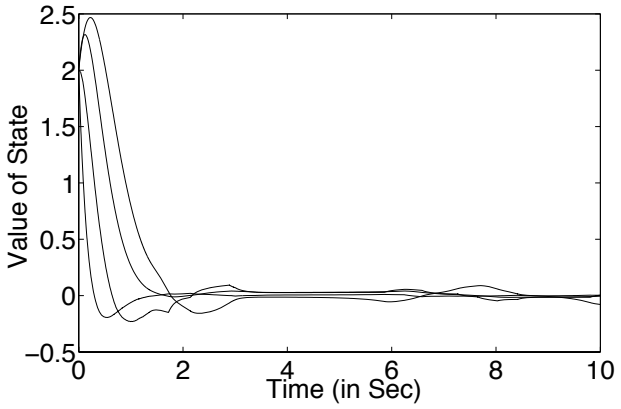
Table 6.1 and Table 6.2 show the CPU time and communication cost for different implementations of the controller for the batch reactor process for 2000s runtime. From Table 6.1, we see that the naive self-triggered implementation takes significantly more CPU time in comparison to the time-triggered implementation. The hybrid implementation brings the required CPU time close to that of the time-triggered implementation. Table 6.2 shows that the number of commu-



(a) TT



(b) ST



(c) Hybrid

Figure 6.2: Evolution of states of a batch reactor process from initial state $\langle 2, 2, 2, 2 \rangle$ for (a) time-triggered (TT), (b) self-triggered (ST), and (c) Hybrid implementation

Implementation	Disturbance Free		Disturbance Scenario 1		Disturbance Scenario 2	
	PowerPC	Leon2	PowerPC	Leon2	PowerPC	Leon2
TT	50251	50251	50251	50251	50251	50251
ST	5731	7422	14735	15786	5974	7422
Hybrid	5868	5891	15847	15868	6932	7343

Table 6.2: Communication cost for different implementations of the controller of the batch reactor process running for 2000s

nications from the plant to the controller improves significantly in self-triggered implementations, as claimed in the existing literature on self-triggered control systems. The Hybrid implementation maintains similar communication cost as the naive self-triggered implementations. For self-triggered implementation, the number of communications is always more for Leon2 processor, as τ_{max} for Leon2 is less than that for PowerPC. For Hybrid implementation, the number of communications is more for Leon2 processor than that for PowerPC as the trigger time computation takes more time with the Leon2 processor than with the PowerPC processor and the system remains in time-triggered mode when the trigger time computation is running.

Figure 6.3 shows how the CPU time required for the controller of the batch reactor process for different implementations using Leon2 processor vary for different duration of runtime between $500s$ and $5000s$ when there is no external disturbance acting on the system. The figure shows that the naive self-triggered implementation always takes significantly more computation time than the time-triggered implementation. However, our hybrid implementation is always close to the time-triggered implementation in terms of computation time, and as time progresses and the memoization table gets filled up with trigger times, the difference between the computation time for our hybrid implementation and that of the time-triggered implementation slowly decreases.

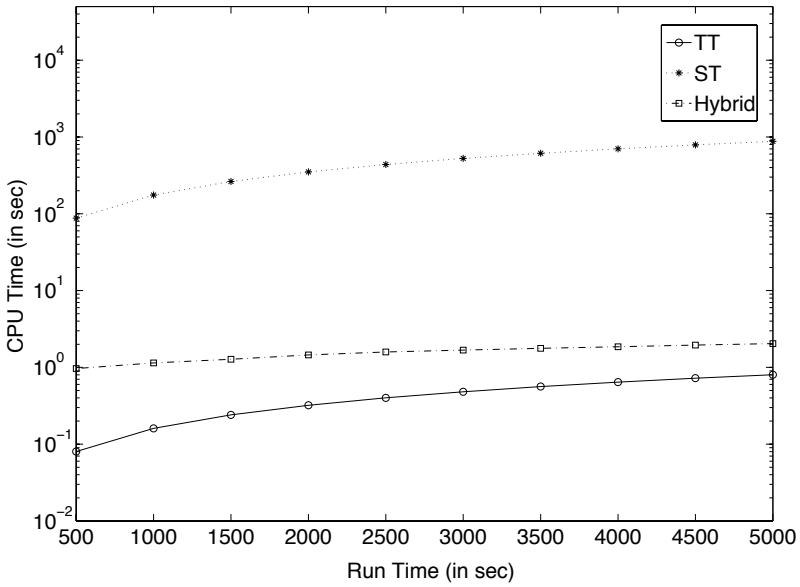


Figure 6.3: CPU time required for batch reactor process for different implementations using the Leon2 processor for different running times in the disturbance-free scenario

Nonlinear System: Jet Engine Compressor. We illustrate our experimental results on nonlinear self triggered control systems using the standard benchmark of a jet engine compressor. The model of the system originally appeared in [KK95b]. In [AT10], it was adapted to translate the desired equilibrium point to the origin as follows:

$$\begin{aligned} \dot{\xi_1} &= -\xi_2 - \frac{3}{2}\xi_1^2 - \frac{1}{2}\xi_1^3 \\ \dot{\xi_2} &= \frac{1}{\beta^2}(\xi_1 - u) \end{aligned} \tag{6.31}$$

where ξ_1 and ξ_2 denote the mass flow and pressure rise in the compressor respectively. The symbol β is a constant positive parameter. The output of the controller u denotes the throttle mass flow.

A control law $u = k(\xi_1, \xi_2)$ was designed in [AT10] to render the system in (6.31) globally asymptotically stable. The control law is given by:

$$u = \xi_1 + \frac{\beta^2}{4}(z(\xi_1^2 + 1) + 3\xi_1^2 y + \xi_1^3 + \xi_1 + 3y) \tag{6.32}$$

where $y = (3\xi_1^2 + 2\xi_2 - \xi_1)/(\xi_1^2 + 1)$ and $z = 2\xi_1 y(y-3) + \xi_1^2(4y-6)$. By substituting u with its corresponding expression, the closed loop system becomes

$$\begin{aligned} \dot{\xi_1} &= -\frac{1}{2}(\xi_1^2 + 1)(\xi_1 + y) \\ \dot{\xi_2} &= -(\xi_1^2 + 1)y \end{aligned} \tag{6.33}$$

The following ISS Lyapunov function is provided in [AT10]

$$V = 1.46\xi_1^2 - 0.35\xi_1 y + 1.16y^2,$$

which is computed using SOStools [PPS04]. Using this Lyapunov function, this can be shown that the state of the closed loop system $\xi = (\xi_1, y)^T$ and the measurement error on the state $e = (e_1, e_2)^T$ are related by the following inequality:

$$0.90\|e\|^2 \leq 0.74\nu^2\|\xi\|^2$$

Now based on the specification that the state ξ is contained in the invariant set $\{\xi \in \mathbb{R}^n | V(\xi) = 27.04\}$, the following formula is provided in [AT10] to compute the trigger time:

$$t_{k+1}(\xi(t_k)) = \frac{29\xi_1(t_k) + \|\xi(t_k)\|^2}{5.36\|\xi(t_k)\|\xi_1(t_k)^2 + \|\xi(t_k)\|^2}\tau^*,$$

where $\tau^* = 7.63ms$ for $\nu = 0.33$.

The operating point of the jet engine is $\xi_1 = 0, \xi_2 = 0$. The state ξ_1 is mass flow, which is a positive quantity. The state ξ_2 is pressure rise, which may be both positive and negative. We choose the memoization region to be $\langle \xi_1 \in [0, 0.5],\ \xi_2 \in [-1, 1] \rangle$. The minimum trigger time $\tau_{min} = \tau^* = 7.63ms$. We choose the maximum trigger time $\tau_{max} = 250ms$, and store the trigger times in the memoization table as a fixed-point number with representation $\langle 8, 10 \rangle$. This enables us to store the trigger time in 1 byte. We assume that we have 256KB memory available to store the memoization table. With this amount of memory, the quantization factor can be chosen to be 0.002 in each dimension, using (6.13). For this quantization

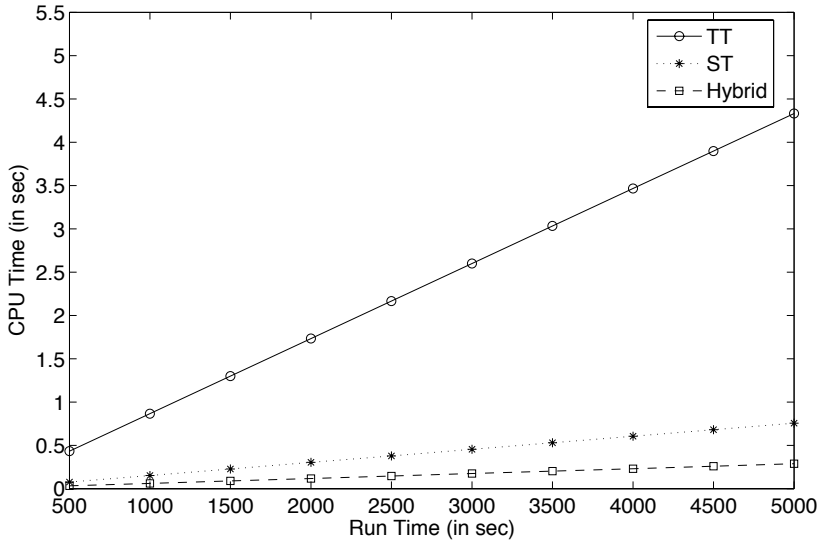


Figure 6.4: CPU time required for the jet engine compressor for different implementations on the PowerPC processor for different running times for disturbance scenario 2

factor, we solve the optimization problem (6.29) to find out the value of τ_e^{max}. The optimization problem is not convex and we solve the problem numerically by dividing the region $\langle \xi_1 \in [0, 0.5],\ \xi_2 \in [-1, 1] \rangle$ into a grid with precision 0.0001 and exhaustively searching all the points on the grid. We find that the value of τ_e^{max} for the values of ξ_1 in the range $[0, 0.25]$ is fairly high (greater than the minimal trigger time 7.63ms). We shift the memoization region in the ξ_1 direction to exclude the region from the memoization region. Thus our memoization region is $\langle \xi_1 \in [0.25, 0.75],\ \xi_2 \in [-1, 1] \rangle$. In the memoization region, τ_e^{max} is 6.634ms.

We consider two scenarios to compare the performance of the hybrid implementation with the time-triggered and self-triggered implementations without memoization. In both scenarios we consider the evolution of the system for $2000s$ in the presence of external disturbances. In the first scenario, the disturbance signal is a periodic pulse signal with amplitude 8, period 400 samples, pulse width 40 samples, zero phase delay and sample time $10ms$. In the second scenario, the disturbance is a normally distributed random signal with mean 1, variance 1, and

Implementation	Disturbance Scenario 1			Disturbance Scenario 2		
	Computation		Commu-	Computation		Commu-
	PowerPC	Leon2	nication	PowerPC	Leon2	nication
TT	1.094s	1.732s	262122	1.094s	1.732s	262122
ST	0.338s	0.473s	18729	0.303s	0.422s	16447
Hybrid	0.190s	0.315s	19009	0.117s	0.211s	16464

Table 6.3: CPU time and communication cost for different implementations of the controller of the jet engine compressor running for 2000s

sample time $1ms$. We do not show results for the scenario when no external disturbance is present, as the memoization-based implementation does not provide any benefit due to not including a region around the origin (ξ_1 is in the range $[0, 0.25]$). Table 6.3 shows the CPU time and the communication cost for the two processors for different implementations of the controller for runs of length $2000s$. The results show that a self-triggered implementation without memoization itself gives significant savings on both the CPU time and number of communications. However, with memoization, we can gain even more in CPU time with a slight increase in the number of communications. The number of communications increases in the hybrid implementation as we need to decrease the trigger time to correct any error arising due to quantization of states in the trigger time computation. On both processors, the time to compute the trigger time is always less than the minimum trigger time τ_{min}. Thus the hybrid implementation never falls back to the time-triggered implementation, and the number of communications for both the processors are the same.

Figure 6.4 shows how the CPU time required for jet engine compressor for different implementations using PowerPC processor vary in the disturbance scenario 2 for different duration of runtime between $500s$ and $5000s$. The figure shows that the savings in CPU time grows with time from the time-triggered implemen-

tation to self-triggered implementation without memoization, and also from the self-triggered implementation without memoization to the hybrid implementation. As time progresses, more and more trigger times are memoized, and the ratio of cache hit to cache miss also increases. Thus the CPU time requirement keeps on decreasing with time for the hybrid implementation.

6.5 Related Work

Memo table based implementations for explicit model predictive control have been suggested before [BMD02, AB09], where the control signals for a compact state space, computed by solving an optimization problem, are stored in a lookup table for fast actuation. We show memo tables allow fast implementation of self-triggered controllers as well, even with dynamic computations in case it is not feasible to pre-compute the entire table.

CHAPTER 7

Static Scheduling

From now on, we will focus on the controller-scheduler co-design problem in the context of implementing multiple control systems on a single processor. In this chapter, we present a framework to design a static scheduler for a number of control applications running on a single processor. The objective of the scheduler is not only to ensure stability of the control systems but also achieve optimal performance in terms of disturbance rejection. We exploit the observation that control systems do not always achieve optimal performance when a control signal is transmitted to the plant in every cycle. We provide a precise analysis of the relationship between the fraction of times when control tasks are "dropped" by the scheduler and the performance of the system. In a second step, we use this information to derive a scheduler that occasionally drops a control signal but ensures that the rate of dropped control signals corresponds to the rate for optimal performance. Thus, in our methodology, we achieve end-to-end performance but keep a separation of concerns: a purely modular, control theoretic approach provides the relationship between the performance of the control systems and the rate of drops, and a purely software-related scheduler synthesis exploits the slack.

Technically, we make two contributions. First, we give a control theoretic calculation that provides a relationship between the long run average of the dropped packets with the performance of a control system. We model control systems in which a scheduler can discard control computations in certain cycles as networked control systems. The action of the scheduler is modeled as a switch that deter-

mines if the control signal reaches the plant (the task is scheduled) or gets dropped (the task is discarded). In the latter case, the actuator values are kept fixed to their values in the previous cycle. Using results from networked control systems [BPZ02], we give bounds on the fraction of drops that still guarantees the stability of the plant, as well as derive a relationship between the upper bound of the $\mathcal{L}_\infty$ to RMS gain of the plant (a standard measure of control systems performance) and the fraction of packet drops, as well as an optimization problem in terms of linear matrix inequalities that can be used to constructively determine this relationship. We note that the relationship between packet-drop rate and performance is not monotonic, as might be expected. In particular, if a system is not schedulable, simply adding more resources to make it schedulable (as done in current practice) may not necessarily yield a better performance.

Second, we provide an automatic technique that takes control tasks, their periods, worst-case execution times and deadlines, as well as the rates at which control signals should be dropped, and outputs a static schedule (if possible) that guarantees that in a given period of time, the number of control signals are dropped in such a way that the optimal performance is achieved for each control loop. Our algorithm sets up the scheduler synthesis problem as a constraint satisfaction question, where the variables encode which task gets run in which time slot, and uses off-the-shelf SMT solvers to find models of the constraints.

We have implemented our methodology in a tool that determines schedulability of multiple control tasks sharing the same computation platform. We expect that a separate tool computes worst case execution times, but use off-the-shelf linear matrix inequality solvers (based on semi-definite programming) as well as SMT solvers to synthesize performance bounds and static schedulers. We have applied our implementation on a standard benchmark of multiple inverted pendulums sharing the same processor [ZSW08], and show how we can completely automatically derive schedulers ensuring optimal performance levels.

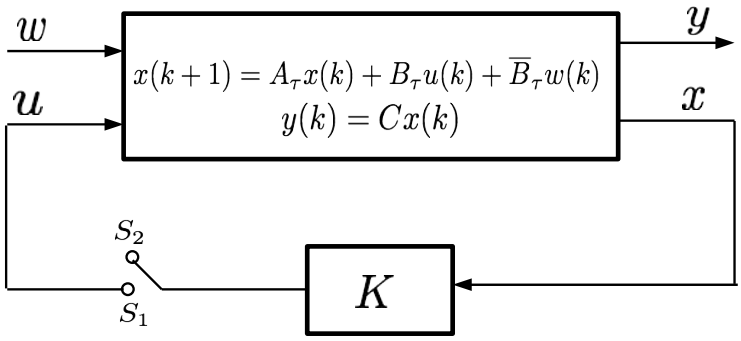


Figure 7.1: Linear control system with dropout

7.1 Control Systems Performance with Limited Computation

We now describe a control theoretic formulation of the behavior of a linear time invariant control system in which the control law may not be computed at every step. We consider the discrete-time linear time-invariant control system in (2.3). However, here our focus is on disturbance rejection, and we ignore the effect of measurement noise.

7.1.1 Scheduled Linear Control Systems

Owing to the transmission delay from the sensor to the controller, and the computation time of the controller, the control signal cannot be applied to the very state based on which the control signal is computed. Rather, the control input at the k-th time step is computed based on the state of the plant at the $(k-1)$-th time step. For simplicity, we assume that this one time unit delay is enough to accommodate the transmission time of the sensor data to the controller and the computation time of the controller. In this chapter, we assume that the sensor data instantaneously reaches the controller and only the computation time to be the only reason for the delay in applying control signal to the plant. In Chapter 8

we will introduce a network between the sensor and the controller, and would deal with the delay introduced by the network.

We consider *state feedback controllers* of the form $u(k) = -Kx(k-1)$, where the matrix K is called the *gain* of the controller. The aim of the controller is to ensure that the closed-loop system has certain properties, such as exponential stability, and a certain performance. Standard control-theoretic computations allow us to obtain state feedback controllers with the desired properties (see [AW90b]).

The intuition behind our model is as follows. In each discrete time step $k = 1, 2, \ldots$, the current state $x(k)$ is observed, and the scheduler tries to schedule the control computation task if possible. If the control task is scheduled, the signal $u(k) = -Kx(k-1)$ is computed and applied to the actuators. In time steps k at which the control signal is not computed (i.e., in which the scheduler does not schedule the control task), we expect the controller to retain its previous value: $u(k) = u(k-1)$.

We model the scheduled control system as a networked control system with a loss of "data packets" between the controller and the plant, that is, we model the link between the controller and the plant as a channel with a switch (see Figure 7.1). In each time step, if the switch is closed (indicating that the scheduler ran the control task), the control signal is the updated control law, but if the switch is open (indicating that the scheduler could not run the control task), the control signal is unchanged from the previous control signal. Cycles in which the switch is open are said to incur *dropouts* of the control signal. Our main result is the effect of dropouts on the performance of the control system.

In Figure 7.1, when the switch is closed (position S_1), the output of the controller is transmitted to the plant, and when the switch is open (position S_2), the output of the switch is held at the previous value. Similar to [BPZ02], the

dynamics of the switch can be modeled formally as follows:

$$\text{When switch is in position } S_1: \quad u(k) = -Kx(k-1), \tag{7.1}$$

$$\text{When switch is in position } S_2: \quad u(k) = u(k-1). \tag{7.2}$$

Define the signal $s(k) = 1$ if the switch is in position S_1 at the k^{th} time step, and $s(k) = 2$ if the switch is in position S_2 at the k^{th} time step. These correspond to the control input being computed or not.

By choosing $X = [x^T, u^T]^T$ as the new state vector, the closed loop system with dropout, depicted in Figure 7.1, is given by:

$$\begin{aligned} X(k+1) &= \widetilde{A}_{s(k)}X(k) + \widetilde{B}_{1s(k)}w(k) \\ y(k) &= \widetilde{C}_{s(k)}X(k), \end{aligned} \tag{7.3}$$

where, using the dynamics of the switch in (7.1) and (7.2), we obtain

$$\widetilde{A}_1 = \begin{bmatrix} A_\tau & B_\tau \\ -K & 0_{m\times m} \end{bmatrix}, \widetilde{B}_{11} = \begin{bmatrix} \overline{B}_\tau \\ 0_{m\times l} \end{bmatrix}, \widetilde{C}_1 = [C\ 0_{p\times m}];$$

and

$$\widetilde{A}_2 = \begin{bmatrix} A_\tau & B_\tau \\ 0_{m\times n} & I_{m\times m} \end{bmatrix}, \widetilde{B}_{12} = \begin{bmatrix} \overline{B}_\tau \\ 0_{m\times l} \end{bmatrix}, \widetilde{C}_2 = [C\ 0_{p\times m}],$$

where $0_{m\times n}$ denotes the zero matrix in $\mathbb{R}^{m\times n}$ and $I_{m\times m}$ denotes the identity matrix in $\mathbb{R}^{m\times m}$.

Note that the successful transmission rate in this paper is the rate at which the switch in Figure 7.1 is in position S_1. Following [HBH99], the *successful transmission rate* r is given by

$$r = \lim_{L\to\infty} \frac{1}{L} \sum_{k=0}^{L} (2 - s(k)).$$

The dropout rate means the rate at which the switch in Figure 7.1 is in S_2. If the successful transmission rate for a control system is r, its dropout rate

is $1 - r$. Clearly, in the former case, the scheduler runs the control task and updates the actuator, and in the latter case, it does not. The following theorem provides a sufficient condition on the successful transmission rate that guarantees the stability of the closed loop system.

Theorem 5. *Consider the LTI control system in* (7.3)*. Let r be the successful transmission rate. Assume that the closed loop system with no dropout and no disturbance is stable (i.e., $\max_i|\lambda_i(A_\tau - B_\tau K)| < 1$, where $\lambda_i(A_\tau - B_\tau K)$ is the i-th eigenvalue of the matrix $A_\tau - B_\tau K$). Then the LTI control system with dropout in* (7.3)*, with no disturbance, is exponentially stable for all*

$$r_{\min} < r \leq 1,$$

where $r_{\min} = \frac{log(\beta_2)}{log(\beta_2)-log(\beta_1)}$, $\beta_1 = \max_i|\lambda_i(\widetilde{A}_1)|^2$, and $\beta_2 = \max_i|\lambda_i(\widetilde{A}_2)|^2$, $\beta_1 < 1$ and $\beta_2 > \beta_1$.

Proof. Following Theorem 6 in [ZBP01] it can be shown that for the LTI control system in (7.3), if there exists a Lyapunov function $V(x(kh)) = x^T(kh)Px(kh)$ and scalars α_1 and α_2 such that

$$\alpha_1^r \alpha_2^{1-r} > 1 \tag{7.4}$$

$$\widetilde{A}_1^T P \widetilde{A}_1 \leq \alpha_1^{-2} P \tag{7.5}$$

$$\widetilde{A}_2^T P \widetilde{A}_2 \leq \alpha_2^{-2} P \tag{7.6}$$

then the system is exponentially stable.

Let $\beta_i = \alpha_i^{-2}$ for $i = 1, 2$. From (7.5) and (7.6) we get that $\beta_1 = \max_i|\lambda_i(\widetilde{A}_1)|^2$ and $\beta_2 = \max_i|\lambda_i(\widetilde{A}_2)|^2$. Now by taking the log in (7.4) we get,

$$r > \frac{log(\beta_2)}{log(\beta_2) - log(\beta_1)}.$$

The constraint $\beta_1 < 1$ and $\beta_2 > \beta_1$ ensures that $r_{min} < 1$. ∎

Note that if $\beta_2 < 1$, then $r_{min} < 0$. If $\beta_2 < 1$, then the open loop system is stable, and a controller is not required for the stability of the system. For any unstable system, $\beta_2 > 1$ and $0 < r_{min} < 1$.

7.1.2 Bound on $\mathcal{L}_\infty$ to RMS gain

The following theorem provides a relationship between the successful transmission rate and the upper bound of the $\mathcal{L}_\infty$ to RMS gain for the control system in (7.3).

Theorem 6. *Consider the discrete time LTI control system in* (7.3) *with the successful transmission rate* r*. The* $\mathcal{L}_\infty$ *to RMS gain is less than positive constant* γ *if there exists a piecewise continuous function* $V : \mathbb{R}^{n+m} \to \mathbb{R}_{\geq 0}$*, such that* $V(0) = 0$*, and* $\gamma_1, \gamma_2 \in \mathbb{R}$ *such that*

$$r\gamma_1^2 + (1-r)\gamma_2^2 < \gamma^2, \tag{7.7}$$

and

$$V\left(\widetilde{A}_i X + \widetilde{B}_{1i} w\right) - V(X) \leq \gamma_i^2 w^T w - y^T y, \text{ for } i = 1, 2. \tag{7.8}$$

Proof. We proceed as in [HBH99], adapting the argument to discrete time control systems. Using inequality (7.8), we have:

$$\begin{aligned} V(X(k+1)) - V(X(k)) &\leq \gamma_i^2 w(k)^T w(k) - y(k)^T y(k) \\ &\leq \gamma_i^2 \|w\|_\infty^2 - y(k)^T y(k), \end{aligned} \tag{7.9}$$

where either $i = 1$ or $i = 2$ because at time instant k, the switch in Figure 7.1 is either closed or open. By summing up inequalities (7.9) for $k = 0, 1, \cdots, l$, we

obtain:

$$V(X(l+1)) - V(X(0)) \leq \gamma_1^2 \begin{pmatrix} \text{total times the} \\ \text{switch is closed} \\ \text{between 0 and } l \end{pmatrix} \|w\|_\infty^2$$

$$+\gamma_2^2 \begin{pmatrix} \text{total times the} \\ \text{switch is open} \\ \text{between 0 and } l \end{pmatrix} \|w\|_\infty^2 - \textstyle\sum_{k=0}^{k=l} y(k)^T y(k).$$

In the limit, when $l \to +\infty$, the total times that the switch is closed is equal to rl and the total times that the switch is open is equal to $(1-r)l$. Therefore, since $X(0) = 0$ in (2.17), we have:

$$V(X(l+1)) \leq \gamma_1^2 rl\|w\|_\infty^2 + \gamma_2^2(1-r)l\|w\|_\infty^2 - \sum_{k=0}^{k=l} y(k)^T y(k).$$

Using (7.7) and $V(X(l+1)) \geq 0$, we obtain:

$$\frac{\frac{1}{l}\sum_{j=0}^{l} y^T(j)y(j)}{\|w\|_\infty^2} \leq r\gamma_1^2 + (1-r)\gamma_2^2 < \gamma^2. \tag{7.10}$$

Therefore, we get:

$$\frac{\limsup_{l\to\infty} \frac{1}{l}\sum_{j=0}^{l} y^T(j)y(j)}{\|w\|_\infty^2} < \gamma^2, \tag{7.11}$$

which completes the proof. ∎

In the next lemma, we show that by choosing $V(X) = X^T P X$, the inequality (7.8) becomes a Linear Matrix Inequality (LMI) for the control system in (7.3).

Lemma 1. *Consider the control system in* (7.3). *Using* $V(X) = X^T P X$, *the inequality (7.8) becomes an LMI as follows:*

$$\begin{bmatrix} \widetilde{A}_i^T P \widetilde{A}_i - P + \widetilde{C}_i^T \widetilde{C}_i & \widetilde{A}_i^T P \widetilde{B}_{1i} \\ \widetilde{B}_{1i}^T P \widetilde{A}_i & \widetilde{B}_{1i}^T P \widetilde{B}_{1i} - \gamma_i^2 \end{bmatrix} \leq 0. \tag{7.12}$$

Proof. In the proof, we drop the arguments of X and w for simplicity.

$$V\left(\widetilde{A}_i X + \widetilde{B}_{1i} w\right) - V(X) = (\widetilde{A}_i X + \widetilde{B}_{1i} w)^T P (\widetilde{A}_i X + \widetilde{B}_{1i} w)$$
$$-X^T P X = X^T \widetilde{A}_i^T P \widetilde{A}_i X + w^T \widetilde{B}_{1i}^T P \widetilde{A}_i X + X^T \widetilde{A}_i^T P \widetilde{B}_{1i} w$$
$$+w^T \widetilde{B}_{1i}^T P \widetilde{B}_{1i} w - X^T P X \leq \gamma_i^2 w^T w - X^T \widetilde{C}_i^T \widetilde{C}_i X.$$

By arranging the terms we obtain:

$$X^T \left[\widetilde{A}_i^T P \widetilde{A}_i - P + \widetilde{C}_i^T \widetilde{C}_i\right] X + w^T \widetilde{B}_{1i}^T P \widetilde{A}_i X \tag{7.13}$$
$$+X^T \widetilde{A}_i^T P \widetilde{B}_{1i} w + w^T \left[\widetilde{B}_{1i}^T P \widetilde{B}_{1i} - \gamma_i^2\right] w \leq 0.$$

The inequality (7.13) can be rewritten as:

$$\begin{bmatrix} X \\ w \end{bmatrix}^T \begin{bmatrix} \widetilde{A}_i^T P \widetilde{A}_i - P + \widetilde{C}_i^T \widetilde{C}_i & \widetilde{A}_i^T P \widetilde{B}_{1i} \\ \widetilde{B}_{1i}^T P \widetilde{A}_i & \widetilde{B}_{1i}^T P \widetilde{B}_{1i} - \gamma_i^2 \end{bmatrix} \begin{bmatrix} X \\ w \end{bmatrix} \leq 0.$$

Since $[x^T \ w^T]$ is an arbitrary vector, the previous inequality is equivalent to the following LMI:

$$\begin{bmatrix} \widetilde{A}_i^T P \widetilde{A}_i - P + \widetilde{C}_i^T \widetilde{C}_i & \widetilde{A}_i^T P \widetilde{B}_{1i} \\ \widetilde{B}_{1i}^T P \widetilde{A}_i & \widetilde{B}_{1i}^T P \widetilde{B}_{1i} - \gamma_i^2 \end{bmatrix} \leq 0, \tag{7.14}$$

which completes the proof. ∎

Note that by choosing $V(X) = X^T P X$ and for a given successful transmission rate r, we can minimize γ, the upper bound of the $\mathcal{L}_\infty$ to RMS induced gain, by solving the following optimization problem:

$$\begin{aligned} &\text{minimize} \quad r\gamma_1^2 + (1-r)\gamma_2^2 \\ &\text{subject to} \\ &\begin{bmatrix} \widetilde{A}_i^T P \widetilde{A}_i - P + \widetilde{C}_i^T \widetilde{C}_i & \widetilde{A}_i^T P \widetilde{B}_{1i} \\ \widetilde{B}_{1i}^T P \widetilde{A}_i & \widetilde{B}_{1i}^T P \widetilde{B}_{1i} - \gamma_i^2 \end{bmatrix} \leq 0 \\ &\qquad\qquad\qquad\qquad \text{for } i = 1, 2 \\ &\gamma_1, \gamma_2 \in \mathbb{R}, \\ &P > 0 \end{aligned} \tag{7.15}$$

For $0 < r \leq 1$, we denote by $\gamma(r)$ the upper bound on the $\mathcal{L}_\infty$ to RMS gain obtained by solving the optimization problem (7.15) for this choice of r.

7.1.3 Scheduled Nonlinear Control Systems

We now provide an extension of the preceding results to nonlinear control systems in discrete time. We consider nonlinear control systems in discrete time given by the following difference equations:

$$\begin{aligned} x(k+1) &= f\left(x(k), u(k), w(k)\right), \\ y(k) &= h\left(x(k)\right), \end{aligned} \tag{7.16}$$

where f and h are smooth maps and $x(k)$, $u(k)$, $w(k)$, and $y(k)$ belong to the same spaces as in (2.3). Here, we consider state feedback controllers of the form $u(k) = k(x(k))$, where k is a smooth map. As in the linear case, the dynamics of the switch can be modeled formally as the following:

$$\text{When switch is in position } S_1 \;:\; u(k) = k\left(x(k-1)\right), \tag{7.17}$$

$$\text{When switch is in position } S_2 \;:\; u(k) = u(k-1). \tag{7.18}$$

By following the same strategies as we did for linear systems and choosing $X = [x^T, u^T]^T$ as the new state vector, the closed loop system with dropout is given by:

$$\begin{aligned} X(k+1) &= \widetilde{f}_{s(k)}\left(X(k), w(k)\right), \\ y(k) &= \widetilde{h}_{s(k)}\left(X(k)\right), \end{aligned} \tag{7.19}$$

where, using the dynamics of the switch in (7.17) and (7.18), we obtain:

$$\begin{aligned} \widetilde{f}_1\left(X(k), w(k)\right) &= \begin{bmatrix} f\left(x(k), u(k), w(k)\right) \\ k\left(f\left(x(k-1), u(k-1), w(k-1)\right)\right) \end{bmatrix}, \\ \widetilde{h}_1\left(X(k)\right) &= h\left(x(k)\right); \end{aligned} \tag{7.20}$$

and

$$\begin{aligned} \widetilde{f}_2\left(X(k), w(k)\right) &= \begin{bmatrix} f\left(x(k), u(k), w(k)\right) \\ u(k) \end{bmatrix}, \\ \widetilde{h}_2\left(X(k)\right) &= h\left(x(k)\right). \end{aligned} \tag{7.21}$$

The following theorem provides a sufficient condition on the successful transmission rate that guarantees the stability of the closed loop system.

Theorem 7 ([HBH99])**.** *Consider the nonlinear control system with dropout in (7.19) with the successful transmission rate r. The system with no disturbance is exponentially stable with decay rate greater than γ if there exists a piecewise continuous function $V : \mathbb{R}^{n+m} \to \mathbb{R}_{\geq 0}$, and γ_1, γ_2, β_1, $\beta_2 \in \mathbb{R}_{>0}$, such that*

$$\beta_1 \|X\|_2^2 \leq V(X) \leq \beta_2 \|X\|_2^2, \tag{7.22}$$

$$\gamma_1^r \gamma_2^{(1-r)} > \gamma > 1, \tag{7.23}$$

and

$$V\left(\widetilde{f}_i\left(X, 0\right)\right) - V(X) \leq \left(\gamma_i^{-2} - 1\right) V(X), \text{ for } i = 1, 2. \tag{7.24}$$

As in the linear case, we can prove a similar relationship between the successful transmission rate and the $\mathcal{L}_\infty$ to RMS gain for the nonlinear control system in (7.19). The proof of the following theorem is the same as the proof of Theorem 6.

Theorem 8. *Consider the nonlinear control system with dropout in (7.19) with the successful transmission rate r. The $\mathcal{L}_\infty$ to RMS gain is less than positive constant γ if there exists a piecewise continuous function $V : \mathbb{R}^{n+m} \to \mathbb{R}_{\geq 0}$, such that $V(0) = 0$, and $\gamma_1, \gamma_2 \in \mathbb{R}$ such that*

$$r\gamma_1^2 + (1 - r)\gamma_2^2 < \gamma^2, \tag{7.25}$$

and

$$V\left(\widetilde{f}_i\left(X, w\right)\right) - V(X) \leq \gamma_i^2 w^T w - y^T y, \text{ for } i = 1, 2. \tag{7.26}$$

Note that if functions f and h in (7.16) and the state feedback controller k are polynomial with respect to their arguments, by using SOS programming [PPS04] we can search for functions V, decay rates and upper bounds of $\mathcal{L}_\infty$ to RMS gain satisfying the inequalities in Theorems 7 and 8 for given values of r.

7.1.4 Finding the Optimal Successful Transmission Rate

We now consider the problem of choosing the successful transmission rate for a given controller to achieve the best performance. The successful transmission rate at which the best performance is achieved is called the *optimal successful transmission rate* and is denoted by r_{opt}. A lower bound on the rate for each system is given by Theorem 5: this is the rate $r_{\min}$ required to ensure stability. An upper bound $r_{\max}$ on the rate is decided by the scheduling constraints. If there is no scheduling constraint, $r_{\max} = 1$.

Now, the following theorem provides the possible successful transmission rates that may achieve optimal performance.

Theorem 9. *The $\mathcal{L}_\infty$ to RMS gain of the discrete time LTI control system in* (7.3) *attains the minimum value for the successful transmission rate to be either at $r_{\min}$ or at $r_{\max}$.*

Proof. Let us assume that the values of γ_1 and γ_2 for which the $\mathcal{L}_\infty$ to RMS gain attains the minimum value are $\gamma_{1\text{opt}}$ and $\gamma_{2\text{opt}}$. Now there may be three cases:

Case 1: $\gamma_{1\text{opt}} > \gamma_{2\text{opt}}$. Note that the LMIs in the constraints (7.15) in the optimization problem in (7.15) do not depend on r. Thus a solution for γ_1 and γ_2 is valid for any successful transmission rate. In this case, if we decrease the value of r for the same values of γ_1 and γ_2, the value of γ also decreases with r. In this case, the minimum value of γ is obtained at $r_{\min}$.

Case 2: $\gamma_{1\text{opt}} > \gamma_{2\text{opt}}$. By using similar argument as Case 1, we can show that the $\mathcal{L}_\infty$ to RMS gain attains the minimal value at $r_{\max}$.

Case 3: $\gamma_{1\text{opt}} = \gamma_{2\text{opt}}$. In this case, γ becomes equal to γ_2 and thus remains constant for any successful transmission rate. Thus $r_{\max}$ and $r_{\min}$ both gives the optimal value for the $\mathcal{L}_\infty$ to RMS gain. ∎

7.1.5 Motivating Example

As a motivating example, we use the model of the inverted pendulum from [ZSW08]. The state-space representation of an inverted pendulum is given by:

$$\begin{aligned} \dot{x} &= Ax + Bu + \overline{B}w; \\ y &= Cx, \end{aligned}$$

where

$$A = \begin{bmatrix} 0 & 1 \\ \frac{g}{l} & \frac{\rho}{ml^2} \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \frac{1}{ml} \end{bmatrix}, \tag{7.27}$$
$$\overline{B} = \begin{bmatrix} 0.1 \\ 0 \end{bmatrix}, \quad C = [0.001,\ 0].$$

In this model, $x = [x_1, x_2]^T$ is the state of the system, with x_1 the angular position and x_2 the angular velocity of the point mass, m is the mass, l is the length of the rod, $g = 9.8\text{m/s}^2$ is acceleration due to gravity, ρ is the rotational friction coefficient, u is the applied force (control input), and w is the disturbance input.

Let us consider an instance of the above system where $\rho = 0.6$, $m = 0.4$ and $l = 0.6$. All values are in S.I. units. We discretize the plant in (7.27) with sampling period of $20ms$. A stabilizing controller for the discretized plant is given by $K_1 = [4.8462\ 0.1800]$. The minimum successful transmission rate for this system to ensure stability is $r_{\min} = 0.6623$.

Now we plot how the upper bound on the $\mathcal{L}_\infty$ to RMS gain varies with the successful transmission rate between $r_{\min}$ and $r_{\max} = 1$. The figure is obtained by quantizing the successful transmission rates between $r_{\min}$ and $r_{\max}$ with a

quantization factor 0.01, and then solving the optimization problem in (7.15) for each choice of the successful transmission rate. For a given controller, we refer to the plot of transmission rate vs. performance as the *performance profile* of the controller. The curve in Figure 7.2 shows the performance profile for the controller K_1.

Note that we use the upper bound on the $\mathcal{L}_\infty$ to RMS gain instead of the $\mathcal{L}_\infty$ to RMS gain for a particular disturbance pattern while constructing the performance profile. Though different successful transmission rate may be the best for different disturbance pattern, we are interested to find out a successful transmission rate for which we can guarantee the minimum bound on the $\mathcal{L}_\infty$ to RMS gain, even if this successful transmission rate may not be the best for all possible disturbance pattern.

Now we come to the problem of choosing the optimal successful transmission rate. Due to scheduling constraints, the value of $r_{\max}$ may be less than 1. The value of r_{opt} depends on the value of $r_{\max}$. In our present example, if $r_{\max} \leq 0.87$ then $\gamma(r_{\min}) \leq \gamma_m(r_{\max})$. Thus, the choice of optimal successful transmission rate is $r_{\min}$, as in that case choosing $r_{\min}$ would give the optimal performance and optimal CPU time usage. If $r_{\max} > 0.87$ then $\gamma(r_{\max}) < \gamma(r_{\min})$. If $r_{\max}$ is permitted by the scheduling constraints to be greater than 0.87, $r_{\max}$ becomes the optimal successful transmission rate. This example illustrates that the scheduling constraints have effect on the choice of the operating successful transmission rate.

7.2 Optimal Performance Scheduler Synthesis

Suppose we have N control systems that are all sharing the same computational resources. Each controller is implemented as a software task C_i that computes the control law. Each task C_i has two parameters $\langle h_i, c_i \rangle$, where h_i denotes the sampling time and c_i the worst-case execution time of the task. A task will be

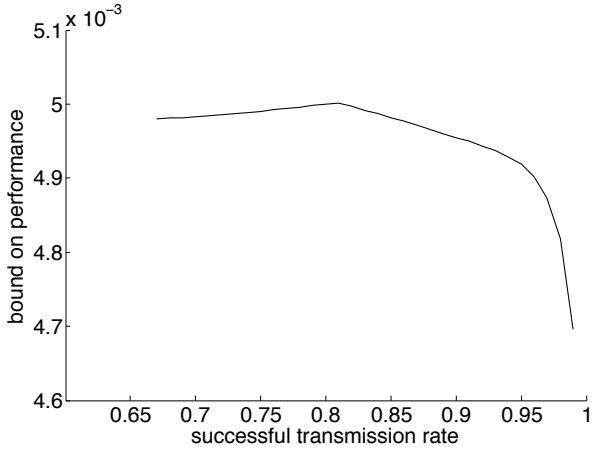


Figure 7.2: The upper bound of the $\mathcal{L}_\infty$ to RMS gain vs successful transmission rate for an inverted pendulum for $K_1 = [4.8462\ 0.1800]$

active immediately after the beginning of a sampling period and, following [LL73], we assume that the deadline for the control task is the end of the sampling period. Suppose we have chosen successful transmission rates r_i for each control task. The system $\{(C_i, r_i) \mid i \in \{1, \ldots, N\}\}$ is *schedulable with dropout* if it is possible to schedule the tasks so that each scheduled task finishes before its deadline, but the scheduler can drop r_i fraction of the task C_i. The following proposition follows using an argument similar to Theorem 7 in [LL73].

Proposition 4. *The system* $\{(C_i, r_i) \mid i \in \{1, \ldots, N\}\}$ *is schedulable with dropout iff* $\sum_{i=1}^{N} r_i c_i / h_i \leq 1$.

Notice that if a system is schedulable with dropout, then it remains schedulable with dropout if the rates are decreased. There are two issues in designing a scheduler. First, how should we choose the rates r_i so that the system is schedulable with dropout? Second, having chosen the rates such that the system is schedulable with dropout, how can we assign a static schedule that schedules all control tasks and ensures the rates chosen?

We now consider the problem of choosing the successful transmission rates. A lower bound on the rate for each system is given by Theorem 5: this is the rate r_{min}

required to ensure stability. Figure 7.2 shows how the upper bound on the $\mathcal{L}_\infty$ to RMS gain varies with the successful transmission rate for an inverted pendulum. As can be seen from Figure 7.2, the upper bound on the $\mathcal{L}_\infty$ to RMS gain does not change monotonically with respect to the successful transmission rate. Ideally, for each system, we should choose the successful transmission rate for which the bound on the $\mathcal{L}_\infty$ to RMS gain attains the minimum. However, when there are multiple control systems, the controllers compete for scheduling resources. Thus it may not be possible to accommodate the best successful transmission rates for all the control systems. For scheduler synthesis, we rather consider a set of successful transmission rates that we call *eligible rates* for scheduler synthesis.

Fix a control system with dropout. A successful transmission rate r is called *eligible* if it satisfies the following two conditions:

- $r \geq r_{\min}$, where r_{min} is as in Theorem 5,
- for each $r' \in [r_{\min}, r)$, we have $\gamma(r') \geq \gamma(r)$.

The eligible rates provide "undominated" solutions: one cannot simultaneously reduce r and get better performance. We would like to find points on the Pareto curve of eligible rates for the N systems.

For each system C_i, we calculate the lower bound $r_{\min,i}$. We discretize the range $[r_{\min,i}, 1]$ with a chosen discretization factor and find the subset E_i of the discrete points that are eligible. We order the points in E_i by the total ordering $\preceq$: for $r_1, r_2 \in E_i$, if $r_1 \preceq r_2$ then $\gamma(r_2) \leq \gamma(r_1)$, that is, "higher" values in $\preceq$ give better performance. Let $r_i^{best} \in E_i$ denote the maximal (in the $\preceq$-ordering) successful transmission rate in E_i.

We now have a multi-objective optimization problem: choose points in $E_1 \times \ldots \times E_N$ that optimize the performance of each system. Though the performance of each control system is independently specified, the controllers compete

for scheduling resources, and we may not be able to choose r_i^{best} for each control system. Instead, we look for undominated solutions.

One way to deal with this problem is to use a ranking method [YH95]. In this method, the objectives in the multi-objective optimization problem are ranked based on their importance. Of the N objectives, the rank 1 is assigned to the most important objective, rank N is assigned to the least important one, and the ranks of the other elements are assigned inversely proportional to their importance. If a number of elements have the same importance, then the average rank is used for all of them. Assume that the control system designer can provide such a ranking for the control systems. These ranks can be used to assign a weight to each objective in the multi-criterion optimization problem. There are several ways to assign the weights based on the ranks, we use the following two formulas [SSE81]:

$$w_i = \frac{\frac{1}{q_i}}{\sum_{j=1}^{N} \frac{1}{q_j}} \qquad w_i = \frac{(N - q_i + 1)}{\sum_{j=1}^{N} (N - q_j + 1)}$$

where q_i and w_i denote the rank and the weight of the i-th element respectively. The weights obtained by these formulas are called *rank reciprocal weights* and *rank sum weights* respectively.

Once the weights have been chosen, we define the *optimal performance scheduler synthesis problem* as choosing rates $r_i \in E_i$ such that the system is schedulable and the weighted sum $w_i\gamma(r_i)$ is minimized.
Formally, we require

$$\begin{array}{lll} \text{minimize} & \sum_{i=1}^{N} w_i\gamma(r_i) & \\ \text{such that} & r_i \in E_i & \text{for each } i \in \{1, \ldots, N\} \\ & \sum_{i=1}^{N} c_i r_i / h_i \leq 1 & \end{array}$$

It is straightforward to show that the multiple-choice knapsack problem [SZ79, KPP04] can be reduced to the above optimization problem. This establishes that the decision version of the optimal-performance scheduler synthesis problem is NP-hard. (In the decision version, we ask if there exists $r_i \in E_i$ such that the

system is schedulable with dropout and $\sum_{i=1}^{N} w_i\gamma(r_i) \leq K$ for an input parameter K.)

Theorem 10. *The optimal performance scheduler synthesis decision problem is NP-hard.*

For the upper bound on the complexity of the problem, we need to be careful because in general the optimization problem (7.15) can only be approximated to a given accuracy ϵ. However, given an ϵ, the LMI can be solved in time polynomial in the size of the input matrices and $\log \frac{1}{\epsilon}$. Thus, the approximate version of the optimal performance scheduler synthesis decision problem, where we ask if $|\sum_{i=1}^{N} w_i\gamma(r_i) - K| \leq \epsilon$ for input parameters K and ϵ can be solved in NP.

7.3 Scheduler Design

As the optimal-performance scheduler synthesis problem is computationally hard, we instead heuristically solve the following simplified version. First, for each system, we find out the minimum successful transmission rates $r_{\min,i}$ using Theorem 5. Second, we find out an upper bound $r_{\max,i}$ on the successful transmission rates for all the control systems, such that the system $\{(C_i, r_{\max,i}) \mid i \in \{1, \ldots, N\}\}$ is schedulable with dropout. If for some i we have $r_{\max,i} < r_{\min,i}$, clearly the system is unschedulable. Otherwise, for each control system individually, we choose the best eligible successful transmission rate $r_{best,i}$ in the range $[r_{\min,i}, r_{\max,i}]$ (i.e., for which $\gamma(r)$ is minimized for $r \in [r_{\min,i}, r_{\max,i}]$). Note that $r_{best,i}$ is either $r_{\min,i}$ or $r_{\max,i}$. We then synthesize a static scheduler, ensuring the chosen rates of successful transmission rates for all the control systems. While the algorithm is not guaranteed to produce an optimal schedule, it heuristically attempts to jointly maximize the performance of all the systems.

We present our overall algorithm in two steps. In Section 7.3.1, we assume that we are given a successful transmission rate for each controller and present a

set of constraints that must be satisfied by any static scheduler that schedules the system with dropout. In Section 7.3.2, we show how these set of constraints can be modified to find an upper bound on the successful transmission rates in such a way that the system is schedulable with dropout when these rates are chosen. Section 7.3.3 summarizes the algorithm.

7.3.1 Synthesis through Constraint Solving

To synthesize a static scheduler, we consider a duration T, called the *basic cycle*, for which we synthesize a schedule. The schedule for any duration $T' > T$ is obtained by repeating the synthesized schedule. Let us assume that each rate r_i is a fraction $\frac{k_i}{K_i}$, for integers k_i and K_i. We also assume, by scaling, that the sampling times h_i of the control systems take integer values. The duration of the basic cycle T is chosen such that the cycle of duration T can accommodate an integer number of tasks after considering dropouts. We set $T = lcm(K_1, K_2, \ldots, K_N) \times lcm(h_1, h_2, \ldots, h_N)$, where lcm stands for the least common multiple.

Let $m_i = \frac{T}{h_i}$ denote the number of task instances of C_i that are generated in the duration T. We introduce m_i variables s_{i1}, s_{i2}, $\ldots s_{im_i}$ for $i \in \{1, \ldots, N\}$. Each variable s_{ij} takes values in $\{0, 1\}$. If the variable $s_{ij} = 1$, this denotes that the jth instance of task C_i (running in the time interval $[(j-1)h_i, jh_i)$ was scheduled and if the variable $s_{ij} = 0$, this denotes that the jth instance of task C_i was dropped by the scheduler. Additionally, we introduce m_i variables t_{i1}, t_{i2}, $\ldots t_{im_i}$ for $i \in \{1, \ldots, N\}$. If $s_{ij} = 1$, i.e., the instance of C_i in the slot $[(j-1)h, jh_i)$ is scheduled, then t_{ij} denotes the time in the interval $[(j-1)h_i, jh_i)$ when the execution of the task begins. Otherwise, if $s_{ij} = 0$, then t_{ij} is set to -1.

Below we present the constraints on the scheduling problem.

1. For each controller i, for each of the m_i tasks in the basic cycle, the instance

of the task C_i is either scheduled or dropped:

$$\bigwedge_{i\in\{1,\cdots,N\}} \bigwedge_{j\in\{1,\cdots,m_i\}} (s_{ij} = 1) \ \vee \ (s_{ij} = 0). \tag{7.28}$$

2. For each controller i, the number of instances of C_i that are scheduled in the basic cycle is equal to $m_i \times r_i$ (which is an integer, by choice of T):

$$\bigwedge_{i\in\{1,\cdots,N\}} \sum_{j=1}^{m_i} s_{ij} = m_i \times r_i. \tag{7.29}$$

3. For all tasks C_i, if the instance of C_i is scheduled in a slot, the start time of the task should be scheduled after the beginning of the slot and the end time of the task should be before the end of the slot.

$$\bigwedge_{i\in\{1,\cdots,N\}} \bigwedge_{j\in\{1,\cdots,m_i\}} (s_{ij} = 1) \implies$$
$$(t_{ij} \geq (j-1) \times h_i) \wedge (t_{ij} + c_{ij} \leq j \times h_i). \tag{7.30}$$

If the instance of the task is dropped, the start time of the task is set to -1.

$$\bigwedge_{i\in\{1,\cdots,N\}} \bigwedge_{j\in\{1,\cdots,m_i\}} (s_{ij} = 0) \implies (t_{ij} = -1) \tag{7.31}$$

4. The time slot assigned for two tasks should not intersect:

$$\bigwedge_{\substack{i,k\in\{1,\cdots,N\}\\ i\neq k}} \bigwedge_{j\in\{1,\cdots,m_i\}} \bigwedge_{l\in\{1,\cdots,m_k\}} (t_{ij} \geq 0) \wedge (t_{kl} \geq 0)$$
$$\implies (t_{ij} + c_i \leq t_{kl}) \vee (t_{kl} + c_k \leq t_{ij}) \tag{7.32}$$

If the constraints are not satisfiable, there does not exist a scheduler for the given successful transmission rates. However, if the constraints are satisfiable we obtain a valid schedule. Further, the schedule obtained by repeating the static schedule every T units of time shows that the system $\{(C_i, r_i) \mid i \in \{1, \ldots, N\}\}$ is schedulable with dropout.

7.3.2 Maximal Scheduler Synthesis

We now present how to find the maximum successful transmission rates for all the controllers that preserve schedulability. To solve this problem, we introduce variables x_i that denote the number of instances of task C_i that are scheduled in one basic cycle, and formulate a vector maximization problem [MA04] where the objective vector $\mathbf{v}$ is given by $\mathbf{v}_i = x_i$. (The variables x_i are proportional to the rates.)

As in Section 7.2, we simplify the multi-objective optimization problem to a single-objective optimization problem by choosing weights and taking the weighted sum of the vector. We assume the control designer additionally provides a priority for each control system. The weight of a control system is derived from its priority using the weight finding formulas introduced in Section 7.2. The single objective function is $\sum_{i=1}^{N} w_i x_i$. We formulate this optimization problem as a constraint solving problem, and find the optimal values for x_i by solving a series of feasibility problems.

First, we modify the set of constraints presented in Section 7.3.1 in the following way. The set of constraints in (7.29) are replaced by the following set of constraints:

$$\bigwedge_{i \in \{1,\cdots,N\}} \sum_{j=1}^{m_i} s_{ij} \geq x_i \tag{7.33}$$

Moreover, we add the following constraint to the set of constraints:

$$\sum_{i=1}^{N} w_i \times x_i > \lambda, \tag{7.34}$$

where λ is a constant. If the constraints are infeasible then λ is certainly an upper bound for the objective function. We iteratively update λ and solve the set of constraints till we find the maximum λ for which the constraints are satisfiable. The satisfying assignment of those constraints gives the maximal rates x_i/T for which the system is schedulable with dropout.

7.3.3 Overall Algorithm

In this section we present the overall algorithm to synthesize a static schedule with dropout. The inputs of the algorithm are the systems C_i with their sampling times h_i, worst case execution times c_i, and priority $\pi_i \in \{1, \ldots, N\}$. The output of the algorithm is a static schedule, if possible. The algorithm has the following steps.

In the first step, using Theorem 5, we find out the minimum successful transmission rates for all the controllers to achieve exponential stability. In the second step, we find out the maximum successful transmission rates by solving the optimization problem described in Section 7.3.2. If the minimum rate is greater than the maximum rate for some system, we stop and say unschedulable.

Now, for each system C_i, we have a range $[r_{\min,i}, r_{\max,i}]$ for the successful transmission rates, and any choice of r_i in this range ensures that the system is schedulable with dropout. For each controller, the optimal successful transmission rate is $r_{\min,i}$ if $\gamma(r_{\min,i}) \leq \gamma(r_{\max,i})$, otherwise, $r_{\max,i}$ is the optimal successful transmission rate for the controller.

Now, by solving the constraints in Section 7.3.1 for the optimal successful transmission rates for each controller, we find a static schedule.

7.4 Evaluation

7.4.1 Implementation

Figure 7.3 shows the toolbus that we have developed to synthesize static schedulers automatically for linear time invariant control systems with dropout. The inputs to our tool are (1) the mathematical description of the plants, (2) the linear controllers, (3) the sampling times, (4) the computation times for the control tasks, and (5) the ranking or priority of the control systems. We assume that

the feedback controllers and the sampling periods have been designed using standard control theoretic methods [AW90b], and the execution times of the control tasks have been calculated using standard techniques [WEE08]. We use a Matlab script to compute the lower bound on the successful transmission rates following Theorem 5. We automatically compute the upper bounds on the rates using Yices [DM06] from the sampling time, computation time, and the ranking of the control systems. Note that while the lower bounds on the rates are independent from each other, the upper bounds need to satisfy the scheduling constraints. It may be the case that the lower bound of the successful transmission rate is bigger than the upper bound for a control system. In this case, we cannot generate a schedule. The feasibility of a schedule is checked using a Matlab script. If there exists a feasible schedule, we find out the rates for which the performance of the control system becomes optimal. For each controller, we we find out the upper bound on the $\mathcal{L}_\infty$ to RMS gain for the minimum and maximum successful transmission rates by solving optimization problem (7.15) using `CVX` [GB11] and choose optimal rate. The optimal rates are then used to form the constraints that are solved using Yices to get the optimal schedule.

To find out the upper bounds of the successful transmission rates, we solve a vector optimization problem using the weights calculated from the ranking of the control systems. The individual objective functions are the number of successfully transmitted packets (denoted by n_i) in a basic cycle with duration T, where $n_i = m_i \times r_i$, and m_i is the number of generated packets for the i-th controller in a basic cycle and r_i is the successful transmission rate of controller i. We solve this optimization problem by solving a number of feasibility problems and using the bisection method [BV04]. Let f denote the value of the objective function. As we want to find the maximal of successful transmission rates maintaining schedulability, we need to maximize the scalarized objective function. We add the constraint $f > C$ in the set of constraints, where C is a constant. We start

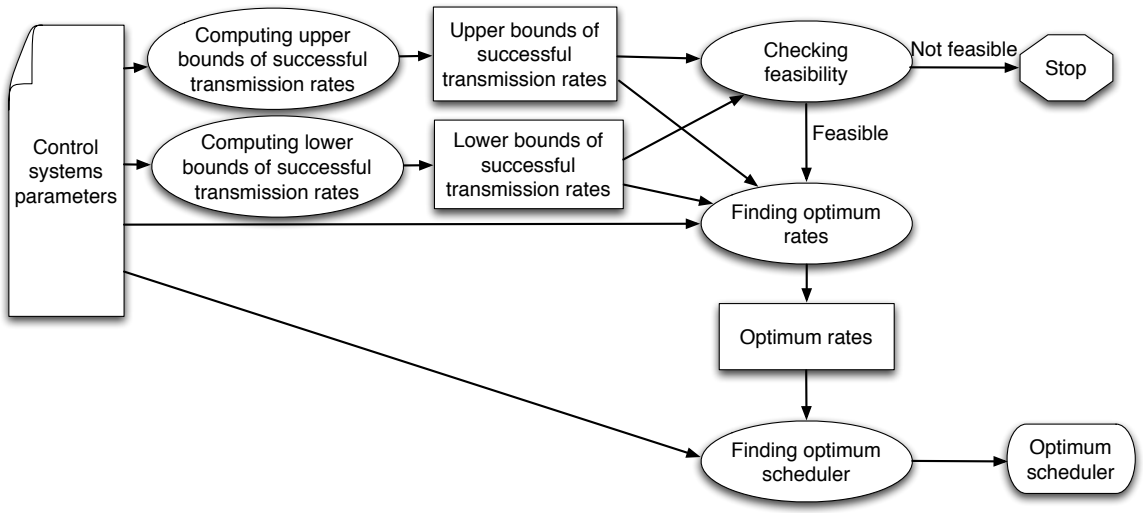


Figure 7.3: Scheduler synthesis toolbus

with $C = 1$. As the sum of the weights is 1, and the values of n_i's are greater than equal to 1, the constraints are satisfiable. We then iteratively find out the maximum value C such that the set of constraints are satisfiable. This is done as follows. At each step, if the constraints are satisfiable, then to choose the next C we multiply the current C by 2. If in any step, the constraints are unsatisfiable, then we choose the next C to be the average of the current value of C and the value of C for which the constraints were satisfiable for the last time. We stop in a step when the constraints are satisfiable and the difference between the previous and the current values of C is below a certain threshold (in our implementation, we choose 0.5). The values of $r_i (= \frac{n_i}{m_i})$'s in the last step provide the upper bounds on the successful transmission rates.

In the last step of our algorithm we need to find out the schedule corresponding to the chosen values of the successful transmission rates for different control systems. The constraints (7.29) in Subsection 7.3.1 are the corresponding set of constraints. As these constraints are equality constraints, they limit the feasible space of the constraints, and it is hard for the SMT solver to find a feasible so-

Systems	Mass (kg)	Length(m)	Priority	Controller gain	Sampling time (s)	Computation time (s)
System 1	0.50	0.50	1	[5.4925 -0.3667]	0.015	0.005
System 2	0.50	0.60	1	[5.2434 -0.2488]	0.015	0.005
System 3	0.50	0.50	1	[5.7505 -0.2024]	0.020	0.005
System 4	0.50	0.60	1	[5.4447 -0.0739]	0.020	0.005

Table 7.1: Control systems parameters

lution. To alleviate this problem, for a control system, we choose a few different rates instead of just one rate, for which the performance of the control system is reasonably good. Now we replace a constraint set (7.29) in Subsection 7.3.1 by the disjunction of the constraints corresponding to different rates. This increases the feasible search space, and Yices can find out a schedule relatively easily.

7.4.2 Experiments

We illustrate our technique by synthesizing a scheduler for four inverted pendulums. The model of such pendulum is given in (7.27). We assume that all pendulums have mass $m = 0.5$, and rotational friction coefficient $\rho = 0.6$. The pendulums differ from each other in their lengths, chosen as $[l_1, l_2, l_3, l_4] = [0.50, 0.60, 0.50, 0.60]$, their sampling times, chosen as $[h_1, h_2, h_3, h_4] = [0.015s, 0.015s, 0.020s, 0.020s]$, and their controllers, designed as $K_1 = [5.4925\ -0.3667]$, $K_2 = [5.2434\ -0.2488]$, $K_3 = [5.7505\ -0.2024]$ and $K_4 = [5.4447\ -0.0739]$. We assume that the computation time for all the controllers is the same and equal to $0.005s$. All constants and variables are expressed in SI units. The parameters of the control systems are summarized in Table 7.1.

Using Theorem 5, we obtain $r_{\min,1} = 0.6588$, $r_{\min,2} = 0.7312$, $r_{\min,3} = 0.6234$, and $r_{\min,4} = 0.6838$, which guarantee that by choosing successful transmission rates bigger than these rates, the inverted pendulums, with no disturbances, are exponentially stable. The obtained maximum successful transmission rates are

Systems	r_{min}	r_{max}	Optimal upper bound of the $\mathcal{L}_\infty$ to RMS gain	Transmission rate used in static scheduler synthesis
System 1	0.6588	0.70	0.0057	0.70
System 2	0.7312	0.80	0.0055	0.80
System 3	0.6234	0.90	0.0055	0.90
System 4	0.6838	0.90	0.0054	0.90

Table 7.2: Experimental results

$r_{\max,1} = 0.70$, $r_{\max,2} = 0.80$, $r_{\max,3} = 0.90$ and $r_{\max,4} = 0.90$, when the ranks of the control systems are chosen to be equal, and we use *rank reciprocal weights*. Since $r_{\min,i} < r_{\max,i}$ for $i = 1, 2, 3, 4$, feasible ranges of the successful transmission rates exist for all the pendulums. For all the control systems $\gamma(r_{max}) < \gamma(r_{min})$. Thus the rates for which we find schedules are 0.70, 0.80, 0.90 and 0.90 respectively for controllers $i = 1, 2, 3, 4$. The optimal rates have been chosen as multiples of 0.05. We synthesize the schedule for these rates which can be roughly the schedule corresponding to the optimal performance. The experimental results are summarized in Table 7.2. Our tool takes 5m7.610s to compute the maximum successful transmission rates for the controllers, and 1m27.986s to synthesize the scheduler.

7.5 Related Work

The co-design problem of feedback controllers and schedulers has been studied in the past [SLS96, RHS97, ACE00, RS00, ZSW08], focusing on the choice of sampling times such that the system is schedulable and each system attains optimal performance. An extensive review of this field can be found in [XS06]. Cervin introduced the idea of feedback scheduling [Cer03] to perform online schedule de-

sign to cope with varying or unknown workloads. Zhang et al. [ZSW08] presented theoretical results on scheduling of a number of single-input single-output (SISOs) control systems to achieve balances among robustness, schedulability, and power consumption. However, the global optimization problems that arise in these works can be hard to solve efficiently. Branicky et al. [BPZ02] proposed a technique to co-design feedback controllers and schedulers for networked control systems [ZBP01] with packet dropouts. They studied the effect of packet dropout on the stability of control systems and applied the rate monotonic scheduling algorithm [Liu00] to schedule controllers in such a way that the controllers maintain stability when the rates of packet dropouts are bounded above. However, they did not consider the tradeoff of performance and packet dropout in their work. In this work we show that it is possible to achieve better performance than just achieving stability by suitably choosing the rate of packet dropout for a controller. The rates of packet dropout are chosen in such a way that all the control tasks are schedulable and individual control systems achieve optimal performance. Our objective is to synthesize a scheduler statically that will provide a schedule following which all the control systems will achieve the best possible performance maintaining fairness.

We automatically synthesize a scheduler considering dropout of packets. We formulate the scheduling problem as a constraint solving problem, and use the SMT solver Yices [DM06] to synthesize a scheduler automatically. Very recently, the merits of using SMT solver for scheduler synthesis have been championed in [Ste08] and [LM10]. Steiner [Ste08] presents an evaluation of scheduler synthesis with Yices for time-triggered multi-hop networks. Legriel and Maler [LM10] present a framework to solve the task graph scheduling problem on a multiprocessor, while achieving the cheapest configuration. Our results fall in the general class of *program synthesis*, and we exploit domain knowledge (e.g., control theoretic performance requirements) to synthesize task schedulers.

CHAPTER 8

Dynamic Scheduling

In this chapter, we describe a methodology and a tool to automatically synthesize optimal controllers for multiple control loops in an integrated architecture in the presence of communication and computation constraints. All the controllers execute on the same CPU and the controllers receive the state of their corresponding plants from the sensors through a network. The network introduces a delay in the transmission of the state. Moreover, it can drop some packets due to interference or corruption. We assume that there is an upper bound on the rate of packet drop by the network, but there is no deterministic mechanism of modeling the drop of individual packets. Our goal is to design controllers for each plant which ensure global asymptotic stability, but in addition, optimally reject disturbances (as measured by the $\mathcal{L}_\infty$ to RMS gain), in spite of network packet losses and shared computational resources.

Unfortunately, when co-designing controllers and schedulers for optimal disturbance rejection, it is known from Chapter 7 that the performance of a controller does not increase monotonically with the rate of successful transmission. Thus, an optimal controller may decide not to compute the control signal even if computational resources are available. In the absence of network losses, we suggested in Chapter 7 a methodology that picks a Pareto-optimal operating point for the controllers and designs a static scheduler that ensures that each controller computes the control signal exactly at the transmission rate for optimal performance. In the presence of network losses, the scheme of Chapter 7 is not applicable, since

the scheduler may need to dynamically adjust the computation of control signals based on the history of packet losses.

In this chapter, our objective is to synthesize controllers for the individual control systems and a dynamic scheduler that can ensure stability and optimal disturbance rejection for all the control systems. We solve this problem using the following three steps.

Scedulability Analysis. We provide a feasibility criterion for a given set of control tasks to be schedulable on a single processor in the presence of packet drops by the network and task drops by the scheduler. For each controller, we compute the maximum successful transmission rate r_{max} (the maximum rate of computing the control task) so that the schedulability can be ensured using earliest deadline first strategy (Section 8.2).

Optimal Controller Synthesis. We give an algorithm to search for optimal controllers in the presence of network losses and computation constraints. Recall that the performance of a controller is not monotonic with the successful transmission rate. In the presence of network losses, the scheduler can only guarantee that the successful transmission rate of a controller is within an operating range (in the interval $[r - r_{net}, r]$ for a rate r chosen by design). The performance of the controller in the operating range can potentially vary significantly. Our goal is to design a controller and an operating range such that the average performance in the operating range is the best possible one over the space of possible stabilizing controllers and their schedulable successful transmission rates $[r_{min}, r_{max}]$. To solve the controller synthesis problem, we solve optimization problems at two levels. First, given a controller, we find a lower bound on its performance for different successful transmission rates. As shown in Chapter 7 the problem of finding the lower bound on the performance for a fixed rate is a convex optimization problem. Second, we find a controller for which the performance is optimal on average in an operating range. This is also a minimization problem, but unfortunately

not convex. To solve this non-convex optimization problem, we use PSO, a local search based stochastic optimization method [KE95, JLY07] (Section 8.3).

Dynamic Scheduler Implementation. We present a dynamic scheduling scheme that ensures the successful transmission rates in the chosen operating rages for the individual controllers, and hence, the performance, of each controller is maintained at the optimal value on average in the presence of network losses (Section 8.4).

We have implemented our controller design methodology in Matlab. We assume that the communication uses the CAN protocol and the processor is shared. We show how an optimal controller can be synthesized under scheduling constraints on the example of five inverted pendulums implemented on a shared processor.

8.1 Networked Control Systems

We assume the following architecture for the system. The state of the plant is sensed at a regular interval and sent to the controller through a network (see Figure 8.1). We assume that the transmission of the plant's state through the network and the computation of the control signal takes less than one sampling period. We divide a sampling period into two sub-periods. At the end of the first sub-period, the state of the plant is available for control computation. In the second sub-period, the control signal is computed and the control signal is directly applied to the plant precisely at the end of the second sub-period (at the end of a period).

Due to network failure some packets from the sensor may be dropped and may never reach the controller. The controller itself may also decide not to compute the control signal in some rounds. The goal of the controller is to maintain a successful transmission rate of the control signal over a period of time so that the control system is exponentially stable and also has the desired performance in

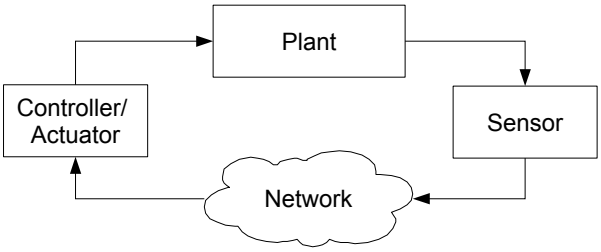


Figure 8.1: Linear control system with dropout

terms of disturbance rejection.

8.1.1 Finding the Operating Rate

We now consider the problem of choosing the successful transmission rate for a given controller to achieve the best performance. In Chapter 7, we have seen that the optimal performance is achieved at a successful transmission rate r_{opt} which is equal to either the minimum successful transmission rate r_{min} or the maximum successful transmission rate r_{max}. Ideally, we should choose the successful transmission rate r_{opt} for which the bound on the $\mathcal{L}_\infty$ to RMS gain attains the minimum value. As the network may also drop packets, it is not possible for the controller to drop the packets according to a static scheduling scheme so that the successful transmission rate is maintained at r_{opt}. Rather, the controller must resort to a dynamic scheduling scheme and maintain the successful transmission rate in a range so that optimal performance is achieved on average.

Let $\gamma_m(r)$ is the mean of the $\mathcal{L}_\infty$ to RMS gains for $r' \in [r - r_{net}, r]$ and the range $[r - r_{net}, r]$ is discretized according to a discretization factor. Our objective is to find the successful transmission rate r so that $\gamma_m(r)$ is minimized among all r in the range $[r_{min} + r_{net}, r_{max}]$. The following theorem shows that we have two choices for such successful transmission rate.

Theorem 11. *The operating successful transmission rate is either* r_{max} *or* $r_{min} + r_{net}$.

Proof. If $r > r_{max}$, then scheduling constraints will be violated. If $r < r_{min} + r_{net}$, then due to packet drop in the network, the successful transmission rate may be less than r_{min}, and the system may be unstable. If we choose any other rate r, $r_{min} + r_{net} < r < r_{max}$, we can use the reasoning of Theorem 9 and show that by shifting the operating rate either towards r_{max} or towards r_{min}, it is possible to decrease the average value of the $\mathcal{L}_\infty$ to RMS gain in the operating region. ∎

8.1.2 Motivating Example

As a motivating example, we use the model of the inverted pendulum in (7.27) in Chapter 7.

Let us consider an instance of the pendulum system where $\rho = 0.6$, $m = 0.4$ and $l = 0.6$. All values are in S.I. units. We discretize the plant in (7.27) with sampling period of $20ms$. A stabilizing controller for the discretized plant is given by $K_1 = [4.8462\ 0.1800]$. The minimum successful transmission rate for this system to ensure stability is $r_{\min} = 0.6623$.

Now we plot how the upper bound on the $\mathcal{L}_\infty$ to RMS gain varies with the successful transmission rate between $r_{\min}$ and $r_{\max} = 1$. The figure is obtained by quantizing the successful transmission rates between $r_{\min}$ and $r_{\max}$ with a quantization factor 0.01, and then solving the optimization problem in (7.15) for each choice of the successful transmission rate. For a given controller, we refer to the plot of transmission rate vs. performance as the *performance profile* of the controller. The curve in Figure 8.2(a) shows the performance profile for the controller K_1.

Now we come to the problem of choosing the operating successful transmission rate. Due to scheduling constraints, the value of $r_{\max}$ may be less than 1. The value of r_{opr} depends on the value of $r_{\max}$. In our present example, if $r_{\max} \leq 0.87$ then $\gamma_m(r_{\min} + r_{\text{net}}) \leq \gamma_m(r_{\max})$. Thus, the choice of operating successful

transmission rate is $r_{\min} + r_{\text{net}}$, as in that case choosing $r_{\min} + r_{\text{net}}$ would give the optimal performance and optimal CPU time usage. If $r_{\max} > 0.87$ then $\gamma_m(r_{\max}) < \gamma_m(r_{\min}+r_{\text{net}})$. If $r_{\max}$ is permitted by the scheduling constraints to be greater than 0.87, $r_{\max}$ can be chosen as the operating successful transmission rate to get better performance. This example illustrates that the scheduling constraints have effect on the choice of the operating successful transmission rate.

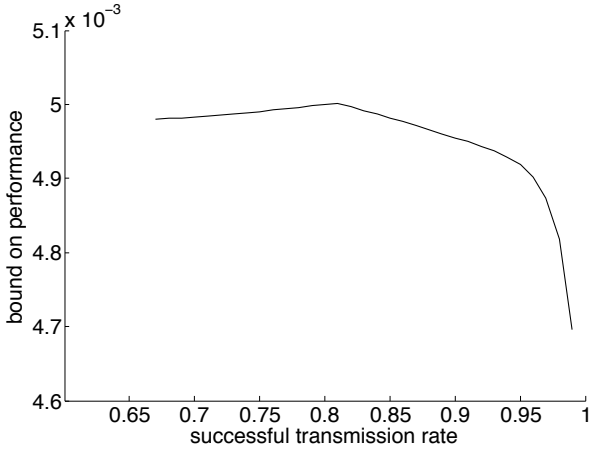
Now let us consider another controller $K_2 = [5.8125\ 0.1249]$. The performance profile for this controller is shown in Figure 8.2(b). The performance profiles of the two controllers K_1 and K_2 illustrate that the performance of the two controllers may be quite different and thus two controllers may have different disturbance rejection capabilities. Thus we have the following controller synthesis question: Given a plant, how to synthesize a stabilizing controller that optimizes the performance under the scheduling constraints?

Thus, in the context of implementation of multiple controllers on a single processor, we consider the following controller-scheduler co-design problem: Given a set of plants we would like to synthesize controllers for them and find the operating points so that the control tasks for all the systems are schedulable and the control systems attain the Pareto optimal performance in terms of the $\mathcal{L}_\infty$ to RMS gain.

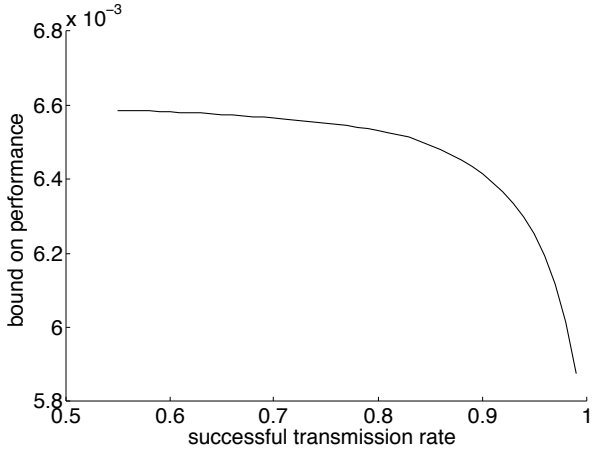
8.2 Schedulability Analysis in the Presence of Packet Dropout

In Section 8.1, we considered network packet losses. Now, we additionally introduce the effect of multiple control loops sharing the same CPU and the network. Since the CPU is shared, we have to schedule the control tasks and ensure that each control task achieves an optimal successful transmission rate in the presence of scheduling constraints and network losses.

Let n denote the number of control systems. Let h_i denote the sampling period



(a) K_1



(b) K_2

Figure 8.2: The upper bound of the $\mathcal{L}_\infty$ to RMS gain vs successful transmission rate for an inverted pendulum for (a) $K_1 = [4.8462\ 0.1800]$ and (b) $K_2 = [5.41250.1489]$

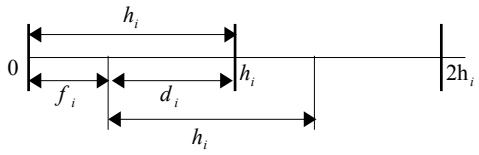


Figure 8.3: Periodic state transmission and control computation

of the i-th control system. The state of the plant of the i-th control system is sampled at the instants $0, h_i, 2h_i, \ldots$, as illustrated in Figure 8.3. The period h_i is divided into two sub-periods f_i and d_i. In the first f_i time duration, the state of the plant is transmitted to the controller. At the end of the first sub-period f_i, the state of the plant is available for the computation of the control signal. During the second sub-period d_i, the control signal is computed and the control signal is applied to the plant at the end of the sampling period. Note that the control computation task arrives periodically with a period h_i, and the deadline of the control computation is d_i. Let c_i denote the worst case computation time for the control signal of the i-th control system.

The control signal may not be applied to the plant in every cycle. The controller may not be able to generate the control signal due to the loss of the packet from the sensor in the network. The controller itself may decide not to compute the control signal to achieve the optimal performance. We denote by r_i the successful transmission rate of the control signal to the plant for the i-th control system.

8.2.1 Computing Message Transmission Times

To find out the value of f_i for the i-th control system, we find out the worst case delivery time of the message from the sensor to the controller. The computation of worst case message delivery time depends on the nature of the protocol used in the transmission of message. We do not address the general problem in this

paper, but use known results about the worst case message delivery time for the CAN protocol [THW95, DBB07]. In our experiments, we use CAN protocol to transmit a message from a sensor to a controller, and use the recurrence relation in [DBB07] to compute the worst case message delivery time. Note that the f_is for different control systems may be different.

8.2.2 Schedulability Analysis of Control Computations

The control computation tasks for the i-th control system arrives at the time instants $f_i, f_i + h_i, f_i + 2h_i, \ldots$. Given h_i, f_i, d_i, c_i and r_i for each control system, we first find out if it is feasible to schedule the control computations on a single processor using the earliest deadline first scheduling strategy. Theorem 12 addresses this feasibility question.

Theorem 12. *The earliest deadline first scheduling algorithm is feasible for the control computations, if for all $t_1 < t_2$, we have*

$$\sum_{i=1}^{n} \eta_i(t_1, t_2) r_i c_i \leq (t_2 - t_1)$$

where

$$\eta_i(t_1, t_2) = \max\{0, \lfloor \frac{t_2 - f_i - d_i}{h_i} \rfloor - \max\{0, \lceil \frac{t_1 - f_i}{h_i} \rceil\} + 1\}$$

Proof. The proof follows from Lemma 3.4 and Lemma 3.5 in [BR90]. ∎

As shown by Baruah and Rosier [BR90], if the f_i's are different for different i, the feasibility problem is coNP-hard in a strong sense, implying that the problem even does not have a pseudo-polynomial solution. However, in the synchronous case, when the start time of the initial computation cycle for all the systems are the same, the feasibility test can by performed in pseudo-polynomial time. To make the start time of the initial computation cycles of all the control systems synchronous, we choose the start time for the computation for all the control

systems to be

$$F = \max_i \ \ f_i$$

The following theorem says that under suitable condition, the schedulability problem for a set of synchronous control systems can be solved in pseudo-polynomial time.

Theorem 13. *Let κ be a fixed constant, $0 < \kappa < 1$, such that $\sum_{i=1}^{n} r_i \frac{c_i}{h_i} \leq \kappa$. Then the problem of testing if all the computation tasks initially activated at the same time are schedulable can be solved in $O(n\ max\{h_i - d_i\})$ time.*

Proof. The proof is similar to the proof of Theorem 3.1 in [BR90]. It can be shown that if all the computation tasks arrives synchronously at the beginning, then the value of t_1 and t_2 in checking the feasibility condition in Theorem 12 can be chosen as 0 and $\frac{\kappa}{1-\kappa} max\{h_i - d_i\}$ respectively. Given that κ, h_i and d_i are constant, the t_2 takes a constant value. As we need to check the feasibility condition in Theorem 12 for each control system, the feasibility problem can be solved in $O(n\ max\{h_i - d_i\})$ time. ∎

The algorithm to check feasibility of scheduling the computation for different control systems by an earliest deadline first scheme is as follows. The algorithm takes as input vectors h, d, c, and r for the control systems. For any time t, we denote by $load(t)$ the amount of processing time required by the already scheduled computations for the control systems. At any time t, $load(t)$ is given by

$$load(t) = \sum_{i=1}^{n} (\lfloor \frac{t - d_i}{h_i} \rfloor + 1) r_i c_i.$$

From Theorems 12 and 13, the system is EDF schedulable iff $load(t)$ is less than or equal to t for each t in the time interval $[0, \frac{\kappa}{1-\kappa} \max\{h_i - d_i\}]$, where κ is chosen as in the proof of Theorem 13. We call this algorithm `testEDFFeasibility`.

8.2.3 Computation of Maximum Successful Transmission Rates

Here we show how to find the upper bound on the rate of successful transmission so that the control tasks are schedulable under the earliest deadline first scheduling scheme. The algorithm to find the maximum successful transmission rates for all the control systems is shown in Algorithm 8.2.1. The algorithm takes as input the vectors h, d and c representing the period, deadline and computation time of the control tasks for all the control systems, respectively. The symbol r denotes the vector representing the successful transmission rates of the control systems. Initially, the components of r are set to 1. The algorithm runs in a loop. At each step, it calls Algorithm `testEDFFeasibility` to check if the control tasks with the rate vector r is schedulable. If yes, vector r is returned as the vector containing the maximum successful transmission rates. Otherwise, the components of r are decremented by using a vector ϵ. We can assign priorities to the control systems and the components of ϵ can be selected in such a way that the control system with a higher priority gets a higher maximum successful transmission rate, and thus a possibility of having a better performance.

8.3 Optimal Controller Synthesis

In this section we present our optimal performance controller synthesis scheme. The objective is to synthesize a controller that achieves the optimal performance in terms of the $\mathcal{L}_\infty$ to RMS gain in the presence of packet dropout in the network and any constraint imposed on the maximum successful transmission rate by the implementation platform. The synthesis algorithm involves solving a minimization problem, which is of non-convex nature. We use *particle swarm optimization* (PSO) [KE95, JLY07], a stochastic local search approach, to solve our problem. PSO iteratively solves an optimization problem by maintaining a population (or *swarm*) of candidate controllers, called *particles*, and moving them around in the

Algorithm 8.2.1: Computation of Maximum Successful Transmission Rates

```
1  function findMaximumRates(h, d, c)
2  begin
3      r := [1 1 ... 1]
4      while r > 0 do
5          result := testEDFFeasibility(h, d, c, r)
6          if result = feasible then
7              return r
8          end
9          r := r − ϵ
10     end
11     return "not feasible"
12 end
```

search-space of possible controllers, trying to minimize the objective function. In each iteration, a particle is assigned a new position and a new velocity that determine its position in the next iteration.

8.3.1 Cost Function

We define a cost function for a controller K based on which we search for an optimal controller. The cost function is based on Theorem 11. The cost function is given by

$$Cost(K) = \begin{cases} \min(\gamma_m(r_{\max}), \gamma_m(r_{\min} + r_{\text{net}})), & \texttt{if } 0 < r_{\min} < 1 \text{ and} \\ & r_{\max} - r_{\min} > r_{\text{net}} \\ \infty & , \text{ otherwise} \end{cases} \tag{8.1}$$

The successful transmission rate r for which the cost function achieves the minimum value is the *operating successful transmission rate* (denoted by r_{opr}) for the controller K. The range $[r_{\text{opr}} - r_{\text{net}}, r_{\text{opt}}]$ is called the *operating successful*

transmission range.

8.3.2 Overall algorithm

The PSO algorithm is used to search for a controller $K \in \mathbb{R}^{m\times n}$, minimizing (8.1). A particle in PSO represents a pole of the closed loop system. The value of each component of the pole is selected from a range $(-1, 1)$ to make sure that the controller K obtained for the pole is a stabilizing controller [AW90b].

The design steps can be summarized as the following:

(1) Uniformly randomly initialize positions and velocities of N closed loop poles p_i, $i = 1, \ldots, N$, where N is the number of particles.

(2) Given any initial pole p_i, find out the corresponding controller K_i [AW90b]. For the controller K_i, compute the cost using the cost function $Cost(K_i)$.

(3) Compare $Cost(K_i)$ to its own best position P_i so far and the global best position. If $Cost(K_i)$ is less than the previous best (resp. the global best), update the best position (resp. the global best) to p_i.

(4) Modify the velocity and position of each p_i according to the rules given in [JLY07].

(5) If the number of iterations reaches the maximum, denoted by $l_{\max}$, then go to Step (6), otherwise go to Step (2).

(6) The pole for which the cost function of the corresponding controller attains the minimal value is the optimal controller.

The algorithm can also be terminated if there is no change in the value of the cost function for a significant number of steps.

8.4 Scheduler Synthesis

In this section, we show the scheduling strategy for the control tasks on a shared processor. The scheduler has to make sure that the rate of control computations eventually reaches the r_{opr} and stays there if there is no packet dropout by the network. If the rate of packet drop by the network is bounded by r_{net}, then the rate of control computation is guaranteed to be in the range $[r_{\text{opr}} - r_{\text{net}}, r_{\text{opr}}]$. Algorithm 8.4.1 presents the pseudo code for the scheduler. In the algorithm, $r(i)$ and $m(i)$ denote the current successful transmission rate and the number of periods occurred so far for the i-th control system. The scheduler runs in an infinite loop. When the scheduler receives the state of a plant for the control computation, the scheduler first checks if scheduling the task would make the rate of scheduled task go above the operating rate r_{opr}. If not, then the control computation is scheduled based on earlier deadline first strategy. As the operating rates for all control systems are below the maximum successful transmission rates, the schedulability of the control tasks is guaranteed.

Theorem 14. *In the steady state, the rate at which the control computation is scheduled for a control system is guaranteed to be in the range* $[r_{\text{opr}} - r_{\text{net}}, r_{\text{opr}}]$.

Proof. Note that the scheduler schedules the control computation only if the successful transmission rate cannot go above r_{opr} after scheduling the computation. Thus, we have to show that when the system reaches the steady state, the value of successful transmission rate cannot go below $r_{\text{opr}} - r_{\text{net}}$. We consider the following two cases.

Case 1: $r_{\text{opr}} \leq 1 - r_{\text{net}}$. In this case, irrespective of the packet dropout by the network, the successful transmission rate eventually reaches r_{opr} as the scheduler always schedules the computation when $r < r_{\text{opr}}$. Once this steady state is reached, due to packet dropout by the network, the successful transmission rate may decrease, but as the rate of packet drop by the network is bounded by r_{net},

Algorithm 8.4.1: Scheduler

```
1  function scheduleControlComputation(r_opr)
2  begin
3      for i = 1...n do
4          r(i) = 0, m(i) = 0
5      end
6      time := 0
7      while true do
8          for i = 1...n do
9              if (time − f_i)%h_i = 0 then
10                 success_new_rate := (r(i)*m(i)+1)/(m(i)+1)
11                 failure_new_rate := (r(i)*m(i))/(m(i)+1)
12                 if The state of the plant is received then
13                     if success_new_rate ≤ r_opr(i) then
14                         Schedule based on EDF strategy
15                         r(i) := success_new_rate
16                     else
17                         // Scheduler drops the computation
18                         r(i) := failure_new_rate
19                     end
20                 else
21                     // Packet drop by network
22                     r(i) := failure_new_rate
23                 end
24                 m(i) := m(i) + 1
25             end
26         end
27         time := time + 1
28     end
29 end
```

the successful transmission rate cannot go below $r_{\text{opr}} - r_{\text{net}}$.

Case 2: $r_{\text{opr}} > 1 - r_{\text{net}}$. Here we consider two extreme cases. In the presence of maximum packet drop by the network, the successful transmission rate can reach $1 - r_{\text{net}}$ in the steady state and remains there. The successful transmission rate $1 - r_{\text{net}}$ is in the range $[r_{\text{opr}} - r_{\text{net}}, r_{\text{opr}}]$. If there is no packet drop by the network before the successful transmission rate reaches the steady state, then the value of the successful transmission rate can reach r_{opr} in the steady state. Once the successful transmission rate is in this state, in the presence of maximum packet dropout by the network in the future, the successful transmission rate cannot go below $r_{\text{opr}} - r_{\text{net}}$. ∎

8.5 Evaluation

8.5.1 Implementation

We implemented our synthesis tool on top of Matlab. We use PSOt [Bir03], a PSO toolbox for Matlab, to solve the synthesis problem. In our experiments, we set the number of the particles in PSO to be $N = 24$ and the maximum number of iterations $l_{\max} = 50$. The synthesis process terminates if there is no change in the value of the objective function for consecutive 25 iterations, or if the number of iterations reaches $l_{\max}$. To solve the convex optimization problem to find the upper bound on the $\mathcal{L}_\infty$ to RMS induced gain for a specific successful transmission rate, we use YALMIP modeling language [Lof04] and SDPT3 semidefinite program solver [TTT03].

We consider that the plant state is transmitted by the sensor using CAN protocol in a single precision floating point format. The plant has two states. Thus for 2 states the data packet from the sensor to the controller contains 8 bytes. We assume that the speed of the CAN bus is $250KBPS$, and the bound on

the drop of packets by the network is 0.05. As shown in [DBB07], the transmission time of a CAN message m with 11-bit identifier and s_m data bytes is given by $(55 + 10s_m)\tau_{bit}$, where τ_{bit} is the time required to transmit 1 bit through the CAN network. With $250KBPS$ speed of the CAN bus, $\tau_{bit} = 0.004ms$. Thus, the transmission of a message from the sensor to the controller requires $0.54ms$.

In our experiments we have used the Truetime simulator [CHL03] to implement the control tasks and simulate the systems under different conditions.

The choice of CAN protocol in our experiments is motivated by the fact that CAN is a widely used protocol in the domain of networked control systems and Truetime supports simulation using CAN protocol. The CAN protocol has recently been proposed to be implemented over wireless network connections [BEC05, Ozc08]. Correctly designed CAN protocol is quite reliable, but the use of wireless network for relaying CAN frames makes it prone to packet dropout. Though we use CAN protocol for our experiments, our technique is applicable to any network protocol for which it is possible to guarantee a bound on the rate of packet dropout.

8.5.2 Experiments

We illustrate our results on synthesizing multiple control systems on a single platform on the example of five inverted pendulums sharing the communication network and a processor. The model of such pendulums is presented in Section 7.1.5. We assume that all pendulums have mass $m = 0.5$, and rotational friction coefficient $\rho = 0.6$. The pendulums differ from each other in their lengths, chosen as $[l_1, l_2, l_3, l_4, l_5]$ =[0.50, 0.50, 0.60, 0.50, 0.60], and their sampling times, chosen as $h = [h_1, h_2, h_3, h_4, h_5] = [15ms, 20ms, 20ms, 25ms, 25ms]$. We assume that the computation time for all the controllers is the same and equal to $5ms$. All constants and variables are expressed in SI units. The parameters of the control

Systems	Mass (kg)	Length (m)	Sampling time (s)	Computation time (s)	r_{max}
System 1	0.50	0.50	0.015	0.005	0.80
System 2	0.50	0.50	0.020	0.005	0.80
System 3	0.50	0.60	0.020	0.005	0.80
System 4	0.50	0.50	0.025	0.005	0.80
System 5	0.50	0.60	0.025	0.005	0.80

Table 8.1: Control systems parameters

systems are summarized in Table 8.1.

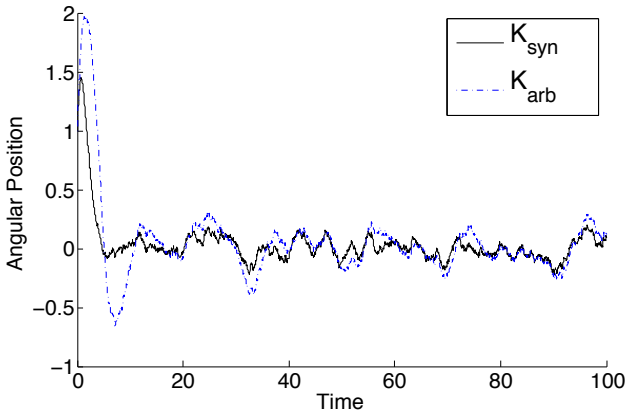
We use the algorithm provided in [DBB07] to compute the worst case message delivery time for the individual control systems. The priorities for the CAN messages are assigned based on the duration of the periods. If the period for a control system is longer, it gets lower priority. The worst case message delivery time for the control systems are given by $[f_1,\ f_2,\ f_3,\ f_4,\ f_5] = [0.54ms,\ 1.08ms,\ 1.62ms,\ 2.16ms,\ 2.70ms]$. To keep the start time of the initial computation cycles for all the control system to be the same, we choose $F = 2.70ms$. With this value of F, we compute the values of the deadlines for the control computation to be $[d_1,\ d_2,\ d_3,\ d_4,\ d_5] = [12.30ms,\ 17.30ms,\ 17.30ms,\ 22.30ms,\ 22.30ms]$. We now use Algorithm 8.2.1 to find out the maximum possible successful transmission rate for all the control systems to satisfy EDF schedulability condition. We use the same weight for all the systems, and the maximum possible successful transmission rate is 0.80 for all the systems. Table 8.2 shows the synthesized controllers for the three pendulums, their minimum successful transmission rate r_{min}, operating successful transmission rate r_{opr}, the value of the cost function, and time required to synthesize the controllers. All experiments were carried out on a notebook running Mac OS 10.7.4 with 2 GHz Intel Core i7 processor and 8 GB 1600 MHz DDR3 memory.

Systems	Controller	r_{min}	r_{opr}	Cost	Synthesis Time (s)
System 1	[5.3284 -0.3643]	0.7478	0.8000	0.0052	2677
System 2	[5.1875 -0.4825]	0.7491	0.8000	0.0047	2244
System 3	[5.1715 -0.2639]	0.7470	0.8000	0.0048	2202
System 4	[5.5158 -0.2726]	0.6596	0.8000	0.0038	2545
System 5	[5.1740 -0.2355]	0.7480	0.8000	0.0045	2417

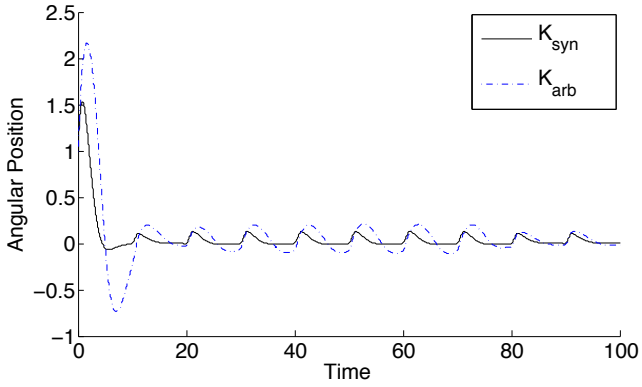
Table 8.2: Synthesized controllers

To judge the quality of the synthesized controllers, we compare the controller synthesized for System 5 with another arbitrarily chosen stabilizing controller. We call the synthesized controller $K_{syn} = [5.1740 - 0.2355]$ and the arbitrarily chosen controller $K_{arb} = [5.0050 \ -0.5772]$. For both the controllers we set the operating successful transmission rate to 0.80. Figure 8.4 shows the evolution of the state angular velocity with time under the action of the two controllers. Figure 8.4(a) shows the evolution of the state when the plant is subjected to a band-limited white noise with noise power 0.1 and sample time 0.01. Figure 8.4(b) shows the evolution of the state when the plant is subjected to a disturbance signal of pulse shape with amplitude 1 unit, period $10s$, pulse width $1s$, and zero phase delay. The figures show that K_{syn} clearly outperforms K_{arb} significantly in terms of the capability of dealing with disturbance.

We also measure the control cost for the two controllers to make sure that the synthesized controller is not consuming too much power. The control cost is measured as the sum of the amplitude of the control signal at the end of each sampling period for a time duration of 100s. For the band-limited white noise disturbance, the control cost for K_{syn} and K_{art} are 582.00 units and 1756.00 units respectively. For the pulse shaped disturbance signal, they are 531.00 units and 1562.00 units respectively. Thus the synthesized controller also uses significantly



(a) band limited white noise



(b) disturbance signal of pulse shape

Figure 8.4: Evolution of angular position with time for the synthesized controller and an arbitrarily chosen controller from initial state $\langle 1, 1 \rangle$ for (a) a band-limited white noise and (b) a disturbance signal of pulse shape

less control power.

8.6 Related Work

There are a number of papers that address the problem of maintaining stability in the presence of network induced delay (e.g. [SQ03, ZSC05]), or packet loss (e.g. [ZBP01, XL07, WC07, LH11]) or simultaneously delay and loss (e.g. [YWC04b, YWC04a, GB07]). However, the problem of designing controllers that achieve op-

timal performance in terms of disturbance rejection in the presence of network induced delay and packet loss was not studied in the past.

CHAPTER 9

Conclusion and Future Work

In this thesis, we have attempted to resolve the issues that arise during the implementation of control systems, and are not taken into account during designing the controllers mathematically using control theory. More specifically, we address the following three problems: First, we show how the stability property can be verified for a physical system under the action of controller software and how to synthesize controller software to minimize the effect of quantization error on the stability quality. Second, we show that the naive implementation of some control algorithms may be infeasible due the computation time required for the control tasks on real platforms and provide a memoization based implementation scheme that guarantees the feasibility of the implementation along with maintaining expected control performance. Third, we address the problem of scheduling control tasks from multiple control systems on a single processor and provide static and dynamic scheduler synthesis strategies to maintain stability and achieve optimal performance in the control systems. So far, control theory, software engineering and real-time system theory have been studied as three independent fields. In this thesis, we show how we can make a confluence of these three fields towards building reliable cyber-physical systems.

Software code for controllers provides a sweet spot for static checkers: while the application domain is often safety-critical —making verification desirable— the code itself usually has statically unrollable loops and statically allocated memory, removing language features that are the bane of most static analyzers. Moreover,

unlike generic software, software for control systems often comes with mathematical models and specifications. This allows a co-ordinated analysis by pushing some of the complexity of software analysis to the model level. Our results show how domain knowledge from control theory on the one hand and program verification and synthesis techniques on the other can interact to provide a solution to reliability problems in cyber-physical systems.

Controller synthesis problems have been widely reduced to convex optimization problems for which efficient algorithms are available. However, when we want to take into account different implementation issues in the controller design phase, we need to solve complex multi-objective optimization problems which are often not amenable to the reduction to convex optimization problems. In this thesis we have showed that combination of convex optimization and stochastic optimization techniques has great potential to solve complex controller synthesis problems.

9.1 Looking Ahead

We end this thesis with an outline of some possible interesting future work:

Bridging the gap between control theory and controller implementation. In this thesis, we have focused on two major implementation issues: (a) the use of finite precision arithmetic and (b) communication delay and packet dropout in the network. There are mainly three performance criteria for controller synthesis: (i) the steady state behavior of the control system, (ii) the control cost and the state cost (LQR cost) and (iii) the capability of disturbance rejection. In my research so far, I have shown how the implementation error due to the use of finite precision arithmetic affects the steady state behavior of the control system, and how packet dropout may be utilized to enhance the disturbance rejection capability of the controller. However, the other combinations of the implementation issues and the performance criteria are also very interesting. For example, how

does the use of finite precision arithmetic affect the LQR/LQG performance and the capability of disturbance rejection? How does the packet dropout affect the steady state behavior and the LQR/LQG performance? In my future research, my aim would be to close these gaps between control theory and the implementation of the control systems by answering questions of the above form. The end-goal is to develop a framework that would be capable of synthesizing controllers that take all implementation constraints into account, and that are Pareto optimal with respect to all the performance criteria.

Dealing with complex cyber-physical systems. In this thesis we have concentrated on classical linear and nonlinear control systems. The next step of the research would be to extend our results for complex control systems, for example, switching control systems. The stability of a switching control system is given by an analysis on dwell time and average dwell time [Lib03]. It would be interesting to study how the implementation error in the controller effect the dwell time parameters of the control systems. Another example may be analyzing robotic controllers for implementation error. In a robot manipulator, the tracking error depends on the quality of the implementation. Currently there is no available formal technique to analyze the effect of implementation error on the tracking quality. The results in this thesis motivates research to deal with such problems.

Study of processor architectures for cyber-physical systems. This thesis discusses issues related to the implementation of multiple control systems on top of a single processor. Recently multicore processors have been introduced in aerospace and automotive domains to enhance the performance of the implemented systems. Though scheduling issues for multi-core processors have received a lot of research attention, the implementation of multiple control systems on a multicore processor has received very little research interest so far. It would be interesting to study how we can synthesize controllers that will have optimal performance while running on multicore processors in the presence of other tasks of

different criticality levels. It would also be interesting to study if we can come up with fundamentally new processor architectures that would be more suitable for building cyber-physical systems than the existing processor architectures. Moreover, it would be worth investigating how suitable FPGAs are for the implementation of cyber-physical systems on top of them.

Abstract interpretation for nonlinear arithmetic. Abstract interpretation based technology has been very efficient in estimating error bound of the fixed-point implementation of a linear controller program, and thus has been very useful in controller synthesis based on the minimization of the effect of the implementation error on the performance of the controller. However, the technique does not work very well for the nonlinear controllers. Our effort on estimating the error bound for a jet engine controller [KK95b] reveals that the abstract interpretation based error estimation technique may end up providing very pessimistic bound. This benchmark motivates research towards enhancing the capabilities of abstract interpretation based tools to deal with nonlinear arithmetic.

References

[AB09] A. Alessio and A. Bemporad. "A Survey on Explicit Model Predictive Control." In *Nonlinear Model Predictive Control*, volume 384 of *LNCIS*, pp. 345–369. Springer, 2009.

[AC00] T. Aamodt and P. Chow. "Embedded ISA Support for Enhanced Floating-Point to Fixed-Point ANSI C Compilation." In *CASES*, 2000.

[ACE00] K.-E. Arzen, A. Cervin, J. Eker, and L. Sha. "An introduction to control and scheduling co-design." In *Proceedings of CDC*, 2000.

[AFP09] F. Alegre, E. Feron, and S. Pande. "Using Ellipsoidal Domains to Analyze Control Systems Software." *CoRR*, **abs/0909.1977**, 2009.

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

[AMS10] A. Anta, R. Majumdar, I. Saha, and P. Tabuada. "Automatic verification of control system implementations." In *Proc. EMSOFT*, pp. 9–18, 2010.

[Arz99] Karl-Erik Årzén. "A Simple Event-Based PID Controller." In *Proc. IFAC*, volume 18, pp. 423–428, 1999.

[ASP10] João Almeida, Carlos Silvestre, and António M. Pascoal. "Self-Triggered State Feedback Control of Linear Plants under Bounded Disturbances." In *Proc. CDC*, pp. 7588–7593, 2010.

[AT10] Adolfo Anta and Paulo Tabuada. "To Sample or not to Sample: Self-Triggered Control for Nonlinear Systems." *IEEE Trans. Automatic Control*, **55**(9), 2010.

[AW90a] K. J. Åström and B. Wittenmark. *Computer-controlled systems: theory and design*. Prentice-Hall, Inc., 2nd edition, 1990.

[AW90b] K. J. Åström and B. Wittenmark. *Computer-controlled systems: theory and design*. Prentice-Hall, Inc., 2nd edition, 1990.

[BCC03] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. "A Static Analyzer for Large Safety-Critical Software." In *PLDI*, pp. 196–207. ACM, 2003.

[BEC05] C. Bayilmis, I. Erturk, and C. Ceken. "Extending CAN segments with IEEE 802.11 WLAN." In *The 3rd ACS/IEEE International Conference on Computer Systems and Applications*, pp. 79–86, 2005.

[BGP09] O. Bouissou, E. Goubault, S. Putot, K. Tekkal, and F. Védrine. "HybridFluctuat: A Static Analyzer of Numerical Programs within a Continuous Environment." In *CAV*, LNCS 5643, pp. 620–626. Springer, 2009.

[Bir03] B. Birge. "PSOt - a particle swarm optimization toolbox for use with Matlab." In *Swarm Intelligence Symposium, 2003. SIS '03. Proceedings of the 2003 IEEE*, pp. 182–186, 2003.

[BKW09] A. Brillout, D. Kroening, and T. Wahl. "Mixed abstractions for floating-point arithmetic." In *FMCAD*, pp. 69–76. IEEE, 2009.

[BMD02] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N. Pistikopoulos. "The explicit linear quadratic regulator for constrained systems." *Automatica*, **38**(1):3–20, 2002.

[BPZ02] M. S. Branicky, S. M. Phillips, and W. Zhang. "Scheduling and Feedback Co-Design for Networked Control Systems." In *Proceedings of CDC*, pp. 1211–1217, 2002.

[BR90] Sanjoy K. Baruah and Louis E. Rosier. "Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic Real-Time Tasks on One Processor." *Real-Time Systems*, **2**:301–324, 1990.

[BR05] P. Belanovic and M. Rupp. "Automated Floating-point to Fixed-point Conversion with the Fixify Environment." In *Proc. Rapid System Prototyping*, 2005.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[Can88] J. Canny. "Some Algebraic and Geometric Computations in PSPACE." In *STOC*, pp. 460–467. ACM, 1988.

[Cer03] A. Cervin. *Integrated Control and Real-Time Scheduling*. PhD thesis, Lund University, 2003.

[CF95a] T. Chen and B. A. Francis. *Optimal sampled-data control systems*. Springer-Verlag, New York, 1995.

[CF95b] T. Chen and B.A. Francis. *Optimal Sampled-Data Control Systems*. Springer-Verlag, 1995.

[CHL03] Anton Cervin, Dan Henriksson, Bo Lincoln, Johan Eker, and Karl-Erik Årzén. "How Does Control Timing Affect Performance? Analysis and Simulation of Timing Using Jitterbug and TrueTime." *IEEE Control Systems Magazine*, **23**(3):16–30, 2003.

[CMU] "Control Tutorial for Matlab and Simulink." Available online at http://www.library.cmu.edu/ctms/ctms/.

[Cou05] P. Cousot. "Integrating Physical Systems in the Static Analysis of Embedded Control Software." In *APLAS*, pp. 135–138, 2005.

[DBB07] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. "Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised." *Real-Time Systems*, **35**(3):239–272, 2007.

[DGP09] D. Delmas, E. Goubault, S. Putot, J. Souyris, K. Tekkal, and F. Védrine. "Towards an Industrial Use of FLUCTUAT on Safety-Critical Avionics Software." In *FMICS*, LNCS 5825, pp. 53–69. Springer, 2009.

[DM06] B. Dutertre and L. de Moura. "A Fast Linear-Arithmetic Solver for DPLL(T)." In *CAV*, LNCS 4144, pp. 81–94. Springer, 2006.

[ECJ] Sean Luke. "The ECJ Owners Manual." Available online at http://www.cs.gmu.edu/ eclab/projects/ecj/docs/manual/manual.pdf.

[EKG12] S. Ebbesen, P/ Kiwitz, and L. Guzzella. "A generic particle swarm optimization function for Matlab." *American Control Conference (to appear)*, June 2012.

[FA08a] E. Feron and F. Alegre. "Control software analysis, Part I Open-loop properties." *CoRR*, **abs/0809.4812**, 2008.

[FA08b] E. Feron and F. Alegre. "Control software analysis, Part II Closed-loop analysis." *CoRR*, **abs/0812.1986**, 2008.

[Fer04] J. Feret. "Static Analysis of Digital Filters." In *ESOP*, LNCS 2986, pp. 33–48. Springer, 2004.

[FHR07] M. Fränzle, C. Herde, S. Ratschan, T. Schubert, and T. Teige. "Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure." *J. SAT*, **1**:209–236, 2007.

[FRC03] Claire F. Fang, Rob A. Rutenbar, and Tsuhan Chen. "Fast, Accurate Static Analysis for Fixed-Point Finite-Precision Effects in DSP Designs." In *ICCAD*, 2003.

[FSI09] G. Fainekos, S. Sankaranarayanan, F. Ivančić, and A. Gupta. "Robustness of model based simulation." In *RTSS*, pp. 345–354. IEEE, 2009.

[GB07] Matas Garca-Rivera and Antonio Barreiro. "Analysis of networked control systems with drops and variable delays." *Automatica*, **43**(12):2054 – 2059, 2007.

[GB11] M. Grant and S. Boyd. "CVX: Matlab Software for Disciplined Convex Programming, version 1.21." http://cvxr.com/cvx, Jan 2011.

[GL94] M. Green and D. J. N. Limebeer. *Linear robust control*. Prentice Hall, August 1994.

[GPB07] E. Goubault, S. Putot, P. Baufreton, and J. Gassino. "Static Analysis of the Accuracy in Control Systems: Principles and Experiments." In *FMICS*, LNCS 4916, pp. 3–20. Springer, 2007.

[HBH99] A. Hassibi, S. P. Boyd, and J. P. How. "Control of asynchronous dynamical systems with rate constraints on events." In *Proceedings of CDC*, volume 2, pp. 1345 –1351, 1999.

[Hes09] J. P. Hespanha. *Linear systems theory*. Princeton University Press, September 2009.

[HF09] Reinhold Heckmann and Christian Ferdinand. "Worst-Case Execution Time Prediction by Static Program Analysis." White paper, AbsInt Angewandte Informatik GmbH, 2009.

[HSV08] W. P. M. H. Heemels, J. H. Sandee, and P. P. J. Van Den Bosch. "Analysis of Event-Driven Controllers for Linear Systems." *Intl. J. of Control*, **81**(4):571–590, 2008.

[IM12] Arnault Ioualalen and Matthieu Martel. "A New Abstract Domain for the Representation of Mathematically Equivalent Expressions." In *SAS*, 2012.

[Jha11] S.K. Jha. *Towards Automated System Synthesis Using SCIDUCTION*. PhD thesis, University of California at Berkeley, 2011.

[JLY07] M. Jiang, Y. P. Luo, and S. Y. Yang. "Stochastic convergence analysis and parameter selection of the standard particle swarm optimization algorithm." *Information Processing Letters*, **102**(1):8–16, April 2007.

[KA02] B. Kisacanin and G. C. Agarwal. *Linear Control Systems*. Kluwer Academic/Plenum Publishers, 2002.

[Kai80] T. Kailath. *Linear systems*. Prentice-Hall, Inc., 1980.

[KE95] J. Kennedy and R. Eberhart. "Particle swarm optimization." *In Proceedings of IEEE International Conference on Neural Networks*, pp. 1942–1948, 1995.

[Kha02] Hassan K. Khalil. *Nonlinear Systems*. Prentice Hall, 2002.

[KK95a] M. Krstic and P.V. Kokotovic. "Lean backstepping design for a jet engine compressor model." *IEEE Conf. Contr. App.*, pp. 1047–1052, 1995.

[KK95b] M. Krstic and P.V. Kokotovic. "Lean backstepping design for a jet engine compressor model." In *Proceedings of IEEE Conf. Control App.*, pp. 1047–1052, 1995.

[Kos09] O. Kosheleva. "Babylonian Method of Computing The Square Root: Justifications Based on Fuzzy Techniques and on Computational Complexity." In *Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS)*, pp. 1–6, 2009.

[KP08] Gal Katz and Doron Peled. "Model Checking-Based Genetic Programming with an Application to Mutual Exclusion." In *TACAS*, pp. 141–156, 2008.

[KPP04] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems.* Springer Verlag, 2004.

[KW98] Ming-Yang Kao and Jie Wang. "Efficient minimization of numerical summation errors." In *Automata, Languages and Programming.* Springer, 1998.

[LAS09] H. Liu, A. Abraham, and V. Snasel. "Convergence analysis of swarm algorithm." *World congress on Nature and Biologically Inspired Computing*, pp. 1714–1719, December 2009.

[LCH07] Michael Lemmon, Thidapat Chantem, Xiaobo Sharon Hu, and Matthew Zyskowski. "On Self-Triggered Full Information H-infinity Controllers." In *Proc. HSCC*, pp. 371–384, 2007.

[LCN07a] J. A. López, C. Carreras, and O. Nieto-Taladriz. "Improved Interval-Based Characterization of Fixed-Point LTI Systems with Feedback Loops." *IEEE Trans. on CAD of Integrated Circuits and Systems*, **26**(11):1923–1932, 2007.

[LCN07b] J. A. López, C. Carreras, and O. Nieto-Taladriz. "Improved Interval-Based Characterization of Fixed-Point LTI Systems with Feedback Loops." *IEEE Trans. on CAD of Integrated Circuits and Systems*, **26**, 2007.

[Leo] "LEON2 Processor." http://vlsicad.eecs.umich.edu/BK/Slots/cache/www.gaisler.com/products/leon2/leon.html.

[LGC06a] D. Lee, A. A. Gaffar, R. C. C. Cheung, O. Mencer, W. Luk, and G. A. Constantinides. "Accuracy-Guaranteed Bit-Width Optimization." *IEEE Trans. on CAD of Integrated Circuits and Systems*, **25**(10), 2006.

[LGC06b] D. Lee, A. A. Gaffar, R. C. C. Cheung, O. Mencer, W. Luk, and G. A. Constantinides. "Accuracy-Guaranteed Bitwidth Optimization." *IEEE Trans. on CAD of Integrated Circuits and Systems*, **25**(10):1990–2000, 2006.

[LH11] Michael Lemmon and Xiaobo Sharon Hu. "Almost sure stability of networked control systems under exponentially bounded bursts of dropouts." In *Proceedings of HSCC*, pp. 301–310, 2011.

[Lib03] Daniel Liberzon. *SWITCHING IN SYSTEMS AND CONTROL*. Birkhauser, 2003.

[Liu00] J. W. S. Liu. *Real-Time Systems*. Prentice-Hall, Inc., 2000.

[LL61] J. LaSalle and S. Lefschetz. *Stability by Lyapunov's Direct Method*. Academic Press, Inc., 1961.

[LL73] C. L. Liu and J. W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment." *Journal of ACM*, **20**(1), 1973.

[LM10] J. Legriel and O. Maler. "Meeting Deadlines Cheaply." Technical Report TR-2010-1, Verimag Research Report, 2010.

[Lof04] J. Lofberg. "YALMIP : a toolbox for modeling and optimization in MATLAB." In *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*, pp. 284–289, 2004.

[LP] "lp_solve, a Mixed Integer Linear Programming (MILP) solver." Available online at http://lpsolve.sourceforge.net/.

[LSG92] K. Liu, R. E. Skelton, and K. Grigoriadis. "Optimal controllers for finite wordlength implementation." *IEEE Transactions on Automatic Control*, **37**(9):1294–1304, September 1992.

[MA04] R. T. Marler and J. S. Arora. "Survey of multi-objective optimization methods for engineering." *Structural and Multidisciplinary Optimization*, **26**(6):369–395, 2004.

[Mar09] Matthieu Martel. "Enhancing the implementation of mathematical formulas for fixed-point and floating-point arithmetics." *Formal Methods in System Design*, **35**(3), 2009.

[MAT09] Manuel Mazo Jr., Adolfo Anta, and Paulo Tabuada. "On Self-Triggered Control for Linear Systems: Guarantees and Complexity." In *Proc. ECC*, 2009.

[Moo66] R. Moore. *Interval Analysis*. Prentice Hall, 1966.

[MPS76] P. McLane, L. Peppard, and K. Sundareswaran. "Decentralized feedback controls for the brakeless operation of multilocomotive powered trains." *IEEE Trans. Autom. Control*, **21**(3):358–363, 1976.

[MPS11] Trent McConaghy, Pieter Palmers, Michiel Steyaert, and Georges G. E. Gielen. "Trustworthy Genetic Programming-Based Synthesis of Analog Circuit Topologies Using Hierarchical Domain-Specific Building Blocks." *IEEE Trans. Evolutionary Computation*, **15**(4):557–570, 2011.

[MSB07] A. Mallik, D. Sinha, P. Banerjee, and H. Zhou. "Low-Power Optimization by Smart Bit-Width Allocation in a SystemC-Based ASIC Design Environment." *IEEE Trans. on CAD of Integrated Circuits and Systems*, **26**(3), 2007.

[MT08] Manuel Mazo Jr. and Paulo Tabuada. "On Event-Triggered and Self-Triggered Control Over Sensor/Actuator Networks." In *Proc. CDC*, pp. 435–440, 2008.

[MT09] Manuel Mazo Jr. and Paulo Tabuada. "Input-to-state Stability of Self-Triggered Control Systems." In *Proc. CDC*, pp. 928–933, 2009.

[OCC07a] W. G. Osborne, R. C. C. Cheung, J. G. F. Coutinho, W. Luk, and O. Mencer. "Automatic Accuracy-Guaranteed Bit-Width Optimization for Fixed and Floating-Point Systems." In *Proc. FPL*, pp. 617–620, 2007.

[OCC07b] W. G. Osborne, R. C. C. Cheung, J. G. F. Coutinho, W. Luk, and O. Mencer. "Automatic Accuracy-Guaranteed Bit-Width Optimization for Fixed and Floating-Point Systems." In *Proc. FPL*, pp. 617–620, 2007.

[OSH09] Roman Obermaisser, Christian El Salloum, Bernhard Huber, and Hermann Kopetz. "From a Federated to an Integrated Architecture." *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, **28**(7):956–965, 2009.

[Ozc08] Ibrahim Ozcelik. "Interconnection of CAN segments through IEEE 802.16 Wireless MAN." *J. Netw. Comput. Appl.*, **31**(4):879–890, 2008.

[PLM08] Riccardo Poli, William B. Langdon, and Nicholas F. McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises, 2008.

[Pow] "PowerPC 5xx Controllers." http://www.freescale.com/webapp/sps/site/taxonomy.jsp?code=DRMCRMPC500MC.

[PPS04] S. Prajna, A. Papachristodoulou, P. Seiler, and P. A. Parrilo. "SOSTOOLS: Control applications and new developments." *in Proceedings of IEEE International Symposium on Computer Aided Control Systems Design*, pp. 315–320, 2004.

[PW07] Andreas Podelski and Silke Wagner. "Region Stability Proofs for Hybrid Systems." In *Proc. FORMATS*, pp. 320–335, 2007.

[RHS97] M. Ryu, S. Hong, and M. Saksena. "Streamlining real-time controller design: From performance specifications to end-to-end timing constraints." In *Proceedings of RTAS*, pp. 91–99, 1997.

[Ros74] H. H. Rosenbrock. *Computer-Aided Control System Design*. Academic Press, 1974.

[RS00] H. Rehbinder and M. Sanfridson. "Integration of off-line scheduling and optimal control." In *Proceedings of ECRTS*, pp. 137–143, 2000.

[San06] J.H. Sandee. *Event-driven control in theory and practice: Tradeoffs in software and control performance*. PhD thesis, Technische Universeteit Endhoven, 2006.

[SF97] J. Stolfi and L. H. Figueiredo. "Self-Validated Numerical Methods and Applications." In *Monograph for 21st Brazilian Mathematics Colloquium, Rio de Janeiro: IMPA*, 1997.

[SLS96] D. Seto, J.P. Lehoczky, L. Sha, and K.G. Shin. "On task schedulability in real-time control systems." In *Proceedings of RTSS*, pp. 13–21, 1996.

[SQ03] Hu Shousong and Zhu Qixin. "Stochastic optimal control and analysis of stability of networked control systems with long delay." *Automatica*, **39**(11):1877–1884, 2003.

[SSE81] W. G. Stillwell, D. A. Seaver, and W. Edwards. "A Comparison of Weight Approximation Techniques in Multiple Utility Decision Making." *Organizational Behavior and Human Performance*, **28**:62–77, 1981.

[Ste08] W. Steiner. "An Evaluation of SMT-based Schedule Synthesis for Time-Triggered Multi-Hop Networks." In *Proceedings of RTSS*, 2008.

[SZ79] Prabhakant Sinha and Andris A. Zoltners. "The Multiple-Choice Knapsack Problem." *Operations Research*, **27**(3):503–515, 1979.

[Tar51] A. Tarski. *A Decision Method for Elementary Algebra and Geometry.* University of California Press, 1951.

[THW95] K. W. Tindell, H. Hansson, and A. J. Wellings. "Calculating Controller Area Network (CAN) message response time." *Control Engineering Parctice*, **3**(8):1163–1169, 1995.

[TTT03] R.H Tutuncu, K.C. Toh, and M.J. Todd. "Solving semidefinite-quadratic-linear programs using SDPT3." *Mathematical Programming Ser. B*, **95**:189–217, 2003.

[VMF03] Manel Velasco, Pau Martí, and Josep M. Fuertes. "The Self-Triggered Task Model for Real-Time Control Systems." In *Work In Progress Proceedings of RTSS*, pp. 67–70, 2003.

[WC07] Jing Wu and Tongwen Chen. "Design of Networked Control Systems With Packet Dropouts." *Automatic Control, IEEE Transactions on*, **52**(7):1314–1319, 2007.

[WEE08] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. "The worst-case execution-time problem – overview of methods and survey of tools." *ACM Trans. Embed. Comput. Syst.*, **7**:36:1–36:53, 2008.

[Wil85] D. Williamson. "Finite wordlength design of digital Kalman filters for state estimation." *IEEE Transactions on Automatic Control*, **30**(10):930–939, October 1985.

[Wil89] D. Williamson. "Optimal finite wordlength linear quadratic regulation." *IEEE Transactions on Automatic Control*, **34**(12):1218–1228, December 1989.

[Win93] G. Winskel. *The formal semantics of programming languages: an introduction.* MIT Press, 1993.

[WL09] Xiaofeng Wang and M.D. Lemmon. "Self-triggered Feedback Control Systems with Finite Gain $\mathcal{L}_2$ Stability." *IEEE Transaction on Automatic Control*, **54**(3):452–467, 2009.

[XL07] Junlin Xiong and James Lam. "Stabilization of linear systems over networks with bounded packet loss." *Automatica*, **43**(1):80–87, 2007.

[XS06] F. Xia and Y. Sun. "Control-scheduling co-design: A perspective on integrating control and computing." *Dynamics of Continuous, Discrete and Impulsive Systems - Series B: Applications and Algorithms, Special Issue on ICSCA'06*, pp. 1352–1358, 2006.

[YH95] K. P. Yoon and C. Hwang. *Multiple attribute decision making: an introduction*. Sage Publications, Inc., 1995.

[YWC04a] Mei Yu, Long Wang, Tianguang Chu, and Fei Hao. "An LMI approach to networked control systems with data packet dropout and transmission delays." In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 4, pp. 3545 – 3550, 2004.

[YWC04b] Mei Yu, Long Wang, Tianguang Chu, and Guangming Xie. "Stabilization of networked control systems with data packet dropout and network delays via switching system approach." In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 4, pp. 3539–3544, 2004.

[Zak03] S. H. Zak. *Systems and Control*. Oxford University Press, 2003.

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

[ZKS07] M. Zamani, M. Karimi-Ghartemani, and N. Sadati. "FOPID controller design for robust performance using particle swarm optimization." *Journal of Fractional Calculus & Applied Analysis (FCAA)*, **10**(2):169–188, 2007.

[ZKS09] M. Zamani, M. Karimi-Ghartemani, N. Sadati, and M. Parniani. "Design of a fractional order PID controller for an AVR using particle swarm optimization." *Control Engineering Practice*, **17**(12):1380–1387, December 2009.

[ZSC05] L. Zhang, Y. Shi, T. Chen, and B. Huang. "A New Method for Stabilization of Networked Control Systems With Random Delays." *Automatic Control, IEEE Transactions on*, **50**(8):1177 – 1181, 2005.

[ZSK09] M. Zamani, N. Sadati, and M. Karimi-Ghartemani. "Design of an H_∞ PID controller using particle swarm optimization." *International Journal of Control, Automation, and Systems (IJCAS)*, **7**(2):273–280, April 2009.

[ZSW08] F. Zhang, K. Szwaykowska, W. Wolf, and V. J. Mooney III. "Task Scheduling for Control Oriented Requirements for Cyber-Physical Systems." In *Proceedings of RTSS*, pp. 47– 56, 2008.