University of California
Santa Barbara

Control under energy and time constraints

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Electrical and Computer Engineering

by

Justin Payne Pearson

Committee in charge:

Professor João P. Hespanha, Chair
Professor Andrew R. Teel
Professor Jason R. Marden
Professor Amr El Abbadi

March 2018

The Dissertation of Justin Payne Pearson is approved.

Professor Andrew R. Teel

Professor Jason R. Marden

Professor Amr El Abbadi

Professor João P. Hespanha, Committee Chair

January 2018

Control under energy and time constraints

by

Justin Payne Pearson

To Jen.

Acknowledgements

I'm very thankful to have met so many nice people during my graduate school career at UC Santa Barbara.

João Hespanha

João, thank you for encouraging me to come back to grad school. You made all this possible. I deeply appreciate your time, attention, and guidance the last several years. Your uncanny intuition about tough problems constantly amazes your students. Thanks to you, I was able to explore several professional and personal intellectual interests, like the Certificate in College and University Teaching. These endeavors really enriched the quality of my life. It was a real pleasure being your student.

Andy Teel

Andy, thanks for all the advice and the miscellaneous chats about day-to-day stuff. Your even-keel attitude steadies the turbulence of grad school. Thanks also for your Hybrid Systems course, which was really mind-expanding. Also, I really enjoyed working on various CCDC projects with you and Kallie.

Jason Marden

Jason, thank you for the encouragement regarding grad school and my role in the department. I really appreciate your willingness to meet with me to help me see the light at the end of the tunnel.

Amr El Abbadi

Professor El Abbadi, thanks for your excellent Distributed Systems course, which helped me pivot to software engineering. Many thanks for your encouragement during that transition.

Funding Agencies

My research was supported by the Institute for Collaborative Biotechnologies through grant W911NF-09-D-0001 from the U.S. Army Research Office, and by the National Science Foundation through grant CNS-1329650, for which I'm very grateful. Moreover, generous fellowships from the UCSB graduate division, the UCSB *Center for Controls, Dynamical Systems, and Computation*, and philanthropist Michael Towbes contributed to my scholarship and my quality of life.

CCDC

The Center for Controls, Dynamical Systems, and Computation provided a steady stream of scholars to speak in a weekly seminar series, and I thank its organizers for their service in maintaining the Center.

Colleagues

I'm grateful for the company and friendship of my colleagues in João's lab. I appreciated Jason's insightful musings, Steven's quick wit, and Hari's perseverence as he straddled the fields of Controls and Biology. David and I worked together for several months and made me wish I had sought more collaborations with my lab-mates. Nevertheless, Henrique and I spent many hours at the whiteboard puzzling over this or that; his great attitude and sense of humor always lifted our

spirits. Combined with Sharad's keen insights, we whiled away many pleasant lunch breaks. Calvin, our long-term undergraduate intern, kept my fingers on the pulse of the undergraduate experience at UCSB; we had many enjoyable conversations about teaching and pedagogy. To our new recruits — Raphael, Murat, and Matina — you are in good hands. And I thank our many visiting scholars, who contributed to the lab's richness: Kyriakos, Masashi, Kunihisa, Michelle, Victor, Adolfo, Marcio, and Rodolfo. Finally, to my colleagues at UCLA — Professor Mani Srivastava, Fatima, Amr, Sean, Yi-Fan, and Andrew Symington — I extend my gratitude for their company and patience during our multi-year, long-distance collaboration during the Roseline project.

Teachers

One of the highlights of this graduate program was my pursuit of the Certificate in College and University Teaching. I benefitted from the pedagogical advice of Lisa Berry and Kim DeBacco in Instructional Development. I truly enjoyed my collaboration with my co-instuctor Professor Jeff Moehlis as we designed the first online course in the College of Engineering. I'm also thankful for Computer Science Professor Jason Isaacs and Department Chair Michael Soltys at Cal State Channel Islands for the opportunity to teach at another college campus.

Staff

A legion of staff members supports our department and I'm so grateful for their help sheparding us students through the program. Ken, Keith, Daryl, and Carlos

keep the hackers at bay. Paul, Avery, and Bear keep our shop stocked, our elevators moving, and the instructional labs functioning despite the undergrads' best efforts. Dimple, Kallie, Kelsey, Robin, and Shannon get the money where it needs to go. And the all-knowing Val keeps a watchful eye over everything to keep our graduate careers streamlined and snag-free. The department is truly blessed to have these fine folks.

Family and Friends

My wife, family, and friends helped me maintain a healthy work-life balance during my grad school career, and I'm very thankful for their continued presence in my life. They mean the world to me, and I'll leave it at that.

Curriculum Vitæ

Justin Payne Pearson

Education

2018	Ph.D. in Electrical and Computer Engineering (Expected), University of California, Santa Barbara
2007	M.S. in Mechanical Engineering, Stanford University
2006	B.S. in Mechanical Engineering, University of California, Santa Barbara

Experience

2012 – 2017	Graduate Student Researcher, University of California, Santa Barbara
2008 – 2012	Controls Engineer, AeroVironment
2016	Instructor of Record, *COMP 150: Introduction to Object Oriented Programming*, California State University at Channel Islands
2015	Certificate in College and University Teaching, UCSB
2015	Mentor, Robotics internship, Institute for Collaborative Biotechnology, UCSB
2015	Instructor of Record, *ENGR 3: Introduction to Programming*, UCSB
2015	Mentor, School of Scientific Thought, Center for Science and Engineering Partnerships, UCSB
2014	Teaching Assistant, *ENGR 3: Introduction to Programming*, UCSB
2014	Teaching Assistant, *ECE 147C: Control System Design Project*, UCSB

Publications

A. Alanwar, F. M. Anwar, Y. Zhang, J. Pearson, J. Hespanha, M. B. Srivastava. "Cyclops: PRU Programming Framework for Precise Timing Applications." *Proc. of the 2017 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, Aug. 2017.

J. Pearson, J. Hespanha, D. Liberzon. "Control with minimal cost-per-symbol encoding and quasi-optimality of event-based encoders." *IEEE Transactions on Automatic Control*, May 2017.

J. Pearson, J. Hespanha, D. Liberzon. "Quasi-optimality of event-based encoders." *Proc. of the 54th Conference on Decision and Control*, Dec. 2015.

J. Pearson, J. Hespanha, D. Liberzon. "Control with Minimum Communication Cost per Symbol." *Proc. of the 53nd Conference on Decision and Control*, Dec. 2014.

Abstract

Control under energy and time constraints

by

Justin Payne Pearson

The performance of a control system is often limited by constraints on timing, bandwidth, and energy. This dissertation explores the trade-offs between constraints on these resources, the control system performance, and the system to be controlled.

We begin by considering a networked control system in which the sensor sends its measurements to the controller over a limited-bandwidth communications channel. We explore the observation that the absence of communication nevertheless conveys information — i.e., nothing communication-worthy occurred. This suggests that energy (or other resources consumed by communication) could be saved using the *timing* of messages to transmit information, rather than the normal practice of transmitting data in the messages themselves. We develop a framework to explore this idea and derive a condition for the existence of a stabilizing controller that captures the trade-off between bandwidth, resource consumption, and the unstable eigenvalues of the linear system to be controlled. A surprising result is that if this condition is satisfied, then one may design a stabilizing controller that consumes resources at an arbitrarily small rate, pro-

vided one has access to a sufficiently precise clock. In an extreme example, a large amount of data is encoded into the precise transmission time of a single bit, and the receiver decodes this data from the time the bit is received. This result quantifies the trade-off between bandwidth and time as resources for transmitting information.

Next, we use our framework to analyze a family of event-based controllers. We show that these controllers can stabilize a system while consuming resources at a rate that is within 2.5 times the theoretically-minimum rate. These event-based controllers are intuitive and easy to implement, and our stability condition quantifies the cost (in additional required communication resources) that a control engineer pays for the convenience of implementing an event-based controller instead of the relatively more complicated controllers from the first section that use the theoretically-minimum communication rate.

A takeaway from these results is that networked and distributed control systems can benefit from precise timing. However, even non-networked systems can benefit from precise timing. We explore this by developing a control architecture that allows a controller running on a non-real-time operating system to run with a high degree of determinacy, even when the OS task scheduler suspends the control task. The architecture employs a small microprocessor to be used as a "real-time processor" that runs independently from the OS and buffers sensor measurement and actuator commands. We implement this on a Beaglebone Black

single-board computer and demonstrate that this architecture can significantly improve a controller's performance in the presence of OS preemption.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Control systems are frequently hampered by constraints on resources like limits on bandwidth, energy, and processor speed. This work explores the theoretical and practical trade-offs between these resources and control system performance. It consists of three chapters, each summarized next.

Control with Minimum Energy Per Symbol (Chapter 2)

Chapter 2 considers the problem of stabilizing a continuous-time linear time-invariant process subject to communication constraints. The basic setup is shown in Figure 1.1, in which a finite capacity communication channel connects the

process sensors to the controller/actuator. An encoder at the sensor sends a symbol through the channel once per sampling time, and the controller determines the actuation signal based on the incoming stream of symbols. The question arises: what is the smallest channel average bit-rate for which a given process can be stabilized?

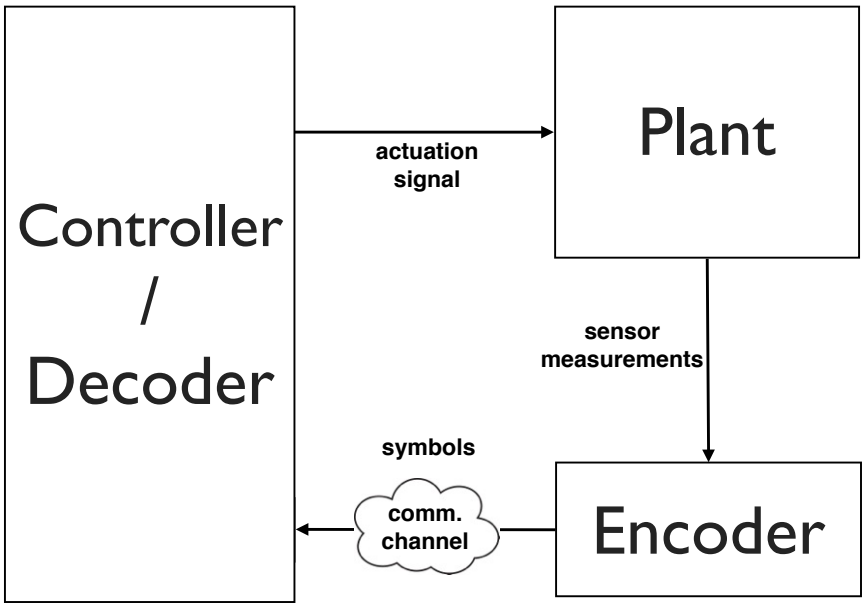

Figure 1.1: The limited-communication control setup. At sampling times, the encoder measures the process state and selects symbols from a finite alphabet to send to the decoder/controller. The decoder/controller constructs the actuation signal for the plant.

Prior work

Variants of the limited-communication environment in Figure 1.1 were also considered in [5, 11, 26, 40, 24, 19] and many other works. As pertains to the present

work, it was shown in [11, 26, 40] that a necessary and sufficient condition for stability can be expressed as a simple relationship between the unstable eigenvalues of the open-loop system matrix and the bit-rate of the communication channel. Extensions of this result have been enthusiastically explored, see [25, 20] and references therein.

Contributions

A starting point for the present work is the observation that an encoder can effectively save communication resources by occasionally not transmitting information — the absence of an explicitly transmitted symbol nevertheless conveys information. We formulate a framework to capture this by supposing that each symbol's transmission costs one unit of communication resources, except for one special free symbol that represents the absence of a transmission.

Within this framework, we define an encoder's *average cost per symbol* – essentially the largest average fraction of non-free symbols emitted by that encoder over all possible symbol streams. This chapter's technical contribution is a necessary and sufficient condition for the existence of a stabilizing controller and encoder/decoder pair obeying a constraint on its average cost per symbol. This condition depends on the channel's average bit-rate, the encoder's average cost per symbol, and the unstable eigenvalues of the open-loop system matrix. The proof is constructive in that it explicitly provides a family of controllers and

encoder/decoder pairs that stabilize the process when the condition holds. The pairs are optimal in the sense that they satisfy the stability condition as tightly as desired. As the constraint on the average cost per symbol is allowed to increase (becomes looser), our necessary and sufficient condition recovers the condition from [11].

Moreover, we show that if an encoder can stabilize the process, then it can do so using arbitrarily small amounts of communication resources per time unit. One way to achieve this is by transmitting only a few non-free symbols per time unit, but being very selective about which transmission period to send them in. Alternatively, the encoder and decoder could share a massive symbol library so that each symbol carries sufficient information about the state.

Finally, a counterintuitive corollary to our main result shows that if the process may be stabilized with average bit-rate r bits per time unit, then there exists a stabilizing controller and encoder/decoder pair using average bit-rate r which uses no more than 50% non-free symbols in any stream of symbols it may transmit.

Quasi-optimality of Event-based control (Chapter 3)

In Chapter 3, we use the framework from Chapter 2 to analyze a family of event-based controllers.

The encoders developed in Chapter 2 are optimal in the sense that they can stabilize a process with an average cost-per-symbol as low as possible. However, they are possibly very complex and difficult to implement. In particular, as an encoder's cost-per-symbol approaches the minimum bound, its codeword library grows to infinite size. In this chapter, we develop an easily-implementable event-based encoder/decoder and compare it to the optimal encoders from Chapter 2.

The framework of Chapter 2 requires symbol transmissions to occur at fixed transmission times. It would therefore appear to prohibit any sort of event-based control, in which events can occur aperiodially and extemporaneously. However, the framework can be regarded as event-based if one interprets non-free symbols as transmission-worthy events and the free symbol as "no transmission."

Prior work

Recent results in event-based control [3, 2, 18, 37] indicate that an encoder can conserve communication resources by transmitting only on a "need-to-know" ba-

sis. Preliminary work in event-based control assumed that the event-detector could transmit infinite-precision quantities across the communication channel to the controller/actuator. To extend this work to finite-bit-rate communication channels, recent works explore event-based *quantized* control, typically introducing an encoder/decoder or quantizer in the communication path to limit the number of bits transmitted. Several recent works offer strategies for event-based quantized control that study trade-offs between quantizer complexity, bit-rate, and minimum inter-transmission intervals. For example, [14] explores an intuitive event-based quantized control scheme that sends single bits based on the state estimation error transitioning between quantization levels. The design in [16] of an event-based quantized control scheme for a disturbed, stable LTI system allows the state trajectory to match as closely as desired the state-feedback state trajectory that would be obtained without communication constraints. In [38] the authors consider the simultaneous co-design of the event-generator and quantizer for the control of a non-linear system using the hybrid system framework from [9]. Sufficient bit-rates for event-triggered stabilizability of nonlinear systems were also studied in [17]. In [39] a method is developed for event-based quantized control design that achieves a desired convergence rate of a Lyapunov function of the state, while guaranteeing a positive lower bound on inter-transmission times and a uniform upper bound on the number of bits in each transmission.

Contributions

In contrast to the optimal encoders introduced in Chapter 2, the proposed event-based encoders are easy to implement but not optimal. However, they are only slightly sub-optimal. Specifically, this chapter's main technical contribution presents a sufficient condition for the existence of an emulation-based controller and event-based encoder/decoder pair. The condition resembles the sufficient condition from Chapter 2, and exceeds it by less than a factor of 2.5, meaning that the proposed event-based encoding scheme needs at most 2.5 times as many communication resources as an optimal encoding scheme requires. This establishes that event-based encoding schemes can offer "order-optimal" performance in communication-constrained control problems.

Preemption-resistant control on a non-real-time operating system (Chapter 4)

The previous chapters indicate that networked control systems can benefit from precise timing by embedding information in the timing of messages. In Chapter 4, we turn our attention away from networked control systems and explore how precise timing can benefit a controller running (locally) on a non-real-time operating system.

Modern computing systems offer many performance-enhancing features like multi-tasking operating systems, multiple layers of caching, and multiprocessor support. However, these features often result in nondeterminstic runtime behavior, making it a challenge to implement control systems on such platforms (see Figure 1.2). Consequently, controllers are often implemented on specialized platforms like "bare metal" microcontrollers, real-time OSs, or FPGAs, which offer finer control of execution and timing. It would be advantageous for a controller architecture to offer both the flexibility of a general-purpose computing platform and also the determinism of a specialized solution. This chapter proposes a controller architecture that addresses this need.

Background and prior work

A program may execute nondeterministically for several reasons. On a multi-threaded processor, the task scheduler may interrupt a task to let another task use the CPU, or to service an interrupt [31]. The time required to fetch data from memory varies wildly depending on whether the data was cached [35, 34]. In a Non-Uniform Memory Access (NUMA) multiprocessor architecture, CPUs are grouped into nodes, each with its own dedicated local memory. The speed of a memory reference therefore depends on whether the desired data resides in the executing processor's local memory or in another node's memory [15, 36]. System Management Interrupts commandeer the CPU and RAM to perform system

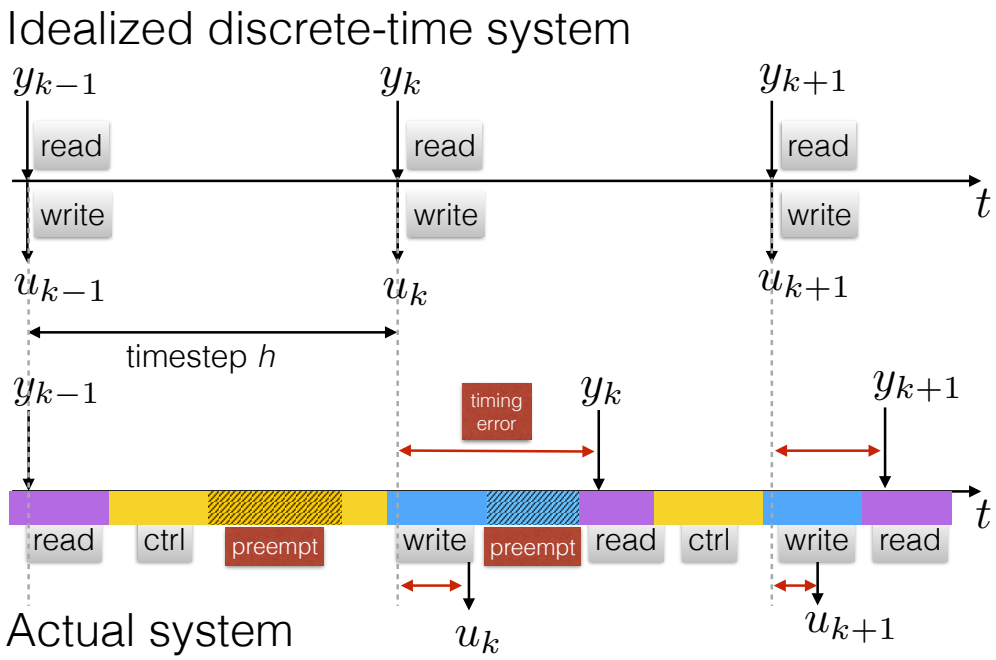

Figure 1.2: Timing in an idealized discrete-time system (top) versus a physical control system running on a non-real-time operating system (bottom). Due to OS preemption and other sources of nondeterminacy, the sensor and actuator signals y_k and u_k neither occur at their intended sample times nor align with each other.

maintenance, e.g., turning on the fan or verifying memory consistency [33].

Several platforms aim to provide determinism in program execution. FPGA designs are synthesized to obey strict user-supplied timing constraints and are therefore well-suited for control applications, see [23]. Matlab's "Simulink Coder" converts Simulink diagrams into code for embedded targets without OSs; [10] studies its use in rapid prototyping of real-time control algorithms. Real-time OSs like VxWorks, Integrity RTOS, μCOS, and FreeRTOS all provide OS-related functionality like multi-tasking, networking, file system support, and memory management while providing timing guarantees. The current RTOSs are surveyed in [4]. RTOS applications to control are explored in [1]. The authors of [6] analyze RTOS task scheduler algorithms. While powerful, these platforms lack the flexibility of a true general-purpose OS in terms of availability of libraries and device drivers.

Work has also been done to modify Linux itself to provide real-time performance guarantees. The OS provided by the "Real-Time Linux" project allows interrupt service routines (ISRs) to be run as regular tasks. Similarly, the Xenomai software augments Linux with a second kernel that runs above the main Linux kernel. The Xenomai kernel can disable the Linux scheduler in order to guarantee timely task execution. RT Linux and Xenomai have found applications in low-latency audio processing [21] and electrical substation automation [32]. They both allow the user to prioritize userspace tasks above interrupts.

However, this practice can result in reduced performance if misconfigured, e.g., dropping network packets due to the network ISR being neglected in favor of the control task. These solutions do not overcome the fundamental problem that when several real-time tasks share the same CPU, preemption is inevitable and is either managed by the OS or by the programmer.

This work implements the proposed control architecture on a Beaglebone single-board computer, running the controller on the primary CPU and utilizing a subsidiary processor core as a real-time I/O coprocessor. The practice of placing application-specific coprocessors beside a general-purpose procesor is a feature of the "ARM big.LITTLE" heterogeneous computing architecture and the family of processors in Texas Instruments' "Open Multimedia Applications Platform" initiative, discussed in [7].

Contributions

The specific contribution of this chapter is a controller architecture that enables a controller to run on a non-real-time OS like Linux, yet maintain precise timing of the sensing and actuation despite OS preemption. This is achieved by performing the sensing and actuation on a dedicated "bare metal" microcontroller that in essence serves as a real-time I/O coprocessor; we refer to this as the "Real-Time Unit" (RTU). Since the OS may preempt the controller at any time, we cannot rely on it for precise sampling or actuation. On the other hand, the

RTU does not run an OS, so it can sample and actuate at precise times without danger of OS preemption. The key idea is to have the RTU buffer time-stamped sensor measurements from the plant and apply buffered time-stamped actuation commands to the plant at precise times. Asynchronously, the controller requests an array of past measurements from the RTU, computes an array of future time-stamped actuation commands, and sends it to the RTU to be executed at the correct times. Consequently, the controller can be preempted but the RTU will continue to apply actuation on its behalf.

We implemented this controller architecture on a Beaglebone Black (BBB) to drive a DC motor. For demonstration purposes, a simple PID controller runs on Linux on the BBB's main 1-GHz CPU, and we use a subsidiary processor core on the BBB as the RTU. The RTU reads a rotary encoder and actuates the motor with PWM every 5 ms with 40 μs accuracy. The PID controller uses a simple method of predicting future measurements to compute an array of future PWM values, which it sends to the RTU. We compare this setup to a PID controller with identical gains that uses the BBB's standard file-based interface for I/O. We observe that although both setups perform well when the CPU is idle, when run alongside several other high-priority tasks the RTU-based setup far out-performs the standard I/O mechanism.

Chapter 2

Control with Minimum Energy Per Symbol

Parts of this chapter come from [30]:

In this chapter we consider the problem of stabilizing a continuous-time linear time-invariant process subject to communication constraints. We develop a framework for exploring the notion that the absence of communication nevertheless conveys information, yet it consumes no communication resources. We

model the absence of a communication by appending a special "free" symbol to the set of symbols offered by the communications channel. Transmitting a normal symbol costs one unit of communications resources, but transmitting the free symbol costs no resources. This yields the notion of an encoder's *average cost per symbol* — essentially the average fraction of non-free symbols sent by the encoder. We then develop a condition under which a stabilizing encoder with the smallest average cost per symbol may be designed.

This chapter is organized as follows. Section 2.2 contains a necessary condition for stability, namely that stability is not possible when our condition does not hold. To prove this result we actually show that it is not possible to stabilize the process with a large class of encoders — which we call M-of-N encoders — that includes all the encoders with average cost per symbol not exceeding a given threshold. Section 2.3 contains a sufficient condition for stability, showing that when our condition *does* hold, there is an encoder/decoder pair that can stabilize the process. We explicitly construct a possible encoding scheme.

2.1 Problem Statement

Consider a stabilizable linear time-invariant process

$$\dot{x} = Ax + Bu, \qquad x \in \mathbb{R}^n, u \in \mathbb{R}^m, \tag{2.1}$$

for which it is known that $x(0)$ belongs to a known bounded set $\mathcal{X}_0 \subset \mathbb{R}^n$. A sensor that measures the state $x(t)$ is connected to the actuator through a finite-data-rate, error-free, and delay-free communication channel, see Figure 2.1.

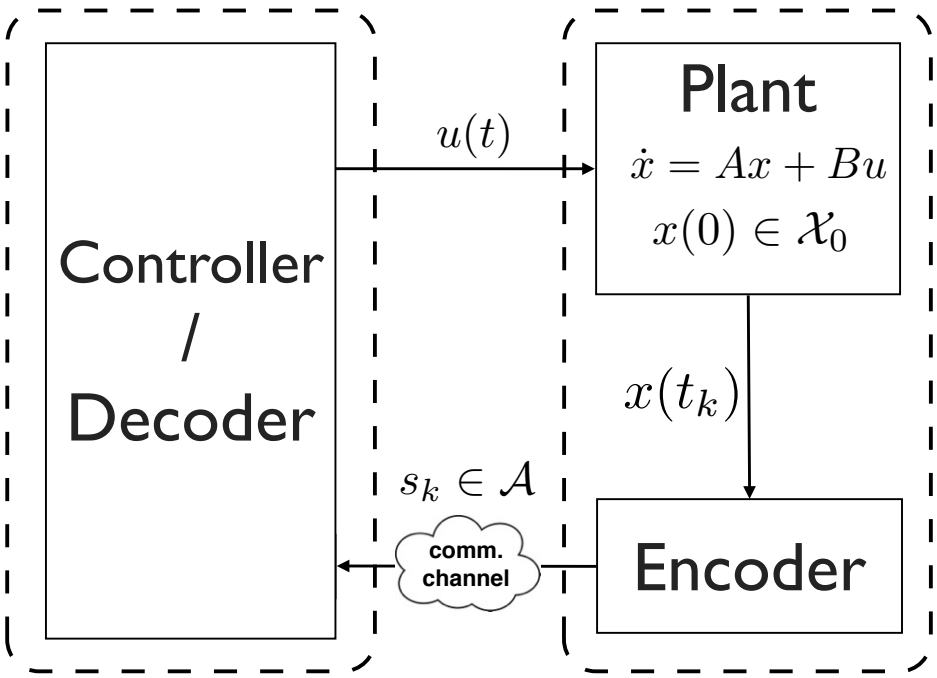

Figure 2.1: The limited-communication setup. At time t_k, the encoder samples the plant state $x(t_k)$ and selects symbol s_k from alphabet $\mathcal{A}$ to send to the decoder/controller. The decoder/controller constructs the actuation signal $u(t)$ for the plant.

An *encoder* collocated with the sensor samples the state at a fixed sequence of *transmission times* $\{t_k \in [0, \infty) : k \in \mathbb{Z}_{>0}\}$, and from the corresponding sequence of measurements $\{x(t_k) : k \in \mathbb{Z}_{>0}\}$ causally constructs a sequence of symbols $\{s_k \in \mathcal{A} : k \in \mathbb{Z}_{>0}\}$ from a nonempty finite alphabet $\mathcal{A}$. Without loss of generality, $\mathcal{A} = \{0, 1, \ldots, S\}$ with $S \coloneqq |\mathcal{A}| - 1$. At time t_k the encoder sends the symbol s_k through the channel to a *decoder/controller* collocated with the actu-

ator, which causally constructs the control signal $u(t)$, $t \geqslant 0$ from the sequence of symbols $\{s_k \in \mathcal{A} : k \in \mathbb{Z}_{>0}\}$ that arrive at the decoder. The sequence of transmission times $\{t_k\}$ is assumed to be monotonically nondecreasing and unbounded (i.e., $\lim_{k\to\infty} t_k = +\infty$). The fact that the sequence of transmission times is fixed *a priori* prevents the controller from communicating information in the transmission times themselves. Note that because the sequence of transmission times is not necessarily strictly increasing, this allows multiple transmissions at a single time instant, which can be viewed as encoding several symbols in the same message.

The non-negative *average bit-rate* r of a sequence of symbols $\{s_k\} \subset \{0, \ldots, S\}$ transmitted at times $\{t_k\}$ is the rate of transmitted information in units of bits per time unit, and is defined as

$$r \coloneqq \log_2(S+1) \ \limsup_{k\to\infty} \frac{k}{t_k}. \tag{2.2}$$

We assume that the symbol $0 \in \mathcal{A}$ can be transmitted without consuming any communication resources, but the other S symbols each require one unit of communication resources per transmission. One can think of the "free" symbol 0 as the absence of an explicit transmission. The "communication resources" at stake may be energy, time, or any other resource that may be consumed in the course of the communication process. In order to capture the average rate

at which an encoder consumes communication resources, we define the *average cost per symbol* of an encoder as follows: We say an encoder has *average cost per symbol not exceeding* γ if there exists a non-negative integer N_0 such that for every symbol sequence $\{s_k\}$ generated by the encoder, we have

$$\frac{1}{N_2} \sum_{k=N_1}^{N_1+N_2-1} I_{s_k \neq 0} \leqslant \gamma + \frac{N_0}{N_2} \quad \forall N_1, N_2 \in \mathbb{Z}_{>0}, \tag{2.3}$$

where $I_{s_k \neq 0} \coloneqq 1$ if the kth symbol is not the free symbol, and 0 if it is. The summation in (2.3) captures the total resources spent transmitting symbols s_{N_1} through $s_{N_1+N_2-1}$, independent of the symbols' transmission times. Motivating this definition of average cost per symbol is the observation that the left-hand side has the intuitive interpretation of the average cost per transmitted symbol between symbols s_{N_1} and $s_{N_1+N_2-1}$. As $N_2 \to \infty$, which corresponds to averaging over a growing window of symbols, the rightmost term vanishes, leaving γ as an upper bound on the average long-term cost per symbol of the symbol sequence. To illustrate the necessity of the N_0 term, note that without it, any symbol sequence with a nonzero symbol at some index k will violate (2.3) for any $\gamma \in [0, 1)$ by picking $N_1 \coloneqq k$ and $N_2 \coloneqq 1$; the presence of the N_0 term allows an encoder to have a very small average cost per symbol while still enabling it to transmit long runs of non-free symbols. Note that because the left-hand side of (2.3) never exceeds 1, every encoder has an average cost per symbol not exceeding c

for any $c \geqslant 1$. Also, note that any encoder with average cost per symbol not exceeding $\gamma = 0$ can transmit at most N_0 non-free symbols for all time, making it unsuitable for stabilization. For these two reasons, any encoder of interest will have an average cost per symbol not exceeding some $\gamma \in (0, 1]$.

Whereas the average bit-rate r depends only on the symbol alphabet $\mathcal{A}$ and transmission times $\{t_k\}$, the average cost per symbol of an encoder/decoder pair depends on every possible symbol sequence it may generate, and therefore may in general depend on the encoder/decoder pair, the controller, process (2.1), and the initial condition $x(0)$.

The specific question considered in this chapter is: under what conditions on the average bit-rate and average cost per symbol do there exist a controller and encoder/decoder pair that stabilize the state of process (2.1)?

2.2 Necessary condition for boundedness with limited-communication encoders

It is known from [11, 26, 40] that it is possible to construct a controller and encoder/decoder pair that stabilize process (2.1) with average bit-rate r only if

$$r \ln 2 \geqslant \sum_{i:\mathbb{R}\lambda_i[A]>0} \lambda_i[A], \tag{2.4}$$

where ln denotes the base-e logarithm, and the summation is over all eigenvalues of A with nonnegative real part. The following result shows that a larger average bit-rate r may be needed when one poses constraints on the encoder's average cost per symbol γ. Specifically, when $\gamma \geqslant S/(S+1)$ the (necessary) stability condition reduces to (2.4), but when $\gamma < S/(S+1)$ an average bit-rate r larger than (2.4) is necessary for stability.

Theorem 1. *Suppose a controller and encoder/decoder pair keep the state of process* (2.1) *bounded for every initial condition $x_0 \in \mathcal{X}_0$. If the encoder uses an alphabet $\{0, \ldots, S\}$, has average bit-rate r, and has average cost per symbol not exceeding γ, then we must have*

$$r\, f(\gamma, S) \ln 2 \geqslant \sum_{i:\mathbb{R}\lambda_i[A]>0} \lambda_i[A], \tag{2.5}$$

where the function $f : [0,1] \times \mathbb{Z}_{>0} \to [0,\infty)$ is defined as

$$f(\gamma, S) \coloneqq \begin{cases} \frac{H(\gamma)+\gamma \log_2 S}{\log_2(S+1)} & 0 \leqslant \gamma < \frac{S}{S+1} \\ 1 & \frac{S}{S+1} \leqslant \gamma \leqslant 1, \end{cases} \tag{2.6}$$

and $H(p) \coloneqq -p\log_2(p) - (1-p)\log_2(1-p)$ is the base-2 entropy of a Bernoulli random variable with parameter p.

It is worth making three observations regarding the function f: First, one can think of $f(\gamma, S)$ as the degradation of an encoder's ability to convey infor-

mation due to the constraint that its average cost per symbol not exceed γ. For example, consider an encoder which transmits symbols 0 and 1, subject to the constraint that the long-term fraction of 1's must not exceed $\gamma = 10\%$. Due to this constraint, each transmitted symbol does not convey 1 bit of information on average, but rather only $f(0.1, 1) = H(0.1) \approx 0.47$ bits. First, $f(\gamma, S)$ is nondecreasing and continuous in γ for any fixed S, as illustrated in Figure 2.2. Second, $f(\gamma, S)$ is monotone nonincreasing in S for any fixed $\gamma \in [0, 1]$, as illustrated in Figure 2.3.

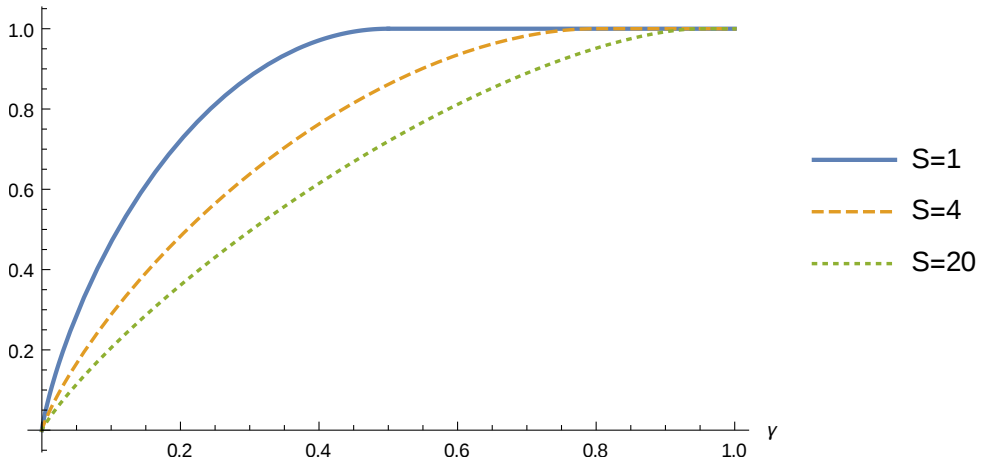

Figure 2.2: A plot of $f(\gamma, S)$ versus γ for $S = 1, 4, 20$.

Therefore, for a fixed r and γ, an encoder can increase its value of $f(\gamma, S)$ "for free" by decreasing S while commensurately decreasing its average transmission period to keep r constant in accordance with (2.2). This implies that smaller alphabets are preferable to large ones when trying to satisfy (2.5) with a given fixed average bit-rate and average cost per symbol.

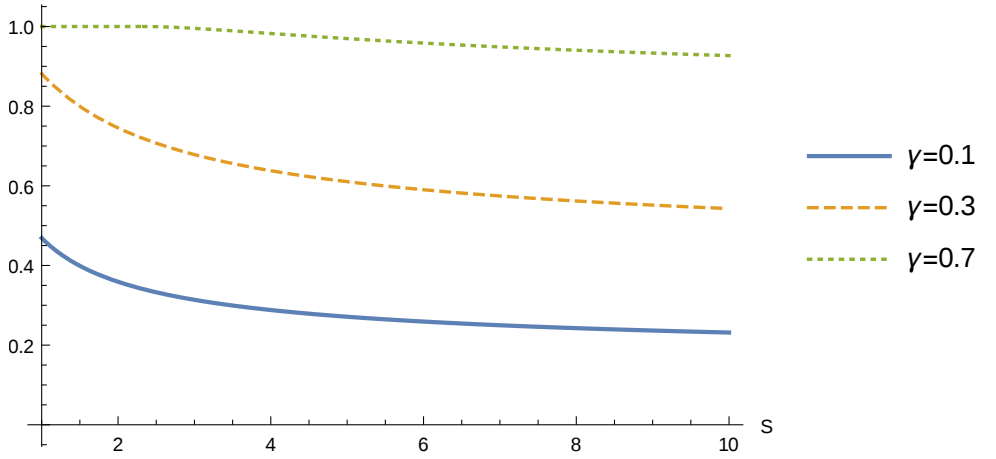

Figure 2.3: A plot of $f(\gamma, S)$ versus S for $\gamma = 0.1, 0.3, 0.7$.

Said another way, an encoder that transmits frequently has more choices as to when precisely to transmit its non-free symbols, so it has more codewords at its disposal and can convey more information.

The intuition is as follows: for a particular fixed average bit-rate, an encoder may either rapidly transmit symbols from a small alphabet or slowly transmit symbols from a large alphabet. In the former case, since the free symbol occupies a larger fraction of the alphabet, it will tend to be used more frequently, resulting in lower resource consumption. For example, the event-based encoder in Chapter 3 achieves an order-optimal performance bound by using $S = 2$ and increasing the transmission rate in order to achieve a suffiently high average bit-rate.

The third observation is that the average *cost per time unit*, which is given by

$$\gamma \limsup_{k \to \infty} \frac{k}{t_k},$$

can be made arbitrarily small while still satisfying (2.5). This can be achieved in several ways:

1. *Large symbol library with infrequent transmissions:* For a given average cost per symbol γ, pick the encoder's transmission times as $t_k \coloneqq kT$ for sufficiently large T so that the average cost per time unit $\gamma \limsup_{k\to\infty} k/t_k = \gamma/T$ is as small as desired. Then, using $r \coloneqq \log_2(S+1)/T$ and leveraging the fact that

$$rf(\gamma, S) = \begin{cases} \frac{H(\gamma)+\gamma \log_2 S}{T} & 0 \leqslant \gamma < \frac{S}{S+1} \\ \frac{\log_2(S+1)}{T} & \frac{S}{S+1} \leqslant \gamma \leqslant 1 \end{cases} \tag{2.7}$$

   is monotone increasing in S for fixed γ, pick S large enough to satisfy (2.5). By choosing a large T and S, this scheme elects to send data-rich symbols only infrequently. The state — although remaining bounded — may grow quite large between these infrequent transmissions. Moreover, the large symbol library may require sizeable computational resources to store and process.

2. *Large symbol library with costly symbols rarely sent:* If the encoder's transmission times $\{t_k\}$ are fixed, pick γ small enough to make the average cost per time unit $\gamma \limsup_{k\to\infty} k/t_k$ as small as desired, then increase S as in the previous case to satisfy (2.5). Like the previous case, this approach

requires processing a large symbol library.

3. *Frequent transmissions with costly symbols rarely sent:* If the number of non-free symbols S is fixed, it is still possible to choose an average cost per symbol γ and transmission times $t_k \coloneqq kT$ so that (2.5) is satisfied and the average cost per time unit $\gamma \limsup_{k\to\infty} k/t_k$ is as small as desired. To verify that this is possible, note that the sequences $\gamma_i \coloneqq e^{-i}$, $T_i \coloneqq e^{-i}\sqrt{i}$, $i \in \mathbb{Z}_{>0}$ have the property that as $i \to \infty$, we have $\gamma_i \to 0$, $T_i \to 0$, and $\gamma_i/T_i \to 0$, but $H(\gamma_i)/T_i \to \infty$, so leveraging (2.7) we conclude that $r_i f(\gamma_i, S) \ln 2 \to \infty$ (where $r_i \coloneqq \log_2(S+1)/T_i$). This means that one can find $i \in \mathbb{Z}_{>0}$ sufficiently large to make the average cost per time unit arbitrarily small and also satisfy the necessary condition (2.5). In practice, to operate with a very small sampling period T, this approach requires an encoder/decoder pair with a very precise clock.

Remark 1. The addition of the "free" symbol effectively increases the average bit-rate without increasing the rate of resource consumption, as seen by the following two observations:

- Without the free symbols, the size of the alphabet would be S and the average bit-rate would be

$$\log_2(S) \limsup_{k\to\infty} \frac{k}{t_k} < \log_2(S+1) \limsup_{k\to\infty} \frac{k}{t_k}.$$

It could happen that this average bit-rate is too small to bound the plant, yet after the introduction of the free symbol, the condition (2.5) is satisfied.

- Since γ is essentially the fraction of non-free symbols, the quantity $r\gamma$ is the number of bits per time unit spent transmitting non-free symbols. But since $f(\gamma, S) \geqslant \gamma$, again we see that the free symbols help satisfy (2.5). To see that $f(\gamma, S) \geqslant \gamma$, observe that for any $S \in \mathbb{Z}_{>0}$, $f(\cdot, S)$ is concave and reaches 1 before the identity function does, hence it is everywhere above the identity function on $(0, 1)$, and it matches the identity function at the endpoints 0 and 1.

2.2.1 Setup and Proof of Theorem 1

We lead up to the proof of Theorem 1 by first establishing three lemmas centered around a restricted but large class of encoders called M-of-N encoders. We first define M-of-N encoders, which essentially partition their symbol sequences into N-length *codewords*, each with M or fewer non-free symbols. Lemma 1 demonstrates that every encoder with a bounded average cost per symbol is an M-of-N encoder for appropriate N and M. Next, in Lemma 2 we establish a relationship between the number of codewords available to an M-of-N encoder and the function f as defined in (2.6). Then, in Lemma 3 we establish a necessary condition for an M-of-N encoder to bound the state of process (2.1). Finally, the proof of Theorem 1 is built upon these three results.

We now introduce the class of M-of-N encoders. For $N \in \mathbb{Z}_{>0}$, $\ell \in \mathbb{Z}_{\geqslant 0}$, we define the ℓ^{th} *N-symbol codeword* to be the sequence $\{s_{\ell N+1}, s_{\ell N+2}, \ldots, s_{\ell N+N}\}$ of N consecutive symbols starting at the index $k = \ell N + 1$. For $M \in \mathbb{R}_{\geqslant 0}$ with $M \leqslant N$, an *M-of-N encoder* is an encoder for which every N-symbol codeword has M or fewer non-free symbols, i.e.,

$$\sum_{k=\ell N+1}^{\ell N+N} I_{s_k \neq 0} \leqslant M, \qquad \forall \ell \in \mathbb{Z}_{\geqslant 0}. \tag{2.8}$$

The total number of distinct N-symbol codewords available to an M-of-N encoder is thus given by

$$L(N, M, S) \coloneqq \sum_{i=0}^{\lfloor M \rfloor} \binom{N}{i} S^i, \tag{2.9}$$

where the ith term in the summation counts the number of N-symbol codewords with exactly i non-free symbols. In keeping with the problem setup, the M-of-N encoders considered here each draw their symbols from the symbol library $\mathcal{A} \coloneqq \{0, 1, \ldots, S\}$ and transmit symbols at times $\{t_k\}$.

An intuitive property of M-of-N encoders is that they have an average cost per symbol not exceeding M/N with $N_0 = 2M$. This result is presented as Lemma 5 in the appendix.

The fact that an M-of-N encoder refrains from sending "expensive" codewords effectively reduces its ability to transmit information: For $M < N$, we have

$L(N, M, S) < L(N, N, S)$ and so a codeword from an M-of-N encoder conveys less information than a codeword from an unconstrained encoder. Specifically, a codeword sent from an M-of-N encoder conveys $\log_2 L(N, M, S)$ bits of information, whereas a codeword from an encoder without the M-of-N constraint conveys $\log_2 L(N, N, S) = N \log_2(S + 1)$ bits.

The next lemma, proved in the appendix, shows that the set of M-of-N encoders is "complete" in the sense that every encoder with average cost per symbol not exceeding a finite threshold γ is actually an M-of-N encoder for N sufficiently large and $M \approx \gamma N$.

Lemma 1. *For any encoder/decoder pair with average cost per symbol not exceeding $\gamma \in (0, 1]$, and every constant $\epsilon > 0$, there exist $M \in \mathbb{R}_{\geqslant 0}$ and $N \in \mathbb{Z}_{>0}$ with $M < N\gamma(1 + \epsilon)$ such that the encoder/decoder pair is an M-of-N encoder.*

The next lemma establishes a relationship between the number of codewords $L(N, M, S)$ available to an M-of-N encoder and the function f defined in (2.6).

Lemma 2. *For any $N \in \mathbb{Z}_{>0}$, $S \in \mathbb{Z}_{\geqslant 0}$ and $\gamma \in [0, 1]$, the function L defined in* (2.9) *and the function f defined in* (2.6) *satisfy*

$$\frac{\ln L(N, N\gamma, S)}{N \ln(S + 1)} \leqslant f(\gamma, S), \tag{2.10}$$

with equality holding only when $\gamma = 0$ or $\gamma = 1$. Moreover, we have asymptotic

equality in the sense that

$$\lim_{N\to\infty} \frac{\ln L(N, N\gamma, S)}{N \ln(S+1)} = f(\gamma, S). \tag{2.11}$$

The left and right sides of (2.10) are plotted in Figure 2.4.

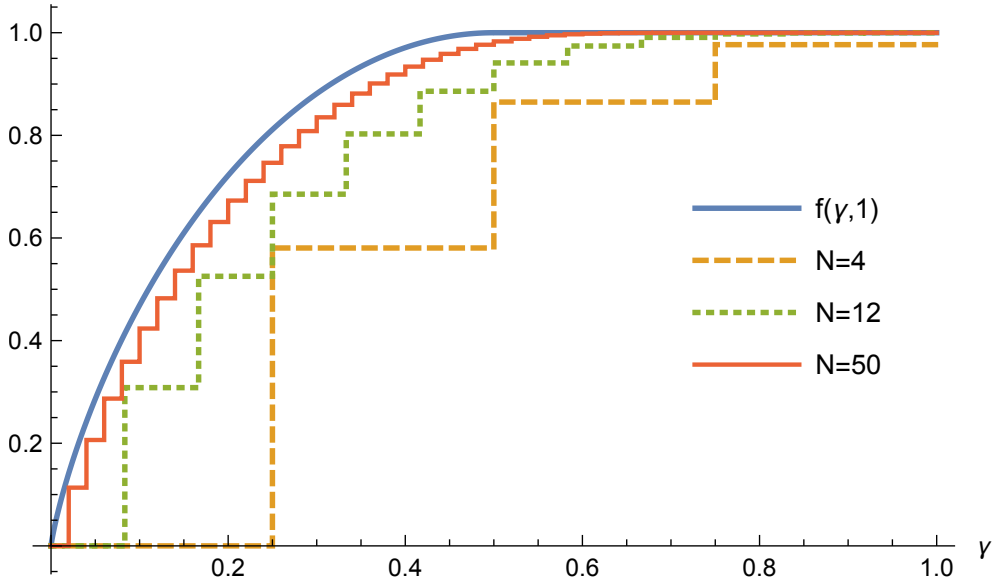

Figure 2.4: A plot of $f(\gamma, S)$ and $\ln L(N, N\gamma, S)/N\ln(S+1)$ versus γ for $S = 1$ and $N = 4, 12, 50$.

Proof of Lemma 2. In this proof we use the base-2 logarithm to match the notation of an information theoretic theorem that we invoke. Let $N \in \mathbb{Z}_{>0}$ and $S \in \mathbb{Z}_{\geqslant 0}$ be arbitrary. First we prove (2.10) for $\gamma \in \left(0, \frac{S}{S+1}\right]$. Applying the Binomial Theorem to the identity $1 = (\gamma + (1-\gamma))^N$, we obtain

$$1 = \sum_{i=0}^{N} \binom{N}{i} \gamma^i (1-\gamma)^{N-i}.$$

Since each term in the summation is positive, keeping only the first $\lfloor N\gamma \rfloor$ terms yields the inequality

$$1 > \sum_{i=0}^{\lfloor N\gamma \rfloor} \binom{N}{i} \gamma^i (1-\gamma)^{N-i}. \tag{2.12}$$

Next, a calculation presented as Lemma 6 in the appendix reveals that

$$\gamma^i (1-\gamma)^{N-i} \geqslant 2^{-N\,H(\gamma)} \, \frac{S^i}{S^{N\gamma}} \tag{2.13}$$

for all $N, S \in \mathbb{Z}_{>0}$, $\gamma \in \left(0, \frac{S}{S+1}\right]$, and $i \in [0, N\gamma]$. Using this in (2.12) and taking $\log_2$ of both sides yields

$$\frac{\log_2 L(N, N\gamma, S)}{N} < H(\gamma) + \gamma \log_2 S. \tag{2.14}$$

By the definition of f, we have $\log_2(S+1) f(\gamma, S) = H(\gamma) + \gamma \log_2 S$ when $\gamma \in \left[0, \frac{S}{S+1}\right]$. Thus, (2.14) proves the strict inequality in (2.10) for $\gamma \in \left(0, \frac{S}{S+1}\right]$. Next, suppose $\gamma \in \left(\frac{S}{S+1}, 1\right)$ and observe from (2.9) that $L(N, M, S)$ is a sum of positive terms whose index reaches $\lfloor M \rfloor$, hence $L(N, N\gamma, S)$ is strictly less than $L(N, N, S)$ for any $\gamma < 1$. We conclude that

$$\begin{aligned} \frac{\log_2 L(N, N\gamma, S)}{N} &< \frac{\log_2 L(N, N, S)}{N} \\ &= \log_2(S+1) \end{aligned} \tag{2.15}$$

$$= \log_2(S+1) f(\gamma, S), \tag{2.16}$$

where the equality in (2.15) follows simply from the fact that $L(N, N, S)$ is the number of all possible codewords of length N and hence equals $(S+1)^N$, and (2.16) follows from the definition of f when $\gamma \in (\frac{S}{S+1}, 1)$. This concludes the proof of the strict inequality in (2.10) for $\gamma \in (0, 1)$. The proof of (2.10) for $\gamma = 0$ follows merely from inspection of (2.10), and the $\gamma = 1$ case follows from the equality in (2.15).

Next we prove the asymptotic result (2.11) using information-theoretic methods. First we prove (2.11) for $\gamma \in [0, \frac{S}{S+1})$. Consider a random variable X parameterized by $S \in \mathbb{Z}_{\geqslant 0}$ and $\gamma \in [0, \frac{S}{S+1})$ which takes values in $\mathcal{X} \coloneqq \{0, 1, \ldots, S\}$ with probabilities given by

$$\mathbb{P}(X = 0) \coloneqq (1 - \gamma)$$

$$\mathbb{P}(X = i) \coloneqq \gamma / S \qquad i = \{1, 2, \ldots, S\}.$$

Following our convention, we call 0 the "free" symbol and $1, \ldots, S$ the "non-free" symbols. To lighten notation we write $p(x) \coloneqq \mathbb{P}(X = x), x \in \mathcal{X}$. The entropy of the random variable X is

$$H(X) \coloneqq -\sum_{i=0}^{S} p(i) \log_2 p(i) = H(\gamma) + \gamma \log_2 S, \tag{2.17}$$

where we have overloaded the symbol H so that $H(\gamma) \coloneqq -\gamma \log_2 \gamma - (1-\gamma)\log_2(1-\gamma)$ is the entropy of a Bernoulli random variable with parameter γ.

Next, for some arbitrary $N \in \mathbb{Z}_{>0}$, we consider N-length sequences of i.i.d. copies of X. Let $\mathcal{X}^N \coloneqq \{(x_1, \ldots, x_N) : x_i \in \mathcal{X}\}$. We use the symbol x^N as shorthand for $(x_1, \ldots, x_N)$, and we use $p(x^N)$ as shorthand for $\mathbb{P}\Big((X_1, X_2, \ldots, X_N) = (x_1, x_2, \ldots, x_N)\Big)$.

Given an N-length sequence $x^N \in \mathcal{X}^N$, the probability that the N i.i.d. random variables $(X_1, \ldots, X_N)$ take on the values in the N-tuple x^N is given by

$$p(x^N) = (1-\gamma)^{N - \sum_{i=1}^N I_{x_i \neq 0}} \; \frac{\gamma^{\sum_{i=1}^N I_{x_i \neq 0}}}{S^{\sum_{i=1}^N I_{x_i \neq 0}}}. \tag{2.18}$$

The summation $\sum_{i=1}^N I_{x_i \neq 0}$ is the number of non-free symbols in the N-tuple x^N. For arbitrary $\epsilon > 0$, define the set $A_\epsilon^{(N)} \subseteq \mathcal{X}^N$ as

$$A_\epsilon^{(N)} \coloneqq \left\{ x^N \in \mathcal{X}^N \;\middle|\; N(\gamma - \delta_\epsilon) \leqslant \sum_{i=1}^N I_{x_i \neq 0} \leqslant N(\gamma + \delta_\epsilon) \right\}, \tag{2.19}$$

where $\delta_\epsilon \coloneqq \epsilon / \log_2 \frac{(1-\gamma)S}{\gamma}$. That is, $A_\epsilon^{(N)}$ is the set of all N-length sequences with "roughly" $N\gamma$ non-free symbols. Using (2.17), (2.18), and the definition of δ_ϵ, we can express the inequalities in (2.19) as

$$A_\epsilon^{(N)} =$$

$$\left\{ x^N \in \mathcal{X}^N \;\middle|\; 2^{-N(H(X)+\epsilon)} \leqslant p(x^N) \leqslant 2^{-N(H(X)-\epsilon)} \right\}. \tag{2.20}$$

Here we relied on the fact that $\frac{(1-\gamma)S}{\gamma} > 1$ for $S \in \mathbb{Z}_{>0}, \gamma \in [0, \frac{S}{S+1})$. In the form of (2.20), we recognize $A_\epsilon^{(N)}$ as the so-called "typical set" of N-length sequences of i.i.d. copies of X as defined in [8]. Theorem 3.1.2 of [8] uses the Asymptotic Equipartition Property of sequences of i.i.d. random variables to prove that for any $\epsilon > 0$, we have

$$(1-\epsilon)2^{N(H(X)-\epsilon)} \leqslant \left|A_\epsilon^{(N)}\right| \tag{2.21}$$

for $N \in \mathbb{Z}_{>0}$ large enough.

Next, we observe that

$$\left|A_\epsilon^{(N)}\right| \leqslant L\left(N, N\left(\gamma + \delta_\epsilon\right), S\right), \tag{2.22}$$

because $|A_\epsilon^{(N)}|$ is the number of N-length sequences with a number of non-frees in the interval $[N(\gamma - \delta_\epsilon), N(\gamma + \delta_\epsilon)]$, whereas the right-hand side counts sequences with a number of non-frees in the larger interval $[0, N(\gamma + \delta_\epsilon)]$. Combining (2.21) and (2.22), we obtain that for any $\epsilon > 0$,

$$\begin{aligned} \frac{1}{N}\log_2(1-\epsilon) + H(\gamma) + \gamma \log_2 S - \epsilon \leqslant \\ \frac{1}{N}\log_2 L\left(N, N\left(\gamma + \delta_\epsilon\right), S\right) \end{aligned} \tag{2.23}$$

for N large enough. Moreover, by (2.10) we have

$$\frac{1}{N}\log_2 L\left(N, N\left(\gamma+\delta_\epsilon\right), S\right) \leqslant$$
$$H\left(\gamma+\delta_\epsilon\right)+\left(\gamma+\delta_\epsilon\right)\log_2 S \tag{2.24}$$

for any $\gamma \in [0, \frac{S}{S+1})$, $N, S \in \mathbb{Z}_{>0}$, and $\epsilon > 0$. Combining these two observations establishes an upper and lower bound on $\frac{1}{N}\log_2 L\left(N, N\left(\gamma+\delta_\epsilon\right), S\right)$. Letting $\epsilon \to 0$, the upper and lower bounds converge to $H(\gamma)+\gamma\log_2 S$, establishing (2.11) for $\gamma \in [0, \frac{S}{S+1})$. Since the upper and lower bounds are continuous in γ, this proves (2.11) for $\gamma = \frac{S}{S+1}$ as well.

Lastly, suppose $\gamma \in (\frac{S}{S+1}, 1]$. Since L is monotonically nondecreasing in its second argument, we have

$$\frac{1}{N}\log_2 L\left(N, N\frac{S}{S+1}, S\right) \leqslant \frac{1}{N}\log_2 L\left(N, N\gamma, S\right). \tag{2.25}$$

Moreover, by (2.10) we have

$$\frac{1}{N}\log_2 L\left(N, N\gamma, S\right) \leqslant \log_2(S+1). \tag{2.26}$$

Combining these establishes an upper and lower bound on $\frac{1}{N}\log_2 L\left(N, N\gamma, S\right)$. Taking $N \to \infty$, the bounds become equal because (2.11) holds for $\gamma = \frac{S}{S+1}$ in the lower bound. Here we relied on the fact that $f(\gamma, S)$ is continuous in γ. We

obtain

$$\lim_{N\to\infty} \frac{1}{N} \log_2 L\left(N, N\gamma, S\right) = \log_2(S+1). \tag{2.27}$$

This concludes the proof of Lemma 2. ■

The following lemma provides a necessary condition for an M-of-N encoder to be able to bound the state of process (2.1).

Lemma 3. *Consider an M-of-N encoder/decoder pair with average bit-rate r using a channel with alphabet $\{0, \ldots, S\}$ (with 0 the free symbol). If the pair keeps the state of process (2.1) bounded for every initial condition, then we must have*

$$r \, \frac{\ln L(N, M, S)}{N \ln(S+1)} \ln 2 \geqslant \sum_{i:\mathbb{R}\lambda_i[A]>0} \lambda_i[A]. \tag{2.28}$$

Proof of Lemma 3. The proof of this result can be constructed using an argument similar to the ones found in [11, 40], which considers the rate at which the uncertainty on the state, as measured by the volume of the set where it is known to lie, grows through the process dynamics (2.1) and shrinks upon the receipt of each N-symbol codeword.

We proceed with a proof by contradiction inspired by [11, 40], which considers the rate at which the uncertainty on the state, as measured by the volume $\mathcal{X}_0 \subset \mathbb{R}^n$

of the set where the initial state is known to lie, grows through process (2.1) and shrinks upon the receipt of information from the encoder. Consider an encoder/decoder pair whose encoder is an M-of-N encoder using symbols $\{0, \ldots, S\}$ and has average bit-rate r. For the sake of contradiction, suppose the controller and encoder/decoder pair keep the state of process (2.1) bounded for every initial condition $x_0 \in \mathcal{X}_0$, but that

$$r < r_{\min} \coloneqq \frac{N \ln(S+1)}{\ln L(N, M, S) \ln 2} \sum_{i:\mathbb{R}\lambda_i[A]>0} \lambda_i[A]. \tag{2.29}$$

After a change of coordinates, process (2.1) can be transformed to

$$\begin{bmatrix} \dot{x}_+ \\ \dot{x}_- \end{bmatrix} = \begin{bmatrix} A_+ & 0 \\ 0 & A_- \end{bmatrix} \begin{bmatrix} x_+ \\ x_- \end{bmatrix} + \begin{bmatrix} B_+ \\ B_- \end{bmatrix} u, \tag{2.30}$$

where $x_+ \in \mathbb{R}^{n_+}$, $x_- \in \mathbb{R}^{n_-}$, $u \in \mathbb{R}^m$, $n_+ + n_- = n$, and the eigenvalues of A are partitioned between $A_+ \in \mathbb{R}^{n_+ \times n_+}$ and $A_- \in \mathbb{R}^{n_- \times n_-}$, with A_+ having the eigenvalues of A with strictly positive real part and A_- the remaining ones. We focus our attention on the unstable subsystem

$$\dot{x}_+ = A_+ x_+ + B_+ u, \qquad x_+ \in \mathbb{R}^{n_+}, u \in \mathbb{R}^m. \tag{2.31}$$

Let $\varphi_+(t; x_0)$ denote the solution of (2.31) in closed-loop, that is, where $u(t)$ is determined by the decoder/controller in response to symbols sent by the encoder.

Suppose that by time t the decoder/controller has observed the specific sequence of symbols $\{s_k : t_k \leqslant t\}$. Define

$$\mathcal{X}_+(t) \coloneqq \big\{x_+ \in \mathbb{R}^{n_+} : \exists x_0 \in \mathcal{X}_0 : \varphi_+(t; x_0) = x_+ \& \quad \mathbf{Enc}(t, x_0) = \{s_k \,:\, t_{\lfloor \frac{k}{N} \rfloor N} \leqslant t\}\big\},$$

where $\mathbf{Enc}(t; x_0)$ denotes the set of codewords that the decoder/controller has observed from the encoder over time interval $[0, t]$ as process (2.1) runs in closed-loop from the initial condition $x(0) = x_0$. The set $\mathcal{X}_+(t)$ is the tightest set of points that the decoder/controller can deduce that the state x_+ lies in at time t, based on the observation of all N-length codewords up to time t. Since the decoder/controller cannot be certain of where $x_+(t)$ lies within $\mathcal{X}_+(t)$, we refer to $\mathcal{X}_+(t)$ as the *uncertainty region*.

Let $\nu(t) := \int_{x \in \mathcal{X}_+(t)} dx$ denote the volume of $\mathcal{X}_+(t)$, and let $\nu^+(t) \coloneqq \lim_{\tau \downarrow t} \nu(\tau)$ and $\nu^-(t) \coloneqq \lim_{\tau \uparrow t} \nu(\tau)$ denote the limits of $\nu(t)$ from above and below.

Let us now explore how the volume of the uncertainty region evolves due to the process. For arbitrary $k \in \mathbb{Z}_{>0}$, consider the open time interval $\big(t_{kN}, t_{kN+N}\big)$ during which the kth codeword is transmitted. Since no complete codewords arrive in this time interval, $\mathbf{Enc}(t; x_0)$ remains constant and the set $\mathcal{X}_+(t)$ simply expands under process (2.31) for $t \in \big(t_{kN}, t_{kN+N}\big)$. By the variation of constants

formula,

$$x_+(t_{kN+N}) = e^{A_+(t_{kN+N}-t_{kN})}x_+(t_{kN}) + u_k, \tag{2.32}$$

where $u_k := \int_{t_{kN}}^{t_{kN+N}} e^{A_+(t_{kN+N}-t_{kN}-\tau)}B_+u(\tau)d\tau$. Therefore

$$\mathcal{X}_+(t) = e^{A_+(t-t_{kN})}\mathcal{X}_+(t_{kN}) + u_k$$

for $t \in \big(t_{kN}, t_{kN+N}\big)$. The volume $\nu^-(t_{kN+N})$ is then given by

$$\nu^-(t_{kN+N}) = \int_{x\in e^{A_+(t_{kN+N}-t_{kN})}\mathcal{X}_+(t_{kN})+u_k} dx. \tag{2.33}$$

Next we define $z := e^{A_+(t_{kN+N}-t_{kN})}x + u_k$ and apply the integral substitution formula

$$\int_{\varphi(U)} g(x)\,dx = \int_U g(\varphi(z))\,|\det(\mathrm{D}\,\varphi)(z)|\;dz \tag{2.34}$$

with the values $U := \mathcal{X}_+(t_{kN})$, $g(x) := 1$, $\varphi(x) := e^{A_+(t_{kN+N}-t_{kN})}x + u_k$, for which $\mathrm{D}\,\varphi = e^{A_+(t_{kN+N}-t_{kN})}$. This yields

$$\nu^-(t_{kN+N}) = \int_{x\in\mathcal{X}_+(t_{kN})} |\det e^{A_+(t_{kN+N}-t_{kN})}|\,dx \tag{2.35}$$

$$= |\det e^{A_+(t_{kN+N}-t_{kN})}|\;\nu^+(t_{kN}). \tag{2.36}$$

Using the fact that $\det e^M = e^{\operatorname{trace} M} = e^{\sum_{i=1}^n \lambda_i[M]}$ for any $n \times n$ matrix M with eigenvalues $\lambda_1[M], \lambda_2[M], ..., \lambda_n[M]$, we conclude that

$$\begin{aligned} \nu^-(t_{kN+N}) &= e^{(t_{kN+N}-t_{kN})\sum_{i=1}^{n_+} \lambda_i[A_+]} \mu(\mathcal{X}_+(t_{kN})) \\ &= e^{(t_{kN+N}-t_{kN})\sum_{i:\mathbb{R}\lambda_i[A]>0} \lambda_i[A]} \nu^+(t_{kN}), \end{aligned} \tag{2.37}$$

where the second equality follows from the fact that the eigenvalues of A_+ are precisely the eigenvalues of A with positive real part. Equation (2.37) establishes the rate of expansion of the uncertainty region between codewords.

Next we characterize how much the uncertainty region shrinks upon the receipt of a codeword.

Let $\mathcal{C} \subset \{0, \ldots, S\}^N$ denote the set of N-length codewords with M or fewer non-free symbols, and note that $|\mathcal{C}| = L(N, M, S)$. Consider $\nu^-(t_{kN})$, the volume of the uncertainty region immediately before a codeword is received at time t_{kN}. Depending on precisely which codeword is received, the volume of the uncertainty region may shrink. To capture this, for each codeword $c \in \mathcal{C}$ let $\nu^+(t_{kN}|c)$ denote the volume of the uncertainty region at time t_{kN} supposing that codeword c is received at that time.

Since for every point x' in the pre-codeword uncertainty region there must exist at least one codeword for which x' is in the post-codeword uncertainty region, we

must have

$$\nu^-(t_{kN}) \leqslant \sum_{c\in\mathcal{C}} \nu^+(t_{kN}|c) \tag{2.38}$$

$$\leqslant |\mathcal{C}| \max_{c\in\mathcal{C}} \nu^+(t_{kN}|c), \tag{2.39}$$

and so there must exist a codeword $c^* \coloneqq \arg\max_{c\in\mathcal{C}} \nu^+(t_{kN}|c)$ for which

$$\begin{aligned}\nu^+(t_{kN}|c^*) &\geqslant \frac{1}{|\mathcal{C}|}\,\nu^-(t_{kN}) \\ &= \frac{1}{L(N,M,S)}\,\nu^-(t_{kN}).\end{aligned}$$

Provided that $\nu^-(t_{kN}) > 0$, there exists a set of initial conditions for which the closed-loop solution results in codeword c^* being transmitted at time t_{kN} and therefore

$$\nu^+(t_{kN}) \geqslant \frac{1}{L(N,M,S)}\nu^-(t_{kN}). \tag{2.40}$$

Thus, there exist initial conditions for which, at time t_{kN}, the post-codeword uncertainty region is at least $1/L(N,M,S)$ times as big as the pre-codeword uncertainty region. In other words, for certain initial conditions, at time t_{kN} the scheme cannot reduce the uncertainty volume by more than a factor of $1/L(N,M,S)$. Iterating (2.37) and (2.40) from time 0 to t_{kN} for arbitrary $k \in \mathbb{Z}_{>0}$, we conclude

that for appropriately selected initial conditions, we will have

$$\nu^+(t_{kN}) \geqslant \frac{1}{L(N,M,S)^k} e^{t_{kN}\sum_{i:\mathbb{R}\lambda_i[A]>0}\lambda_i[A]}\nu(0) \tag{2.41}$$

$$= e^{t_{kN}\sum_{i:\mathbb{R}\lambda_i[A]>0}\lambda_i[A]-k\ln L(N,M,S)}\nu(0) \tag{2.42}$$

Next, let us consider the consequences of our limited bit-rate $r < r_{\min}$. Define $\delta \coloneqq r_{\min} - r$ so that $r_{\min} - \delta/2 > r$. Using the definition of r from (2.2) we find that

$$\frac{r_{\min} - \delta/2}{\log_2(S+1)} > \limsup_{k\to\infty} \frac{k}{t_k}, \tag{2.43}$$

meaning that $(r_{\min} - \delta/2)/\log_2(S+1)$ is an eventual upper bound of the sequence $\{k/t_k\}$, and therefore also of the sequence $\{kN/t_{kN}\}$. This means that for any $\epsilon > 0$, there exists $K \in \mathbb{Z}_{>0}$ such that for all $k > K$ we have

$$\frac{kN}{t_{kN}} < \frac{r_{\min} - \delta/2}{\log_2(S+1)} + \epsilon. \tag{2.44}$$

In particular, pick $\epsilon = \delta/(4\log_2(S+1))$ in (2.44). Using the definition of $r_{\min}$ from (2.29) and straightforward algebraic manipulations yields

$$t_{kN}\left(\frac{\delta \ln L(N,M,S)}{4N\log_2(S+1)}\right) < t_{kN}\sum_{i:\mathbb{R}\lambda_i[A]>0}\lambda_i[A] - k\ln L(N,M,S) \qquad \forall k > K. \tag{2.45}$$

The left-hand side is unbounded because the sequence $\{t_k\}$ is unbounded. Hence, we conclude that

$$\lim_{k\to\infty}\left(t_{kN}\sum_{i:\mathbb{R}\lambda_i[A]>0}\lambda_i[A]-k\ln L(N,M,S)\right)=\infty. \tag{2.46}$$

Note that this is the exponent in (2.42), which means that the volume of sets $\{\mathcal{X}_+(t)\}$ grows to infinity as $t\to\infty$, which in turn means that we can find values for the state in these sets arbitrarily far apart for sufficiently large t and thus arbitrarily far from the origin. We thus conclude that the controller and encoder/decoder pair cannot stabilize the process. ■

Now we are ready to prove Theorem 1.

Proof of Theorem 1. If $\gamma=0$, then the encoder transmits at most N_0 non-free symbols, and therefore cannot bound an unstable system for all time. We assumed that the encoding scheme keeps the state of process (2.1) bounded, so we must have $\sum_{i:\mathbb{R}\lambda_i[A]>0}\lambda_i[A]=0$, and so (2.5) is satisfied trivially. Now suppose $\gamma>0$. By Lemma 1, for any $\epsilon>0$ there exist $M\in\mathbb{R}_{\geqslant 0}$ and $N\in\mathbb{Z}_{>0}$ with $M<N\gamma(1+\epsilon)$ for which the encoder/decoder is an M-of-N encoder. Since the state of the process is kept bounded, by Lemma 3 we have

$$\sum_{i:\mathbb{R}\lambda_i[A]>0}\lambda_i[A]\leqslant r\,\frac{\ln L(N,M,S)}{N\ln(S+1)}\,\ln 2. \tag{2.47}$$

Since L is monotonically nondecreasing in its second argument and $M < N\gamma(1+\epsilon)$, we have

$$r \, \frac{\ln L(N, M, S)}{N \ln(S+1)} \leqslant r \, \frac{\ln L(N, N\gamma(1+\epsilon), S)}{N \ln(S+1)}. \tag{2.48}$$

Lemma 2 implies that

$$r \, \frac{\ln L(N, N\gamma(1+\epsilon), S)}{N \ln(S+1)} \leqslant r f(\gamma(1+\epsilon), S). \tag{2.49}$$

Combining these and letting $\epsilon \to 0$, we obtain (2.5). This completes the proof of Theorem 1. ■

2.3 Sufficient condition for stability with limited-communication encoders

The previous section established a necessary condition (2.5) on the average bit-rate and average cost per symbol of an encoder/decoder pair in order to bound process (2.1). In this section, we show that with a strict inequality this condition is also sufficient for a stabilizing encoder/decoder to exist. The proof is constructive in that we provide the encoder/decoder.

The proposed scheme is sometimes called *emulation-based* because the encoder/decoder emulates a stabilizing state-feedback controller $u = Kx$. This

state-feedback controller cannot be used in the limited-communication environment considered in this chapter because the infinite-precision state $x(t) \in \mathbb{R}^n$ cannot be sent over the channel and hence is unavailable to the controller. Instead, in emulation-based control, the state-feedback controller is coupled to an encoder/decoder pair that estimates the state as $\hat{x}(t)$, resulting in the control law $u(t) = K\hat{x}(t),\ t \geqslant 0$.

Theorem 2. *Assume that $A + BK$ is Hurwitz. For every $S \in \mathbb{Z}_{\geqslant 0}$, $r \geqslant 0$, and $\gamma \in [0, 1]$ satisfying*

$$rf(\gamma, S) \ln 2 > \sum_{i:\mathbb{R}\lambda_i[A]>0} \lambda_i[A], \tag{2.50}$$

where the function f is defined in (2.6)*, there exists an emulation-based controller and an M-of-N encoder/decoder pair that uses S non-free symbols, has average bit-rate not exceeding r, has an average cost per symbol not exceeding γ, and exponentially stabilizes process* (2.1) *for every initial condition $x_0 \in \mathcal{X}_0$.*

Remark 2. The encoding scheme that follows relies on a strict inequality in (2.50) for the existence of a suitable M-of-N encoder, and as that gap shrinks to 0, the codeword length N becomes unbounded. In contrast, we will see that the event-based encoding scheme presented in Chapter 3 has the property that if its corresponding data-rate condition (3.4) holds with equality, the scheme bounds the state of the process, *cf.* Remark 6.

The proof of Theorem 2 uses the following lemma, proved in the appendix, which establishes a useful coordinate transformation for the error system of an emulation-based controller.

Lemma 4. *Consider the process and the (open-loop) state estimator*

$$\dot{x}(t) = Ax(t) + Bu(t), \quad x(t_k) = x_0 \quad \forall t \in [t_k, t_{k+1}) \tag{2.51}$$

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t), \quad \hat{x}(t_k) = \hat{x}_0 \quad \forall t \in [t_k, t_{k+1}). \tag{2.52}$$

There exists a time-varying matrix $P(t) \in \mathbb{R}^{n\times n}$ such that for any $t_k, t_{k+1}, x_0, \hat{x}_0$, the state estimation error

$$e(t) \coloneqq P(t)(x(t) - \hat{x}(t)) \tag{2.53}$$

satisfies

$$e_i(t) = e^{a_i(t-t_k)} G_i(t - t_k) e_i(t_k), \quad e_i(t) \in \mathbb{R}^{d_i}, \tag{2.54}$$

for all $t \in [t_k, t_{k+1})$ and all $i \in \{1, \ldots, n_b\}$, where n_b is the number of real Jordan blocks in the real Jordan normal form of A, a_i is the real part of the eigenvalue associated with Jordan block i, and d_i is the geometric multiplicity of that eigen-

value; the time-varying real matrix $G_i(t)$ has the form

$$G_i(t) \coloneqq \begin{bmatrix} 1 & t & \frac{t^2}{2!} & \dots & \frac{t^{d_i-1}}{(d_i-1)!} \\ & 1 & t & & \\ & & \ddots & & \\ & & & & \\ & & & & 1 \end{bmatrix} \in \mathbb{R}^{d_i \times d_i} \tag{2.55}$$

if the ith Jordan block corresponds to a real eigenvalue, and

$$G_i(t) \coloneqq \begin{bmatrix} I_2 & I_2 t & I_2 \frac{t^2}{2!} & \dots & I_2 \frac{t^{d_i-1}}{(d_i-1)!} \\ & I_2 & I_2 t & & \\ & & \ddots & & \\ & & & & \\ & & & & I_2 \end{bmatrix} \in \mathbb{R}^{2d_i \times 2d_i} \tag{2.56}$$

if it corresponds to a complex conjugate pair, where $I_2 \coloneqq \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. *Moreover, there exists a positive scalar ϵ_P for which*

$$\sigma_{\min}(P(t)) \geqslant \epsilon_P \qquad \forall t \geqslant 0, \tag{2.57}$$

where $\sigma_{\min}(\cdot)$ denotes the smallest singular value of a matrix.

2.3.1 Proof of Theorem 2

The basic idea of the proof is as follows. The encoder and decoder each run internal copies of the process to compute an estimate $\hat{x}$ of the state. Since there is no channel noise, the encoder's and decoder's state estimates will be equal, which corresponds to an information pattern "encoder class 1a" in the terminology of [41].

The encoder monitors the state estimation error and periodically transmits symbols to the decoder that essentially encode a quantized version of the error, making sure that the average cost per symbol does not exceed γ. The decoder then uses those symbols to update its state estimate $\hat{x}$.

Definition of the encoding and decoding scheme

We first select the integers M and N for our M-of-N encoder. Assume that S, r, and γ satisfy (2.50), so that

$$\eta \coloneqq rf(\gamma, S) \ln 2 - \sum_{i:\mathbb{R}\lambda_i[A]>0} \lambda_i[A] > 0. \tag{2.58}$$

In view of (2.10) and (2.11), we conclude that we can pick N sufficiently large to satisfy

$$rf(\gamma, S) \ln 2 - r \, \frac{\ln L(N, N\gamma, S)}{N \ln(S+1)} \ln 2 < \eta/2, \tag{2.59}$$

and we then define $M \coloneqq N\gamma$. By Lemma 5 in the appendix, this encoder has an average cost per symbol not exceeding γ.

Now we specify which N-length codewords will be transmitted. Here is the basic idea: The encoder and decoder each estimate the state of the process as $\hat{x}(t)$ as defined in (2.52), with $t_0 \coloneqq 0$ and $\hat{x}(t_0) \coloneqq 0$. The encoder monitors the state estimation error $e(t) \coloneqq P(t)(x(t) - \hat{x}(t))$, where $P(t)$ is determined by Lemma 4. For each of the n_b error subsystems $e_i(t) \in \mathbb{R}^{d_i}$ given by (2.54) we employ a *sub-encoder* i that monitors $e_i(t)$ and every T_i time units (to be defined shortly) transmits to the decoder a set of N-length codewords with M or fewer non-free symbols from the alphabet $\{0, \ldots, S\}$. The chosen set of codewords is essentially the index of the d_i-dimensional quantization cell in which $e_i(kT_i) \in \mathbb{R}^{d_i}$ lies. Based on this set of codewords, the encoder and decoder each adjusts their state estimates, and the procedure repeats.

We now define the scheme formally. We first select the transmission periods T_i: partition the n_b error systems based on whether or not they are stable:

$$\mathcal{S} \coloneqq \{i \in \{1, \ldots, n_b\} : a_i < 0\}$$

$$\mathcal{U} \coloneqq \{i \in \{1, \ldots, n_b\} : a_i \geqslant 0\},$$

where a_i is the real part of the ith eigenvalue of A. For the subsequent argument, in the case that $a_i = 0$ we add a small positive number to it so that (2.50) still

holds, and use the same label a_i to denote this number. Note that, in contrast with the previous section, we treat eigenvalues with zero real part as unstable.

The error dynamics for e_i with $i \in \mathcal{S}$ are stable and so there is no need to transmit information on behalf of e_i, $i \in \mathcal{S}$, since these errors will converge to zero exponentially fast. So there is no need to define T_i for $i \in \mathcal{S}$. For $i \in \mathcal{U}$, we select the transmission period for sub-encoder i to be

$$T_i \coloneqq c_i \frac{\ln L(N, M, S)}{a_i} \frac{1}{1 + \eta/(2 \sum_{i:\mathbb{R}\lambda_i[A]>0} \lambda_i[A])}, \tag{2.60}$$

where the positive integer c_i is chosen large enough so that T_i satisfies

$$\sum_{j=0}^{d_j-1} \frac{T_i^j}{j!} < e^{\kappa T_i} \tag{2.61}$$

where $\kappa \coloneqq a_i \eta/(4 \sum_{i:\mathbb{R}\lambda_i[A]>0} \lambda_i[A]) > 0$.

Note that for those eigenvalues whose real part was 0, the transmission period can be arbitrarily large (but finite) because the positive number that was added to them can be arbitrarily small.

Now we specify how the sub-encoder i selects which codeword to transmit. For $i \in \mathcal{S}$ no symbols are transmitted. For $i \in \mathcal{U}$, the ith sub-encoder initializes with $L_{i,0} \coloneqq \sup_{x_0 \in \mathcal{X}_0} \|x_0\|_\infty$ and at time kT_i, $k \in \mathbb{Z}_{>0}$, performs the following steps:

1. Divide the d_i-dimensional box $e^{(a_i+\kappa)T_i} L_{i,k-1}[-1, 1]^{d_i}$ into $L(N, M, S)^{c_i d_i}$

smaller boxes of equal size by dividing each of its d_i dimensions into $L(N, M, S)^{c_i}$ intervals of equal length. The sub-encoder i determines in which of these boxes the error $e_i(kT_i)^-$ lies and transmits this information to the decoder. Since there are $L(N, M, S)^{c_i d_i}$ boxes, this requires sending exactly $c_i d_i$ M-of-N codewords.

Let $B_{i,k} \subset \mathbb{R}^{d_i}$ denote the indicated box, $b_{i,k} \in \mathbb{R}^{d_i}$ denote the box's center, and $w_{i,k}$ denote the transmitted set of codewords. Note that set $B_{i,k} - b_{i,k} \subset \mathbb{R}^{d_i}$ is a cube centered at 0.

2. Update the state estimate as

$$\hat{x}(kT_i)^+ = \hat{x}(kT_i)^- + I_i' b_{i,k}, \tag{2.62}$$

where $\hat{x}^+(t) \coloneqq \lim_{\tau \downarrow t} \hat{x}(\tau)$ and $\hat{x}^-(t) \coloneqq \lim_{\tau \uparrow t} \hat{x}(\tau)$, and the matrix $I_i \in \mathbb{R}^{d_i \times n}$ "extracts" from the error $e(t)$ its component $e_i(t)$ such that $e_i(t) = I_i e(t)$. Specifically,

$$I_i \coloneqq \begin{bmatrix} 0_{d_i \times d_1} & 0_{d_i \times d_2} & \dots & I_{d_i \times d_i} & \dots & 0_{d_i \times d_{n_b}} \end{bmatrix}.$$

3. Define

$$L_{i,k} \coloneqq \sup_{z \in B_{i,k} - b_{i,k}} \|z\|_\infty \tag{2.63}$$

The sequences $\{w_{i,k}\}$, $\{B_{i,k}\}$, $\{b_{i,k}\}$, and $\{L_{i,k}\}$ are available both to the encoder and the decoder, so the decoder can maintain and update its own state estimate via Step 2, which is used by the state feedback controller $u \coloneqq K\hat{x}$. We now show that the proposed encoding/decoding scheme satisfies the conditions of Theorem 2, namely that the state goes to 0 and that the average bit-rate is at most r.

The scheme exponentially stabilizes the process

From process (2.1) and the definition of $e(t)$ in (2.53), the control law $u = K\hat{x}$ results in the following closed-loop dynamics:

$$\dot{x}(t) = (A + BK)x(t) - BKe(t). \tag{2.64}$$

Since $A + BK$ is Hurwitz, the state $x(t)$ converges exponentially to 0 provided that $e(t) \to 0$ exponentially. We now prove that $e(t) \to 0$ exponentially under the proposed scheme.

The basic idea is as follows: On one hand, in view of (2.54) and (2.61), the error $e_i(t)$ grows in magnitude by a factor less than $e^{(a_i+\kappa)T_i}$ in the T_i time units between the transmission of sets of codewords. On the other hand, every T_i time units the ith sub-encoder sends $L(N, M, S)^{c_i d_i}$ codewords, allowing the ith sub-decoder to reduce its uncertainty of $e_i(t)$ by a factor of $L(N, M, S)^{c_i d_i}$. We will show that condition (2.50) in Theorem 2 implies that $L(N, M, S)^{c_i d_i} > e^{(a_i+\kappa)T_i}$,

meaning that the sub-decoder's uncertainty in $e_i(t)$ shrinks faster than the error dynamics expands $e_i(t)$. Therefore the decoder can determine $e(t)$ and drive it to 0.

First we prove by induction that the rule (2.62) for updating the state estimate guarantees that $\|e_i(kT_i)^+\|_\infty \leqslant L_{i,k}$. From the definition of $e(t)$ and I_i we have

$$e_i(kT_i)^- = I_i e(kT_i)^- = I_i \left(x(kT_i)^- - \hat{x}(kT_i)^- \right). \tag{2.65}$$

Solving the update rule (2.62) for $\hat{x}(kT_i)^-$ and substituting the result into (2.65) yields

$$\begin{aligned} e_i(kT_i)^- &= I_i \left(x(kT_i)^- - \left(\hat{x}(kT_i)^+ - \ I_i' b_{i,k} \ \right) \right) \\ &= e_i(kT_i)^+ + b_{i,k}, \end{aligned} \tag{2.66}$$

where we used the fact that $x(kT_i)^- = x(kT_i)^+$ due to the continuity of the solution $x(t)$. Next, suppose by the induction hypothesis that $\|e_i((k-1)T_i)^+\|_\infty \leqslant L_{i,k-1}$. Then we have

$$\|e_i((k-1)T_i)^+\|_\infty \leqslant L_{i,k-1} \tag{2.67}$$

$$\Leftrightarrow \qquad e_i((k-1)T_i)^+ \in L_{i,k-1}[-1,1]^{d_i} \tag{2.68}$$

$$\Rightarrow \qquad e_i(kT_i)^- \in e^{a_i T_i} \|G_i(T_i)\|_\infty L_{i,k-1}[-1,1]^{d_i} \tag{2.69}$$

$$\Rightarrow \qquad e_i(kT_i)^- \in e^{(a_i+\kappa)T_i} L_{i,k-1}[-1,1]^{d_i}, \tag{2.70}$$

where (2.69) holds because $e_i(t)$ follows the dynamics (2.54) between transmissions, and (2.70) follows because T_i was chosen to satisfy (2.61) and we have $\|G_i(T_i)\|_\infty = \sum_{j=0}^{d_j-1} T_i^j/j!$.

Moreover, the set in (2.70) is precisely the box in Step 1 of the proposed scheme, so therefore we must have $e_i(kT_i)^- \in B_{i,k}$. Applying (2.66) yields $e_i(kT_i)^+ \in B_{i,k} - b_{i,k}$, and therefore

$$\|e_i(kT_i)^+\|_\infty \leqslant \sup_{z \in B_{i,k}-b_{i,k}} \|z\|_\infty =: L_{i,k}. \tag{2.71}$$

This demonstrates that $\|e_i(kT_i)^+\|_\infty \leqslant L_{i,k}$ for all $k \in \mathbb{Z}_{>0}$.

From Step 1 of the encoding scheme, the length $L_{i,k}$ is essentially the side-length of the cube $B_{i,k}$. The set $B_{i,k}$ was constructed by dividing every dimension of $e^{(a_i+\kappa)T_i} L_{i,k-1}[-1,1]^{d_i}$ into $L(N,M,S)^{c_i}$ pieces. Therefore the lengths $L_{i,k}$ are recursively related via

$$L_{i,k} = \frac{e^{(a_i+\kappa)T_i}}{L(N,M,S)^{c_i}} L_{i,k-1}, \tag{2.72}$$

and therefore

$$L_{i,k} = e^{Rk} L_{i,0}, \tag{2.73}$$

where

$$R \coloneqq \ln\Big(\frac{e^{(a_i+\kappa)T_i}}{L(N,M,S)^{c_i}}\Big). \tag{2.74}$$

The transmission period T_i and κ were chosen in (2.60) to satisfy

$$\frac{e^{(a_i+\kappa)T_i}}{L(N,M,S)^{c_i}} < 1, \tag{2.75}$$

and so $R < 0$. Therefore the event boundaries $L_{i,k}$ shrink to 0 at an exponential rate.

This implies that $e_i(t) \to 0$ exponentially, as follows. For any time t we have $t = kT_i + \underline{t}$, where $k \coloneqq \lfloor t/T_i \rfloor$ and $\underline{t} \in [0, T_i)$. Therefore

$$\begin{aligned}
\|e_i(t)\|_\infty &= \|e_i(kT_i + \underline{t})\|_\infty && (2.76)\\
&\leqslant e^{a_i \underline{t}} \|G_i(\underline{t})\|_\infty \|e_i(kT_i)^+\|_\infty && (2.77)\\
&\leqslant e^{a_i T_i} \|G_i(T_i)\|_\infty \|e_i(kT_i)^+\|_\infty && (2.78)\\
&\leqslant e^{a_i T_i} \|G_i(T_i)\|_\infty L_{i,k} && (2.79)\\
&= e^{a_i T_i} \|G_i(T_i)\|_\infty L_{i,0} e^{Rk} && (2.80)\\
&\leqslant e^{a_i T_i} \|G_i(T_i)\|_\infty L_{i,0} e^{-R} e^{Rt/T_i}, && (2.81)
\end{aligned}$$

where (2.77) follows from the error dynamics (2.54), (2.79) follows from (2.71), and (2.80) follows from (2.73). Since $R < 0$, this establishes that $e_i(t) \to 0$ at an

exponential rate.

Since this holds for all i, $e(t)$ exponentially converges to 0 as well. Therefore by (2.64), the state $x(t)$ exponentially converges to 0.

The scheme's average bit-rate does not exceed r

Since each sub-encoder is transmitting independently, the average bit-rate of this encoding scheme as a whole is simply the sum of the sub-encoder's average bit-rates. For $i \in \mathcal{S}$, the ith sub-encoder never transmits. For $i \in \mathcal{U}$, every T_i time units the ith sub-encoder sends $c_i d_i$ codewords, each from a codeword library of length $L(N, M, S)$. Therefore its average bit-rate is $r_i \coloneqq c_i d_i \log_2 L(N, M, S)/T_i$. The encoder's total average bit-rate is therefore

$$\sum_{i \in \mathcal{U}} r_i = \log_2 L(N, M, S) \sum_{i \in \mathcal{U}} \frac{c_i d_i}{T_i}.$$

Leveraging (2.60) yields

$$\sum_{i \in \mathcal{U}} r_i \leqslant \frac{1}{\ln 2} \left(1 + \frac{\eta}{2 \sum_{i : \mathbb{R}\lambda_i[A] > 0} \lambda_i[A]} \right) \sum_{i \in \mathcal{U}} d_i a_i. \tag{2.82}$$

Since $\mathcal{U}$ contains the non-negative real parts of the eigenvalues of A, we have

$$\sum_{i \in \mathcal{U}} a_i = \sum_{i : \mathbb{R}\lambda_i[A] > 0} \lambda_i[A].$$

From this and (2.82) we conclude that

$$\begin{aligned}\sum_{i\in\mathcal{U}} r_i &\leqslant \frac{1}{\ln 2}\left(\sum_{i:\mathbb{R}\lambda_i[A]>0}\lambda_i[A]+\frac{\eta}{2}\right)\\ &< r\frac{\ln L(N,M,S)}{N\ln(S+1)} \leqslant r,\end{aligned} \tag{2.83}$$

where in (2.83) we leveraged (2.58) and (2.59) and then used the fact that L is nonincreasing in its second argument and so $L(N,M,S) \leqslant L(N,N,S) = (S+1)^N$. We conclude that this encoding scheme has average bit-rate less than r.

This concludes the proof of Theorem 2. ■

An unexpected consequence of Theorems 1 and 2 is that when it is possible to drive the state of process (2.1) to 0 with a given average bit-rate r, one can always find M-of-N encoders that stabilize it for (essentially) the same average bit-rate and average cost per symbol not exceeding $S/(S+1)$, i.e., approximately a fraction $1/(S+1)$ of the symbols will not consume communication resources. In the most advantageous case, the encoder/decoder use the alphabet $\{0,1\}$ and the encoder's symbol stream consumes no more than 50% of the communication resources.

The following summarizes this observation.

Corollary 1. *If process* (2.1) *can be bounded with an encoder/decoder pair with average bit-rate* r*, then for any* $\epsilon > 0$ *and* $S \in \mathbb{Z}_{>0}$ *there exists an* M*-of-*N *encoder using alphabet* $\{0,\ldots,S\}$ *with average bit-rate* $r+\epsilon$ *and average cost per symbol*

not exceeding $S/(S+1)$ that bounds its state.

Proof of Corollary 1. Since the original encoder/decoder pair bounds the state, then by (2.4) we have

$$\begin{aligned}\sum_{i:\mathbb{R}\lambda_i[A]>0} \lambda_i[A] &\leqslant r \ln 2 < (r+\epsilon)\ln 2 \\ &= (r+\epsilon) f\left(\frac{S}{S+1}, S\right) \ln 2.\end{aligned}$$

Applying Theorem 2 completes the proof. ■

The price paid for using an encoder/decoder with average cost per symbol close to $S/(S+1)$ is that it may require prohibitively long codewords (large N) as compared to an encoder with higher average cost per symbol. To see this, note that $f(\gamma, S) = 1$ when $\gamma \in [S/(S+1), 1]$ and recall that $\ln L(N, N\gamma, S)/N$ is monotonically nondecreasing in γ and N. Hence, with r and S fixed, one can decrease γ from 1 toward $S/(S+1)$ and still satisfy (2.59) by increasing N. This can be seen in Figure 2.4.

Remark 3. In the problem statement, $x(0)$ was assumed to belong to a known bounded set. If the region $\mathcal{X}_0$ is not precisely known, the proposed scheme could be modified by introducing an initial "zooming-out" stage as described in [5], where the encoder picks an arbitrary box to quantize and successively zooms out at a super-linear rate until the box captures the state.

2.4 Numerical example

In this subsection we present a numerical example of the M-of-N encoding scheme presented in this chapter.

Consider process (2.1) with

$$A \coloneqq \begin{bmatrix} 57 & -25 \\ 125 & -53 \end{bmatrix} \quad B \coloneqq \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad K \coloneqq \begin{bmatrix} -7 \\ 3.784 \end{bmatrix}, \tag{2.84}$$

for which $\lambda[A] = 2 \pm 10i$ and K is the state-feedback gain of a stabilizing emulation-based controller. Suppose the initial condition is known to lie in the box $\mathcal{X}_0 \coloneqq \{(x_1, x_2) : -1 \leqslant x_i \leqslant 2\}$, and that $x(0) \coloneqq (1, -1)$. Using the coordinate transformation from Lemma 4 yields the open-loop error system $\dot{e}_i(t) = 2e_i(t)$ for $i \in \{1, 2\}$. Note that although the two error components grow at the same rate, their initial conditions are different: $e_1(0) = -3$, $e_2(0) = 2$.

With average bit-rate $r \coloneqq 10$, average cost per symbol $\gamma = 0.2$, and alphabet $\mathcal{A} \coloneqq \{0, 1\}$, the sufficient bound (2.50) is satisfied. Following the encoder design in Subsection 2.3.1, we pick $N \coloneqq 10$, $M \coloneqq 2$, and $T_i = 1.9$ for $i \in \{1, 2\}$. There are $L(N, M, S) = 56$ length-10 codewords with 2 or fewer non-free symbols. In accordance with the encoder design in Subsection 2.3.1, at time kT_i, $k \in \mathbb{Z}_{>0}$, sub-encoder i measures the scalar $e_i(kT_i)$, quantizes it into one of 56 bins — one per codeword — and transmits the appropriate 10-symbol codeword to the

decoder. The two sub-encoders each transmit up to 2 non-free symbols every 1.9 time units, resulting in a total average rate of resource consumption of 2.1 non-free transmissions per time unit. Then the encoder and decoder each update their state estimate according to (2.62). One observes the state $x(t)$ of the closed-loop system converging to 0.

Figure 2.5 demonstrates this, and Figure 2.6 shows the state component $x_1(t)$ converging to 0.

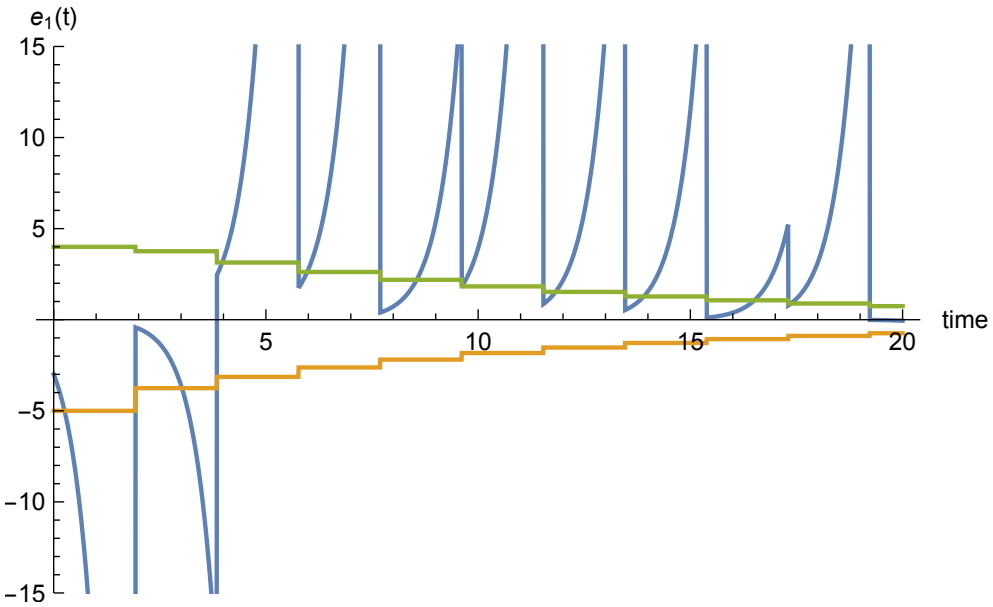

Figure 2.5: Plot of the closed-loop state estimation error component $e_1(t)$ (blue) and the endpoints $\alpha_{1,k}$ (orange) and $\beta_{1,k}$ (green) of the bounding sub-intervals drawn as continuous lines for ease of viewing. At transmission time kT_1 $k \in \mathbb{Z}_{>0}$, the decoder receives a codeword and adjusts the error to be within $[\alpha_{1,k}, \beta_{1,k}]$.

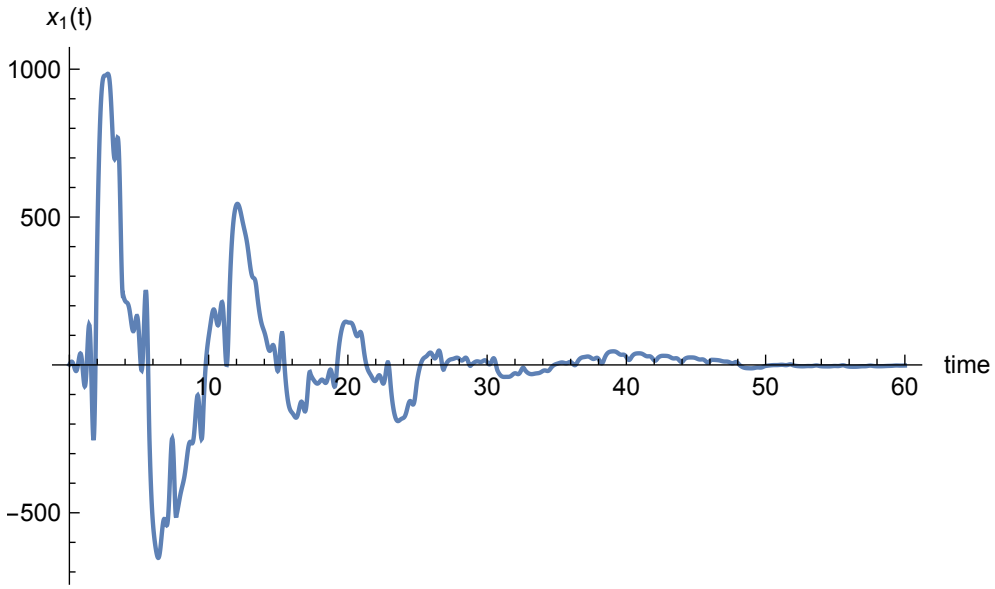

Figure 2.6: Plot of the closed-loop state $x_1(t)$ exponentially decaying to 0 using the encoding scheme described in Subsection 2.3.1.

2.5 Conclusion

In this chapter, we considered the problem of bounding the state of a continuous-time linear process under communication constraints. We considered constraints on both the channel average bit-rate and the encoding scheme's average cost per symbol. Our main contribution was a necessary and sufficient condition on the process and constraints for which a bounding encoder/decoder/controller exists. In the absence of a limit on the average cost per symbol, the conditions recovered previous work. A surprising corollary to our main result was the observation that one may impose a constraint on the average cost per symbol without necessarily needing to loosen the average bit-rate constraint. Specifically, we proved that if a process may be bounded with a particular average bit-rate, then there exists

a (possibly very complex) encoder/decoder that can bound it with that same average bit-rate, while using no more than 50% non-free symbols on average. One would expect that the prohibition of some codewords would require that the encoder necessarily compensate by transmitting at a higher average bit-rate, but this not the case.

Another surprising result was the observation that, for any constraint on average bit-rate and average cost per symbol satisfying the necessary and sufficient conditions for stability, one can always construct a stabilizing encoder with an arbitrarily small average cost per time unit. In many communication-constrained control problems this is the quantity of interest. We observed that constructing such an encoder boils down to either having precisely-synchronized clocks between the encoder and decoder, or storing a large symbol library on the encoder and decoder.

Chapter 3

Quasi-optimality of Event-based control

Parts of this chapter come from [30]:

In the last chapter we constructed an N-of-M encoding scheme that stabilizes process (2.1) provided that the bit-rate and average cost condition (2.50) holds. This scheme may be difficult to implement in practice if the encoder/decoder pair use a large number of codewords. In this section we present an *event-based*

encoding scheme that is easy to implement and does not require storing a large set of codewords. Instead, it uses a library of only three symbols $\{-1, 0, 1\}$ and does not group them into codewords. The basic idea is to monitor in parallel each one-dimensional component of the error system, and as long as it stays inside a fixed interval, send the free symbol 0. A non-free symbol is sent only when the one-dimensional component of the error leaves the interval: send -1 if the error exited the left side of the interval and send 1 if it exited out the right side. Communication resources are therefore consumed only upon the occurrence of this event, justifying the label *event-based*. The proposed scheme resembles the distributed-sensor scheme of [41], in that each coordinate of a plant measurement is sent by a dedicated encoder to a central decoder.

The proposed scheme has similarities with the one from Section 2.3 in the following ways: the encoder and decoder each estimate the process as $\hat{x}$ using (2.52); the emulation-based controller is $u \coloneqq K\hat{x}$, where K is a stabilizing state-feedback gain; Lemma 4 decouples the error system into n_b sub-systems; each of n_b sub-encoders monitors the d_i-dimensional component of the error and transmits a block of symbols every T_i time units; only the unstable systems $\mathcal{U}$ require transmission. If A is diagonalizable over $\mathbb{C}$, then this event-based encoding scheme reduces to the one proposed in [29].

3.1 Definition of the event-based scheme

Unlike the scheme from Section 2.3, this scheme differs in what symbols are sent and how the state estimate $\hat{x}$ is updated: For $i \in \mathcal{U}$, at time kT_i, $k \in \mathbb{Z}_{>0}$ (with T_i to be determined shortly), the sub-encoder i monitors the d_i scalar components $e_{i,j}(t) \in \mathbb{R}$, $j \in \{1, \ldots, d_i\}$ of $e_i(t)$, and for each one sends a symbol $s_{i,j}(k) \in \{-1, 0, 1\}$ according to

$$s_{i,j}(k) = \begin{cases} -1 & e_{i,j}(kT_i) < -L_j \\ 0 & e_{i,j}(kT_i) \in [-L_j, L_j] \\ 1 & e_{i,j}(kT_i) > L_j \end{cases} \qquad k \in \mathbb{Z}_{>0}, \tag{3.1}$$

with the *event boundaries* $L_j > 0$ also to be determined shortly. The encoder and decoder then each update their state estimates as

$$\hat{x}(kT_i)^+ = \hat{x}(kT_i)^- + P(kT_i)^{-1}\mathbf{v}_{i,j}\Delta_{i,j}(s_{i,j}(k)),$$
$$i \in \{1, \ldots n\},\ k \in \mathbb{Z}_{>0}, \tag{3.2}$$

where the unit vector $\mathbf{v}_{i,j} \in \mathbb{R}^{d_i}$ satisfies $e_{i,j}(t) = \mathbf{v}_{i,j}' e(t)$, $\hat{x}(t)^+$ and $\hat{x}(t)^-$ denote limiting values of $\hat{x}(t)$ from above and below t, $P(t)$ is from Lemma 4, and the

decoding function $\Delta_{i,j} : \{-1, 0, 1\} \to \mathbb{R}$ is defined as

$$\Delta_{i,j}(s) \coloneqq \begin{cases} -\frac{L_j}{2}(1 + \exp(a_i T_i)) & s = -1 \\ 0 & s = 0 \\ \frac{L_j}{2}(1 + \exp(a_i T_i)) & s = 1, \end{cases} \tag{3.3}$$

where $a_i \coloneqq \mathbb{R}\lambda_i[A]$ is defined as before. Note that the nonzero values of $\Delta_{i,j}$ are merely the midpoints of the intervals $[L_j, L_j \exp(a_i T_i)]$ and $[-L_j, -L_j \exp(a_i T_i)]$.

The event-based encoding/decoding scheme and controller are described in pseudo-code as Algorithms 1 and 2 below.

Algorithm 1. *(Encoder)*

Set state estimate $\hat{x}(0) \leftarrow 0$

Continuously compute state estimate $\hat{x}(t)$ *from* (2.52)

for *each sub-encoder* $i \in \mathcal{U}$ *in parallel,* ***do***

for *time* $t = kT_i$, $k \in \{1, 2, \ldots\}$ ***do***

measure state $x(t)$ *and compute* $e_i(t)$ *from* (2.53)

for *each scalar component* $e_{i,j}(t)$, $j \in \{1, \ldots, d_i\}$, ***do***

compute $s_{i,j}(k)$ *from* (3.1) *and transmit it to decoder*

update $\hat{x}(t)$ *from* (3.2)

end for

end for

end for

Algorithm 2. *(Decoder)*

Set state estimate $\hat{x}(0) \leftarrow 0$

Continuously compute state estimate $\hat{x}(t)$ *from* (2.52)

Continuously compute actuation signal $u(t) \coloneqq K\hat{x}(t)$

for *each sub-decoder* $i = 1$ *to* n *in parallel,* ***do***

for *time* $t = kT_i$, $k = 1, 2, \ldots$ ***do***

receive $s_{i,j}(k)$ *from the encoder*

update $\hat{x}(t)$ *from* (3.2)

end for

end for

This concludes the description of the event-based encoder/decoder pair, except for the precise choice of the transmission periods T_i and the event boundaries L_j.

3.2 Main result and proof

The following result states that if the average bit-rate and average cost per symbol satisfy a particular condition, then one can choose transmission periods T_i and event boundaries L_j for which this scheme obeys the communication constraints and bounds the process state.

Theorem 3. *Consider process* (2.1)*, and assume that* $A + BK$ *is Hurwitz. For every* $\gamma \in [0, 1]$ *and* $r > 0$ *satisfying*

$$r \, \frac{h^{-1}(\gamma)}{\ln 3} \, \ln 2 \geqslant \sum_{i:\mathbb{R}\lambda_i[A]>0} \lambda_i[A], \tag{3.4}$$

$$h(x) \coloneqq \frac{x}{\ln \frac{2}{e^x - 1}}, \; x \in (0, \ln 3), \; h(0) \coloneqq 0, \tag{3.5}$$

there exists an emulation-based controller and event-based encoder/decoder pair of the type described above that keeps the state of the process bounded for every initial condition in $\mathcal{X}_0$*; the encoder has average bit-rate not exceeding* r *and has average cost per symbol not exceeding* γ*.*

Remark 4. For the special case of $n = 1$ (scalar system) and $\gamma = 1$ (no power constraint), this event-based encoding scheme selects the transmission period $T \coloneqq h^{-1}(1)/\lambda = \ln 2/\lambda$ and hence bounds the state estimation error within the interval $[-2L, 2L]$.

Remark 5. Whereas the necessary and sufficient bounds from Theorems 1 and 2 had the term $f(\gamma, S)$, the event-based encoding bound in (3.4) has the term $h^{-1}(\gamma)/\ln 3$. The ratio

$$g(\gamma, S) \coloneqq \frac{f(\gamma, S)}{\frac{h^{-1}(\gamma)}{\ln 3}} \tag{3.6}$$

captures the factor by which the event-based bound exceeds the tight theoret-

ical bound developed in the previous sections. This factor is a function of the encoder's average cost per symbol γ and the alphabet size S, and is plotted in Figure 3.1 for $S = 2$ and $S = 1$. Since the event-based encoder has $S = 2$, the $g(\gamma, 2)$ curve provides a "fair" comparison between the event-based encoder and all other encoders with alphabet size $S = 2$. The $g(\gamma, 1)$ curve compares the event-based encoder with all other encoders with the smallest (most efficient) alphabet, $S = 1$. We observe:

- $g(\gamma, 1) < 2.43$ for all $\gamma \in (0, 1]$.
- $g(\gamma, 2) < 2.0$ for all $\gamma \in (0, 1]$.
- $g(1, S) = \ln 3 / \ln 2 \approx 1.58$ for all $S \in \mathbb{Z}_{>0}$.

The first point guarantees that this encoding and control scheme is never more than 2.43 times more conservative than the optimal bound established in Theorems 1 and 2. Specifically, if a given process may be bounded with a certain average bit-rate r, then there exists an average bit-rate $\widetilde{r}$ not exceeding $2.43r$ such that this event-based scheme can bound the process using average bit-rate $\widetilde{r}$. The second point establishes that this event-based scheme never requires more than twice the average bit-rate of any stabilizing N-of-M encoding scheme that, like this scheme, uses a three-symbol alphabet. The third point states that as the communication constraint relaxes ($\gamma \to 1$), this event-based encoding scheme is only 1.58 times more conservative than the optimal average bit-rate bound from

Theorems 1 and 2.

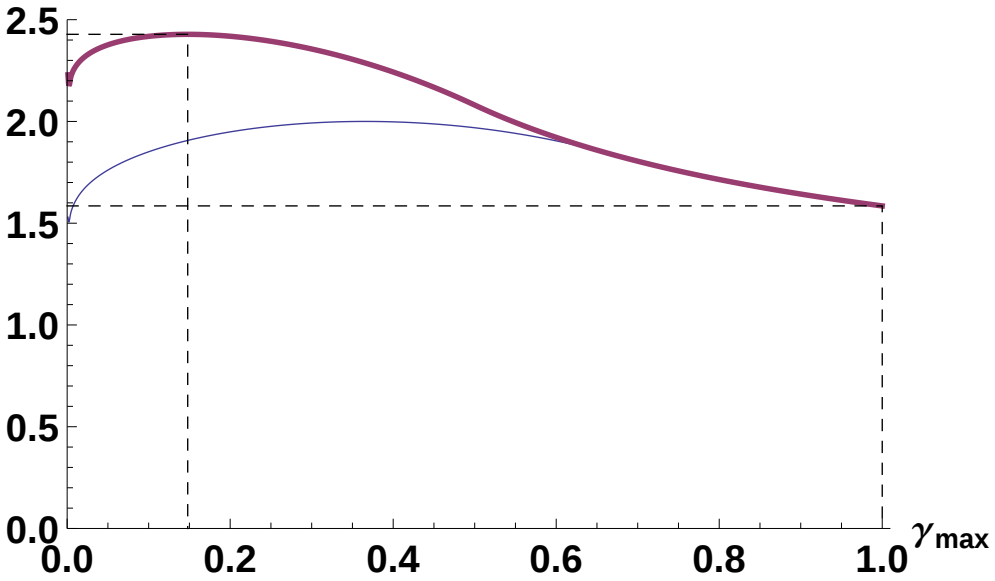

Figure 3.1: Plot of $g(\gamma, S)$ (defined in (3.6)) versus γ, for $S = 1$ (thick solid line) and $S = 2$ (thin solid line).

A consequence of $g(\gamma, S) > 1$ is that event-based encoders are sub-optimal in the following sense: if r, γ, and S satisfy (3.4), then there exists $\widetilde{r} \coloneqq r/g(\gamma, S) < r$ for which $\widetilde{r}$, γ, and S satisfy (2.50). Therefore, whenever Theorem 3 could be invoked with (r, γ, S) to build a stabilizing event-based encoding scheme, one could instead invoke Theorem 2 with $(\widetilde{r}, \gamma, S)$ to construct a stabilizing M-of-N encoding scheme with a smaller average bit-rate. This is the price paid for the convenience of the simple event-based logic as opposed to having to implement an encoder/decoder with a (possibly quite large) library of M-of-N codewords.

Remark 6. In Remark 2 it was noted that the sufficiency result in Theorem 2 would not bound the process state if the data-rate condition (2.50) held only with equality. In contrast, if the present data-rate inequality (3.4) holds with equality,

the following event-based scheme bounds the state of the process, as we will show in the proof of Theorem 3. However, the two sufficiency results of Theorem 2 and Theorem 3 are consistent in the sense that if their data-rate conditions [(2.50) and (3.4) respectively] hold with strict inequality, then exponential stabilization can be achieved, with the rate of exponential convergence determined by the "gap" in the inequality. To see this for the present scheme, suppose (3.4) holds with strict equality and let $\overline{x}(t) \coloneqq e^{\epsilon t}x(t)$, where $\epsilon > 0$ is small enough that

$$r\frac{h^{-1}(\gamma)}{\ln 3}\ln 2 > \sum_{i:\mathbb{R}\lambda_i[A]>0}\lambda_i[A] + n\epsilon, \tag{3.7}$$

and suppose $A + \epsilon I + BK$ is Hurwitz. Applying Theorem 3 to the $\overline{x}$ system provides a controller and encoder/decoder that bounds $\overline{x}$. However,

$$\|\overline{x}(t)\| \leqslant c \qquad \Leftrightarrow \qquad \|x(t)\| < ce^{-\epsilon t}, \tag{3.8}$$

so the state $x(t)$ converges to 0 exponentially fast.

3.2.1 Proof of Theorem 3

The main idea behind the proof is to show that, when assumption (3.4) holds, it is possible to allocate the available average bit-rate among sub-encoders in such a way that each sub-encoder has a sufficiently large average bit-rate to bound its components of the state estimation error.

For the sub-encoder $i \in \mathcal{U}$, we pick the transmission period T_i as

$$T_i \coloneqq h^{-1}(\gamma)/(a_i + \eta), \tag{3.9}$$

where the definition of h is from (3.5) and $\eta > 0$ satisfies

$$r\frac{h^{-1}(\gamma)}{\ln 3}\ln 2 \geqslant \sum_{i:\mathbb{R}\lambda_i[A]>0} \lambda_i[A] + n\eta. \tag{3.10}$$

As mentioned above, no information needs to be sent on behalf of the stable systems $i \in \mathcal{S}$.

The event boundaries $L_j > 0$ are chosen as follows. Define

$$\underline{\tau}_i \coloneqq \frac{1}{a_i}\ln\left(\frac{2}{e^{(a_i+\eta)T_i}-1}\right). \tag{3.11}$$

Note that $\infty > \underline{\tau}_i > 0$ because $a_i > 0$ for $i \in \mathcal{U}$ and $\frac{2}{e^{(a_i+\eta)T_i}-1} > 1$ by our choice of T_i. Next, pick $1 > \phi > 0$ sufficiently small so that

$$\phi < e^{-T_i}/4 \tag{3.12}$$

$$\underline{\tau}_i \leqslant \frac{1}{a_i}\ln\left(\frac{2}{(e^{a_i T_i}(1+2e^{T_i}\phi)-1+2e^{\underline{\tau}_i}\phi}\right) \tag{3.13}$$

for all $i \in \mathcal{U}$. Finally, define the event boundaries recursively as

$$L_n \coloneqq \sup_{x_0 \in \mathcal{X}_0} \|P(0)x_0\|_\infty \tag{3.14}$$

$$L_j \coloneqq \frac{1}{\phi} \sum_{l=j+1}^{n} L_l \quad j \in \{1, \ldots, n-1\}. \tag{3.15}$$

The proof proceeds in three parts: First we establish that this choice of transmission periods results in an average bit-rate that does not exceed r. Then we prove that the scheme bounds the state of process (2.1). Lastly we prove the scheme's average cost per symbol does not exceed γ.

The scheme's average bit-rate does not exceed r

For $i \in \mathcal{U}$, sub-encoder i sends d_i symbols from the alphabet $\{-1, 0, 1\}$ every T_i time units, resulting in an average bit-rate of

$$r_i \coloneqq d_i \log_2 3 / T_i, \tag{3.16}$$

and so the average bit-rate used by the encoder as a whole is simply

$$\sum_{i \in \mathcal{U}} r_i = \log_2 3 \sum_{i \in \mathcal{U}} \frac{d_i}{T_i} = \frac{\log_2 3}{h^{-1}(\gamma)} \sum_{i \in \mathcal{U}} d_i(a_i + \eta) \tag{3.17}$$

$$= \frac{\log_2 3}{h^{-1}(\gamma)} \left(\sum_{i : \mathbb{R}\lambda_i[A] > 0} \lambda_i[A] + n\eta \right) \leqslant r, \tag{3.18}$$

where the last inequality follows from hypothesis (3.4). Hence, this encoding scheme uses an average bit-rate of r or less.

The scheme stabilizes the process

Next we show that this controller and event-based encoder/decoder pair bound the state of process (2.1). In view of (2.64), this is ensured if $e(t)$ is bounded. Since $e_i(t) \to 0$ for $i \in \mathcal{S}$, we focus on $e_i(t)$ for $i \in \mathcal{U}$.

We proceed with an inductive proof that the sequence $\{e_{i,j}(kT_i)^+\}_{k\in\mathbb{Z}_{>0}}$ is bounded for $i \in \mathcal{U}$, $j \in \{1, \ldots, d_i\}$. The base of induction $k = 0$ follows from the definition of L_j in (3.14). Next we prove that $e_{i,j}(kT_i)^+ \in [-L_j, L_j]$ provided that $e_{i,l}(kT_i - T_i)^+ \in [-L_l, L_l]$ for $l \in \{j, \ldots, d_i\}$. If $e_{i,j}(kT_i - T_i)^+$ is so small that it does not grow outside the box $[-L_j, L_j]$ by the next timestep, then we naturally have $e_{i,j}(kT_i)^+ \in [-L_j, L_j]$. On the other hand, suppose at a specific time t^* satisfying $kT_i - T_i \leqslant t^* < kT_i$, the scalar error $e_{i,j}(t^*)$ grows to the boundary of the box $[-L_j, L_j]$; without loss of generality suppose $e_{i,j}(t^*) = L_j$. Up to T_i time units later, the timestep kT_i occurs and the sub-encoder i transmits $s_{i,j}(k) = 1$ to the decoder. Upon receiving symbol 1, the decoder knows from the encoding scheme (3.1) that the scalar error $e_{i,j}(kT_i)^-$ immediately before the transmission must have exceeded the event boundary L_j and hence $e_{i,j}(kT_k)^- > L_j$. Moreover,

$$|e_{i,j}(kT_i)^-| = |\mathbf{v}'_{i,j} e_i(kT_i)^-| \tag{3.19}$$

$$= |\mathbf{v}'_{i,j} e^{a_i T_i} G_i(T_i) e_i(kT_i - T_i)^+| \tag{3.20}$$

$$\leqslant e^{a_i T_i} \left| \sum_{l=0}^{d_i - j} \frac{T_i^l}{l!} e_{i,j+l}(kT_i - T_i)^+ \right| \tag{3.21}$$

$$\leqslant e^{a_i T_i} \Big(|e_{i,j}(kT_i - T_i)^+| + \sum_{l=1}^{d_i - j} \frac{T_i^l}{l!} \sum_{l=1}^{d_i - j} |e_{i,j+l}(kT_i - T_i)|^+ \Big) \tag{3.22}$$

$$\leqslant e^{a_i T_i} \left(L_j + \sum_{l=1}^{d_i - j} \frac{T_i^l}{l!} \sum_{l=1}^{d_i - j} L_{j+l} \right) \tag{3.23}$$

$$\leqslant e^{a_i T_i} L_j \left(1 + e^{T_i} \phi\right), \tag{3.24}$$

where $\mathbf{v}_{i,j} \in \mathbb{R}^{d_i}$ is a unit vector satisfying (3.19), (3.20) follows from the error dynamics (2.54) in Lemma 4, (3.21) follows from the definition of the matrix $G_i(T_i)$, (3.22) follows from the triangle inequality, (3.23) follows from the induction hypothesis, and (3.24) follows by the definition of ϕ, and by upper-bounding the sum $\sum_{l=1}^{d_i - j} T_i^l / l!$ by e^{T_i}. Therefore the decoder can conclude that

$$e_{i,j}(kT_i)^- \in (L_j, L_j \, e^{a_i \, T_i}(1 + e^{T_i} \phi)]. \tag{3.25}$$

We can express the scalar error $e_{i,j}(kT_i)^-$ as the overall error vector $e(kT_i)^- \in \mathbb{R}^n$ times an appropriate unit vector:

$$e_{i,j}(kT_i)^- = \mathbf{v}'_{i,j} e(kT_i)^- \tag{3.26}$$

$$= \mathbf{v}'_{i,j} P(kT_i)(x(kT_i)^- - \hat{x}(kT_i)^-). \tag{3.27}$$

Rearranging the update rule (3.2) yields an expression for $\hat{x}(kT_i)^-$:

$$\hat{x}(kT_i)^- = \hat{x}(kT_i)^+ - P(kT_i)^{-1}\mathbf{v}_{i,j}\Delta_{i,j}(1). \tag{3.28}$$

Substituting this into (3.27) yields

$$\begin{aligned} e_{i,j}(kT_i)^- &= \mathbf{v}_{i,j}' P(kT_i)\left(x(kT_i)^- - \right. \\ &\qquad \left. \hat{x}(kT_i)^+ + P(kT_i)^{-1}\mathbf{v}_{i,j}\Delta_{i,j}(1)\right) \\ &= \mathbf{v}_{i,j}' P(kT_i)\left(x(kT_i)^- - \hat{x}(kT_i)^+\right) + \Delta_{i,j}(1) \\ &= e_{i,j}(kT_i)^+ + \Delta_{i,j}(1), \end{aligned}$$

where we used the fact that $x(kT_i)^- = x(kT_i)^+$ due to the continuity of the solution $x(t)$. Substituting this into (3.25) and simplifying yields

$$e_{i,j}(kT_i)^+ + \Delta_{i,j}(1) \in (L_j, L_j\, e^{a_i T_i}(1 + e^{T_i}\phi)] \tag{3.29}$$

which is equivalent to

$$e_{i,j}(kT_i)^+ \in \left(-\frac{L_j(e^{a_i T_i} - 1)}{2}, \frac{L_j(e^{a_i T_i}(1 + 2e^{T_i}\phi) - 1)}{2}\right]. \tag{3.30}$$

Recall that T_i was chosen to satisfy $h(a_i T_i) = \gamma \leqslant 1$. Applying h^{-1} to this yields $a_i T_i \leqslant \ln 2$, and so $e^{a_i T_i} \leqslant 2$. Combining this with the upper bound (3.12) on ϕ

yields

$$\frac{L_j(e^{a_i T_i}(1 + 2e^{T_i}\phi) - 1)}{2} < L_j. \tag{3.31}$$

Applying this to (3.30) establishes that

$$e_{i,j}(kT_i)^+ \in (-L_j, L_j) \tag{3.32}$$

and completes the inductive proof that the sequence $\{e_{i,j}(kT_i)^+\}_{k\in\mathbb{Z}_{>0}}$ is bounded. Since this holds for arbitrary $j \in \{1, \ldots, d_i\}$, the sequence $\{e_i(kT_i)^+\}_{k\in\mathbb{Z}_{>0}} \subset \mathbb{R}^{d_i}$ is also bounded. Following a similar argument to (2.76), we conclude that $e_i(t)$ is bounded for any $t \geqslant 0$. Since $e_i(t)$ is bounded for all $i \in \mathcal{U}$ and $e_j(t) \to 0$ for $j \in \mathcal{S}$, this controller and encoder/decoder pair bound the estimation error. Therefore the state is bounded for all time as well.

The scheme's average cost per symbol does not exceed γ

Lastly we prove that this encoding scheme has average cost per symbol not exceeding γ. The symbol stream emitted by the encoder is comprised of the $|\mathcal{U}|$ individual symbol sequences $\{s_{i,j}(k)\}_{k\in\mathbb{Z}_{>0}}$, $i \in \mathcal{U}$, $j \in \{1, \ldots, d_i\}$. We first show that each individual symbol sequence has average cost per symbol not exceeding γ. Then we show that superimposing these sequences preserves this property.

Consider the scalar error component $e_{i,j}(t)$, $i \in \mathcal{U}$, $j \in \{1, \ldots, d_i\}$. By (3.32)

we have $|e_{i,j}(kT_i)^+| < L_j$ with strict inequality. So there will be a strictly positive period of time with duration $\tau_{i,j} > 0$ starting at time kT_i until $e_{i,j}(t)$ grows to leave the $[-L_j, L_j]$ box. During this time, no non-free symbols will be transmitted. The "dead time" $\tau_{i,j}$ is simply the amount of time required for the bound $L_j\Big(e^{a_i T_i}(1+2e^{T_i}\phi)-1\Big)/2$ in (3.30) to grow to size L_j. Specifically, the dead time $\tau_{i,j}$ satisfies $|e_{i,j}(\tau_{i,j}+kT_i)| = L_j$ provided that $|e_{i,j}(kT_i)| \leqslant L_j\Big(e^{a_i T_i}(1+2e^{T_i}\phi)-1\Big)/2$. We now prove that the parameters $\underline{\tau}_i$ were chosen so that

$$|e_{i,j}(\underline{\tau}_i + kT_i)| \leqslant L_j \tag{3.33}$$

provided that

$$|e_{i,j}(kT_i)^+| \leqslant L_j\Big(e^{a_i T_i}(1+2e^{T_i}\phi)-1\Big)/2, \tag{3.34}$$

and therefore $\underline{\tau}_i$ lower-bounds the dead time $\tau_{i,j}$. Following a similar process to (3.19), we have

$$\begin{aligned}
|e_{i,j}(\underline{\tau}_i + kT_i)| &= |\mathbf{v}'_{i,j} e_i(\underline{\tau}_i + kT_i)| && (3.35)\\
&= |\mathbf{v}'_{i,j} e^{a_i \underline{\tau}_i} G_i(\underline{\tau}_i) e_i(kT_i)^+| && (3.36)\\
&\leqslant e^{a_i \underline{\tau}_i}\left|\sum_{l=0}^{d_i-j} \frac{\underline{\tau}_i^l}{l!} e_{i,j+l}(kT_i)^+\right| && (3.37)\\
&\leqslant e^{a_i \underline{\tau}_i}\Big(|e_{i,j}(kT_i)^+|
\end{aligned}$$

$$+ \sum_{l=1}^{d_i-j} \frac{\underline{\tau}_i^l}{l!} \sum_{l=1}^{d_i-j} |e_{i,j+l}(kT_i)|^+ \Big) \tag{3.38}$$

$$\leqslant e^{a_i \underline{\tau}_i} \Big(L_j \frac{e^{a_i T_i}(1+2e^{T_i}\phi) - 1}{2} + \sum_{l=1}^{d_i-j} \frac{\underline{\tau}_i^l}{l!} \sum_{l=1}^{d_i-j} L_{j+l} \Big) \tag{3.39}$$

$$\leqslant e^{a_i \underline{\tau}_i} L_j \left(\frac{e^{a_i T_i}(1+2e^{T_i}\phi) - 1}{2} + e^{\underline{\tau}_i}\phi \right) \tag{3.40}$$

$$\leqslant L_j, \tag{3.41}$$

where $\mathbf{v}_{i,j} \in \mathbb{R}^{d_i}$ is a unit vector satisfying (3.35), (3.36) follows from the error dynamics (2.54) in Lemma 4, (3.37) follows from the definition of the matrix $G_i(\underline{\tau}_i)$, (3.38) follows from the triangle inequality, (3.39) follows from the premise (3.34) and also (3.32), (3.40) follows by the definition of ϕ, and by upper-bounding the sum $\sum_{l=1}^{d_i-j} \underline{\tau}_i^l/l!$ by $e^{\underline{\tau}_i}$, and (3.41) follows from (3.13). We conclude that $\underline{\tau}_i \leqslant \tau_{i,j}$.

Therefore by (3.11) we have

$$\tau_{i,j} \geqslant \underline{\tau}_i \coloneqq \frac{1}{a_i} \ln \left(\frac{2}{e^{(a_i+\eta)T_i} - 1} \right) \tag{3.42}$$

$$= \left(\frac{a_i + \eta}{a_i} \right) \left(\frac{T_i}{h((a_i+\eta)T_i)} \right) \tag{3.43}$$

$$\Leftrightarrow \qquad \frac{T_i}{\tau_{i,j}} \leqslant \frac{a_i}{a_i+\eta}\gamma < \gamma, \tag{3.44}$$

where (3.43) and (3.44) follow from the definitions of h and T_i. This establishes a bound on the number of non-free transmissions as follows. Consider the symbol

sequence $\{s_{i,j}(k)\}_{k\in\mathbb{Z}_{>0}}$ emitted by this encoding scheme. Let N_2, N_1 be arbitrary positive integers, and let $N_{\text{nf}} \coloneqq \sum_{k=N_1}^{N_1+N_2-1} I_{s_{i,j}(k)\neq 0}$ be the number of non-free symbols among symbols $s_{i,j}(N_1), \ldots, s_{i,j}(N_1+N_2-1)$. Let $t_l, l \in \{1, \ldots, N_{\text{nf}}\}$ be the time that the lth non-free transmission occurred. The t_l satisfy $N_1 T_i \leqslant t_1 < \ldots < t_{N_{\text{nf}}} \leqslant (N_1+N_2-1)T_i$. Only free symbols are transmitted in the time interval $[t_l, t_l + \tau_{i,j})$, and so

$$t_l \geqslant \tau_{i,j} + t_{l-1}, \qquad \forall l = 2, \ldots, N_{\text{nf}}. \tag{3.45}$$

Iterating this formula over l, we obtain

$$t_{N_{\text{nf}}} \geqslant \tau_{i,j}(N_{\text{nf}} - 1) + t_1. \tag{3.46}$$

Rearranging this and using the facts that $N_1 T_i \leqslant t_1$ and $t_{N_{\text{nf}}} \leqslant (N_1+N_2-1)T_i$, we obtain

$$\sum_{k=N_1}^{N_1+N_2-1} I_{s_{i,k}\neq 0} =: N_{\text{nf}} \leqslant \frac{T_i}{\tau_{i,j}} N_2 + 1 \leqslant \gamma N_2 + 1,$$

where we leveraged (3.44). This implies the average cost per symbol condition (2.3), so we conclude that for any $i \in \mathcal{U}$ and any $j \in \{1, \ldots, d_i\}$, the symbol sequence $\{s_{i,j}(k)\}_{k\in\mathbb{Z}_{>0}}$ has average cost per symbol not exceeding γ.

Finally we show that superimposing the symbol streams results in a stream

with average cost per symbol not exceeding γ. Let $N_1, N_2 \in \mathbb{N}$ be arbitrary positive integers, and let $\mathcal{J}_i, i \in \mathcal{U}$ partition $\{N_1, N_1+1, \ldots, N_1+N_2-1\}$ such that $\mathcal{J}_i$ is the set of indices between N_1 and N_1+N_2-1 where the transmitted symbol was sent by sub-encoder i. Then $\sum_{i\in\mathcal{U}} |\mathcal{J}_i| = N_2$, and we obtain

$$\begin{aligned}\sum_{k=N_1}^{N_1+N_2-1} I_{s_{i,k}\neq 0} &= \sum_{i\in\mathcal{U}} \sum_{k\in\mathcal{J}_i} I_{s_{i,k}\neq 0} \\ &\leqslant \sum_{i\in\mathcal{U}} (\gamma|\mathcal{J}_i| + N_{0,i}) \\ &= \gamma N_2 + N_0,\end{aligned}$$

where $N_0 \coloneqq \sum_{i\in\mathcal{U}} N_{0,i}$. The inequality comes from leveraging (2.3) for each sub-encoder on its respective index interval $\mathcal{J}_i$. This completes the proof of Theorem 3. ■

3.3 Numerical example

In this subsection we present a numerical example of the event-based encoding scheme presented in this chapter. As in the numerical example of the M-of-N encoder from Section 2.4, consider process (2.1) with A, B, and K defined in (2.84).

Whereas the M-of-N encoding scheme from in Section 2.4 stabilized the system with a average bit-rate of $r = 10$ and an average cost per symbol of $\gamma = 0.2$,

note that $r = 10$ and $\gamma = 0.2$ do not satisfy the sufficient bound (3.4) so they cannot be used in Theorem 3 to construct a stabilizing event-based scheme. Instead, we use $r \coloneqq 21$, leaving $\gamma \coloneqq 0.2$ as before. This satisfies (3.7) with $\epsilon = 0.1$, so we apply Theorem 3 to obtain an encoder/decoder and controller that together bound the system $\overline{x}(t) \coloneqq e^{0.1t}x(t)$, and therefore $x(t)$ decays exponentially. This is illustrated in Figures 3.2 and 3.3.

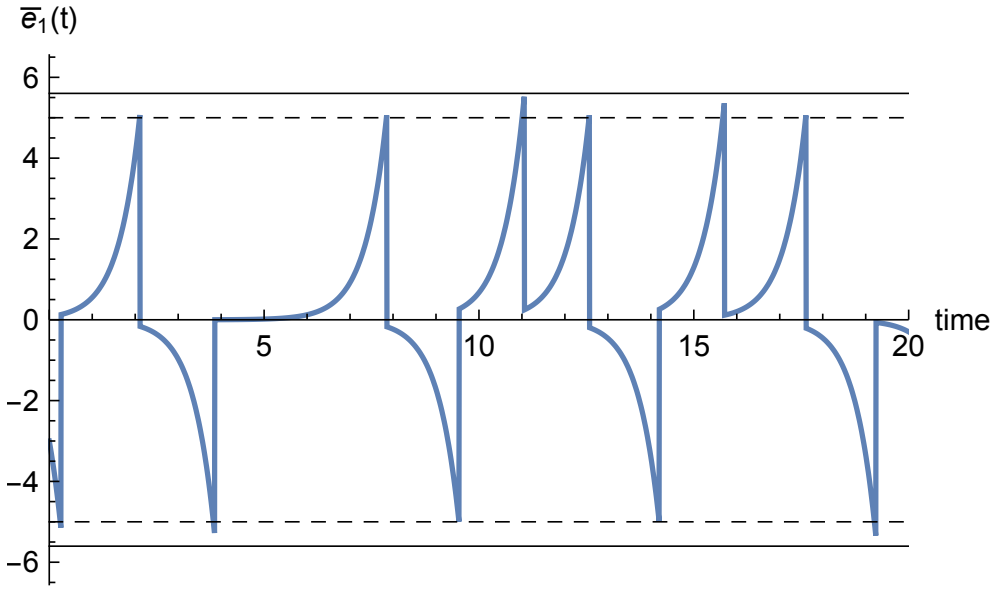

Figure 3.2: Plot of the closed-loop state estimation error component $\overline{e}_1(t)$ for the $\overline{x}(t)$ system, using the event-based encoding scheme. Once the error leaves $[-L_1, L_1]$ (thin dashed lines), a non-free symbol is transmitted at the next transmission time. The error stays bounded between $-L_1 e^{(a_1+0.1)T_1}$ and $-L_1 e^{(a_1+0.1)T_1}$ (thick dashed lines). Unlike the encoder from Section 2.3 in Figure 2.5, the transmission of non-free symbols is event-triggered and non-periodic.

Recall that the two sub-encoders of the codeword-based encoder from Section 2.4 each transmit up to 2 non-free symbols every 1.9 time units, resulting in a total average rate of resource consumption of 2.1 non-free transmissions per

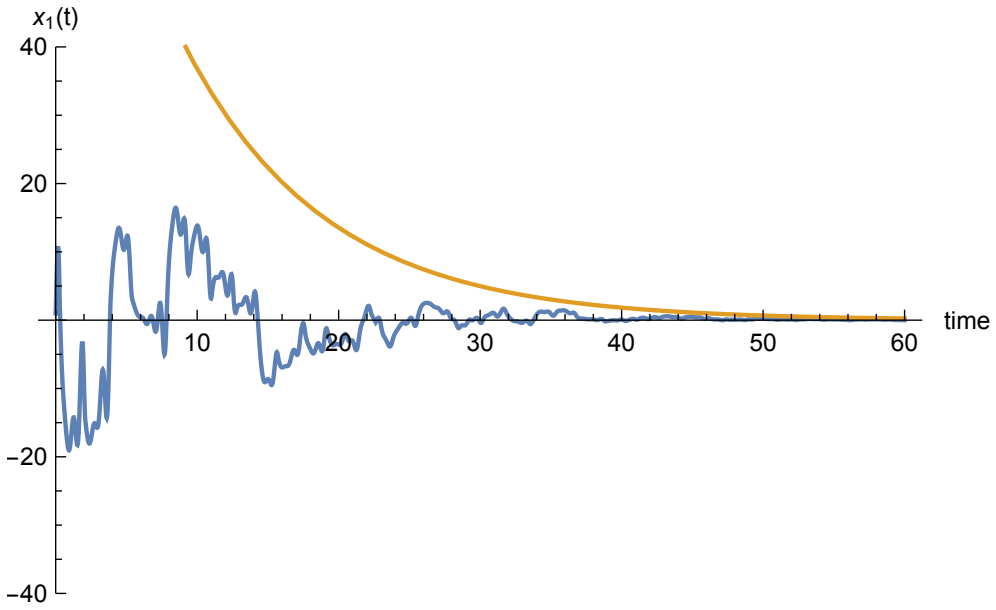

Figure 3.3: Plot of the closed-loop state $x_1(t)$ exponentially decaying to 0 using the event-based encoding scheme described in Section 3. The curve $100e^{-0.1t}$ is plotted for reference.

time unit. On the other hand, the event-based encoder's two sub-encoders each transmit a symbol every 0.151 time units, and a fraction $\gamma = 0.2$ of these symbols are non-free. Therefore this event-based encoder consumes communication resources at a total average rate of 2.65 non-free transmissions per time unit. This is in accordance with Remark 6: this larger rate of consumption is the price paid for using an easier-to-implement event-based encoding scheme.

3.4 Conclusion

In this chapter examined an event-based controller based on the framework from Chapter 2. We proved its average bit-rate requirements were order-optimal with

respect to the necessary and sufficient condition for stabilizability from Chapter 2. This supports the use of event-based controllers in limited-communication control schemes.

The controller in the proposed event-based scheme required state feedback. This could be extended to an output-feedback setting by embedding a state observer in the encoder, which is the subject of future work.

Chapter 4

Preemption-resistant control on a non-real-time operating system

In this chapter we consider the problem of stabilizing a system with unpredictable timing due to the controller running on a non-real-time operating system. We propose a method of implementing a discrete-time control algorithm on a non-real-time operating system so that the sensing and actuation occur at precise times, even if the OS preempts the control task.

4.1 Real-time I/O coprocessor concept

We first describe the architecture of our control and sensing/actuation scheme. Figure 4.1 illustrates the basic idea, which is that the controller and sensor/actu-

ator execute on two separate processors. The controller runs on a non-real-time OS like Linux, whereas the sensing and actuation are performed by a dedicated "bare-metal" microcontroller called the Real-Time Unit (RTU). The RTU contains two circular buffers, one of size n_s for time-stamped sensor measurements, and one of size n_a for time-stamped actuator commands. At each timestep, the RTU reads, time-stamps, and saves a new sensor measurement in the sensor buffer, and then applies the appropriate actuator command from the actuator buffer. The controller and RTU may be co-located on the same circuit-board or even within a single system-on-a-chip.

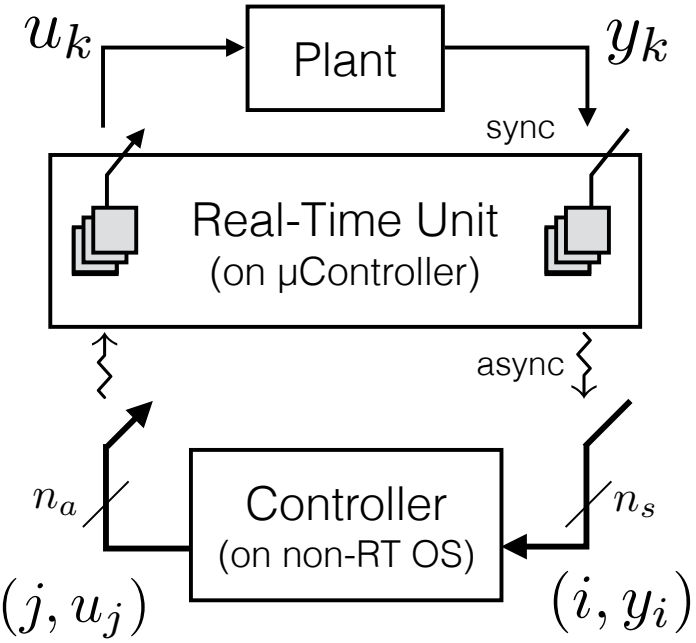

Figure 4.1: Schematic of the control architecture. The real-time I/O coprocessor measures sensors y_k and applies actuator values u_k every T_s time units from its two buffers. Asynchronously, the controller retrieves the n_s most recent sensor values and transmits n_a time-stamped actuator values for the RTU to apply to the plant.

We now explain Algorithms 1 and 2 below, which summarize the code that runs on the controller and the RTU.

Algorithm 3. *1: (Controller)*

2: ***while*** *true* ***do***

3: *Retrieve the RTU's sample buffer.*

4: *Generate a list of n_a time-stamped actuator values.*

5: *Send the list to the RTU.*

6: *Wait for next timestep.*

7: ***end while***

Algorithm 4. *1: (Real-Time Unit (RTU))*

8: ***while*** *true* ***do***

9: ***if*** *controller requested data,* ***then***

10: *Send the sensor buffer to the controller.*

11: ***end if***

12: ***if*** *controller sent a new actuation schedule,* ***then***

13: *Copy the schedule to the actuation buffer.*

14: ***end if***

15: *Check the time.*

16: ***if*** *sample time $T_s k$, $k \in \mathbb{N}$ just elapsed,* ***then***

17: *Sample sensors and store with time-stamp k.*

18: ***if*** *(k, u_k) is in the actuation buffer,* ***then***

19: *Apply input u_k to the plant.*

20: ***else***

21: *Apply default input to the plant.*

22: ***end if***

23: ***end if***

24: ***end while***

(Algorithm 1: Controller.) At an arbitrary time t, the controller requests the RTU's measurement buffer and receives n_s time-stamped measurements $(k-n_s+1, y_{k-n_s+1}), \ldots, (k, y_k)$, where $k \coloneqq \lfloor t/T_s \rfloor$ is the index of the last timestep before time t. The controller then computes a list of n_a time-stamped actuator values. The resulting actuation sequence could follow the retrieved sample sequence by starting at sample $k+1$, e.g., $(k+1, u_{k+1}), \ldots, (k+n_a, u_{k+n_a})$. However, if it is known that the controller will take at least C sample times to run, the controller may instead compute and send actuation signals to be applied at sample times $k+C+1, \ldots, k+C+n_a$. In either case, the controller then transmits the actuation sequence to the RTU's actuator buffer. Note that because the controller may be preempted, the controller's actions occur asynchronously with respect to the RTU's sampling and actuation times. Section 4.1.1 discusses the issues of generating future actuation sequences.

(Algorithm 2: RTU.) The RTU loop starts by checking whether the controller requested data or delivered a new actuation schedule. If so, the RTU transfers

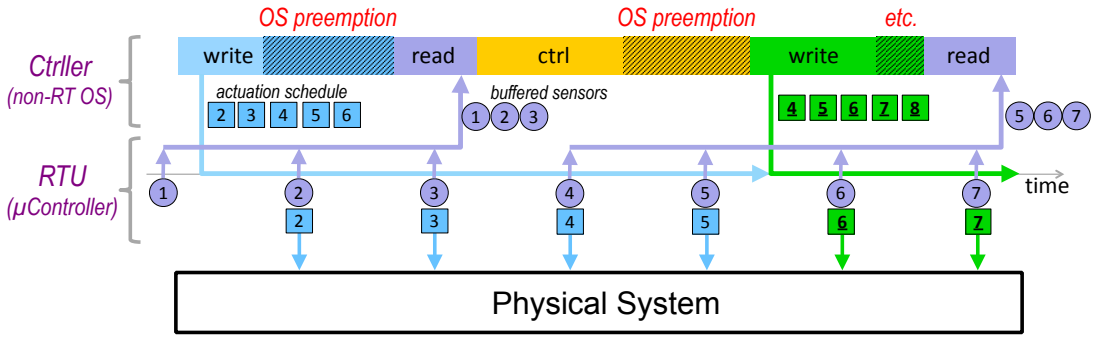

Figure 4.2: The RTU buffers sensor measurements (circles) and executes buffers of time-stamped actuator commands (squares) from the controller.

sensor data to the controller or copies new actuation commands into the RTU's private buffer. At sample time $t = T_s k$, $k = 1, 2, \ldots$, the RTU reads the sensor measurement y_k and stores the time-stamped measurement (k, y_k) in its circular buffer. It then searches its actuation buffer for an actuation command of the form (k, u_k) and applies u_k to the plant over the time interval $[T_s k, T_s(k+1))$. If the actuation buffer does not contain a command for timestep k, the RTU applies some default actuation, e.g., u_{k-1} or 0.

The RTU's ability to sample at precise times depends crucially on its ability to check the time rapidly. Consequently, it is important that the RTU be able to execute Algorithm 2 lines 8–13 quickly. Therefore the interconnection between the controller and the RTU needs to be fast, e.g., a shared memory. Similarly, the actual sampling and actuation also needs to happen quickly (lines 15–21).

Figure 4.2 illustrates this architecture with buffer sizes $n_s = 3$ and $n_a = 5$. At each sample time $t = T_s k$, $k \in \mathbb{N}$, the RTU reads and stores a sensor measurement. At some time between T_s and $2T_s$, the controller delivers an actuation

schedule (k, u_k), $k = 2, \ldots, 6$, then gets preempted. Despite the controller being preempted, the RTU executes the actuation schedule. Some time between $3T_s$ and $4T_s$ the controller requests the sensor buffer, which contains (k, y_k), $k = 1, 2, 3$. The controller then begins computing the actuation sequence (k, u_k), $k = 4, \ldots, 8$, but gets preempted partway through. Later, between $5T_s$ and $6T_s$, the controller awakens and delivers the actuation sequence. Note that because of preemption, the new actuation schedule arrives too late to apply the new (underlined) actuator values intended for sample times $4T_s$ and $5T_s$; instead, the RTU applied actuator commands u_4 and u_5 from the previous actuation schedule. After time $7T_s$ the controller receives the sample buffer with measurements for $k = 5, 6, 7$. Note that sample y_4 was overwritten and so is not available to the controller.

The key idea in this architecture is that the closed-loop system can tolerate some amount of OS preemption because the RTU continues to gather measurements and apply actuation even while the controller is asleep. The aim is for the controller to provide a sufficient number of future actuator values so the RTU can continue to stabilize the plant if the controller gets preempted. Even though the future actuations are applied "open-loop", we shall see that they are better than holding the actuators constant until the controller awakens.

4.1.1 Building the actuation schedule

The architecture proposed here requires the controller to produce, at each sample time k, an actuation schedule with control values for the next n_a future sample times $k+1, k+2, \ldots, k+n_a$. Two options are available: a model-free approach that generates the future control signals without an explicit model for the process and a model-based approach that uses such a model.

To describe both approaches consider a discrete-time nonlinear controller expressed by the following state-space model

$$z_{k+1} = f(z_k, y_k, r_k), \qquad u_k = g(z_k, r_k), \tag{4.1}$$

where the y_k denote sensor measurements, the u_k actuation values, and the r_k reference signals.

The model-free approach generates the n_a future actuator commands

$$u_{k+1}, u_{k+2}, \ldots, u_{k+n_a}$$

using polynomial extrapolation. Assuming that the measurement sequence can be approximated by a polynomial of degree q, one can use the previous $q+1$ measurements

$$y_{k-q}, \ldots, y_{k-1}, y_k$$

to predict $n_a - 1$ future measurements

$$y_{k+1}, y_{k+2}, \ldots, y_{k+n_a-1}$$

. Feeding these to the controller (4.1), one obtains the desired future actuator commands

$$u_{k+1}, u_{k+2}, \ldots, u_{k+n_a}$$

. As we shall see in Section 4.2, even a low order polynomial (linear extrapolation with $q = 1$) can be used to obtain good results.

When a plant model is available, the accuracy of the predicted measurements can be improved. Assuming a linear plant model of the form

$$x_{k+1} = Ax_k + Bu_k, \qquad y_k = Cx_k + Du_k, \tag{4.2}$$

if the plant's state x_k can be directly measured or estimated, one can estimate future measurements by directly solving the process model (4.2), which leads to

$$\hat{y}_{k+i} = CA^i\hat{x}_k + \Big(\sum_{j=0}^{i-1} CA^{i-j-1}Bu_{k+j}\Big) + Du_{k+i},$$
$$\forall i \in \{1, 2, \ldots, n_a - 1\}, \tag{4.3}$$

where $\hat{x}_k$ denotes the state estimate at time k and $u_{k+1}, u_{k+2}, \ldots, u_{k+i}$ a sequence

of future control signals constructed based on the controller model (4.1) and previous measurement estimates obtained by (4.3). The use of the plant model (4.2) permits a more accurate estimate of future measurements and consequently a better schedule for the future controls. However, our initial experiments indicate that this approach does not yield significant gains unless the sample time is fairly large.

Linearity of the process model in (4.2) was assumed solely for simplicity of presentation, as the sequence of estimated outputs can easily be generated for a nonlinear process model, provided that the process' state can be measured or estimated. Model predictive control (for either linear or nonlinear plants) is especially attractive for this type of architecture, as it automatically produces a sequence of future controls.

4.2 Experimental results

In this section we compare the performance of a PID controller driving a DC motor when it uses a standard file-based I/O interface versus using a real-time I/O coprocessor.

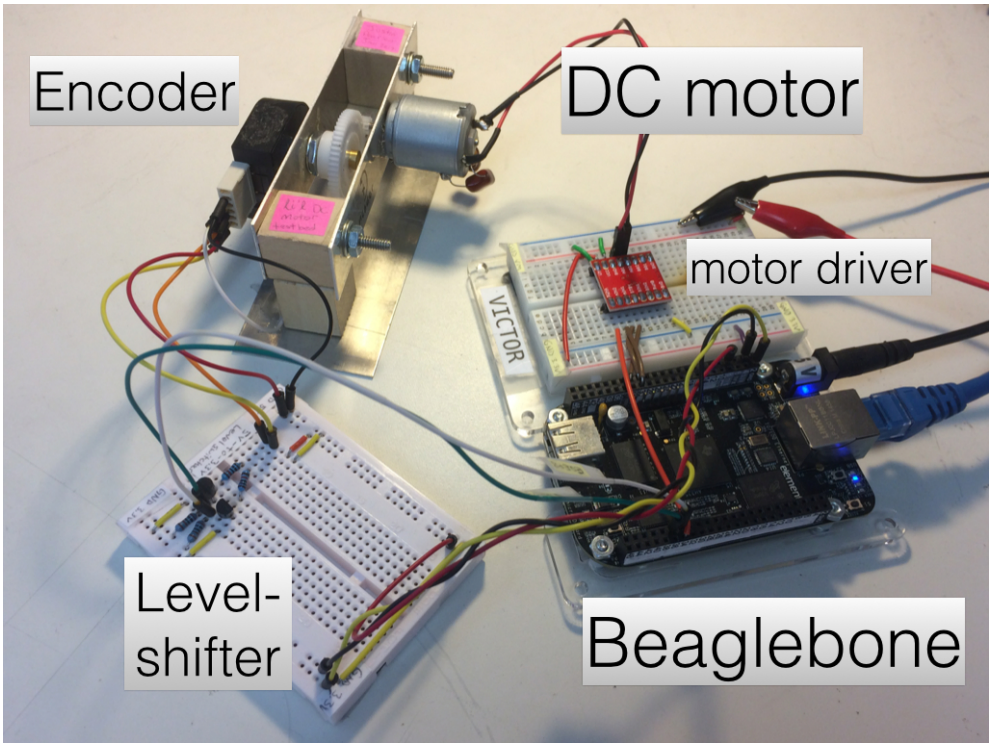

Figure 4.3: Picture of the hardware setup. A Beaglebone Black drives a DC motor and measures its shaft angle using a rotary encoder.

4.2.1 Hardware

Figure 4.3 shows our hardware setup. A TB6612FNG motor driver drives a hobby-grade permanent-magnet DC motor from a 5-volt power supply. The motor driver takes a 50 kHz PWM signal and three discrete 3.3 V signals which determine the motor direction. The motor shaft angle is measured by a US Digital rotary optical encoder. The encoder has 4096 counts per rotation and outputs a quadrature-encoded pulse (QEP) signal. A 5V-to-3.3V level-shifting circuit scales the QEP signal.

The controller runs on a Beaglebone Black (BBB) single-board computer [22].

The BBB has a 1-GHz processor, 512 MB RAM, HDMI video, an ethernet port, and a USB port. It ships with Debian Linux installed on its 4 GB flash memory. It is powered by a Texas Instruments Sitara AM3358BZCZ100 processor, which contains ADC, PWM, QEP, and GPIO peripherals.

For a real-time I/O coprocessor, we use one of the two "Programmable Real-Time Units" (PRUs) included in the Sitara microcontroller. Designed for real-time applications, each PRU is a 32-bit 200-MHz RISC processor core that executes independently from the main CPU, has its own 8 kB data RAM, and has full access to the peripherals on the Sitara. The CPU and PRU have access to each other's memory and can therefore exchange time-stamped sensor and actuator data quickly. The PRU is not pipelined, making its execution simpler and more deterministic: register-level instructions run in 1 cycle (5 ns), and memory instructions to the PRU's local memory take 3 cycles. The PRU's data RAM does not share a bus with the main RAM, so the PRU can read and write to it without the risk of bus contention with the main memory. A cycle counter register within the PRU allows it to track time in increments of 5 ns. For these reasons, the PRU is well-suited for use as a RTU.

We implemented the control architecture described in Section 4.1 on the PRU with circular buffer sizes of $n_a = n_s = 32$. The sensor buffer stored time-stamped QEP samples from the rotary encoder, whereas the actuator buffer stored time-stamped PWM and GPIO commands for the motor driver. To coordinate data

transfer between the CPU and the PRU, we double-buffered the sensor and actuator arrays in the PRU data RAM and implemented rudimentary mutual-exclusion semaphores. Accounting for the time to sample, actuate, and communicate, our implementation on the PRU achieved sampling and actuation timing accuracy of 40 μs.

4.2.2 Controller Design

A DC motor can be modeled as a series connection of a resistor, inductor, and back-EMF voltage source, resulting in the dynamic equations

$$\begin{aligned} V_m &= iR + L\dot{i} + K_1\omega \\ J\dot{\omega} &= -b\omega + K_2 i + \tau_{\text{ext}} \\ \dot{\theta} &= \omega, \end{aligned} \tag{4.4}$$

where i is the current through the motor, R and L are the resistance and inductance of the motor windings, θ is the motor shaft angle, ω is the angular velocity, V_m is the voltage applied across the motor, K_1 and K_2 are motor constants, b is the friction coefficient, J is the angular moment of inertia of the motor, and τ_{ext} is any external torque imposed on the motor shaft.

Equations (4.4) form a 3rd-order linear dynamical system. System identification was performed on the motor system using ARX on voltage and angle data

to obtain the following 3rd-order discrete-time linear model of the DC motor:

$$\frac{\Theta(z)}{V(z)} = \frac{-0.5898z^{-1} - 1.121z^{-2} - 0.2757z^{-3}}{1 - 1.586z^{-1} + 0.3719z^{-2} + 0.2136z^{-3}}. \tag{4.5}$$

The model's sample time was $T_s \coloneqq 0.005$ s. The model was validated with additional input/output data.

A PID controller was designed using Matlab's PIDTuner with the discrete-time transfer function

$$C(z) = k_p + k_i \frac{T_s}{z-1} + k_d \frac{1}{T_f + T_s/(z-1)}, \tag{4.6}$$

where $k_p = -0.0304$, $k_i = -0.106$, $k_d = -8.73\text{e}{-4}$, and $T_f = 0.00405$. The parameter T_f is the time-constant of a first-order filter on the derivative term. This controller was implemented as an IIR filter in C. To produce a future actuation schedule, the two most recent angle measurements were used to linearly extrapolate measurements for the future motor angles, and the PID controller was run on the tracking error between those predicted measurements and a known reference signal. Specifically, given the sensor buffer ending with sample k, the controller computed

$$\Delta \coloneqq \theta_k - \theta_{k-1}$$

$$\theta_{k+i} \coloneqq \theta_k + \Delta i$$

$$v_{k+i} \coloneqq c(\theta_{k+i}, \theta_{k+i-1}, \theta_{k+i-2}, v_{k+i-1}, v_{k+i-2}),$$

for $i = 1, \ldots, n_a$, where $c(\cdot)$ is the IIR representation of the controller (4.6), and θ_k and v_k are the angle measurement and voltage command at the kth sample time.

4.2.3 Results

Figure 4.4 shows the result of the two PID controllers as they track a triangle-wave reference signal on motor shaft angle. The top two plots show the performance of the PID controller when it uses the standard I/O mechanism on the BBB, wherein the peripherals appear as normal files. The lower two plots show the response of the PID controller when it uses the PRU as a real-time I/O coprocessor. The second and fourth plots show the time between each iteration of the PID control loop running on the main processor.

Table 4.1: RMS reference-tracking error of the controllers under idle and heavy system load.

	PID using standard I/O	PID using RTU I/O
Idle	12.0	11.3
Heavy	144	36.3

For the first two seconds, the PID controllers each run with essentially sole control of the CPU. There are no major OS preemptions during $t < 2$ and the second and fourth plots show that each iteration takes the intended sample

time $T_s = 0.005$ s. We observe that the two control configurations have similar performance during $t < 2$. At $t = 2$, several higher-priority CPU-heavy tasks were spawned. The spikes in the second and fourth plots during $t > 2$ correspond to controller preemptions, sometimes lasting 10 times the sample period. Whereas the PID controller using the standard I/O interface is heavily disrupted by these preemptions, we observe that the RTU-based controller runs much more smoothly due to the PRU buffering future control signals. The root-mean-square tracking errors for each controller under idle and heavily-loaded processor conditions are shown in Table 4.1. Similar results were obtained as the priorities of the competing tasks were changed to vary the frequency and durations of the OS preemptions.

4.3 Conclusion

In this chapter, we presented a controls architecture that pairs a real-time I/O coprocessor with a controller on a non-real-time operating system. The RTU enables sampling and actuation at precise times, even when the controller is preempted by the OS. This enables control designers to reap the benefits of an OS with minimal concern for the timing uncertainties associated with the OS task scheduler. We demonstrated the platform's utility by designing a preemption-resistant PID controller on a Beaglebone Black that uses its Programmable Real-time Unit as a real-time I/O coprocessor. The RTU-based PID controller

out-performed the standard PID controller in the presence of large OS preemptions. Future directions of this work include using a more sophisticated method of forward-prediction for the actuation sequence, such as model predictive control. Also we intend to extend this architecture to multiple controllers distributed across a network.

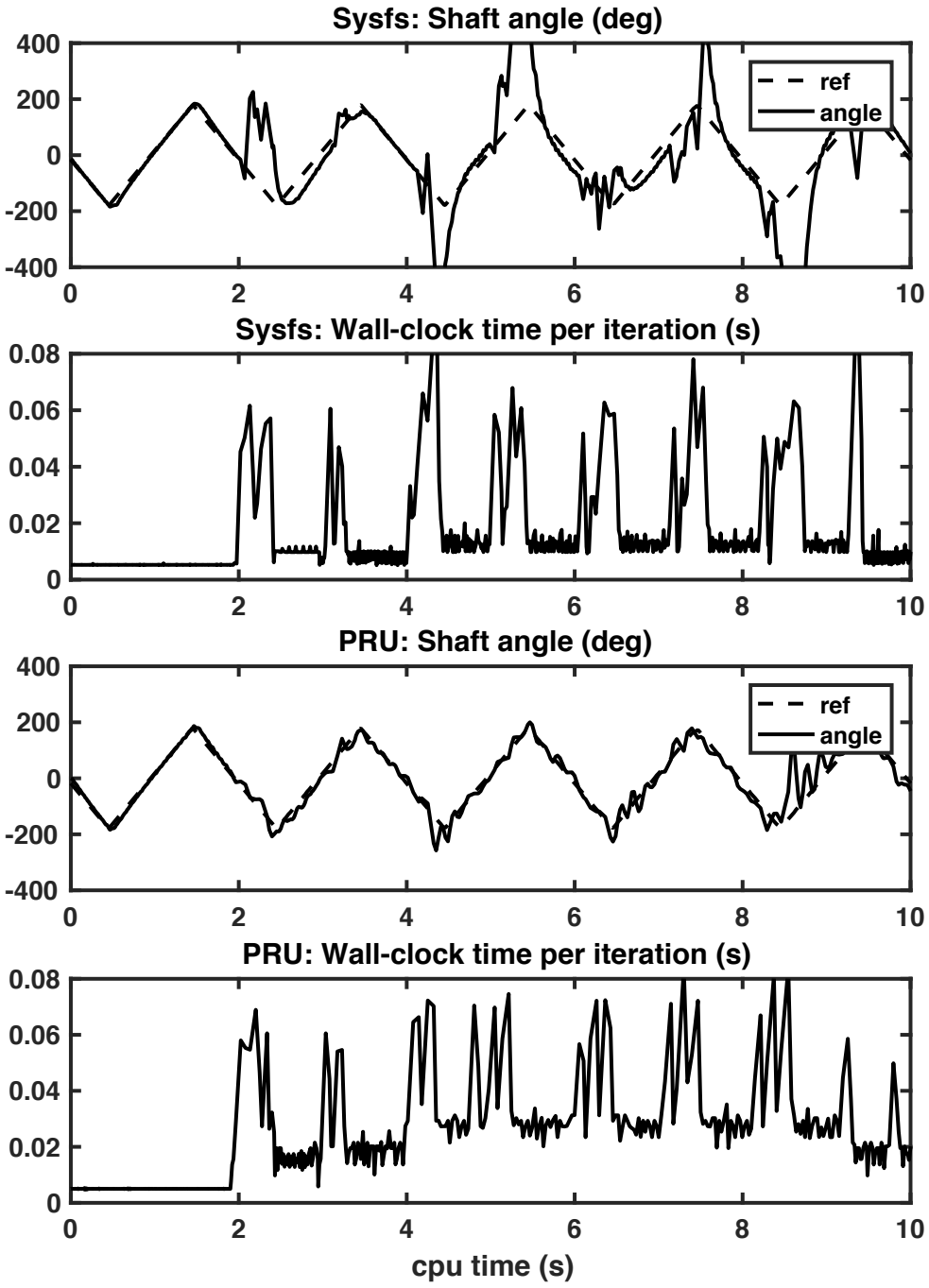

Figure 4.4: Both the standard PID controller and the PRU-based PID controller have similar performance under idle ($t < 2$). However, when subjected to OS preemption ($t > 2$), the PRU out-performs the standard one.

Appendix A

Proofs of lemmas

Proof of Lemma 1. Let $\ell \in \mathbb{Z}_{\geqslant 0}$ be arbitrary. Since the pair's average cost per symbol is at most γ, (2.3) holds for some $N_0 \in \mathbb{Z}_{>0}$. Rearranging (2.3) yields

$$\sum_{k=N_1}^{N_1+N_2-1} I_{s_k \neq 0} \leqslant N_2\gamma + N_0, \quad \forall N_1, N_2 \in \mathbb{Z}_{>0}. \tag{A.1}$$

Let N be any positive integer greater than $(N_0+1)/\epsilon\gamma$ and define $M \coloneqq \lfloor N\gamma + N_0 + 1 \rfloor$. Invoking (A.1) for $N_1 \coloneqq \ell N + 1$ and $N_2 \coloneqq N$ yields

$$\begin{aligned} \sum_{k=\ell N+1}^{\ell N+N} I_{s_k \neq 0} &\leqslant N\gamma + N_0 \leqslant M \\ &\leqslant N\gamma + N_0 + 1 < N\gamma(1+\epsilon). \end{aligned} \tag{A.2}$$

Therefore we have found an M and N satisfying $M < N\gamma(1+\epsilon)$ and moreover (A.2) implies the condition (2.8) defining M-of-N encoders. This completes the proof. ■

Proof of Lemma 4. There exists a real invertible matrix $Q \in \mathbb{R}^{n\times n}$ that transforms A to its real Jordan normal form, namely

$$Q^{-1}AQ = \Lambda \coloneqq \mathbf{diag}\left(J_1, \ldots, J_{n_b}\right),$$

where the J_i are real Jordan blocks: for real eigenvalue a_i with geometric multiplicity d_i, the corresponding real Jordan block $J_i \in \mathbb{R}^{d_i\times d_i}$ has the form

$$\begin{bmatrix} a_i & 1 & & \\ & \ddots & & \\ & & & 1 \\ & & & a_i \end{bmatrix}; \tag{A.3}$$

for a complex conjuguate pair of eigenvalues $a_i \pm jb_i$ with multiplicity d_i, the associated real Jordan block $J_i \in \mathbb{R}^{2d_i\times 2d_i}$ has the form

$$\begin{bmatrix} \Lambda_i & I_2 & & \\ & \ddots & & \\ & & & I_2 \\ & & & \Lambda_i \end{bmatrix}, \tag{A.4}$$

where the 2-by-2 matrix $\Lambda_i \in \mathbb{R}^{2\times 2}$ has the form

$$\Lambda_i \coloneqq \begin{bmatrix} a_i & b_i \\ -b_i & a_i \end{bmatrix}. \tag{A.5}$$

Next, define the time-varying invertible block-diagonal matrix $R(t) \in \mathbb{R}^{n\times n}$, $t \geqslant 0$ as

$$R(t) \coloneqq \mathbf{diag}\left(R_1(t), \ldots, R_{n_b}(t)\right) \tag{A.6}$$

where $R_i(t) \coloneqq I_{d_i} \in \mathbb{R}^{d_i}$ if J_i corresponds to a real eigenvalue a_i, and $R_i(t) \coloneqq \mathbf{diag}(\Theta_i(t)^{-1}) \in \mathbb{R}^{2d_i\times 2d_i}$ if J_i corresponds to a complex conjugate eigenvalue $a_i \pm jb_i$, where

$$\Theta_i(t) \coloneqq \begin{bmatrix} \cos(b_i t) & -\sin(b_i t) \\ \sin(b_i t) & \cos(b_i t) \end{bmatrix} \in \mathbb{R}^{2\times 2}. \tag{A.7}$$

Let $P(t) \coloneqq R(t)Q^{-1}$, $t \geqslant 0$. We have

$$e(t) \coloneqq P(t)(x(t) - \hat{x}(t)) \tag{A.8}$$

$$= R(t)Q^{-1}e^{At}(x(0) - \hat{x}(0)) \tag{A.9}$$

$$= R(t)Q^{-1}e^{Q\mathbf{diag}(J_i)Q^{-1}t}(x(0) - \hat{x}(0)) \tag{A.10}$$

$$= R(t)e^{\mathbf{diag}(J_i)t}Q^{-1}(x(0) - \hat{x}(0)) \tag{A.11}$$

$$= R(t)e^{\mathbf{diag}(J_i)t}e(0) \tag{A.12}$$

$$= R(t)\mathbf{diag}(e^{J_i t})e(0), \tag{A.13}$$

where (A.11) follows from a well-known property of the matrix exponential, and (A.12) follows the definition of $e(0)$ and the observation that $R(0)$ is the identity matrix. A well-known property of real Jordan blocks is that

$$e^{J_i t} = e^{a_i t}\begin{bmatrix} 1 & t & \frac{t^2}{2!} & \cdots & \frac{t^{d_i-1}}{(d_i-1)!} \\ & 1 & t & & \\ & & \ddots & & \\ & & & & \\ & & & & 1 \end{bmatrix}$$

if the real Jordan block J_i corresponds to a real eigenvalue, and

$$e^{J_i t} = e^{a_i t} \begin{bmatrix} \Theta_i(t) & \Theta_i(t)t & \Theta_i(t)\frac{t^2}{2!} & \dots & \Theta_i(t)\frac{t^{d_i-1}}{(d_i-1)!} \\ & \Theta_i(t) & \Theta_i(t)t & & \\ & & \ddots & & \\ & & & & \\ & & & & \Theta_i(t) \end{bmatrix}$$

if it corresponds to a complex conjugate pair. In terms of $R_i(t)$ and $G_i(t)$ these equations become simply

$$e^{J_i t} = e^{a_i t} R_i(t)^{-1} G_i(t). \tag{A.14}$$

Using this in (A.13) yields

$$\begin{aligned} e(t) &= R(t)\mathbf{diag}(e^{a_i t}\mathbf{diag}(R_i(t)^{-1}G_i(t)))e(0) \\ e(t) &= \mathbf{diag}(R_i(t))\mathbf{diag}(R_i(t)^{-1})\mathbf{diag}(e^{a_i t}G_i(t))e(0) \\ e(t) &= \mathbf{diag}(e^{a_i t}G_i(t))e(0), \end{aligned}$$

implying (2.54).

Lastly, it is straightforward to verify that the minimum singular value of $R_i(t)$ is

$\sigma_{\min}(R_i(t)) = 1$ for any t. Moreover, since Q is invertible, there exists $\epsilon > 0$ for which $\sigma_{\min}(P(t)) \geqslant \epsilon$ for all t. This concludes the proof. ■

Lemma 5. *For any $N \in \mathbb{Z}_{>0}$ and $M \in \mathbb{R}_{\geqslant 0}$ with $M \leqslant N$, every M-of-N encoder has average cost per symbol not exceeding M/N.*

Proof of Lemma 5. Suppose M and N are fixed and consider a sequence of N_2 symbols starting at index N_1, for arbitrary $N_1, N_2 \in \mathbb{Z}_{>0}$. This index sequence $\{N_1, \ldots, N_1 + N_2 - 1\}$ overlaps or partially overlaps with at most $\lceil N_2/N \rceil + 1$ of the fixed N-symbol codewords. Each codeword has at most M non-free symbols. Therefore the number of non-free symbols in the sequence is upper-bounded by

$$\begin{aligned}\sum_{k=N_1}^{N_1+N_2-1} I_{s_k \neq 0} &\leqslant M\left(\lceil N_2/N \rceil + 1\right) \\ &\leqslant M(N_2/N + 2) = \frac{M}{N} N_2 + 2M. \end{aligned} \tag{A.15}$$

We let $N_0 \coloneqq 2M$ and rearrange terms to obtain (2.3), the definition of average cost, with $\gamma = M/N$. ■

Lemma 6. *The following inequality holds for all $N, S \in \mathbb{Z}_{>0}$, $q \in (0, S/(S+1)]$, and $i \in [0, Nq]$:*

$$q^i (1-q)^{N-i} \geqslant 2^{-N\,H(q)}\, \frac{S^i}{S^{Nq}} \tag{A.16}$$

where $H(q) \coloneqq -q \log_2 q - (1-q)\log_2(1-q)$ *is the base-*2 *entropy of a Bernoulli random variable with parameter* q*.*

Proof of Lemma 6. Let N, S, q, and i take arbitrary values from the sets described in the lemma's statement. Since $\log_2$ is a monotone increasing function, $\log_2(q/(1-q))$ for $q > 0$ is maximized at the right endpoint value, $q = S/(S+1)$, where it equals $\log_2 S$. This leads to

$$\log_2 q - \log_2(1-q) \leqslant \log_2 S \tag{A.17}$$

for all $S \in \mathbb{Z}_{>0}$ and $q \in (0, S/(S+1)]$. Next, $i \in [0, Nq]$ by assumption, therefore $i - Nq \leqslant 0$. Multiplying (A.17) by $i - Nq$ and straightforward algebraic manipulation yields

$$\begin{aligned} & i \log_2 q + (N-i)\log_2(1-q) \\ & \geqslant Nq \log_2 q + N(1-q)\log_2(1-q) + (i-Nq)\log_2 S \\ & = -NH(q) + (i-Nq)\log_2 S, \end{aligned}$$

where the equality follows from the definition of $H(q)$. Raising 2 to the power of both sides, (A.16) follows. ∎

Appendix B

Beaglebone Black / DC motor test-bed

(This project is hosted online at GitHub [28].)

B.1 Summary

This section describes how to interact with the Beaglebone Black's I/O from the command line. We use the Beaglebone Black to drive a DC motor with pulse-width modulation through a motor driver, and measures the motor's shaft angle with a rotary encoder that uses quadrature-encoded pulses.

It does this entirely from the command line; there is no C or Python code needed. This is possible because you can do I/O through special files called "sysfs

entries". For example, after loading the PWM device-tree overlay (see below), you can set the PWM duty cycle and period from the command line with the following commands:

```
$ echo 100000 > /sys/devices/ocp.3/pwm_test_P8_34.12/period  # nanosec
$ echo  10000 > /sys/devices/ocp.3/pwm_test_P8_34.12/duty    # nanosec
```

(Note: a bug in the PWM driver flips the polarity, meaning that a 10000-ns duty cycle with a 100000-ns period results in a 90% high square-wave and not a 10% square wave as you'd expect.)

B.2 Hardware setup

Here is the hardware setup:

- $3 5V DC motor from Sparkfun
  - Note: Solder capacitors between the motor terminals and case to reduce inductive kickback into the BBB. Without these capacitors, sometimes BBB will hang when simultaneously driving the motor and reading the ADCs.
- $30 Dual H-bridge motor driver from Sparkfun (model: TB6612FNG)
- $100 rotary encoder from US Digital (model: S1-1024-250-NE-B-D)
- Outputs 5V EQEP signal, 4096 EQEP ticks per revolution
- Use the `bone_eqep1` device-tree overlay

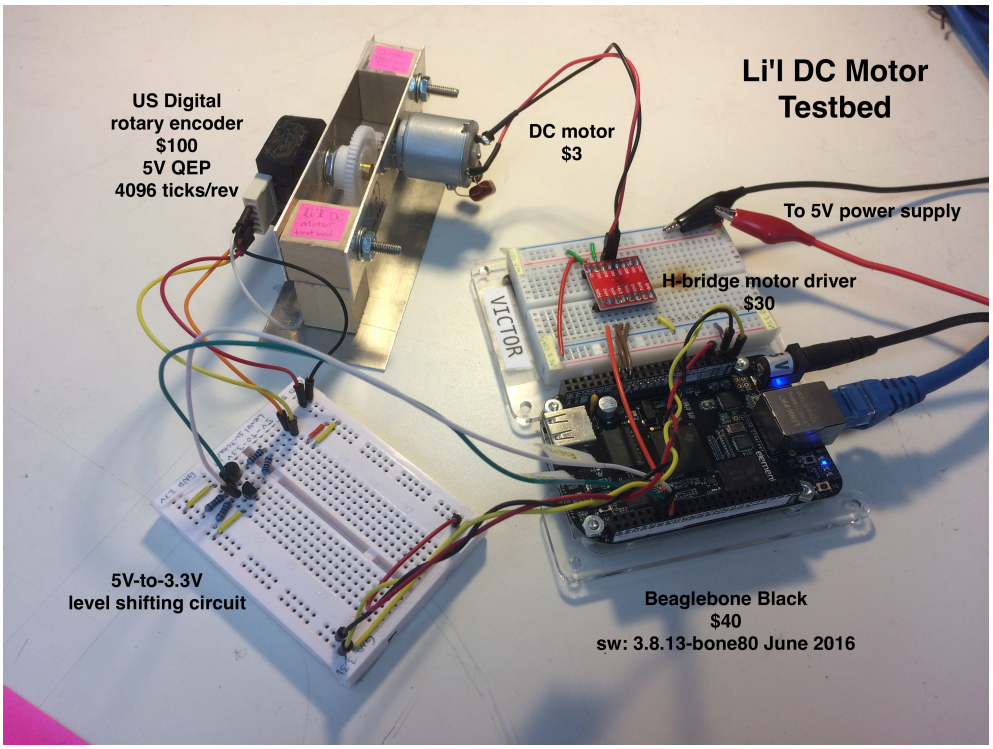

Figure B.1: The motor setup.

- transistors for 5V-to-3.3V conversion between rotary encoder and BBB

Note: It would be best to drive the motor using both H-bridges in the motor driver, driving them in parallel. The driver is rated for 1A continuous current with 3A peak; driving the 2.5-Ohm motor with 5V across results in 2A continuous current.

B.3 Software setup

There is no software setup; just set up the hardware and then run `run-lil-dc-motor.sh`.

After the script runs, it will leave the PWM, EQEP, and GPIO sysfs entries enabled.

The script `run-lil-dc-motor.sh` (see [28]) shows how to configure the Beaglebone Black to drive the DC motor and read its position with a rotary encoder. The script does this:

- loads the PWM and EQEP device-tree overlays
- loads the GPIO sysfs entries
- runs the motor clockwise
- runs the motor counter-clockwise
- sets PWM's "run" to 0 and motor driver's "standby" GPIO pin to low (standby)

Note: This code was developed on the following Beaglebone Linux kernel (found with `uname -a`):

```
Linux beaglebone 3.8.13-bone80 \#1 SMP Wed Jun 15 17:03:55 UTC 2016
armv7l GNU/Linux
```

B.4 Background

You can control various peripherals on the Beaglebone Black from the command line. For example, to configure pin `P9_31` as an output and set it high (3.3V), do this:

```
echo 110   > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio110/direction
echo 1     > /sys/class/gpio/gpio110/value
```

The number `110` is the number of the GPIO that's wired up to header pin `P9_31`. The mapping between BBB header pins (`P9_31`) and GPIO numbers (`110`) is weird, see Derek Molloy's Exploring Beaglebone Figures (pdf 1, pdf 2)

Line 1 creates `/sys/class/gpio/gpio110/` with various files inside like `direction` and `value`. They are not "real" files on disk; the kernel catches reads/writes to the files inside `gpio110/` and invokes a special kernel module to interact with the GPIO hardware. This method of interfacing with hardware is called "sysfs".

Line 2 writes the string "out" to the `direction` sysfs file. This causes the GPIO kernel module to configure the GPIO as an output.

Line 3 writes the string "1" to the `value` sysfs file. This causes the GPIO KM to set the GPIO to 3.3V (high).

Appendix C

Beaglebone C I/O library

(This project is hosted online at GitHub [27].)

C.1 Introduction

We present a simple C library for interacting with the I/O peripherals (PWM, GPIO, EQEP) on the Beaglebone Black. This library makes it easy to use C to interface with the Beaglebone Black's PWM, GPIO, and EQEP sysfs entries that permit users to modify I/O from userspace. For simplicity, only standard syscall functions are used: `open`, `close`, `read`, and `write`.

We demonstrate the library with code that drives a DC motor, see Figure C.1 and Figure C.2. It reads the motor shaft angle with a EQEP-based rotary encoder and drives the motor with PWM through a motor driver. The motor driver draws

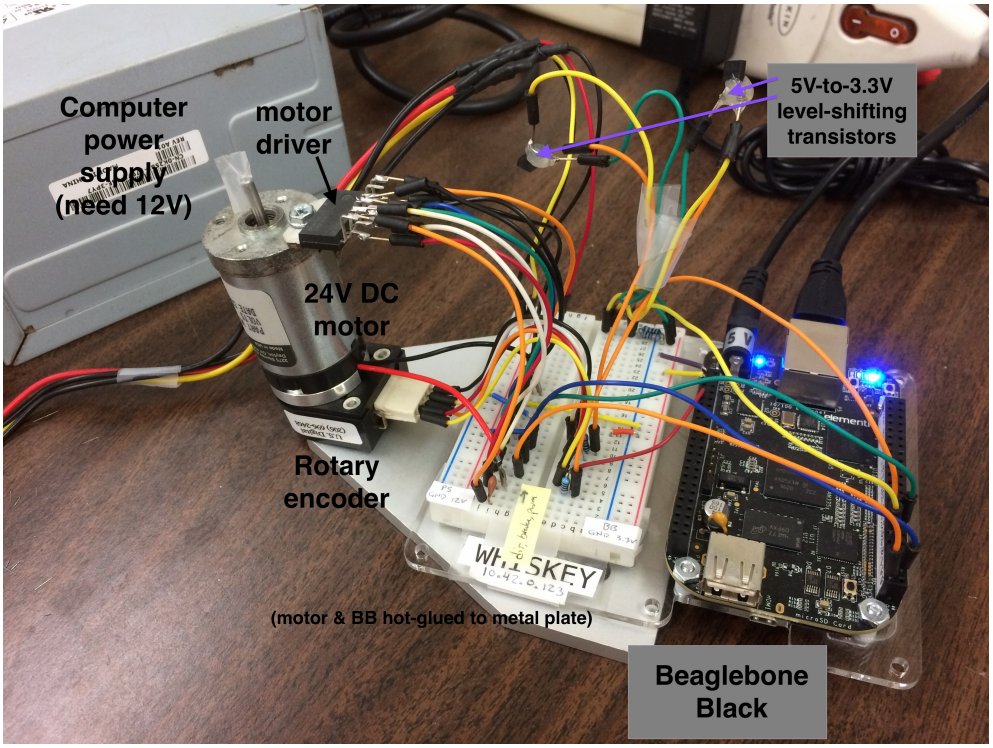

Figure C.1: The Beaglebone Black, motor driver, and 24V DC motor that we drive with the BBB C I/O library.

power through a disused desktop power supply's 12-Volt line.

Some Python code is also provided for comparison. It uses the built-in Adafruit_BBIO library and Nathaniel Lewis's eqep.py module.

A lot of stuff is hard-coded for expediency. This makes it easy for newcomers to learn how to use C to interface with the Beaglebone's sysfs entries, without getting bogged down with C++ classes or device-tree overlays.

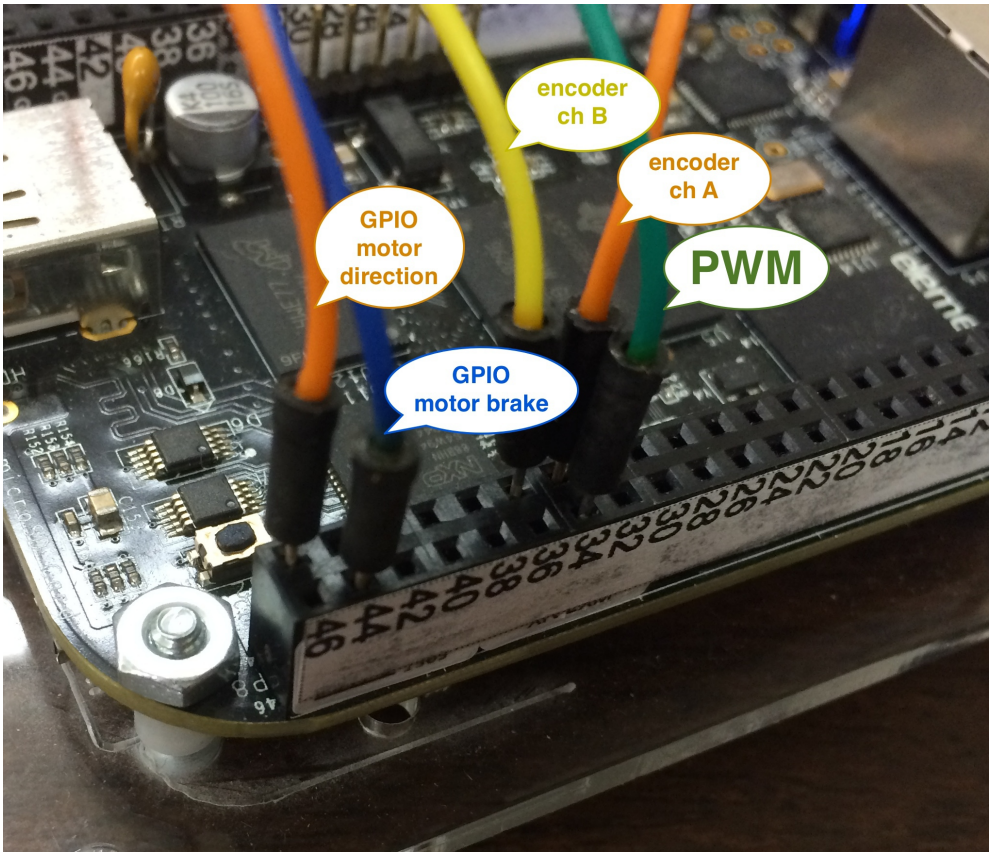

Figure C.2: How the motor driver is wired into the Beaglebone Black.

C.2 Quick-start

C.2.1 Configure Beaglebone, build, & run

- Plug in the BB's 5V power plug. If the 4 blue LEDs don't start blinking in 5 seconds, unplug it and re-plug it.
- Then, ssh into the BB from your laptop. (The BB's IP address is hard-coded as 10.42.0.123, so make your laptop 10.42.0.2 or something.)
- Note: it's possible to have a wireless Internet connection while being ssh'd to the Beaglebone over wired ethernet. See this for setup on Ubuntu.

```
ssh debian@10.42.0.123
sudo su
date -s "13 Dec 2013 13:43"  # or whatever
cd Beaglebone-Motor-Demo/C
./run.sh
```

The `run.sh` script does 3 things:

1. Loads the PWM, GPIO, and EQEP device-tree overlays necessary to run the demo. It essentially does

```
export SLOTS=$(find /sys/devices -name slots)
echo am33xx_pwm     > $SLOTS
```

```
echo bone_pwm_P8_34 > $SLOTS

echo bone_eqep1     > $SLOTS

echo 70             > /sys/class/gpio/export

echo 73             > /sys/class/gpio/export
```

Moreover, it also generates a header file `sysfs-paths.h` that just `#defines` the paths of the PWM, GPIO and EQEP sysfs entries so that functions in `bb-simple-sysfs-c-lib.h` can use them.

Originally, I hard-coded the sysfs paths in `bb-simple-sysfs-c-lib.h`. But it turns out that the directories sometimes change between reboots, e.g., sometimes

`echo bone_pwm_P8_34 > $SLOTS`

results in a directory

`/sys/devices/ocp.3/pwm_test_P8_34.18/`

but sometimes it is called

`/sys/devices/ocp.3/pwm_test_P8_34.12/`.

2. Compiles the library (`bb-simple-sysfs-c-lib.c/h`), tests (`tests.c`), and main (`main.c`) programs.

3. Runs `main`.

C.2.2 Handy BB commands

- Shutdown: `# shutdown -hP now`

- Reboot: `# reboot`

C.2.3 Turn motor in C

```
#include "bb-simple-sysfs-c-lib.h"

void main() {

  setup();


  printf("Shaft angle BEFORE (deg): %lf\n", shaft_angle_deg());

  duty(50); // 50% duty cycle

  cw();     // clockwise

  unstby(); // disable 'stby' GPIO on motor driver

  run();    // set 'run' sysfs entry for PWM


  sleep(1); // let it run for a sec.


  duty(0);  // set 'duty' to 0

  stop();   // turn off 'run'

  stby();   // set 'stby' GPIO on motor driver


  printf("Shaft angle AFTER (deg): %lf\n", shaft_angle_deg());
```

```
    shutdown();


  }
```

C.2.4 Proportional controller in C

```
int main ( int argc, char *argv[] ) {


  setup();

  unstby();

  run();

  cw();


  double kp = -.015;

  double dt = 0.1;      // sec, time per iteration

  double max_time = 10; // sec, max time of sim

  int num_iters = max_time / dt;

  double freq = 1;      // Hz, rate of ref angle change


  int i=0;

  for( i=0; i<num_iters; i++ ) {
```

```
    double angle = shaft_angle_deg();
    double ref = 180 * sin(2.0 * M_PI * freq * dt * i); // deg
    double error = ref-angle;
    double v = kp * error;
    voltage(v);
    usleep(dt*1000000.0);
  }

  stop();
  stby();
  shutdown();

return 0;
}
```

C.3 Hardware setup

The hardware consists of:

- DC motor (Globe Motors 405A336)
- Motor driver (LMD18201T)
- Rotary encoder (US Digital)

- Beaglebone Black
- Dell desktop power supply
- 2 10-nF capacitors for motor driver
- Two 2N3906 transistors used for 5V-to-3.3V level-shifting the EQEP sensor

Figure C.3 shows the wiring schematic for this hardware setup.

In particular, note that:

- The motor driver has inputs for PWM, direction, and brake.
- Pin P8_34 is the PWM.
- Pin P8_45 (GPIO) ctrls motor direction.
- Pin P8_44 (GPIO) ctrls motor brake (standby).
- The rotary encoder puts out 5V, but the BB's GPIOs require 3.3V; the transistor circuits perform level-shifting from 5V to 3.3V.
- The rotary encoder's EQEP signal is read by the BBB's EQEP peripheral.

C.4 Software

The file `bb-simple-sysfs-c-lib.c/h` provides a very thin C interface to the Beaglebone Black's PWM, GPIO, and EQEP sysfs entries.

For expediency, I hard-coded the sysfs entries for the PWM, two GPIOs, and EQEP in `bb-simple-sysfs-c-lib.h`:

```
#define PWM_PATH          "/sys/devices/ocp.3/pwm_test_P8_34.18/"
```

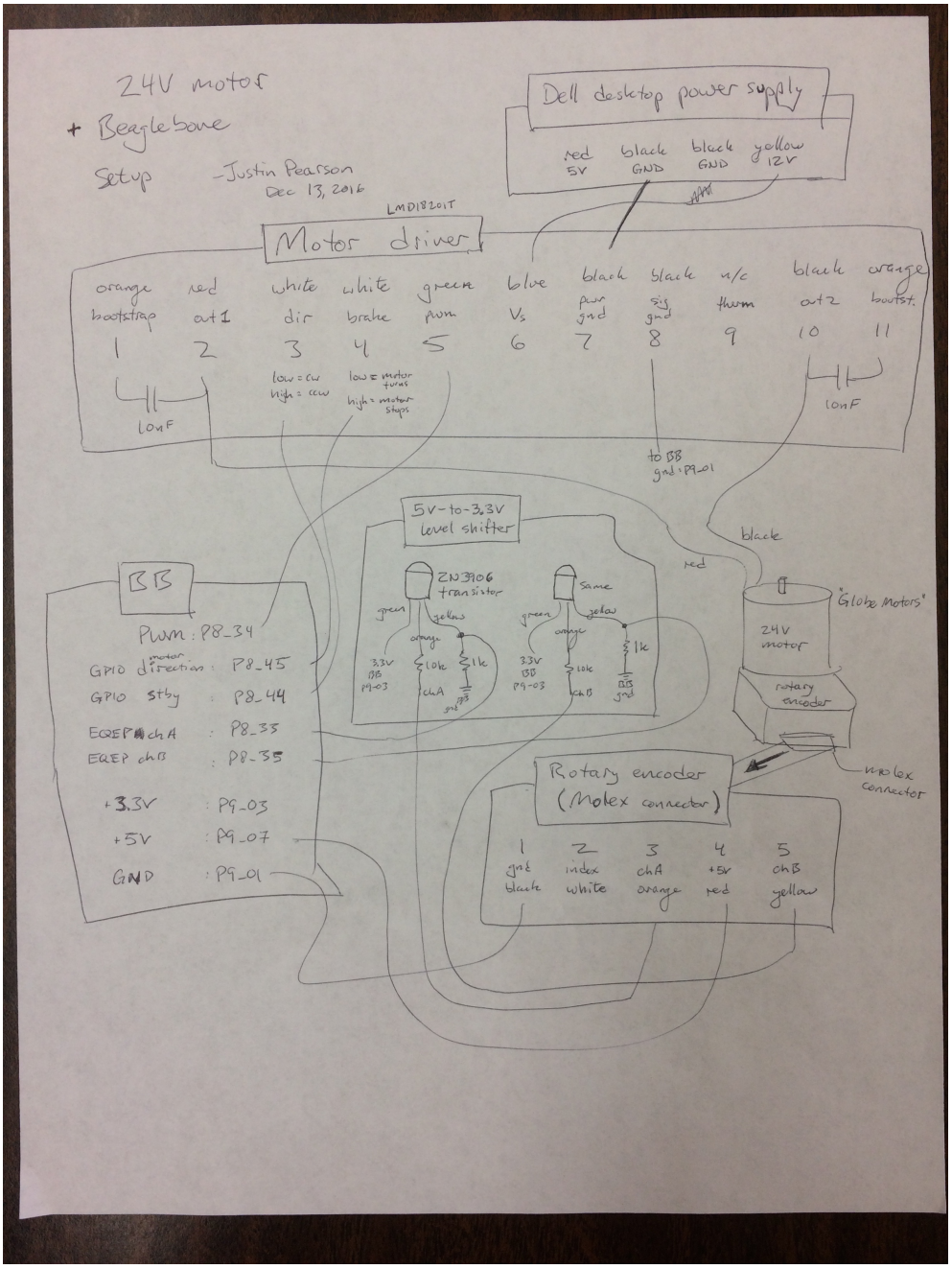

Figure C.3: Wiring schematic of the Beaglebone Black, LMD18201T motor driver, and 5V-to-3.3V level-shifting circuit.

```
#define GPIO_MOTORDIR_PATH "/sys/class/gpio/gpio70/"
#define GPIO_STBY_PATH     "/sys/class/gpio/gpio73/"
#define EQEP_PATH        "/sys/devices/ocp.3/48302000.epwmss/48302180.eqep/"
```

If these sysfs directories don't exist, execute the following lines to create them:

```
$ export SLOTS=$(find /sys/devices -name slots)
$ echo am33xx_pwm > $SLOTS
$ echo bone_pwm_P8_34 > $SLOTS
$ echo bone_eqep1 > $SLOTS
$ echo 70 > /sys/class/gpio/export
$ echo 73 > /sys/class/gpio/export
```

Notes:

- the `slots` file on my machine lives at `/sys/devices/bone_capemgr.9/slots`.
- The Exploring Beaglebone book's Fig 6-6 shows that P8_45 (that I connected to the motor driver's "direction" pin) is GPIO 70, and P8_44 (I connected to "brake" / standby) is GPIO 73.
- The EQEP directory may be named slightly different; find the precise one with

```
$ find /sys/devices/ -iname "*qep*"
```

- The same goes for the PWM; use `find /sys/devices/ -name duty` to find it.

- Running Python's Adafruit library wipes out the sysfs entries, e.g,

  Adafruit_BBIO.PWM.cleanup(),

  so you will have to re-echo them to recreate them.

The motor driver draws power from a Dell desktop power supply's 12V line. I hard-coded the PWM period to 50kHz. The rotary encoder seems to have a resolution of 1500 lines per revolution:

```
#define MAX_VOLTAGE 11.7 // Volts, Dell desktop power supply
#define NS_PER_PWM_PERIOD 20000 // ns per PWM period
#define NS_PER_PWM_PERIOD_STR "20000"
#define EQEP_PER_REV 1500 // I counted by hand, rough estimate
```

C.4.1 C functions provided

- PWM
  - `stop()` / `run()`: write 0 / 1 to the "run" PWM sysfs entry
  - `rawduty(char* c, int len)`: write a string to the "duty" sysfs file: "20000" is 0% duty cycle, "0" is 100% duty cycle
  - `duty( double d )`: write 0 - 100% to the "duty" sysfs file

  - `voltage( double v)`: convert voltage v into a duty cycle & GPIO direction and change them appropriately

- GPIO
  - `stby()` / `unstby()`: set P8_44 to 1 / 0
  - `cw()` / `ccw()`: set P8_45 to 1 / 0

- EQEP
  - `int eqep_counts()`: read eqep "position" file as an int
  - `double shaft_angle_deg()`: gets eqep position and converts to degrees

C.5 Details / Notes

- Note: PWM period is set in something like

  `/sys/devices/ocp.3/pwm_test_P8_34.18/period`

  with units of "ns per PWM cycle".

- Note: in sysfs, the 'duty' file is given in ns, not %. Ex: if period is set to 20000 (ns), then duty takes value between 0 (for 100% duty cycle) to 20000 (for 0% duty cycle)

- Note: polarity is switched on pwm:

- to do 0% duty cycle, you must write same value to

  `/sys/devices/ocp.3/pwm_test_P8_34.18/duty`

  as you wrote to

  `/sys/devices/ocp.3/pwm_test_P8_34.18/period`.

- To get 100% duty, must write 0 to duty.

C.5.1 Background: sysfs entries

The BBB uses a sysfs filesystem to provide a userspace interface to the hardware.

For example, set up a 50kHz PWM on pin P8_34 like this:

```
echo bone_pwm_P8_34 > /sys/devices/bone_capemgr.9/slots
```

(Note that the location of `slots` changes between versions of the BBB kernel.)

That creates the directory

```
/sys/devices/ocp.3/pwm_test_P8_34.18/   (your .18 may be different)
```

with files like `duty`, `period`, and `run`. Now turn on the PWM:

```
echo 20000 > period  # 20000 ns per PWM cycle => 50kHz
echo 10000 > duty    # 50% duty cycle
echo 1 > run
```

C.5.2 Hardware setup notes

- cw 1 rev: eqep changes by -1450
- ccw 1 rev: eqep changes by 1500

- the motor may have a gearbox inside.
- stby low: motor turns; hi: motor stops
- pwm 10%: just barely turns. stutters. sometimes stops
- dir pin low: motor turns cw; high: ccw

C.5.3 Sign conventions

For the `shaft_angle_deg()` and `voltage()` functions:

- ccw is positive angle
- positive motor voltage turns motor cw
- cw 1 rev => -1500 encoder ticks

C.5.4 Motor specifications

- motor coil resistance: 14 ohms
- motor coil inductance: 11.52 mH
- rotary encoder: 1500 lines / rev, roughly

C.5.5 Troubleshooting

The EQEP driver isn't included in the stock BBB kernel, so the command

`echo bone_eqep1 > $SLOTS`

will fail in `dmesg`; update kernel to latest with

```
cd /opt/scripts/tools/
git pull
sudo ./update_kernel.sh
sudo reboot
```

(Source: `http://elinux.org/Beagleboard:BeagleBoneBlack_Debian`)

Now you should have

```
# find /lib/firmware -iname "*qep*"
/lib/firmware/bone_eqep0-00A0.dtbo
/lib/firmware/bone_eqep1-00A0.dtbo
/lib/firmware/bone_eqep2b-00A0.dtbo
/lib/firmware/bone_eqep2-00A0.dtbo
```

Check your new OS version with `uname -a`:

```
Linux beaglebone 3.8.13-bone81 #1 SMP
Fri Oct 14 16:04:10 UTC 2016 armv7l GNU/Linux
```

Make a shell variable `SLOTS` pointing to your `slots` file that organizes your DTOs:

```
$ export SLOTS=$(find /sys/devices -name slots)
```

On my BB, `$SLOTS` is `/sys/devices/bone_capemgr.9/slots`.

Load Device Tree Overlays:

```
$ echo am33xx_pwm > $SLOTS
$ echo bone_pwm_P8_34 > $SLOTS
$ echo bone_eqep1 > $SLOTS
```

Have them added automatically by adding to `/boot/uboot/uEnv.txt`:

```
optargs=capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMIN \
capemgr.enable_partno=BB-ADC,bone_pwm_P8_34,am33xx_pwm,bone_eqep1
```

Appendix D

PASM syntax highlighter

This section contains instructions for configuring Sublime Text 3 to syntax-highlight the `pasm` code used by the Beaglebone Black's Programmable Real-time Unit (PRU). Figure D.1 shows what this looks like. The pasm commands are listed in the following Texas Instruments webpage [12, 13].

D.1 Installation instructions

1. Copy the code in Section D.2 into a new file `pasm.sublime-syntax`. Note: in the `match:` lines, remove the newlines and backslashes so as to make a single long line.

2. Put the file `pasm.sublime-syntax` in the place where Sublime Text looks

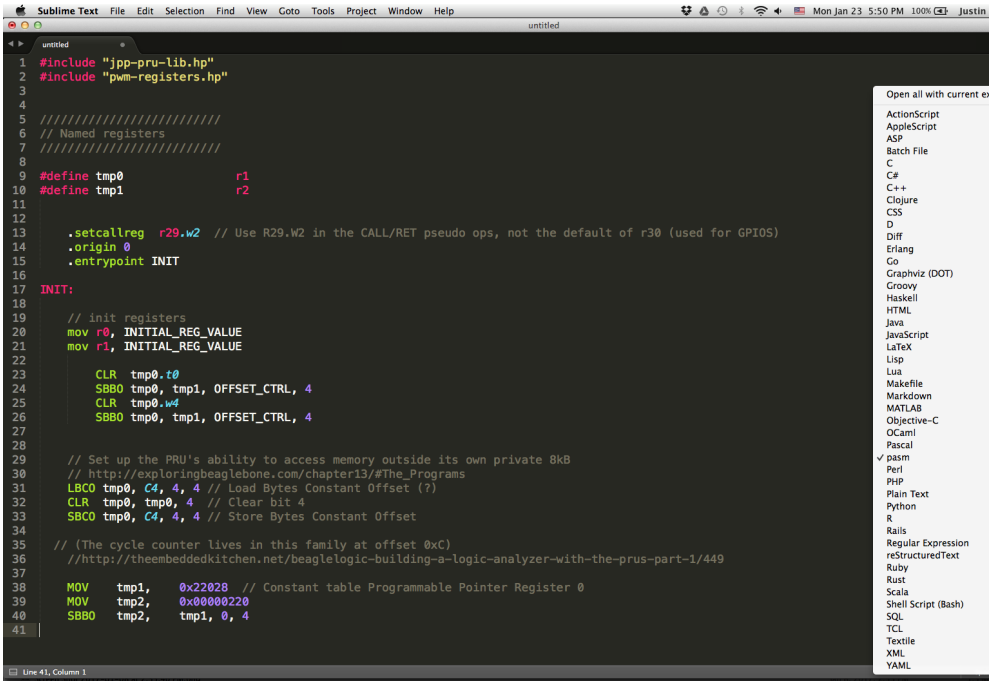

Figure D.1: After installing `pasm.sublime-syntax`, Sublime Text 3 displays pasm code correctly syntax-highlighted.

for syntax definitions:

- **Mac:** `/Users/justin/Library/Application Support/Sublime Text 3/Packages/User/`
- **Linux:** `~/.config/sublime-text-3/Packages/User/`

3. Restart Sublime Text.

4. In the lower-right language selection, you should see 'pasm' (see Figure D.1).

D.2 Syntax-highlighting code

```
%YAML 1.2

---

name: pasm

file_extensions: p

scope: source.pasm


contexts:

  main:


    # Assembly instructions

    - match: \b(ADD|ADC|SUB|SUC|RSB|RSC|LSL|LSR|AND|OR|XOR|NOT\

                |MIN|MAX|CLR|SET|SCAN|LMBD|MOV|LDI|MVIB|MVIW\

                |MVID|LBBO|SBBO|LBCO|SBCO|LFC|STC|ZERO|JMP|JAL\

                |CALL|RET|QBGT|QBGE|QBLT|QBLE|QBEQ|QBNE|QBA\

                |QBBS|QBBC|WBS|WBC|HALT|SLP|LOOP|add|adc|sub\

                |suc|rsb|rsc|lsl|lsr|and|or|xor|not|min|max\

                |clr|set|scan|lmbd|mov|ldi|mvib|mviw|mvid|lbbo\

                |sbbo|lbco|sbco|lfc|stc|zero|jmp|jal|call|ret\
```

```
            |qbgt|qbge|qblt|qble|qbeq|qbne|qba|qbbs|qbbc\
            |wbs|wbc|halt|slp|loop)\b
    scope: entity.name.function

# Dot commands
- match: \.origin|\.entrypoint|\.setcallreg|\.macro|\
         \.mparam|\.endm|\.struct|\.ends|\.u8|\.u16|\
         \.u32|\.assign|\.enter|\.leave|\.using
    scope: entity.name.function

# Preprocessor directives
- match: (#include|#define|#undef|#error\
          |#ifdef|#ifndef|#endif|#else)
    scope: keyword.control.import.include

# Comments
- match: //.*$
    scope: comment

# Numbers: 123
- match: \b[0-9]+\b
```

```
    scope: constant.numeric


# Hex numbers: 0xC, 0xff
- match: \b(0x[0-9A-Fa-f]+)\b
  scope: constant.numeric


# Binary numbers: 0b11011000
- match: \b(0b[01]+)\b
  scope: constant.numeric


# Labels: "INIT_ADC:"
- match: ([0-9a-zA-Z_]+)(:)
  scope: keyword.control


# Registers: R3
- match: ([rR]\d\d?)
  scope: storage


# Constant registers: C12
- match: ([cC]\d\d?)
  scope: storage.type
```

```
    # Bits & words: R3.t15, C12.w3
    - match: \.[tw]\d\d?
      scope: storage.type

    # Strings: "hello world"
    - match: \".*\"
      scope: string
```

Bibliography

[1] R Alur, K-E Arzen, John Baillieul, TA Henzinger, Dimitrios Hristu-Varsakelis, and William S Levine. *Handbook of networked and embedded control systems*. Springer Science & Business Media, 2007.

[2] K.J. Aström. Event based control. In *Analysis and Design of Nonlinear Control Systems: In Honor of Alberto Isidori*, page 127. Springer Verlag, 2007.

[3] K.J. Aström and B.M. Bernhardsson. Comparison of Riemann and Lebesgue sampling for first order stochastic systems. In *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, volume 2, pages 2011 – 2016, dec. 2002.

[4] Sanjeev Baskiyar and Natarajan Meghanathan. A survey of contemporary real-time operating systems. *Informatica (Slovenia)*, 29(2):233–240, 2005.

[5] R.W. Brockett and D. Liberzon. Quantized feedback stabilization of linear systems. *Automatic Control, IEEE Transactions on*, 45(7):1279–1289, Jul 2000.

[6] G. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Real-Time Systems Series. Springer US, 2011.

[7] J. Chaoui, K. Cyr, S. de Gregorio, J. P. Giacalone, J. Webb, and Y. Masse. Open multimedia application platform: enabling multimedia applications in third generation wireless terminals through a combined risc/dsp architecture. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, volume 2, pages 1009–1012 vol.2, 2001.

[8] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley, 2012.

[9] R. Goebel, R.G. Sanfelice, and A. Teel. *Hybrid Dynamical Systems: Modeling, Stability, and Robustness*. Princeton University Press, 2012.

[10] R. Grepl. Real-time control prototyping in matlab/simulink: Review of tools for research and education in mechatronics. In *2011 IEEE International Conference on Mechatronics*, pages 881–886, April 2011.

[11] João Pedro Hespanha, Antonio Ortega, and Lavanya Vasudevan. Towards the control of linear systems with minimum bit-rate. In *Proc. of the Int. Symp. on the Mathematical Theory of Networks and Syst.*, Aug. 2002.

[12] Texas Instruments. *PRU Assembly Instructions*, 2017 (accessed January 23, 2017). `http://processors.wiki.ti.com/index.php/PRU_Assembly_Instructions`.

[13] Texas Instruments. *PRU Assembly Reference Guide*, 2017 (accessed January 23, 2017). `http://processors.wiki.ti.com/index.php/PRU_Assembly_Reference_Guide`.

[14] E. Kofman and J.H. Braslavsky. Level crossing sampling in feedback stabilization under data-rate constraints. In *Decision and Control, 2006 45th IEEE Conference on*, pages 4423–4428, Dec 2006.

[15] Christoph Lameter. Numa (non-uniform memory access): An overview. *Queue*, 11(7):40:40–40:51, July 2013.

[16] Daniel Lehmann and Jan Lunze. Event-based control using quantized state information. In *Proc. of the 2nd IFAC Workshop on Distributed Estimation and Control in Networked Systems (NecSys'10)*, pages 1–6, Sep. 2010.

[17] Lichun Li, Xiaofeng Wang, and Michael Lemmon. Stabilizing bit-rates in quantized event triggered control systems. In *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control*, pages 245–254. ACM, 2012.

[18] Jan Lunze and Daniel Lehmann. A state-feedback approach to event-based control. *Automatica*, 46(1):211 – 215, 2010.

[19] A. Matveev and A. Savkin. Multirate stabilization of linear multiple sensor systems via limited capacity communication channels. *SIAM Journal on Control and Optimization*, 44(2):584–617, 2005.

[20] A. Matveev and A. Savkin. An analogue of Shannon information theory for detection and stabilization via noisy discrete communication channels. *SIAM Journal on Control and Optimization*, 46(4):1323–1367, 2007.

[21] Andrew McPherson and Victor Zappi. An environment for submillisecond-latency audio and sensor processing on beaglebone black. In *Audio Engineering Society Convention 138*. Audio Engineering Society, 2015.

[22] Derek Molloy. *Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux*. John Wiley & Sons, 2014.

[23] E. Monmasson and M. N. Cirstea. Fpga design methodology for industrial control systems — a review. *IEEE Transactions on Industrial Electronics*, 54(4):1824–1842, Aug 2007.

[24] Girish N. Nair and Robin J. Evans. Exponential stabilisability of finite-dimensional linear systems with limited data rates. *Automatica*, 39(4):585 – 593, 2003.

[25] G.N. Nair. A nonstochastic information theory for communication and state estimation. *Automatic Control, IEEE Transactions on*, 58(6):1497–1510, June 2013.

[26] G.N. Nair and R.J. Evans. Communication-limited stabilization of linear systems. In *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, volume 1, pages 1005 –1010, 2000.

[27] Justin Pearson. *Beaglebone Motor Demo*, 2017 (accessed Feb 13, 2017). `https://github.com/justinpearson/Beaglebone-Motor-Demo`.

[28] Justin Pearson. *Beaglebone drives a motor from the command line*, 2017 (accessed Jan 27, 2017). `https://github.com/justinpearson/Beaglebone-motor-from-command-line`.

[29] Justin Pearson, João Pedro Hespanha, and Daniel Liberzon. Quasi-optimality of event-based encoders. In *Proc. of the 54nd Conf. on Decision and Contr.*, pages 4800–4805, Dec. 2015.

[30] Justin Pearson, João Pedro Hespanha, and Daniel Liberzon. Control with minimal cost-per-symbol encoding and quasi-optimality of event-based encoders. *IEEE Trans. on Automat. Contr.*, 62(5):2286–2301, May 2017.

[31] Jan Reineke, Björn Wachter, Stefan Thesing, Reinhard Wilhelm, Ilia Polian, Jochen Eisinger, and Bernd Becker. A Definition and Classification of Timing Anomalies. In Frank Mueller, editor, *6th International Workshop on Worst-Case Execution Time Analysis (WCET'06)*, volume 4 of *OpenAccess Series in Informatics (OASIcs)*, Dagstuhl, Germany, 2006. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[32] Stefano Rinaldi, Paolo Ferrari, and Matteo Loda. Synchronizing low-cost probes for iec61850 transfer time estimation. In *Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS), 2016 IEEE International Symposium on*, pages 1–6. IEEE, 2016.

[33] H. Sanchez, B. Kuttanna, T. Olson, M. Alexander, G. Gerosa, R. Philip, and J. Alvarez. Thermal management system for high performance powerpc/sup tm/ microprocessors. In *Proceedings IEEE COMPCON 97. Digest of Papers*, pages 325–330, Feb 1997.

[34] Alan J. Smith. Disk cache — miss ratio analysis and design considerations. *ACM Trans. Comput. Syst.*, 3(3):161–203, August 1985.

[35] Daniel J. Sorin, Mark D. Hill, and David A. Wood. A primer on memory consistency and cache coherence. *Synthesis Lectures on Computer Architecture*, 6(3):1–212, 2011.

[36] Per Stenström, Truman Joe, and Anoop Gupta. Comparative performance evaluation of cache-coherent numa and coma architectures. *SIGARCH Comput. Archit. News*, 20(2):80–91, April 1992.

[37] P. Tabuada. Event-triggered real-time scheduling of stabilizing control tasks. *Automatic Control, IEEE Transactions on*, 52(9):1680 –1685, sept. 2007.

[38] P. Tallapragada and N. Chopra. On co-design of event trigger and quantizer for emulation based control. In *American Control Conference (ACC), 2012*, pages 3772–3777, June 2012.

[39] P. Tallapragada and J. Cortes. Event-triggered stabilization of linear systems under bounded bit rates. *ArXiv e-prints*, May 2014.

[40] S. Tatikonda and S. Mitter. Control under communication constraints. *Automatic Control, IEEE Transactions on*, 49(7):1056 – 1068, july 2004.

[41] Sekhar C. Tatikonda. *Control under communication constraints*. PhD thesis, Massachusetts Institute of Technology, 2000.