A QUANTITATIVE FRAMEWORK FOR LAYERED MULTIRATE CONTROL

Toward a Theory of Control Architecture

NIKOLAI MATNI, AARON D. AMES, and JOHN C. DOYLE

PHOTO CREDIT: CALTECH

Complex engineered and natural control systems, such as those used in robotics, the power grid, human sensorimotor control, and the Internet, are characterized by needing to operate robustly and reliably across many spatiotemporal scales despite being implemented using highly constrained hardware and software. Remarkably, a universal design pattern centered around *layered control architectures* (*LCAs*) has emerged to address these challenges across vastly different domains. These LCAs are the central object of study of this article (see "Summary").

INTRODUCTION

Before proposing a broad definition of LCAs, we consider a familiar representative example from aerospace engineering, namely, the widely used guidance, navigation, and control (GNC) approach to aircraft control. Here, the overall task of flying an aircraft from an initial location to a goal location is decomposed into tractable subproblems, as illustrated in Figure 1, taken from the Apollo mission documentation [1]:

- *Guidance* determines a desired trajectory from the aircraft's current location to a goal location, in addition to nominal control actions, such as changes in forward and rotational velocity, for following the desired trajectory.
- *Navigation* is tasked with estimating the aircraft's state from onboard sensors, such as accelerometers and gyroscopes, and external signals, such as GPS.
- *Control* applies forces directly to the aircraft via actuators, for example, steering, thrust and aileron deflection, in order to execute the

Digital Object Identifier 10.1109/MCS.2024.3382388
Date of current version: 24 May 2024

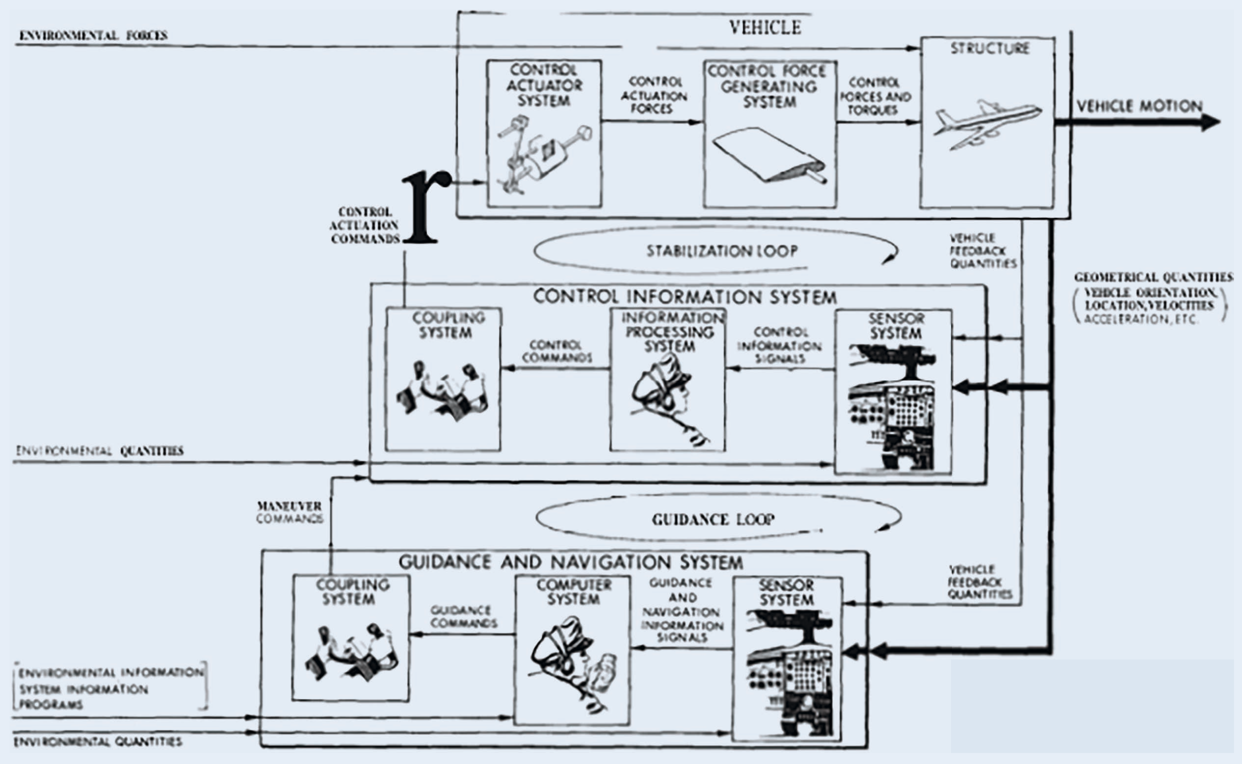
1066-033X/24©2024IEEE

trajectory planned by the guidance, all while maintaining aircraft stability.

We highlight some salient features of the GNC approach that we aim to capture in our broader theory of LCAs. The first and most important aspect is that an overall complex task (aircraft control) is decomposed into modular subtasks (GNC) of different complexity that operate at different frequencies over different spatiotemporal resolutions. These control modules, or as we call them, *layers*, are allowed to interact but only via well-defined interfaces. For example, the control layer must operate at a high frequency, as it is tasked with stabilizing the unstable aircraft dynamics about a nominal trajectory in the face of an uncertain and dynamic environment and hence is limited to simple feedback laws that can be implemented in real time [for example, proportional derivative (PD) control or a linear quadratic regulator (LQR)]. In contrast, the guidance layer, which must contend with vast spatiotemporal scales in planning an aircraft's route, typically issues commands at a much slower frequency than the control layer, as it must solve a longer-horizon trajectory planning problem. Despite this modularization, the layers are nevertheless coupled via the exchange of a reference trajectory from guidance to control and a tracking error from control to guidance. Enabling both guidance and control is the navigation layer, which is responsible for aircraft state estimation.

Summary

This article focuses on the need for a rigorous theory of *LCAs* for complex engineered and natural systems, such as power systems, communication networks, autonomous robotics, bacteria, and human sensorimotor control. All deliver extraordinary capabilities, but they lack a coherent theory of analysis and design, partly due to the diverse domains across which LCAs can be found. In contrast, there is a core universal set of control concepts and theory that applies very broadly and accommodates necessary domain-specific specializations. However, control methods are typically used only to design algorithms in components within a larger system designed by others, typically with minimal or no theory. This points toward a need for natural but large extensions of robust performance, from control to the full decision and control stack. It is encouraging that the successes of extant architectures, from bacteria to the Internet, are due to strikingly universal mechanisms and design patterns. This is largely due to convergent evolution by natural selection and not intelligent design, particularly when compared with the sophisticated design of components. Our aim here is to describe the universals of architecture and sketch tentative paths toward a useful design theory.



ENVIRONMENTAL FORCES
VEHICLE
CONTROL ACTUATOR SYSTEM
CONTROL ACTUATION FORCES
CONTROL FORCE GENERATING SYSTEM
CONTROL FORCES AND TORQUES
STRUCTURE
VEHICLE MOTION
CONTROL ACTUATION COMMANDS
STABILIZATION LOOP
VEHICLE FEEDBACK QUANTITIES
GEOMETRICAL QUANTITIES
VEHICLE ORIENTATION, LOCATION, VELOCITIES ACCELERATION, ETC.
CONTROL INFORMATION SYSTEM
COUPLING SYSTEM
CONTROL COMMANDS
INFORMATION PROCESSING SYSTEM
CONTROL INFORMATION SIGNALS
SENSOR SYSTEM
ENVIRONMENTAL QUANTITIES
MANEUVER COMMANDS
GUIDANCE LOOP
GUIDANCE AND NAVIGATION SYSTEM
COUPLING SYSTEM
GUIDANCE COMMANDS
COMPUTER SYSTEM
GUIDANCE AND NAVIGATION INFORMATION SIGNALS
SENSOR SYSTEM
VEHICLE FEEDBACK QUANTITIES
ENVIRONMENTAL INFORMATION SYSTEM INFORMATION PROGRAMS
ENVIRONMENTAL QUANTITIES

FIGURE 1 The GNC architecture used for Apollo missions (taken from [1]).

Despite the surprising universality of LCAs, they have yet to be a central object of study within the systems and controls community.

A Model LCA

This article seeks to initiate a *quantitative study of LCAs*, such as the one described previously. To ground our discussion of LCAs, we begin with the "model LCA" in Figure 2, which is composed of three layers, which broadly decompose across timescales and complexity/flexibility (by convention, we place slower, more complex layers "higher" in the stack and faster, more rigid layers "lower"):

» *Decision making*: The top layer operates at the slowest frequency of the architecture but is tasked with making complex logical decisions. Primarily the domain of semantic logic and other discrete decision-making tools, the decision-making layer establishes mission objectives (for example, which locations to visit via goal waypoints) and other system actions (for example, delivering a payload and exchanging or collecting information). In the context of GNC, this would be a higher "mission layer" that specifies the goal location.

» *Trajectory planning*: The intermediate layer, sitting between decision making and feedback control, operates at a moderate frequency to generate trajectories that accomplish the mission objectives specified by the decision-making layer. Typical techniques employed at this layer include optimization-based [for example, model predictive control (MPC) and mixed-integer programming] and sampling-based (for example, rapidly exploring random tree search) methods. The generated trajectories, which are constrained to satisfy the mission objectives specified by the decision-making layer, are transmitted to the feedback control layer. This is precisely the guidance layer in GNC.

» *Feedback control*: The bottom layer operates at the fastest frequency of the architecture and is tasked with tracking the trajectories generated by the planning layer. This layer is the home of feedback control, and while offline computation to synthesize control gains may be sophisticated and expensive, online evaluation is typically constrained to be simple, fast, and rigid. In addition to ensuring that the system tracks the desired trajectory, feedback control also provides robustness to high-frequency and dynamic disturbance processes. This is the control layer in GNC.

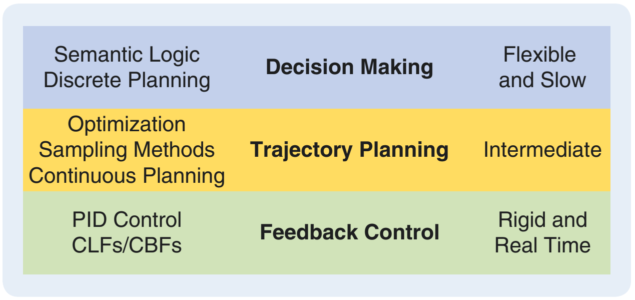


FIGURE 2 LCAs are ubiquitous across natural and engineered systems. We seek to initiate a quantitative study of LCAs based on the illustrated three-layer abstraction. PID: proportional-integral-derivative; CLF: control Lyapunov function; CBF: control barrier function.

A small note on terminology is in order before proceeding. Although the word *feedback* appears only in the bottom layer, it should be understood that some degree of feedback, either implicit or explicit, is present at all layers. For example, if trajectory planning is implemented using MPC, implicit feedback is provided by measuring the current system state. Thus, we ask the reader to interpret the use of the word feedback as indicating *real-time explicit feedback control* unless described otherwise.

This architectural pattern or similar ones, which should be familiar to control theorists, appear consistently and broadly across domains despite both extreme diversity in the systems on which they are deployed and the remarkable advances in sensing, actuation, and computation that have occurred over the past decades. Despite the surprising universality of LCAs, they have yet to be a central object of study within the systems and controls community. This article is motivated by this current gap in the literature.

A Brief Overview of Control Architecture Research

While we defer more detailed literature reviews to appropriate sections, we pause to highlight that this article builds upon and is inspired by a rich literature, both academic and industrial, on process control and automation architecture. Work providing a qualitative perspective about control architectures can be found in [2] and [3]. While these works place a heavier emphasis on industrial applications and implementations, such as the use of programmable logic controllers to implement distributed control systems, they also touch upon topics core to this article. An interesting observation is that both papers acknowledge the importance of control architecture while also recognizing its mercurial and difficult-to-define nature. Although different terminology is used, a layered and multirate perspective is provided in both. Indeed, using MPC for planning and simple feedback control (PD control) for tracking is identified as a common design pattern and is one that we revisit in great detail in the sequel. We view these important qualitative perspectives as complementary to the frameworks we propose and as further supporting the need for a more rigorous quantitative framework for reasoning about LCAs. Domain-specific work centered around control architecture can be found in [4] for smart grid applications, in [5] for cyberphysical system applications, and in [6] and [7] for Internet congestion control. Once again, we see the key themes of this manuscript, such as layered multirate control implemented using diverse components, discussed. For example, the templates proposed in [4] can be directly mapped to the proposed layered strategies in LCAs via optimal control decomposition. Lee et al. [5] propose a five-layer architecture (called the 5C architecture), with each layer having different complexity and spatiotemporal scope, and initial quantitative methodologies for layering as optimization decomposition can be found in [6] and [7]. This latter perspective serves as a key starting point for the framework proposed in this article. Finally, we note that although not the subject of this article, an important enabling technology for control architecture design will inevitably be appropriate *modeling languages and frameworks*, which may, for instance, be inspired by or build upon systems modeling language [8].

Article Organization

The rest of the article is broadly organized into three parts. In the first part, which includes the "LCAs via Optimal Control Decomposition" and "LCAs for Robotic Systems" sections, we propose a framework for *deriving* LCAs via optimal control decomposition. We then provide concrete instantiations of LCAs for robotic systems to illustrate the already impressive practical impact of layered control system design. In the second part, composed of the "Architecture Design as Multicriterion Optimization" and "A Case Study in Sensorimotor Control" sections, we propose an alternative perspective and frame architecture design as multicriterion optimization. A key takeaway of this section is that matching diversity across layers with diversity in control tasks can lead to LCAs that perform better than any individual layer could on its own. We illustrate these concepts in the section "A Case Study in Sensorimotor Control." Finally, in the third part, which includes the "Key Concepts in Control Architecture" section, we indulge in a more speculative discussion and introduce qualitative definitions of what we believe to be other key concepts in control architecture. Finally, we end with the "Conclusions" section.

LCAs VIA OPTIMAL CONTROL DECOMPOSITION

We propose a minimal quantitative framework for deriving and reasoning about LCAs, such as those illustrated in Figure 2. Our starting point is a control policy synthesis problem that captures the key ingredients of modern complex systems that LCAs have evolved to address, namely, 1) the mix of discrete/logical decision making with continuous dynamics and control and 2) the diversity in timescales at which different layers of a system (and its environment) evolve. Inspired by the layering as optimization decomposition [6], [7] approach to layered architectures, originally applied to network congestion control, our strategy is to systematically decompose the overall synthesis problem into tractable subproblems, each associated with a specific layer.

We consider *specifications* that the system must meet and *safety constraints* that the system must obey. We restrict ourselves to specifications expressed using formal logic, although alternative formulations are certainly possible. These specifications describe system goals; for example, in robotic applications, such a goal might be to navigate to a target or to perform a household task. Safety constraints, in contrast, are often expressed in terms of set membership constraints on a system's physical state; for example, in aerospace applications, such safety constraints may be expressed in terms of state/input inequality constraints enforcing the aerodynamic flight envelope. Design problems also typically include auxiliary performance objectives (for example, fuel efficiency, speed, and robustness), which are optimized subject to the specification and safety constraints. The goal, then, is to find the most efficient system design, as measured by the auxiliary performance objectives, that meets the system specifications and safety constraints.

Our framework thus centers around an *overall synthesis problem* that seeks to find a control policy that satisfies high-level specifications, subject to system dynamics as well as state and input constraints. In the interest of clarity, we omit auxiliary performance objectives in the initial formulation but highlight natural ways in which they can be incorporated throughout. After defining the overall synthesis problem, we show that through suitable *relaxations and decompositions*, 1) the three-layer architecture described previously can be derived, and 2) familiar optimization-based trajectory planning and feedback control algorithms emerge naturally in an attempt to minimize the errors induced by these relaxations and decompositions.

Notation

We use subscripts to denote continuous time; for example, x_t is the state x at time $t \in \mathbb{R}$. We use parenthesized numbers to denote discrete time; for example, $x(k)$ is the state x at discrete-time step $k \in \mathbb{N}$. We use boldface to denote infinite-horizon signals in both continuous and discrete time; that is, $\boldsymbol{x} = (x_t)_{t \geq 0}$ or $\boldsymbol{x} = (x(0), x(1), \ldots)$, depending on the context. We use the notation $x(T_1 : T_2)$ to denote finite-horizon discrete-time signals; that is, $x(T_1 : T_2) = (x(T_1), x(T_1+1), \ldots, x(T_2))$. Finally, we use parentheses to concatenate two vectors; that is, if $a \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$, then $c = (a, b) \in \mathbb{R}^{n+m}$.

The Overall Synthesis Problem

We assume that the system specifications are defined using a form of temporal logic, with linear temporal logic (LTL) and signal temporal logic (STL) being the most commonly used in the controls community; in the sequel, we use *TL to denote such general temporal logics. In particular, we let the system goals be specified by a given *TL formula φ defined over a finite set $\mathcal{AP}$ of atomic propositions.

The system state evolves according to the continuous-time dynamics

$$\Sigma: \quad \dot{x}_t = f(x_t, u_t) \tag{1}$$

where $x_t \in \mathbb{R}^n$ is the system state and $u_t \in \mathbb{R}^m$ is the control input. To simplify exposition, we assume nominal dynamics without any model uncertainty or process noise. Of course, real systems are subject to both, and how to systematically account for such uncertainty in LCAs remains an important open problem (see the "Robust LCAs" section). The state and control inputs are subject to safety constraints of the form $x_t \in \mathcal{X} \subseteq \mathbb{R}^n$ and $u \in \mathcal{U} \subseteq \mathbb{R}^m$.

We also define, for a suitable sampling time τ, the corresponding discrete-time model

$$\Sigma_d: \quad x(k+1) = f_d(x(k), u(k)) \tag{2}$$

where $x(k) = x_{k\tau}$, $u(k) = u_{k\tau}$, and f_d is a discretization of the continuous-time dynamics f. Albeit somewhat cumbersome, we introduce both continuous- and discrete-time dynamics to highlight the *multirate* nature of typical LCAs, wherein the decision-making, trajectory planning, and feedback control layers all operate at different *loop rates*; that is, each layer recomputes or updates its action at a different frequency. Where the switch from continuous- to discrete-time models is made in the LCA is often subject to computational constraints and loop rate requirements, which are in turn dictated by system specifications, safety constraints, and dynamics; see the "Multirate LCAs" and "Continuous-Time LCAs" sections. Nevertheless, a common design pattern, which we adopt here, is to use continuous-time models for real-time feedback control (to emphasize fast loop rates) and discrete-time models for both trajectory planning and decision making.

To verify the satisfaction of the *TL specifications φ, we assume the existence of a *labeling* function $L: \mathcal{X} \to 2^{\mathcal{AP}}$ that associates a *label* encoding the TRUE atomic propositions at every state $x \in \mathcal{X}$. Finally, we define the trace, or run, of system Σ under a control input sequence $\boldsymbol{u}$ to be the sequence

$$\xi(\boldsymbol{x}, \boldsymbol{u}) := (L(x(0)), u(0))((L(x(1)), u(1))\ldots$$

where the signals $\boldsymbol{x} = (x(0), x(1), \ldots)$, $\boldsymbol{u} = (u(0), u(1), \ldots)$ are discrete-time trajectories from the sampled system Σ_d. If such a trace satisfies the specification φ, we write $\xi(\boldsymbol{x}, \boldsymbol{u}) \vDash \varphi$.

We can now finally pose the overall synthesis problem as finding a possibly time-varying state feedback policy $u_t: \mathcal{X} \to \mathcal{U}$ such that

$$\begin{aligned} &\xi(\boldsymbol{x}, \boldsymbol{u}) \vDash \varphi, \\ &\quad \dot{x}_t = f(x_t, u(x_t)),\ x_0 \text{ given} \\ &\quad x_t \in \mathcal{X},\ u_t(x_t) \in \mathcal{U} \quad \forall t \geq 0. \end{aligned} \tag{3}$$

Example 1 (Running Example: Robot Navigation)

We use a simple robot navigation problem as a running example to illustrate the concepts introduced in this section (see Figure 3). Suppose the system is a *Dubins' car* (also called a *unicycle*) with dynamics

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{4}$$

and that the state and input constraints are the simple box constraints

$$\mathcal{X} = \left\{(x_1, x_2, \theta) \,\middle|\, \max\{|x_1|, |x_2|\} \leq 1, |\theta| \leq \frac{\pi}{2}\right\}$$
$$\mathcal{U} = \{(u_1, u_2) \,|\, \max\{|u_1|, |u_2|\} \leq 1\}.$$

To specify the system goals, we define the sets $\mathcal{X}_1 = \{(x_1, x_2, \theta) \,|\, x_1^2 + x_2^2 \leq 0.1^2\}$ and $\mathcal{X}_2 = \{(x_1, x_2, \theta) \,|\, x_1 \geq 0.9, x_2 \geq 0.9\}$. The task specification is for the robot to first visit set $\mathcal{X}_1$ and then visit $\mathcal{X}_2$, that is, to go to a small circle in the center of the space and then go to the upper-right corner (see Figure 4). This can be expressed in LTL via the specification

$$\varphi = F\left(\mathcal{X}_1 \wedge F\mathcal{X}_2\right) \tag{5}$$

where $\wedge$ and F are the `and` and `eventually` atomic propositions, respectively.

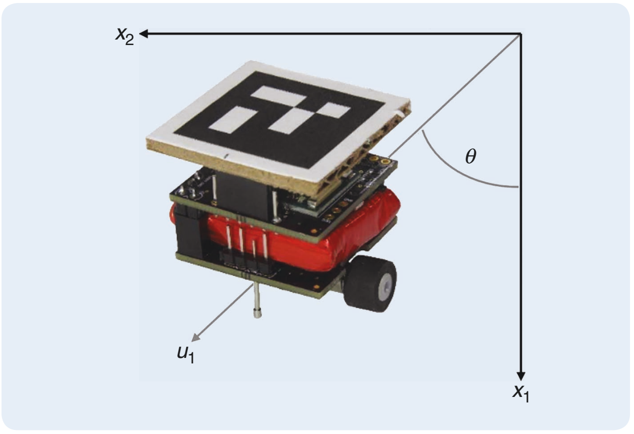


FIGURE 3 A mobile robot (the GritBot) modeled as a Dubins' car. The GritBot utilizes an LCA to allow for the safe implementation of user algorithms [9].

Layering via Problem Decomposition and Relaxation

Toward our goal of deriving an LCA, we strategically rewrite the global problem (3) by introducing redundant variables and subsequently relaxing consistency constraints among these redundant variables to allow for modularization across layers. We emphasize that while problem (3) is fundamental—in that it is dictated by the physics of the system as well as the specification and safety constraints of the problem—and that the proposed framework of problem decomposition and relaxation is foundational to a theory of LCAs, the particular realizations of these ideas that follow are *architectural design choices*. They are by no means unique, although they are chosen to be broadly representative of approaches taken in the literature.

Decision-Making Layer

The *TL specifications φ are defined over a discrete state and input spaces, whereas the global problem (3) is defined over continuous state and input spaces. Toward bridging this gap, we assume that the continuous state space $\mathcal{X}$ admits a partitioning $\mathcal{S}$. Ideally, such a partition is such that for any cell $s \in \mathcal{S}$, we have that $L(x) = L(y)$ for all $x, y \in s$; that is, all states in a partition satisfy the same atomic propositions and are "semantically equivalent." However, if the partition of the state space is coarse, this may not hold true, and partitioning may introduce conservatism. It is natural to consider this partition as defining a discrete state space for a Markov decision process (MDP), and with slight abuse of notation, we use $s \in \mathcal{S}$ to denote such a discrete state as well. Similarly, a discrete action space $\mathcal{A}$ is induced by this partition and the system Σ, allowing us to define the MDP dynamics $s(i+1) = f_{\text{MDP}}(s(i), a(i))$. The MDP dynamics evolve in discrete time, with dynamics defined to be consistent with traces of the sampled system Σ_d. However, the MDP dynamics typically correspond to a *subsampling* of Σ_d; that is, $s(i)$ is determined by $x(k\delta)$, for $\delta \in \mathbb{N}_+$, a discrete sampling time. In general, this subsampling may be irregular, for example, if each time step of the MDP is associated with a change in a discrete state, but note that our model can capture this phenomenon by including the null action $\varnothing$ such that $s(i+1) = f_{\text{MDP}}(s(i), \varnothing) = s(i)$ for all $s(i) \in \mathcal{S}$.

Example 2 (Running Example: Robot Navigation)

A possible discrete state space $\mathcal{S}$, with $|\mathcal{S}| = 14^2$, is displayed in Figure 4(b), where each square corresponds to a discrete state $s \in \mathcal{S}$. This induces corresponding discrete actions $\mathcal{A} = \{\uparrow, \downarrow, \leftarrow, \rightarrow, \varnothing\}$ and MDP dynamics corresponding to those of a typical grid world problem (for clarity of exposition, we assume that diagonal movement is not allowed). In particular, letting $s = (s_1, s_2)$ denote the (x_1, x_2) grid position, we then have

$$(s_1(i+1), s_2(i+1)) = \begin{cases} (s_1(i), s_2(i)) & a(i) = \varnothing \\ (s_1(i), s_2(i)+1) & a(i) = \uparrow \\ (s_1(i), s_2(i)-1) & a(i) = \downarrow \\ (s_1(i)+1, s_2(i)) & a(i) = \rightarrow \\ (s_1(i)-1, s_2(i)) & a(i) = \leftarrow \end{cases} \tag{6}$$

where we show only dynamics for allowable actions a that would not take the system outside of the state space $\mathcal{S}$. Note that due to the resolution of state-space discretization, the set $\mathcal{S}_1$ is an underapproximation of the corresponding continuous set $\mathcal{X}_1$.

Given this definition and toward the goal of isolating a decision-making layer, we introduce redundant discrete planning variables $\boldsymbol{s}$ and $\boldsymbol{a}$, which are subject to the MDP dynamics f_{MDP}:

$$\begin{aligned} &\xi(\boldsymbol{s}, \boldsymbol{a}) \vDash \varphi, s(i+1) = f_{\text{MDP}}(s(i), a(i)), s(0) \ni x(0) \\ &x(k) \in s(\lfloor k/\delta \rfloor), x(k) = x_{k\tau}, \forall k \in \mathbb{N} \\ &\dot{x}_t = f(x_t, u_t(x_t)), x_0 \text{ given} \\ &x_t \in \mathcal{X}, u_t(x_t) \in \mathcal{U}, \forall t \geq 0. \end{aligned} \tag{7}$$

Slightly abusing notation, here, the first line replaces the trace over continuous variables $(\boldsymbol{x}, \boldsymbol{u})$ with one defined over discrete variables $(\boldsymbol{s}, \boldsymbol{a})$ satisfying the MDP dynamics, and we write $s(0) \ni x(0)$ to emphasize that the discrete-state initial condition $s(0)$ must be consistent with the continuous-state

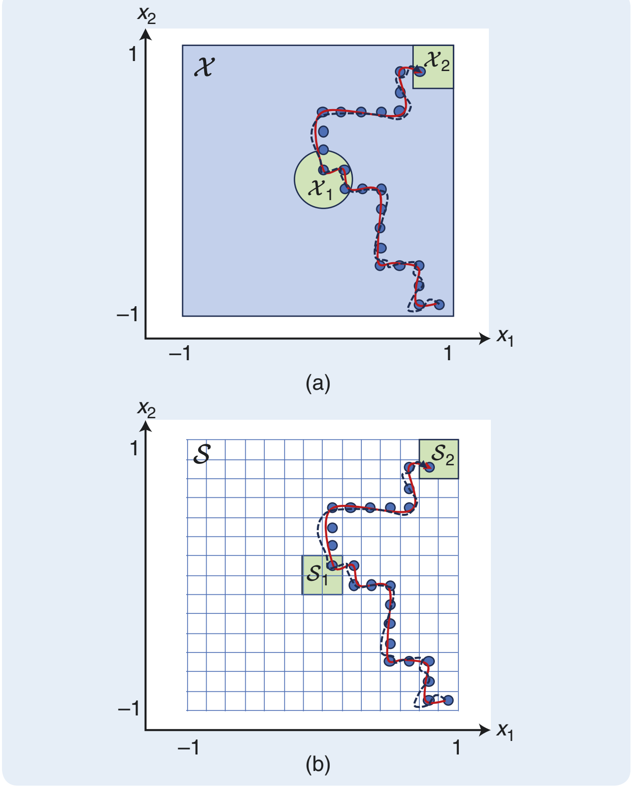


FIGURE 4 (a) Continuous and (b) discrete state space in the robot navigation running example. We see that due to the resolution of the partition, the set of discrete states $\mathcal{S}_1$ is an underapproximation of the corresponding first objective set $\mathcal{X}_1$. The blue circles illustrate the discrete state-space trace defined by the decision-making layer; the red arrow shows the reference trajectory generated by the planning layer, using the blue circles as waypoint constraints; and the black dashed line is the actual system evolution, as driven by the feedback control layer on the continuous-time dynamics.

initial condition $x(0)=x_0$. The second line enforces consistency between the remainder of the discrete plan of the first line and the underlying continuous control system Σ. In particular, the constraint $x(k)\in s(\lfloor k/\delta \rfloor)$ ensures that the continuous-state discrete-time trace $(x(0), x(1), \ldots)$ induces the correct discrete-state trace $(s(0), s(1), \ldots)$. Similarly, the constraint $x(k)=x_{k\tau}$ ensures consistency between the continuous-state discrete-time trace $(x(0), x(1), \ldots)$ and the continuous-state trajectory $x_{t\geq 0}$.

Our first relaxation is to decompose problem (7) by isolating the discrete planning problem

$$\xi(\boldsymbol{s}, \boldsymbol{a}) \vDash \varphi,\ s(i+1)=f_{\mathrm{MDP}}(s(i), a(i)),\ s(0) \ni x(0) \tag{8}$$

which we view as the decision-making layer problem. Here, the decision-making layer assumes that any discrete plan $(\boldsymbol{s}, \boldsymbol{a})$ can be realized by the underlying continuous control system Σ. Solving the decision layer problem (8), for example, using *TL synthesis methods, yields a discrete state and action plan $(\boldsymbol{s}, \boldsymbol{a})$ that satisfies the specification φ. We show next how this high-level plan can be used to define a trajectory planning problem using a similar decomposition and relaxation technique. We note that *TL synthesis methods are computationally expensive, and hence, replanning at this layer is typically done at a slower timescale, with the faster lower layers used to mitigate unexpected disturbances in the interim.

Example 3 (Running Example: Robot Navigation)
The decision-making layer problem for this example becomes one of finding a state/action trace $(\boldsymbol{s}, \boldsymbol{a})$ that satisfies the specification (5), subject to the discrete-time discrete-state dynamics (6).

Trajectory Planning Layer

Removing the decision-making layer problem (8) from problem (7) leaves us with

$$\begin{aligned} & x(k)\in s(\lfloor k/\delta \rfloor),\ x(k)=x_{k\tau},\ \forall k\in\mathbb{N} \\ & \dot{x}_t = f(x_t, u_t(x_t)),\ x_0 \text{ given} \\ & x_t\in\mathcal{X},\ u_t(x_t)\in\mathcal{U},\ \forall t\geq 0. \end{aligned} \tag{9}$$

Toward the goal of isolating a trajectory planning layer, we introduce a redundant continuous-state discrete-time trajectory variable $\boldsymbol{r}=(r(0), r(1), \ldots)$, constrained to be consistent with the discrete-time state $\boldsymbol{x}=(x(0), x(1), \ldots)$. The resulting strategically rewritten *equivalent* problem is then given by

$$\begin{aligned} & r(k)\in s(\lfloor k/\delta \rfloor),\ r(k)\in\mathcal{X} \\ & r(k)=x(k),\ x(k)=x_{k\tau},\ \forall k\in\mathbb{N} \\ & \dot{x}_t = f(x_t, u_t(x_t)),\ x_0 \text{ given} \\ & x_t\in\mathcal{X},\ u_t(x_t)\in\mathcal{U},\ \forall t\geq 0. \end{aligned} \tag{10}$$

The first line isolates a trajectory generation problem, defined now over the reference trajectory variable $r(k)$, ensuring that 1) the trajectory is consistent with the discrete plan, as enforced by $r(k)\in s(\lfloor k/\delta \rfloor)$, and 2) the trajectory is safe, as enforced by $r(k)\in\mathcal{X}$. As above, the second line enforces coupling between different layers: $r(k)=x(k)$ ensures that the trajectory and system states are consistent, and once again, $x(k)=x_{k\tau}$ ensures consistency between the discrete-time system Σ_d and the continuous-time system Σ.

Our next relaxation is to decompose problem (10) by isolating a middle-layer trajectory planning problem, which takes the form of the feasibility problem

$$r(k)\in s(\lfloor k/\delta \rfloor),\ r(k)\in\mathcal{X},\ \forall k\geq 0,\ r(0)=x(0). \tag{11}$$

Here, we drop all consistency constraints that $r(k)=x(k)=x_{k\tau}$ except for those enforcing the initial condition $r(0)=x(0)$, and a reference trajectory $\boldsymbol{r}$ that satisfies the state constraint $r(k)\in\mathcal{X}$ and specification constraints $r(k)\in s(\lfloor k/\delta \rfloor)$ is searched for. Due to this decomposition and relaxation of the constraints, the reference trajectory produced by solving feasibility problem (11) is not guaranteed to be dynamically feasible, and therefore, the tracking error between the true system state $x(k)$ and the planned trajectory $r(k)$ should be accounted for when enforcing the specification and safety constraints $r(k)\in\mathcal{X}\cap s(\lfloor k/\delta \rfloor)$. Letting $C(k) := \mathcal{X}\cap s(\lfloor k/\delta \rfloor)$ denote the intersection of the specification and safety constraints at discrete-time step k, define the tightened constraint set $\overline{C}(k)$, and replace the specification and safety constraints with $r(k)\in\overline{C}(k)$. How to appropriately tighten this constraint set depends on the feedback control layer, but once the tracking error has been characterized, standard tools can be used.

To promote reference trajectories that are easy to track by the underlying continuous system Σ, the feasibility problem (11) is often modified to produce approximately dynamically feasible solutions. For example, it is common to decompose the control policy into a feedforward term u_{ff}, depending on the reference trajectory, and a feedback term u_{fb}, depending on the system state (or more specifically, on the tracking error). For example, a typical such decomposition is to simply set $u_t(x_t)=u_{\mathrm{ff}}(r(\lfloor t/\tau \rfloor))+u_{\mathrm{fb}}(e_t)$ for tracking error $e_t := x_t - r(\lfloor t/\tau \rfloor)$). Another standard approach is to assume that the reference trajectory obeys simplified planning dynamics $r(k+1)=f_{\mathrm{plan}}(r(k), u_{\mathrm{ff}}(k))$. A common choice for these simplified planning dynamics is to use a *reduced-order* model defined by $y(k+1)=f_{\mathrm{rom}}(y(k), v(k))$, where $y\in\mathbb{R}^p$ and $v\in\mathbb{R}^s$, with $p\leq n$ and $s\leq m$. This case can be integrated into the proposed framework by replacing the consistency constraint $r(k)=x(k)$ with a reduced-order consistency constraint $y(k)=\pi_x(x(k))$, for $\pi_x:\mathbb{R}^n\to\mathbb{R}^p$, some projection map encoding the model order reduction. To distinguish this important special case, in the sequel, we reserve $r(k)$ for *full-order* reference trajectories, that is, reference trajectories defined over the entire state with both $r(k), x(k)\in\mathbb{R}^n$, and use $y(k)$ to denote reduced-order model states, as these are often used as *tracked outputs* at the feedback control layer.

Finally, we make the following additional modifications. First, toward employing a receding horizon control approach, we restrict the trajectory planning problem to be over a finite horizon N. Second, we encode additional desirable properties of the trajectory, such as smoothness, via a running cost function $C(r,u)$ and a terminal cost $C_N(r)$. Integrating these elements with those described previously yields the planning problem solved at discrete-time step k:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=k}^{k+N-1} C(r(i), u_{\text{ff}}(i)) + C_N(r(k+N)) \\ \text{subject to} \quad & r(i+1) = f_{\text{plan}}(r(i), u_{\text{ff}}(i)),\, r(k) = x(k) \\ & r(i) \in \overline{C}(i),\, i = k, k+1, \ldots, k+N. \end{aligned} \tag{12}$$

The planning problem (12) is typically solved and implemented in a receding horizon fashion, for example, via MPC, and thus is limited to being resolved at an intermediate frequency (that is, more frequently than the decision layer problem but less frequently than the feedback control layer).

We end by noting that this is but one approach to defining a planning problem given a discrete state plan $\boldsymbol{s}$. Alternative approaches consider, for example, loss functions that penalize deviations of the reference trajectory $\boldsymbol{r}$ from particular waypoints that are consistent with the discrete state plan $\boldsymbol{s}$. It is hopefully clear that problem (12) is equivalent to such an approach, up to hard/soft constraints.

Example 4 (Running Example: Robot Navigation)

By converting the state trace $(s(0), s(1), \ldots)$ computed by the decision-making layer from discrete (s_1, s_2) coordinates to continuous (x_1, x_2) coordinates, for example, by choosing the centroid of cell (s_1, s_2), these can be used to define a sequence of *waypoints* $((p_1(0), p_2(0)), (p_1(1), p_2(1)), \ldots)$, with $(p_1(i), p_2(i)) \in \mathbb{R}^2$, that can be used as *state constraints* within the planning layer. These waypoints are indicated with blue circles in Figure 4. In Figure 4(a), we illustrate their use as waypoints for continuous trajectory planning, and in Figure 4(b), we demonstrate their use as a feasible state trace in the discretized state space $\mathcal{S}$.

We formulate the planning problem using a reduced-order linear model composed of decoupled single-integrator dynamics in the x_1 and x_2 directions; that is, we set $y(k) = (y_1(k), y_2(k))$, $v(k) = (v_1(k), v_2(k))$, and $f_{\text{rom}}(y(k), v(k)) = (y_1(k) + \tau v_1(k), y_2(k) + \tau v_2(k))$. In this case, we are using a reduced-order model that projects out the angle θ and that introduces feedforward linear velocity inputs (v_1, v_2). We use the waypoints $((p_1(0), p_2(0)), (p_1(1), p_2(1)), \ldots)$ to define constraints on the trajectory of the form $|y_1(k) - p_1(\lfloor k/\delta \rfloor)| \le \Delta$ for Δ half the length of the square cells defining the discrete state space and idem for the x_2 coordinate; that is, we ask that the reference trajectory follow the sequence of discrete cells defined by the discrete-state trace $(s_1(i), s_2(i))$ but allowing appropriate time within each cell, as dictated by the sampling rate δ used at the decision-making layer.

We additionally impose smoothness and control effort penalties in the objective and constrain the reference trajectory to satisfy tightened state constraints [here, we assume that the feedback control layer can guarantee a tracking error of at most 0.05 in either of the (x_1, x_2) coordinates]. The resulting planning problem solved at time step k over a horizon N is then given by

$$\begin{aligned} \text{minimize} \quad & \sum_{i=k}^{k+N-1} \| y(i+1) - y(i) \|_2^2 + \| v(i) \|_2^2 \\ \text{subject to} \quad & y(i+1) = f_{\text{rom}}(y(i), v(i)) \\ & y(k) = (x_1(k), x_2(k)) \\ & |y_1(i) - p_1(\lfloor i/\delta \rfloor)| \le \Delta - 0.05 \\ & |y_2(i) - p_2(\lfloor i/\delta \rfloor)| \le \Delta - 0.05 \\ & y(i) \in \bar{\mathcal{X}} = \left\{ (x_1, x_2) \,\middle|\, \max_j |x_j| \le 0.95 \right\} \\ & i = k, k+1, \ldots, k+N. \end{aligned} \tag{13}$$

An illustrative example of the resulting reference trajectory is shown in red in Figure 4(a).

Real-Time Feedback Control Layer

Finally, we consider the feedback control layer. At discrete-time step k, given a solution $(r(k:k+N), u_{\text{ff}}(k:k+N))$ to the planning layer problem (12), we must contend with the remaining constraints, now truncated to the planning horizon N:

$$\begin{aligned} & r(i) = x(i),\, x(i) = x_{i\tau},\, i = k, k+1, \ldots, K \\ & \dot{x}_t = f(x_t, u_t(x_t)),\, x_0 \text{ given} \\ & x_t \in \mathcal{X},\, u_t(x_t) \in \mathcal{U},\, \forall t \in [k\tau, (k+N)\tau]. \end{aligned} \tag{14}$$

We relax this problem by 1) removing the state constraint $x_t \in \mathcal{X}$, as this is addressed within the planning problem (12), and 2) allowing for the state $\boldsymbol{x}$ to deviate from the reference trajectory $\boldsymbol{r}$, as the reference trajectory $\boldsymbol{r}$ is not expected to be dynamically feasible:

$$\begin{aligned} \text{minimize} \quad & \int_{k\tau}^{(k+N)\tau} \left(\| e_s \|_Q^2 + \| u_{\text{fb},s} \|_R^2 \right) ds \\ \text{subject to:} \quad & \dot{x}_t = f(x_t, u_{\text{ff},t} + u_{\text{fb},t}),\, x_0 \text{ given} \\ & u_{\text{ff},t} + u_{\text{fb},t} \in \mathcal{U}. \end{aligned} \tag{15}$$

In the above, for a positive semidefinite matrix P, we let $\|z\|_P^2 := z^T P z$, define the tracking error $e_t := x_t - r(\lfloor t/\tau \rfloor)$, and slightly abuse notation by letting $u_{\text{ff},t} = u_{\text{ff}}(\lfloor t/\tau \rfloor)$ be the zero-order holds of their corresponding discrete-time signal. This problem can be viewed as a "best effort" tracking controller.

Loop rate constraints lead to either offline computed feedback policies that approximately solve the problem (such as LQR tracking), to simple-to-implement approaches such as PD control, or to myopic simplifications that can be solved in real time via, for example, quadratic programming. All these approaches can be viewed as further relaxations of the above tracking problem (15). While widely used, the proposed relaxation (15) and its extensions typically lack tracking error guarantees. To address this concern, recent work has leveraged Lyapunov-based techniques to certify tracking error bounds, which in turn allow for a principled tightening of the constraints used in the planning layer. We highlight some of these techniques, as applied to robotic LCAs, in the next section.

Example 5 (Running Example: Robot Navigation)

While many feedback control approaches are possible, here, we take this opportunity to briefly introduce differential flatness-based control and show how it can be used in this context. To synthesize a feedback controller that tracks the reduced-order model reference trajectory $(y_t, v_t) = (y(\lfloor t/\tau \rfloor), v(\lfloor t/\tau \rfloor))$, we first identify the *flat outputs* [10] such that the state and inputs can be written as a function of these flat outputs and derivatives. For the unicycle dynamics, a flat output is $\xi = (x_1, x_2)$, as verified by the relationship

$$\begin{bmatrix} x_1 \\ x_2 \\ \theta \end{bmatrix} = \begin{bmatrix} \xi_1 \\ \xi_2 \\ \arctan\left(\frac{\dot{\xi}_2}{\dot{\xi}_1}\right) \end{bmatrix} =: x_\flat(\xi, \dot{\xi}) \tag{16}$$

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \sqrt{\dot{\xi}_1^2 + \dot{\xi}_2^2} \\ \frac{\dot{\xi}_1 \ddot{\xi}_2 - \dot{\xi}_2 \ddot{\xi}_1}{\dot{\xi}_1^2 + \dot{\xi}_2^2} \end{bmatrix} =: u_\flat(\xi, \dot{\xi}, \ddot{\xi}). \tag{17}$$

To synthesize a feedback controller, it is convenient to define the flat state $z := (\xi_1, \xi_2, \dot{\xi}_1, \dot{\xi}_2) = (x_1, x_2, \dot{x}_1, \dot{x}_2) \in \mathbb{R}^4$ and flat control input $a := (\ddot{\xi}_1, \ddot{\xi}_2) = (\ddot{x}_1, \ddot{x}_2) \in \mathbb{R}^2$, resulting in the linear dynamics

$$\dot{z}_t = \underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{A^c_{\text{rom}}} z_t + \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}}_{B^c_{\text{rom}}} a_t \tag{18}$$

for $z_t \in \mathbb{R}^4$ and $a_t \in \mathbb{R}^2$. With slight abuse of notation, we may then write $x_t = x_\flat(z_t)$ and $u_t = u_\flat(z_t, a_t)$ by making the appropriate identifications between $(\xi_t, \dot{\xi}_t, \ddot{\xi}_t)$ and (z_t, a_t). We can then synthesize a feedforward-plus-feedback policy using, for example, PD control by setting

$$a_t = \dot{v}_t - K^\flat_P((z_1, z_2)_t - (y_1, y_2)_t) - K^\flat_D((z_3, z_4)_t - v_t) \tag{19}$$

for appropriately tuned positive definite matrices $K^\flat_P$ and $K^\flat_D$. Note that in this case, the feedforward term $\dot{v}_t$ can be approximated via a finite difference; that is, $\dot{v}_t \approx (v(\lfloor t/\tau \rfloor) - v(\lfloor t/\tau \rfloor - 1))/\tau$.

We now discuss how to translate the flat state z_t and flat input a_t to a control input u_t composed of feedforward and feedback terms given current measurements of $z_t = (x_1, x_2, \dot{x}_1, \dot{x}_2)_t$. We define the feedforward term to be given precisely by the mapping from flat state and input to control input:

$$u_{\text{ff},t}(z_t, a_t) = u_\flat(z_t, a_t).$$

If the mapping $u_\flat$ exactly captures the system dynamics, then no additional feedback term will be required. However, in practice, the Dubins' car is often used as a reduced-order model for planning trajectories for more complex systems, such as quadrupeds [see Figure S1 in "Multirate LCAs in Practice" as well as the "Example 6 (Running Example: Robot Navigation)" section]. Thus, a feedback term to ensure that flat and actual states match is additionally required. One such option is, again, a PD controller:

$$u_{\text{fb},t}(z_t) = -K_P((x_1, x_2)_t - (z_1, z_2)_{t+\tau}) - K_D((\dot{x}_1, \dot{x}_2)_t - (z_3, z_4)_{t+\tau})$$

for positive definite matrices K_P and K_D. Note here that the flat look-ahead state $z_{t+\tau}$ is obtained by forward integrating the flat dynamics (18) with the a_t given as in (19) and $z_t = (x_1, x_2, \dot{x}_1, \dot{x}_2)_t$ obtained from hardware measurements. The final control input is then the sum of both the feedforward and feedback terms; that is,

$$u_t(z_t, a_t) = u_{\text{ff},t}(z_t, a_t) + u_{\text{fb},t}(z_t) \tag{20}$$

as suggested in the feedback control problem (15). A conceptual rendering of the resulting evolution of the actual system state is presented with a dashed black line in Figure 4.

Discussion

We obtained an LCA by introducing redundant variables and suitable relaxations to decompose the overall synthesis problem (3) into tractable decision-making, trajectory generation, and feedback control subproblems, as specified in (8), (12), and (15), respectively. This highlights another key feature of LCAs: they allow for intractable global problems to be decomposed into tractable subproblems, often with minimal loss in performance or efficiency. We note that the proposed decomposition is only one of certainly many approaches and is chosen to be consistent with the rest of the article. Indeed, while the above framework provides a more formal perspective on LCAs, it still has an element of "art" to it. In particular, *how to relax the overall synthesis problem* as well as *how to bridge the simplifications among the different layers* of the resulting architecture are still up to the designer. Nevertheless, by posing the problem in this way, there is a natural nested optimization structure that emerges that may allow for more principled methods of LCA design to be defined. We highlight next some key open questions and concepts that we do not treat in depth but that certainly deserve more investigation.

What About the Hardware?

Each layer described previously delineates a *functional component* of an overall decision and control stack. Equally important are the *physical substrates* used to implement these functional layers. For example, a motor used in a robotic system to actuate a joint is composed of different scales of components, ranging from circuit elements to microprocessors to motor components. While these physical substrates are closely related to the layers they implement, they are distinct; to make this explicit, we use the term *levels* for physical substrates and reserve layers for functional components. We expand on this idea, and introduce other key concepts of LCAs not touched upon here, in the "Key Concepts in Control Architecture" section. Furthermore, in the "Architecture Design as Multicriterion Optimization" section, we present a quantitative framework to inform how to choose diverse hardware to implement diverse functionality as a function of diversity in the control task at hand, and we instantiate this perspective in the "A Case Study in Sensorimotor Control" section.

How Many Layers Should There Be?

This section presents an approach to deriving an LCA with three layers, each operating at different spatiotemporal resolutions. While these three layers, namely, decision making, trajectory generation, and feedback control, are commonly found in complex engineered systems, this pattern is by no means the only one possible. Indeed, all the concepts introduced above can be applied recursively, leading to layers of layers. For example, there can be several layers of trajectory planning, operating at different loop rates, using different planning models, and planning over different horizons; see, for example, the "Continuous-Time LCAs" section. Similarly, nested control loops are a standard control design pattern that can be interpreted as different layers of real-time feedback control. While we present a framework for deriving layers given an overall problem formulation, we still lack quantitative tools for deciding how many layers there should be as well as what information should be exchanged among them. This is undoubtedly a key open question.

Multirate Control

In the above, we hint at the role of multirate control in LCAs that seeks to address implicit timing constraints. Low-layer feedback control (15) operates in (near) real time, trajectory generation (12) at a slower rate, and decision making (8) at a slower rate still. This suggests that ideas from singular perturbation analysis and timescale separation (see, for example, [11] and [12] and the references therein) may also be used to provide further rigor to the approach. We further explore multirate LCAs in robotic systems in the "Multirate LCAs" section.

Robust LCAs

We omit uncertainty due to process noise and modeling errors in our development. However, practical LCAs must be robust to these effects. Promising approaches to tackling uncertainty among layers include the use of robust Lyapunov certificates for guaranteeing the bounded tracking error of a reference trajectory by the feedback control layer (see, for example, the "Continuous-Time LCAs" section) and, more generally, the use of assume–guarantee contracts [13], [14]. These approaches are intuitive and effective, but it is nevertheless of interest to investigate whether such certificates can be derived by applying similar decompositions and relaxations to a robust overall synthesis problem that explicitly acknowledges uncertainty in its initial formulation.

Layered Sensing Architectures

We also emphasize that although our focus in this section has been on fully observed state feedback control problems, analogous layered decompositions for sensing and output feedback problems need to be developed. A promising starting point is to recognize that different sensors, ranging from semantically rich and complex sensors (for example, cameras and lidar) to simple single-output sensors [for example, inertial measurement units (IMUs) and gyroscopes], are naturally assigned to each of the decision-making (for example, computer vision and semantic segmentation), trajectory generation (for example, visual inertial odometry + simultaneous localization and mapping), and feedback (for example, IMUs) layers.

Learning in LCAs

The use of rich perceptual sensors, such as cameras, invariably introduces learning into the resulting LCAs, which is a topic we cannot hope to do justice within the scope of this article. This is, however, an exciting and important direction to be explored, with learning and data-driven techniques poised to make a significant impact in designing effective LCAs.

Related Work

The framework proposed above is inspired by a rich literature seeking to establish principles of LCA design, although it is not necessarily explicitly identified as such.

Layering as Optimization Decomposition

The layering as optimization decomposition (see [6] and [7] and the references therein) and the reverse/forward engineering (see [15] and [16] and the references therein) paradigms have been particularly fruitful in tackling Internet and power grid control problems, respectively. Both of these frameworks can be loosely viewed as using the dynamics of the system to implement a distributed optimization algorithm through vertical (layering) and horizontal (distributed) decomposition. These methods ensure that the state of the system converges to a setpoint that optimizes a utility function. These approaches can scale to large systems by taking advantage of the structure underlying the utility optimization problem and can simultaneously identify and guarantee stability around an optimal equilibrium point. Nevertheless, they do not explicitly consider optimal control, and in particular, transients, in their analysis, making them an important but incomplete first step toward a theory of LCAs.

Decision Making and Continuous Control

This line of work seeks to make explicit that although formal specifications are inherently discrete, in order to ensure that a system satisfies them, designers must account for continuous dynamics and control. One line of work seeks to reformulate *TL specifications into continuous control tasks through the use of control Lyapunov functions (CLFs) and control barrier functions (CBFs) [17], [18], [19]; see, for example, [20] and [21]. An alternative approach is to encode *TL constraints via mixed-integer linear constraints in robust/optimal control problems [22], [23], [24] or to abstract the continuous control problem into an uncertain finite-state MDP and use robust dynamic programming [25]. Closely related is the work of Fan et al. [26], wherein decision making is done via SAT-based trajectory planning methods, which solve a satisfiability problem over quantifier-free linear real arithmetic. Other representative works that explicitly acknowledge the inherently hybrid (discrete/continuous) nature of the decision-making and control problem, and that seek to bridge the gap in a principled way, include reactive planning approaches [27] and the use of CBFs for determining the magnitude of disturbance that a system can be subject to while still ensuring the satisfaction of STL specifications [28]. Barrier functions have also been applied to partially observable MDPs in the context of distribution temporal logic [29] and coherent risk measures (such as conditional value at risk) [30], [31] to enforce safety constraints at a planning level. More broadly, risk-aware planning and control is considered in [32], [33], and [34]. Implicit in all of the above is a layered architecture wherein the high-layer decision-making component operates on a discrete abstraction of the underlying continuous-time system, and similarly, the underlying continuous-time system implements planning/control layers in order to meet the plan specified by the top decision-making layer.

Trajectory Generation and Continuous Control

Approaches to dynamics-aware trajectory generation typically follow a classic two-phase approach, wherein first a graph is constructed whose nodes are collision-free configurations and whose edges correspond to feasible paths among these configurations; see, for example, [35] and the references therein. More recent approaches based on graphs of convex sets [36]; motion primitives [37], [38]; optimization-based methods [39]; control Lyapunov, barrier, and contraction metrics [40], [41], [42], [43], [44], [45]; and reachability techniques [46] have also been proposed to bridge the gap between low-layer fast-timescale control and middle-layer intermediate-timescale trajectory generation. The common theme in all these approaches is the use of simplified dynamics in the trajectory generation layer, allowing for fast replanning, and a low-layer feedback controller that provides certifiable guarantees on the tracking error. Finally, most closely related to the framework presented in the previous discussion are the results found in [47], [48], [49], and [50], wherein it is shown that a two-layer trajectory generation/feedback control LCA can be obtained by suitably relaxing consistency constraints between the state and reference trajectory. A key feature of this approach is that the trajectory planning problem is augmented with a tracking penalty regularizer that promotes dynamic feasibility of the synthesized reference signal.

LCAs FOR ROBOTIC SYSTEMS

LCAs have long found use in robotic systems; empirically, it is well known that this is the best (and arguably only) way to implement controllers in practice. Yet, despite this empirical evidence, there is very little analysis of LCAs. Conversely, while the control community applies rigorous approaches to controller synthesis, it is often applied only to a single layer. This points to a unique opportunity for the controls community: reverse engineering and analyzing the LCAs deployed on robotic systems that have proved useful in practice.

To put the central role of LCAs in robotic systems in context, one should first consider the hardware itself. A robotic system, broadly defined, typically consists of three main components: sensors used for perception, a central processor, and motor controllers used for driving actuators. Concrete examples of this include cameras mounted on a legged robot for localization and mapping and proximity sensors on a vehicle for advanced driver assistance. In this context, LCAs (as shown in Figure 5) are often deployed relative to these physical levels on hardware (see the "Key Concepts in Control Architecture" section for more examples of how levels and layers interact in LCAs). Perception leads to a decision-making layer operating at a discrete/semantic level of abstraction, the central processor leads to reference signal generation using reduced-order models, and, finally, at the actuator level, real-time algorithms instantiate feedback control. (The robotics literature refers to these different layers as high-level, midlevel, and low-level control. We, however, argue that these are better viewed through the lens of LCAs and hence use the layered terminology defined in the previous sections.)

This section gives concrete instantiations of LCAs for robotic systems. We start by defining robotic system dynamics and subsequently work our way up the layers of a typical instantiation of LCAs for robotic control. A goal of this section is to highlight the importance of multirate control in the context of LCAs. We start with the feedback control layer, termed the *real-time feedback control layer* herein to highlight the fast loop rate at which it is implemented. We then discuss trajectory planning paradigms and end by highlighting approaches to their integration with the real-time feedback control layer. We forego a discussion of the decision-making layer for the sake of brevity. See [51] for a formal inclusion of decision making with the methods presented in this section.

Robot Dynamics

Robotic systems are inherently governed by nonlinear equations of motion. These represent the physical evolution of the system and are typically obtained from Euler–Lagrange equations:

$$D(q_t)\ddot{q}_t + C(q_t,\dot{q}_t)\dot{q}_t + G(q_t) = Bu_t \tag{21}$$

where $q_t \in Q$ are the configuration variables of the system, $\dot{q}_t \in T_qQ$ is a vector of velocities (which take values in the tangent space to the configuration space), $D(q_T)$ is the inertia matrix, $C(q_t,\dot{q}_t)$ is the Coriolis matrix, $G(q_t)$ contains the gravity-related terms, and B is the actuation matrix. Defining the state vector, $x_t = (q_t,\dot{q}_t) \in TQ$, where, for simplicity, we can work with a local coordinate chart of Q wherein $TQ \cong \mathbb{R}^n$ for n even, allows for the formulation of a control system affine in the control input:

$$\dot{x}_t = f(x_t) + g(x_t)u_t \tag{22}$$

where $f:\mathbb{R}^n \to \mathbb{R}^n$ and $g:\mathbb{R}^n \to \mathbb{R}^n \times \mathbb{R}^m$ can be directly obtained from (21). The end result is a control system of the form (1), with the additional structural property that the control input appears in an affine fashion, an observation that has important ramifications for controller synthesis.

Real-Time Feedback Control Layer

We begin by tackling the feedback control problem (15) for robotic systems. We assume that a trajectory, containing both a (reduced-order) reference trajectory and a feedforward control term, is available. We discuss approaches to solving this trajectory problem after addressing the feedback control problem. Emphasis is placed throughout on the need for *real-time* feedback control.

Linear Control

For robotic systems, the most common form of real-time controller (historically called "low-level control") is a simple linear feedback controller acting on the error, such as a PD controller implemented at the motor control layer. These controllers are *highly* effective in practice when implemented properly. It is important to stress that this is not due to the dynamics being linear (they are not), nor does it imply that the dynamics are even locally linear (again, they are not). Rather, these controllers work well exactly because of their use within an LCA, running at a fast loop rate and actuating as a function of tracking error. This second observation, that linear control actuates only on the error (and thus is independent of model information) is critical. To provide a precise instantiation of real-time linear controllers, consider a continuous-time reference signal $\boldsymbol{r} = (\boldsymbol{r}_q, \dot{\boldsymbol{r}}_q)$ that we wish to track (decomposed into a reference position and velocity) and an associated feedforward control input $\boldsymbol{u}_{\text{ff}}$. Then, the simplest form of feedforward and feedback control becomes

$$u_t = u_{\text{ff},t} + \ddot{r}_{q,t} - K_P(q_t - r_{q,t}) - K_D(\dot{q}_t - \dot{r}_{q,t}) \tag{23}$$

$$=: u_{\text{ff},t} + \ddot{r}_{q,t} + u_{\text{fb},t}(q_t - r_{q,t}, \dot{q}_t - \dot{r}_{q,t}) \tag{24}$$

for K_P, K_D positive definite matrices [see, for example, the Dubins' car control policy (20)]. Given the decoupled nature of this controller, it can be deployed in a decentralized fashion, that is, actuator by actuator, on the motor controllers at a very fast loop rate (faster than 1 kHz). It is important to note that many variations of this controller are possible. For example, an integral term can be added, or a reference velocity signal from a higher layer can be tracked, in which case the K_P term might be removed.

The fact that linear controllers can stabilize the nonlinear dynamics associated with a robotic system can be made rigorous in certain cases. To this end, assume that the robotic system is fully actuated; that is, B is invertible, and, for simplicity, take $B = I$. Then, picking $u_{\text{ff},t} = G(q_t)$ in (23) results in asymptotic stability of the tracking of the reference signal $\boldsymbol{r}$. To see this, let $\boldsymbol{e}_q = \boldsymbol{q} - \boldsymbol{r}_q$ and $\dot{\boldsymbol{e}}_q = \dot{\boldsymbol{q}} - \dot{\boldsymbol{r}}_q$ be the position and velocity error in tracking the reference signal $\boldsymbol{r}$. Define the error signal $\boldsymbol{e} := (\boldsymbol{e}_q, \dot{\boldsymbol{e}}_q)$, and consider the Lyapunov function candidate

$$V(e) = \frac{1}{2}\dot{e}_q^T D(e_q + r_q)\dot{e}_q + \frac{1}{2}e_q^T K_P e_q$$

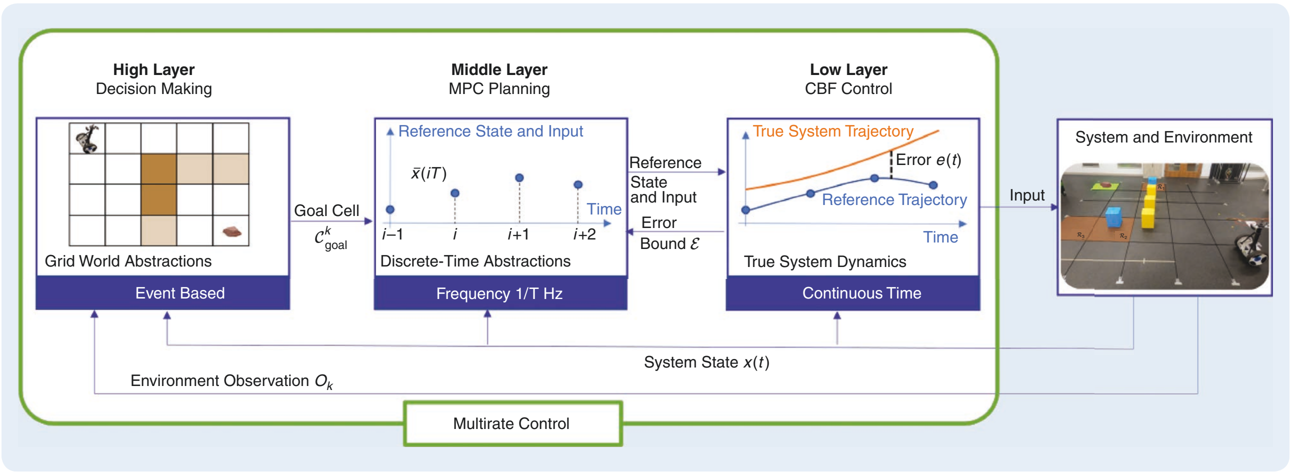


FIGURE 5 Multirate robotic control can be viewed through the lens of layered architectures (taken from [51]).

which is positive definite since $D(q)$ is symmetric positive definite. Then, differentiating V along solutions of (21) yields

$$\dot{V}(e) \leq -\dot{e}_q^T K_D \dot{e}_q \leq 0$$

as $\dot{D}(q,\dot{q}) - 2C(q,\dot{q})$ is skew symmetric. Invoking LaSalle's principle then shows asymptotic stability of $(e_q, \dot{e}_q) = (0,0)$; that is, it shows that the reference signal is asymptotically tracked.

Nonlinear Control

The use of Lyapunov functions in certifying linear controllers' ability to asymptotically track reference signals points to nonlinear controllers, based on Lyapunov functions, that can achieve improved performance. Indeed, to maximize performance in robotic systems, it is necessary to exploit the full nonlinear dynamics of the system, which can be done only with nonlinear controllers. These controllers must, however, be synthesized in a way that yields both theoretical guarantees while also being deployable in practice; that is, they must be implementable at fast loop rates (>1 KHz). With this in mind, a key attribute of the nonlinear controllers we define next is that they can be expressed as convex optimization problems that can be solved quickly, for example, linear programs and quadratic programs (QPs).

With the goal of driving the error signal $\boldsymbol{e} = (\boldsymbol{e}_q, \dot{\boldsymbol{e}}_q)$ to zero exponentially, consider a CLF V satisfying

$$\begin{aligned} &k_1 \| e \|^c \leq V(e) \leq k_2 \| e \|^c \\ &\inf_{u \in \mathcal{U}} \dot{V}(e,u) \leq -\lambda V(e) \end{aligned} \tag{25}$$

for $c, k_1, k_2, \lambda > 0$. Importantly, due to the affine nature of the dynamics (22), $\dot{V}$ is affine in the input u,

$$\dot{V}(e,u) = \frac{\partial V}{\partial e}(f(x) + g(x)u - \dot{r})$$

and can therefore be expressed as a QP when $\mathcal{U} = \mathbb{R}^p$:

$$\begin{aligned} u_{\text{fb}}(e) = &\operatorname*{argmin}_{u \in \mathbb{R}^p} \| u - u_{\text{ff}} \|^2 \\ &\text{subject to} \quad \dot{V}(e,u) \leq -\lambda V(e) \end{aligned} \tag{26}$$

which computes a minimal deviation from the desired feedforward control input $u_{\text{ff},t}$ while tracking the reference signal exponentially: $e_t = x_t - r_t \to 0$. Importantly, there are many variations of QP-based controllers that are used for real-time control in robotic systems, that is, those that utilize the dynamics as a constraint and can therefore account for constraints on forces and moments in real time. In all cases, the fact that these are QPs means that they can be implemented in real time at loop rates of 1 kHz or greater, even on complex robotic systems like walking robots.

Multirate LCAs

In LCAs, planning layers typically operate using *reduced-order models*. These are simpler, usually lower-dimensional, representations of the components of the full-order dynamics (21) of interest, designed to capture essential behavior needed for the control task. Thus, reduced-order models are often application dependent, and their generation is often heuristic in nature: some of the most common reduced-order models used for robotic control are the single integrator, double integrator, and unicycle. These are often leveraged for control synthesis in the context of kinematic models, for example, for mobile robots. The overarching goal in the design of these reduced-order models is the ability to generate reference signals that are (approximately) dynamically feasible and can be tracked well by a real-time feedback controller.

This trajectory generation is typically performed over a longer horizon, requiring more computation time; it is therefore natural to view the interplay of trajectory generation and feedback control through the lens of *multirate LCAs*, that is, through the lens of LCAs for which the controllers at different layers operate at different frequencies or rates. This can be captured using continuous models (such as singular perturbation theory [11], [12]). Yet, in the case of robotic systems, it is advantageous to be more concrete about the timescale separations present among layers. One way to explicitly capture this is through the use of discrete-time reduced-order models at the planning layer and continuous-time full-order models at the real-time feedback control layer. An additional advantage of discrete-time models at the planning layer is that they can provide an effective means of generating reference trajectories; this discrete instantiation better allows for planning forward in time, for example, through MPC.

To that end, consider a linear discrete-time reduced-order model

$$y(k+1) = A_{\text{rom}} y(k) + B_{\text{rom}} v(k) \tag{27}$$

that will be used by the trajectory planning layer, where the reduced-order model state $y \in \mathbb{R}^p$ and inputs $v \in \mathbb{R}^s$ are typically (but not necessarily) of a lower dimension than full-order state $x \in \mathbb{R}^n$ and inputs $u \in \mathbb{R}^m$; that is, $p \leq n$, and $s \leq m$. The reduced-order state is often related to the full system state via a projection map: $\pi_x(x) = y$. For robotic systems, a commonly used projection is $\pi_x(x) = \pi_x(q,\dot{q}) = q$; that is, one considers reduced-order models on the configuration variables only. Additionally, $v \in \mathbb{R}^s$ is an auxiliary input to the reduced-order model that is used to generate a reference signal sent to the real-time controller; we use v to denote this auxiliary input, as it can often be interpreted as a velocity command. Analogously, we typically require an embedding of the auxiliary input v into the full-order dynamics (22) via $\pi^v(v) = u$. While this section focuses on discrete-time linear reduced-order models, neither feature (discrete time or linear) is essential. In the "Continuous-Time LCAs" section, we explore the use of nonlinear continuous-time reduced-order models within multirate LCAs.

Proceeding with the discrete-time linear reduced-order model (27), we follow the approach proposed in intermediate subproblem (10) to couple the discrete-time reduced-order model state $y(k)$ and the underlying continuous-time dynamics (22). As in [52], we model the dynamics of the combined multirate LCA composed of a planning layer operating in discrete time with sampling period τ together with the full-order control system (22) defined on $\mathcal{T} := \cup_{k\in\mathbb{N}_{\geq 0}} \mathcal{T}_k$, with $\mathcal{T}_k := (k\tau,(k+1)\tau)$, as follows:

$$\begin{aligned} \text{Slow: } & y(k+1) = A_{\text{rom}}y(k) + B_{\text{rom}}v(k), && k\in\mathbb{N}_{\geq 0} \\ \text{Fast: } & \dot{x}_t = f(x_t) + g(x_t)(u_t + u_{\text{ff}}(k)), && t\in\mathcal{T}_k \\ \text{Coupling: } & y(k) = \pi_x(x_{k\tau}), \quad u_{\text{ff}}(k) = \pi^v(v(k)). && \end{aligned} \tag{28}$$

Here, the planning layer operates at a slow loop rate, defined by the sampling rate τ, in discrete time on a reduced-order model, while the real-time feedback controller operates at a fast timescale (represented by a continuous-time evolution). Analogous to the coupling constraints in subproblem (10), the reduced- and full-order models are coupled via π_x and π^v, where $\pi^v(v(k))$ is held constant over the interval $(k\tau,(k+1)\tau)$ on which the "fast" low-layer feedback controller operates and $\pi_x(x_{k\tau})$ is used to update the current state of the reduced-order model every discrete step.

The goal is to synthesize controllers u and v for the fast and slow dynamics in a synergistic fashion to achieve an overall control objective. In particular, suppose we synthesize a controller, $v(k) = v_{\text{fb}}(y(k))$, that achieves a control objective for the reduced-order linear model (27); that is, the closed-loop system $y(k+1) = A_{\text{rom}}y(k) + B_{\text{rom}}v_{\text{fb}}(y(k))$ drives $y(k)\to y_g$ for a desired goal state y_g. The evolution of the discrete-time system can also be used to give a set of tracking goals for the real-time feedback controller expressed by the error terms: $e_{k,t} = (\pi_x(x_t) - y(k+1))$. [Note that this causes a discrete jump in the error every discrete step. To avoid this, a smooth function of time $r_t(k)$ on $\mathcal{T}_k$ can be defined such that $\boldsymbol{r}_{k\tau}(k) = y(k)$ and $\boldsymbol{r}_{(k+1)\tau}(k)(k) = y(k+1)$, wherein the error term becomes $\boldsymbol{e}_k = (\pi_x(x) - \boldsymbol{r}_k)$. This can be achieved by converting the discrete-time system $y(k+1) = A_{\text{rom}}y(k) + B_{\text{rom}}v(k)$ into a continuous-time system $\dot{y} = A^c_{\text{rom}}y + B^c_{\text{rom}}v$, with v implemented in a sample-and-hold fashion; that is, A^c_{rom} and B^c_{rom} are defined by

$$A^c_{\text{rom}} = \frac{1}{T}\log(A_{\text{rom}}), \qquad B^c_{\text{rom}} = A^c_{\text{rom}}(A_{\text{rom}} - I)^{-1}B_{\text{rom}}$$

when the log and inverse are well defined. In the case when they are not, one can assume that the discrete-time system came from Euler integration, $A_{\text{rom}} = (I + A^c_{\text{rom}}T)$ and $B_{\text{rom}} = B^c_{\text{rom}}T$, to obtain $A^c_{\text{rom}} = (A_{\text{rom}} - I)/T$ and $B^c_{\text{rom}} = B_{\text{rom}}/T$. In either case, the result is a continuous reference signal:

$$r_t(k) = e^{A^c_{\text{rom}}(t-k\tau)}y(k) + \left(\int_{k\tau}^{t} e^{A^c(t-s)}ds\right)B^c v_{\text{fb}}(y(k)), \qquad t\in\mathcal{T}_k.$$

Alternatively, the discrete-time system (27) can be replaced by a continuous-time system at the slow control layer, as done in [52], to generate a smooth reference signal a priori.]

A controller can be synthesized that ideally drives this error to zero, such as a linear controller, as in (23), which here takes the form

$$u(e_{k,t},x(k\tau)) = \overbrace{\pi^v\circ v_{\text{fb}}\circ\pi_x(x_{k\tau})}^{u_{\text{ff}}(x_{k\tau})} + \underbrace{\left(\frac{\partial\pi_x}{\partial x}\right)^T K_P(\pi_x(x_t) - y(k+1))}_{u_{\text{fb}}(e_{k,t})} \tag{29}$$

or using the Lyapunov controller in (26), with e replaced by $e_{k,t}$ and u_{ff} replaced by $u_{\text{ff}}(x(k\tau))$ for $t\in\mathcal{T}_k$. The end result is the closed-loop multirate system

Slow: For $k\in N_{\geq 0}$:

$$y(k+1) = A_{\text{rom}}\pi_x(x_{k\tau}) + B_{\text{rom}}v_{\text{fb}}\circ\pi_x(x_{k\tau}) \tag{30}$$

Fast: For $\boldsymbol{e}_{k,t} = (\pi_x(x_t) - y(k+1))$ and $t\in\mathcal{T}_k$:

$$\dot{x}_t = f(x_t) + g(x_t)(u_{\text{ff}}(x_{k\tau}) + u_{\text{fb}}(e_{k,t})). \tag{31}$$

Here, the state $x_{k\tau}$ at the beginning of the sampling period informs the next iteration of the slow dynamics, while the slow dynamics inform the fast dynamics through the feedforward input (which depends on v_{fb}) and $e_{k,t}$, which drives the system to the next desired setpoint $y(k+1)$ over the interval $\mathcal{T}_k$. To provide a specific example of the generation of closed-loop policies, we begin with slow controller synthesis viewed as a planning problem.

Slow Trajectory Generation

We view trajectory generation as a planning problem (12), which can be solved using MPC. In particular, consider a goal state y_g for the reduced-order model, obtained, for example, from a higher decision layer, with the objective of synthesizing a controller that achieves this objective subject to a safety constraint expressed as state constraints $\mathcal{S} = \{y\in\mathbb{R}^p : h(y)\geq 0\}$ and input constraints $\mathcal{V}$; that is, the system must evolve such that $y(k)\in\mathcal{S}$ and $v(k)\in\mathcal{V}$ for all $k\geq 0$. To this end, we can formulate an MPC problem resembling that proposed in (12), with an $N\geq 1$ planning horizon and positive (semi)definite cost matrices $Q, Q_K\succeq 0$ and $R\succ 0$, as a QP:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=k}^{k+N-1}(\|y(i) - y_g\|^2_Q + \|v(i)\|^2_R) \\ & \quad + \|y(k+N) - y_g\|^2_{Q_N} \\ \text{subject to} \quad & y(i+1) = A_{\text{rom}}y(i) + B_{\text{rom}}v(i) \\ & y(k) = \pi_x(x_{\tau k}) \\ & y(i), y(k+N)\in\mathcal{S}, \quad v(i), v(k+N)\in\mathcal{V} \\ & i = k,\ldots,k+N-1 \end{aligned} \tag{32}$$

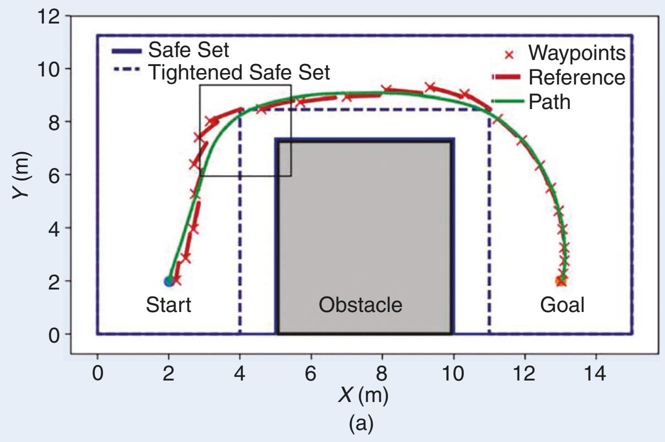
with $\|y\|^2_Q = y^TQy$. At each discrete-time step $k\geq 0$, problem (32) is solved to produce a sequence of nominal reduced-order model states $\boldsymbol{y}^* = y^*(k:k+N)$ and inputs $\boldsymbol{v}^* = v^*(k:k+N-1)$. The planning layer controller is then chosen as $v(k) = v_{\text{MPC}}(y(k)) := v^*(k)$, as is the standard approach in MPC.

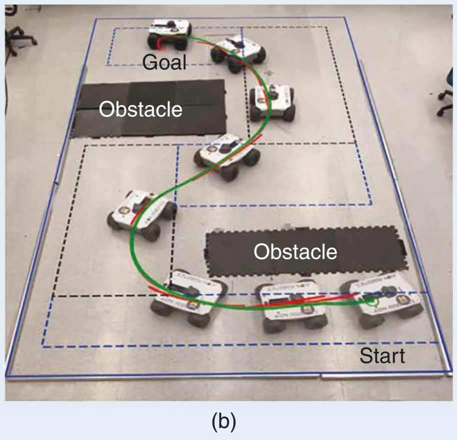
Multirate LCAs in Practice

To highlight the practical impact of the multirate LCAs we present, we provide an overview of successful experimental implementations in safe navigation, safe locomotion, and data-driven locomotion. In all cases, a multirate LCA facilitates the ability to realize controllers in practice. The commonality of the approaches and, more specifically, architectures in these disparate applications on different hardware platforms shows the broad applicability of LCAs for robot control.

SAFE NAVIGATION

Consider the problem of safe navigation with a ground robot [S1]—in this case, both a wheeled vehicle and a quadruped. Following the "Example 6 (Running Example: Robot Navigation)" section, the differential flatness of the Dubins' car is utilized to create a linear system (18) that is discretized as in (34). Using this discrete-time linear reduced-order model, an MPC problem is formulated as in (32), wherein safety is enforced (avoiding obstacles) while planning a path toward a goal. This generates a discrete-time reference trajectory that is sent to the Dubins' car and tracked with a real-time controller as in (35). This paradigm is illustrated in Figure S1. In particular, the discretely updated reference signals are shown (in red) along with the tracking of these reference signals by the real-time controller (in green). Importantly, the input to the Dubins' car model can be viewed as a reference velocity that can be tracked on hardware with onboard controllers. This further layering allows for the experimental deployment on both a wheeled vehicle and a quadruped, as, again, shown in Figure S1.





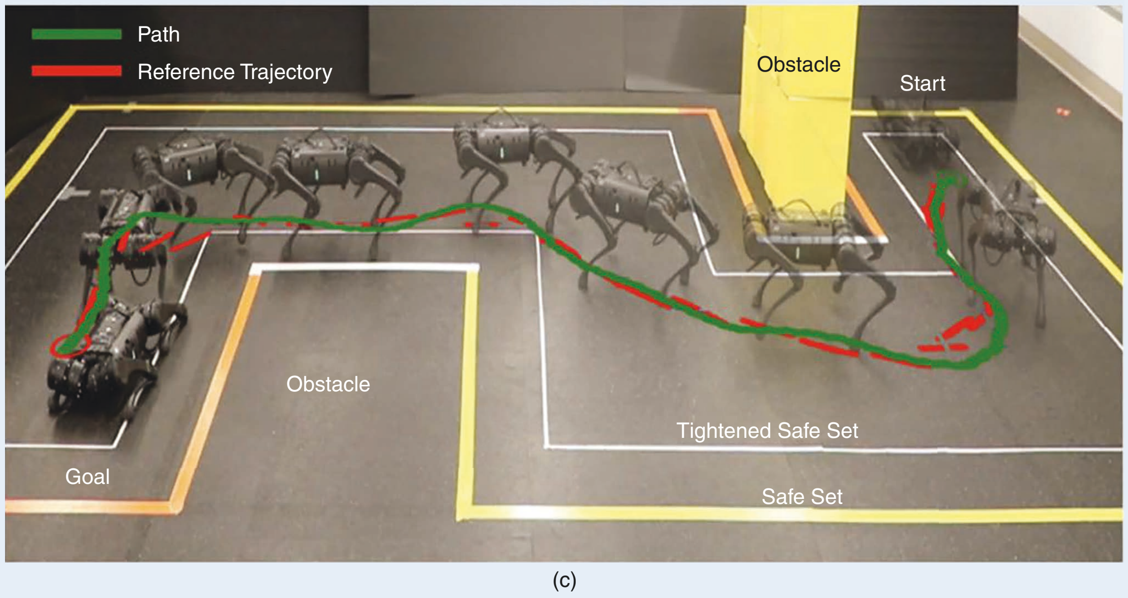


FIGURE S1 A multirate LCA using the Dubins' car and differential flatness (from [S1]). (a) Reference trajectories are generated with a discrete-time linear reduced-order model and tracked by the Dubins' car model. These signals can be passed to hardware and tracked onboard with real-time controllers, on both (b) a wheeled vehicle and (c) a quadruped.

SAFE LOCOMOTION

As noted throughout the "Multirate LCAs" section, LCAs provide an effective paradigm for enforcing safety constraints on complex robotic systems by enforcing safety (framed as set invariance) at both the planning layer [for example, in the MPC problem (32) as a state constraint] and at the real-time control layer via a CBF [for example, as in the QP (33)]. To demonstrate this, consider the *stepping stone problem*, where the goal is for a legged robot to precisely place its feet on a series of stepping stones. This is safety critical in that if this foot placement target is missed by the feet, the robot will fall. Additionally, it requires a layered approach, in that the system must maintain safety while also remaining dynamically balanced.

In [S2], an LCA formulation was implemented on a quadrupedal robot (ANYmal) to realize stepping stone behavior experimentally. This is illustrated in Figure S2. In particular, a safety constraint is implemented at the planning layer via a CBF (via MPC with a kinematic reduced-order model) and at the real-time control level (as a CBF constraint in a whole-body controller). Implementing CBFs at both layers resulted in no failures (missing the stepping stone) over 140 steps. Without the LCA framework, more failures were observed; that is, just enforcing a CBF constraint at the real-time layer leads to five failures, while implementing CBFs at only the planning layer leads to six failures. This demonstrates the practical utility of LCAs for safety-critical systems.

DATA-DRIVEN LOCOMOTION

Finding reduced-order models on which to instantiate multirate control can be challenging, often requiring domain-specific knowledge. This can be addressed by learning reduced-order models via data-driven methods. To provide an example of this paradigm, consider again the problem of legged locomotion, where reduced-order models are used at the planning layer. The goal is to learn this model and deploy the learned model experimentally.

Following [S3] and [S4], we learn a linear reduced-order model at the planning layer and leverage this model to pose an MPC problem. In particular, given sufficiently rich (persistently excited) data collected from the robot, Hankel matrices can be used to exactly determine the forward evolution

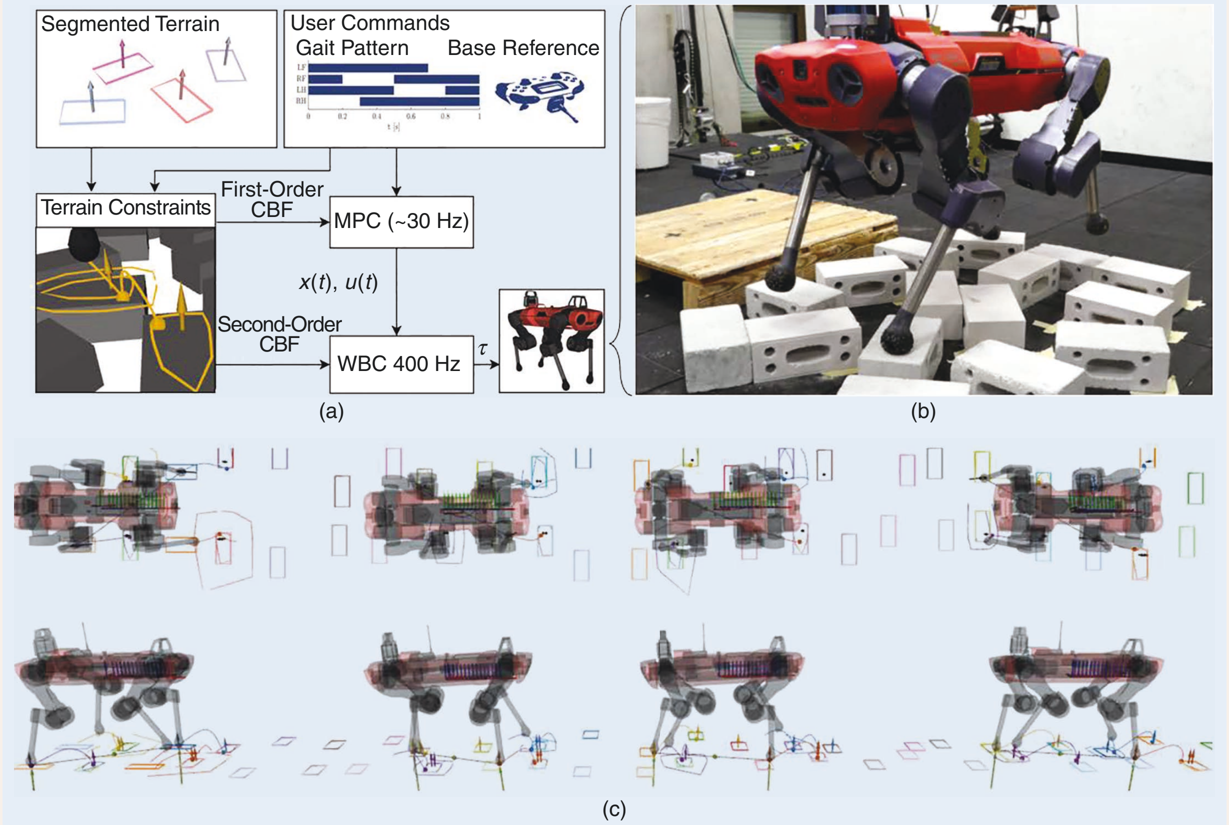


FIGURE S2 (a) An LCA used to realize dynamic walking on stepping stones (from [S2]). (b) This framework was implemented on ANYmal, with (c) the result of walking as illustrated. WBC: whole body control; RF: right front; LF: left front; RH: right hind; LH: left hind.

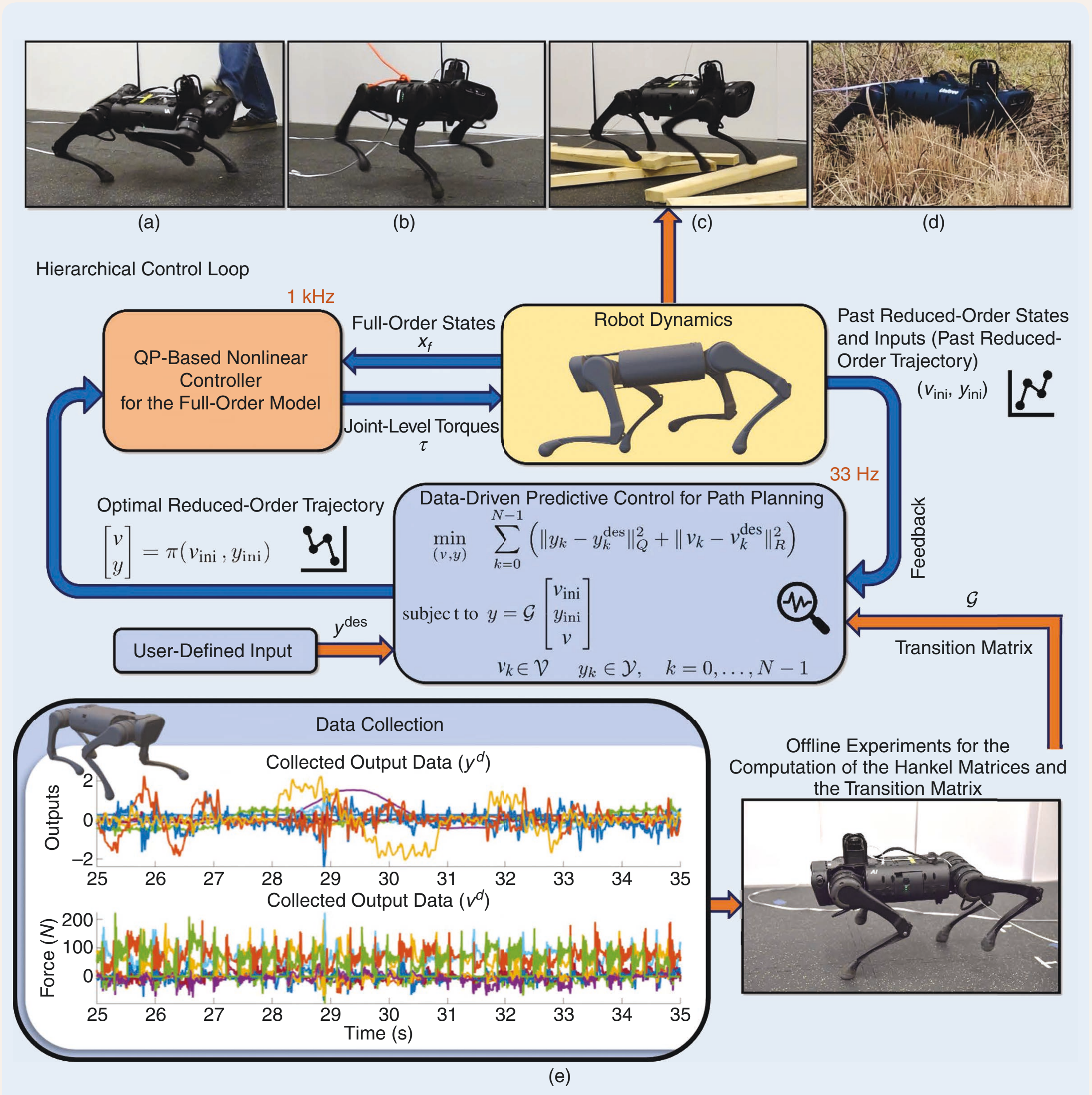
(Continued)

Multirate LCAs in Practice (*Continued*)

of linear time-invariant systems [S5]. Following [S6], given a user-defined reduced-order model state $y \in \mathbb{R}^p$ and input $v \in \mathbb{R}^s$, one can define the *data-driven state transition matrix* $\mathcal{G}(\mathtt{data})$ over N steps:

$$y_{k:k+N} = \mathcal{G}(\mathtt{data})\begin{bmatrix} v_{\text{ini}} \\ y_{\text{ini}} \\ v_{k:k+N} \end{bmatrix}. \qquad \text{(S1)}$$

Here, v_{ini} and y_{ini} are the reduced-order inputs and state observed in the past over an estimation horizon T_{ini}. Equation (S1), which defines linear relationships between the control input and state over the next N steps, can be used as a constraint in the MPC problem (32) instead of the explicit dynamic constraint $y(k+1) = A_{\text{rom}}y(k) + B_{\text{rom}}v(k)$.



(a)
(b)
(c)
(d)
Hierarchical Control Loop
1 kHz
QP-Based Nonlinear Controller for the Full-Order Model
Full-Order States x_f
Joint-Level Torques τ
Robot Dynamics
Past Reduced-Order States and Inputs (Past Reduced-Order Trajectory)
$(v_{\text{ini}}, y_{\text{ini}})$
33 Hz
Feedback
Optimal Reduced-Order Trajectory
$\begin{bmatrix} v \\ y \end{bmatrix} = \pi(v_{\text{ini}}, y_{\text{ini}})$
Data-Driven Predictive Control for Path Planning
$\min_{(v,y)} \sum_{k=0}^{N-1} \left(\|y_k - y_k^{\text{des}}\|_Q^2 + \|v_k - v_k^{\text{des}}\|_R^2\right)$
subject to $y = \mathcal{G}\begin{bmatrix} v_{\text{ini}} \\ y_{\text{ini}} \\ v \end{bmatrix}$
$v_k \in \mathcal{V} \quad y_k \in \mathcal{Y}, \quad k = 0, \ldots, N-1$
User-Defined Input
y^{des}
$\mathcal{G}$
Transition Matrix
Data Collection
Collected Output Data (y^d)
Outputs
Collected Output Data (v^d)
Force (N)
Time (s)
Offline Experiments for the Computation of the Hankel Matrices and the Transition Matrix
(e)

FIGURE S3 A data-driven LCA on a quadrupedal robot (from [S3]). Data are collected from the robot to determine the data-driven state transition matrix $\mathcal{G}(\mathtt{data})$, which is then used in an MPC problem to plan trajectories. The reduced-order model state and input are passed to a nonlinear optimization-based controller at the real-time layer to control the robot. Dynamic walking results that is robust to (a) pushes, (b) external pulls, (c) rough terrain, and (d) natural terrain, based on the (e) hierarchical control loop and data collection.

(Continued)

Multirate LCAs in Practice *(Continued)*

This approach was experimentally deployed on a quadruped robot, as described in Figure S3. The robot considered has 18 degrees of freedom, and thus, the state is 36D ($x \in \mathbb{R}^{36}$). A reduced-order model is considered with a 10D state $y \in \mathbb{R}^{10}$ consisting of select body positions, velocities, and orientations. The reduced-order input $v \in \mathbb{R}^{12}$ consists of ground reaction forces. With this reduced-order model, data are collected, $\mathcal{G}(\mathtt{data})$ is computed, and (S1) is leveraged in an LCA via an MPC problem at the planning layer, which is subsequently implemented on the robot via a nonlinear real-time controller similar to (33). The end result is robust data-driven dynamic walking.

REFERENCES

[S1] D. R. Agrawal, H. Parwana, R. K. Cosner, U. Rosolia, A. D. Ames, and D. Panagou, "A constructive method for designing safe multirate controllers for differentially-flat systems," *IEEE Contr. Syst. Lett.*, vol. 6, pp. 2138–2143, 2022, doi: 10.1109/LCSYS.2021.3136465.

[S2] R. Grandia, A. J. Taylor, A. D. Ames, and M. Hutter, "Multi-layered safety for legged robots via control barrier functions and model predictive control," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Piscataway, NJ, USA: IEEE, 2021, pp. 8352–8358, doi: 10.1109/ICRA48506.2021.9561510.

[S3] R. T. Fawcett, K. Afsari, A. D. Ames, and K. A. Hamed, "Toward a data-driven template model for quadrupedal locomotion," *IEEE Robot. Autom. Lett.*, vol. 7, no. 3, pp. 7636–7643, Jul. 2022, doi: 10.1109/LRA.2022.3184007.

[S4] R. T. Fawcett, L. Amanzadeh, J. Kim, A. D. Ames, and K. A. Hamed, "Distributed data-driven predictive control for multi-agent collaborative legged locomotion," 2022, *arXiv:2211.06917*.

[S5] J. C. Willems, P. Rapisarda, I. Markovsky, and B. L. De Moor, "A note on persistency of excitation," *Syst. Control Lett.*, vol. 54, no. 4, pp. 325–329, 2005, doi: 10.1016/j.sysconle.2004.09.003.

[S6] J. Coulson, J. Lygeros, and F. Dörfler, "Data-enabled predictive control: In the shallows of the DeePC," in *Proc. 18th Eur. Control Conf. (ECC)*, Piscataway, NJ, USA: IEEE, 2019, pp. 307–312, doi: 10.23919/ECC.2019.8795639.

Fast and Safe Feedback Control

The solution $(y^\star, v^\star)$ to each MPC subproblem solved at discrete-time step k can further be leveraged to synthesize a lower-layer real-time feedback controller. In particular, we can transmit these solutions to the fast lower layer to define the error signal $e_{k,t} = (\pi_x(x_t) - y^\star(k+1))$, which can in turn be driven to zero using the Lyapunov controller (26) (replacing e with $e_{k,t}$) and using $u_{\mathrm{ff}}(k) = \pi^v(v_{\mathrm{MPC}}(y(k)))$. While this will drive $\pi_x(x) \to y^\star(k+1))$ (the next step produced by the MPC problem) with the input from the MPC problem $v_{\mathrm{MPC}}(y(k))$ as a reference, there is no guarantee that the safety constraints will be satisfied over the time interval $\mathcal{T}_k$. To address this shortcoming, we can combine the Lyapunov controller (26) with the CBF controller (37) into an optimization problem that resembles optimization problem (15). However, in this case, we can exploit the control-affine structure of the dynamics (22) to obtain a QP through the use of Lyapunov and barrier functions:

$$\begin{aligned} u_{\mathrm{fb}}(e_{k,t}) = \ & \underset{u \in \mathbb{R}^p,\, \delta \in \mathbb{R}}{\operatorname{argmin}} \| u - \pi^v(v_{\mathrm{MPC}}(y(k))) \|^2 + p\delta^2 \\ \text{subject to} \quad & \dot{V}(e_{k,t}, x_t, u) \leq -\lambda V(e_{k,t}) + \delta \\ & \dot{h}_{\pi_x}(x_t, u) \geq -\alpha h_{\pi_x}(x_t) \end{aligned} \tag{33}$$

with $h_{\pi_x} := h \circ \pi_x$ assumed to be a valid CBF:

$$\sup_{u \in \mathbb{R}^m} \underbrace{\left[\frac{\partial h}{\partial y} \frac{\partial \pi_x}{\partial x} (f(x) + g(x)u) \right]}_{\dot{h}_{\pi_x}(x_t, u)} \geq -\alpha h_{\pi_x}(x).$$

Here, $p > 0$ is a penalty associated with the relaxation term $\delta > 0$, which can be interpreted as an instantaneous analog to the tracking cost (15) that penalizes $\| x - r \|_Q^2$. Note that input constraints can also be added to the QP (33), but this would require a relaxation of the CBF condition or the input constraints to ensure feasibility (see [53], which considers the interplay between continuous and discrete dynamics in the context of input constraints).

The solution $u_{\mathrm{fb}}(e_{k,t})$ to the QP (33) can be used in the multirate dynamics (31), which, when combined with the MPC problem (30), yields a closed-loop multirate controller instantiated via an LCA. The power of the closed-loop multirate LCA, as opposed to a single-layer feedback controller using Lyapunov and barrier functions, is that information from the MPC problem encodes knowledge of future system behavior via the reduced-order model. Conversely, the fast layer accounts for the full nonlinear dynamics of the system that are not present in the slow layer. Thus, we are able to synthesize an LCA that leverages the nonlinear dynamics of the system at the real-time feedback control layer while looking ahead to future behaviors defined in the trajectory planning layer via a synergistic coupling of the two. Formal guarantees for the multirate LCA presented here can be found in [51] and [52].

Example 6 (Running Example: Robot Navigation)

Consider again the Dubins' car, which, for the moment, we view as the full-order dynamics. We aim to instantiate a discrete-time planning layer via MPC. To determine the corresponding reduced-order model, we leverage the fact that the dynamics of the Dubins' car $\dot{q}_t = f_{\mathrm{rom}}(q_t)u_t$, with $q = (x_1, x_2, \theta)$ and $u = (u_1, u_2)$, as given in (4), are differentially flat per the "Example 5 (Running Example: Robot Navigation)" section. For the flat output $\xi = (x_1, x_2)$, denote the relationships among the states, inputs, and flat outputs by $q = q_\flat(\xi, \dot{\xi})$ and $u = u_\flat(\xi, \dot{\xi}, \ddot{\xi})$. Note that we make a slight deviation from the notation used in the "Example 5 (Running Example: Robot Navigation)" section to be consistent with the configuration space notation defined in this section and use q to denote the state, rather than x.

To apply the approach outlined in this section, we can forward integrate the flat continuous-time linear dynamics (18) over the time interval $\mathcal{T}_k = [k\tau, (k+1)\tau]$ to obtain the discrete-time reduced-order model

$$y(k+1) = \underbrace{e^{A^c_{\text{rom}}\tau}}_{A_{\text{rom}}} y(k) + \underbrace{\int_0^\tau e^{A^c_{\text{rom}}s} B^c_{\text{rom}} ds}_{B_{\text{rom}}} v(k). \tag{34}$$

Here, $y(k) = (\xi(k), \dot{\xi}(k)) \in \mathbb{R}^4$, and $v(k) = \ddot{\xi}(k) \in \mathbb{R}^2$. We note that here, the reduced-order model is actually higher dimensional but is "reduced" in complexity by being linear. Utilizing this system, a feedback controller $v_{\text{fb}}(y(k))$ can be synthesized. For example, this can be chosen to be the result of the MPC problem (32); that is, $v_{\text{fb}}(y(k)) = v_{\text{MPC}}(y(k))$. The result is the error $e_{k,t} = (q_t - q_\flat(y(k+1)))$ and a feedforward input $u_{\text{ff}}(y(k)) = u_\flat(y(k), v_{\text{fb}}(y(k)))$. This can be used to synthesize a linear feedback controller of the form (29), modified slightly to exploit differential flatness, as in (20):

$$\begin{aligned} u(e_{k,t}, y(k)) &= u_{\text{ff}}(y(k)) + K_P e_{k,t} \\ &=: u_{\text{ff}}(y(k)) + u_{\text{fb}}(e_{k,t}) \end{aligned} \tag{35}$$

for K_P, a positive definite matrix.

The final feedback controller is given by setting $y(k) = (x_{1,k\tau}, x_{2,k\tau}, \dot{x}_{1,k\tau}, \dot{x}_{2,k\tau})$. This paradigm is deployed experimentally in "Multirate LCAs in Practice." Alternatively, in the expression above, we could consider a continuous reference signal $\boldsymbol{e}_t = (q_t - q_\flat(y_t))$, where y_t is the solution to (18) given a feedback controller $v = K_{\text{fb}} y$, that is, by solving $\dot{y} = (A + BK_{\text{fb}})y$ with initial condition $y(k\tau) = (x_{1,k\tau}, x_{2,k\tau}, \dot{x}_{1,k\tau}, \dot{x}_{2,k\tau})$. This paradigm is deployed experimentally as described in "Continuous-Time LCAs in Practice" and is discussed in more detail in the next section.

Continuous-Time LCAs

We observe that we can further layer the control architecture defined in the "Example 6 (Running Example: Robot Navigation)" section by viewing the unicycle as a reduced-order model wherein continuous multirate control can be applied. That is, we can view $\mu_{\text{fb}}(e_{k,t}, y(k))$ as a reference velocity [see the "Example 7 (Running Example: Robot Navigation)" section], which we want a more complex robot to track, that is, a quadruped or drone, as described in "Continuous-Time LCAs in Practice." The end result is a three-layer architecture with two planning layers and one feedback control layer: 1) a slower discrete-time planning layer using a linear model, 2) an intermediate reference signal generation layer using a continuous-time unicycle model, and 3) a fast feedback control layer for the tracking of the reference signal by the underlying complex robotic system. This observation highlights that layers can often be added in a fairly modular fashion, allowing for the benefits of each layer to be enjoyed. This section explores the bottom two layers of the LCA described previously, namely, the interplay between a continuous-time reference signal generation layer and a real-time feedback control layer, in more detail. Implicit throughout is the assumption that the loop rates at each layer, and the communication among layers, happen sufficiently fast. We focus on safety-critical systems, wherein safe reference signals are generated by the trajectory generation layer to be tracked by the real-time control layer. We show that formal guarantees of safety can be obtained for these LCAs and that, importantly, this architecture enables theory to be widely deployed in practice.

Safe Reference Signal Generation

Consider a continuous-time reduced-order model

$$\dot{y}_t = f_{\text{rom}}(y_t, v_t) \tag{36}$$

where, as above, the reduced-order dynamics f_{rom}, state $y_t \in \mathbb{R}^p$, and auxiliary input $v_t \in \mathbb{R}^s$ are chosen to be simpler than the full equations of motion (21) while nevertheless capturing the essential features of the control problem at hand. We recall that we assume that the reduced-order state y is related to the full system state x via the projection $\pi_x(x) = y$ and that the auxiliary input v can be embedded into the full dimensional input space via the embedding $\pi^v(v)$.

Assume now that the reduced-order model (36) is used to generate a desired reduced-order state trajectory $\boldsymbol{y}^d$ and a corresponding feedback law $v^d(y_t)$ for the auxiliary input, for example, via the techniques described in the previous section. Now consider the objective of ensuring that the reference signal satisfies a safety constraint, encoded by making the set $\mathcal{S} = \{y \in \mathbb{R}^p : h(y) \geq 0\}$ forward invariant, for some differentiable function $h : \mathbb{R}^p \to \mathbb{R}$. We can leverage CBFs if h satisfies the CBF condition with respect to the reduced-order dynamics:

$$\sup_{v \in \mathbb{R}^s} \dot{h}(y,v) = \sup_{v \in \mathbb{R}^s} \left[\frac{\partial h}{\partial y} f_{\text{rom}}(y,v) \right] \geq -\alpha h(x) \tag{37}$$

where $\alpha > 0$ is a positive constant. (Note that, more generally, α can be chosen to be an extended class $\mathcal{K}$ function. We opt for a positive constant for simplicity of exposition.) If f_{rom} is affine in the auxiliary control input v, this inequality can be expressed as a QP of the form (26), with the result being a safety filter operating on the reduced-order model within the reference signal generating layer:

$$\begin{aligned} v_{\text{safe}}(y) = \; & \underset{v \in \mathbb{R}^s}{\operatorname{argmin}} \; \| v - v^d(y) \|^2 \\ & \text{subject to } \; \dot{h}(y,v) \geq -\alpha h(y). \end{aligned} \tag{38}$$

This safety filter can then be integrated into reference signal generation in a variety of ways, for example, forward integrating the closed-loop dynamics $\dot{y}_t = f_{\text{rom}}(y_t, v_{\text{safe}}(y_t))$ to generate a reference signal $\boldsymbol{r}_y$ with corresponding error $e_{y,t} = y_t - r_{y,t}$, which can then be tracked with a linear or nonlinear controller, that is, replacing e in (26) with e_y. Alternatively, the safe input v_{safe} can be tracked by the real-time controller, as described below.

Real-Time Feedback Control

To provide a concrete example of the use of continuous-time reduced-order models at the planning layer coupled with real-time feedback controllers, consider the case when we have a kinematic reduced-order model with $y = q$; that is, our reduced-order model operates on the configuration variables, $\dot{q}_t = f_{\mathrm{rom}}(q_t, v_t)$, where now the auxiliary input v_t is naturally associated with the generalized velocities $\dot{q}_t$. Consider a safe velocity, $v_{\mathrm{safe}}(q)$, generated from the QP (38). Following [54], assume that this velocity is passed to a real-time controller via the error signal $\dot{e}_{\mathrm{safe},t} := \dot{q}_t - v_{\mathrm{safe}}(q_t)$; that is, the real-time controller takes the safe velocity from the reduced-order model as a reference, with the goal of tracking this reference signal. Assume a real-time feedback controller $u_t = u_{\mathrm{fb}}(x_t, v_{\mathrm{safe}}(q_t))$ that can exponentially track this reference velocity, for example, via the controller (26), with e_t replaced by $\dot{e}_{\mathrm{safe},t}$, resulting in exponentially fast tracking:

$$\left\| \dot{e}_{\mathrm{safe},t} \right\| \leq M e^{-\lambda t} \left\| \dot{e}_{\mathrm{safe},0} \right\| \tag{39}$$

for $M, \lambda > 0$, with the error calculated along solutions of the closed-loop system $\dot{x}_t = f(x_t) + g(x_t) u_{\mathrm{fb}}(x_t, v_{\mathrm{safe}}(q_t))$. The following theorem, adapted from [55], provides formal guarantees for the reduced-order model-based LCA applied to the full-order dynamics (21). For experimental implementations related to this formal result, see "Continuous-Time LCAs in Practice."

Theorem 1

Consider a control system (22), where $x = (q, \dot{q})$, and a safe set $\mathcal{S} = \{q \in Q : h(q) \geq 0\}$. Assume that h has a bounded gradient; that is, there exists $K_h > 0$, subject to $\|\partial h / \partial q\|_2 \leq K_h$, for all $q \in \mathcal{S}$. Let $v_{\mathrm{safe}}(q)$ be the safe velocity given by the QP (38), with corresponding error $\dot{e}_{\mathrm{safe}} = \dot{q} - v_{\mathrm{safe}}(q)$ satisfying (39). If $\lambda > \alpha$, safety is achieved for the full-order dynamics (22):

$$(q(0), \dot{e}_{\mathrm{safe}}(0)) \in \mathcal{S}_M \quad \Rightarrow \quad q(t) \in \mathcal{S}, \quad \forall t \geq t_0 \tag{40}$$

where

$$\mathcal{S}_M = \left\{ (q, \dot{e}_{\mathrm{safe}}) \in \mathbb{R}^{2n} : h(q) - \frac{K_h M}{\lambda - \alpha} \| \dot{e}_{\mathrm{safe}} \|_2 \geq 0 \right\}. \tag{41}$$

Note that to certify "fast enough" tracking by the real-time controller, a Lyapunov certificate can be used [54] (see Figure 6). Assume that the real-time feedback controller tracks the error $\dot{e}_{\mathrm{safe}}$ per a Lyapunov function, as in (25): $\dot{V}(\dot{e}_{\mathrm{safe}}) \leq -\lambda V(\dot{e}_{\mathrm{safe}})$. Then, for any differentiable $v_{\mathrm{safe}}(q)$ satisfying the CBF condition (37), safety for the full-order dynamics is achieved if $\lambda > \alpha$:

$$(q(0), \dot{e}_{\mathrm{safe}}(0)) \in \mathcal{S}_M \quad \Rightarrow \quad q(t) \in \mathcal{S}_V, \quad \forall t \geq t_0$$

where

$$\mathcal{S}_V = \{ (q, \dot{e}_{\mathrm{safe}}) \in R^{2n} : \; h_V(q, \dot{e}_{\mathrm{safe}}) \geq 0 \}$$
$$h_v(q, \dot{e}_{\mathrm{safe}}) := -V(q, \dot{e}_{\mathrm{safe}}) + \alpha_e h(q), \quad \alpha_e = \frac{(\lambda - \alpha) k_1}{K_h}.$$

Interestingly, this result is established by synthesizing a CBF for the full system dynamics, h_V, using the CBF for the reduced-order model, h, together with the Lyapunov function for the tracking controller, V.

Example 7 (Running Example: Robot Navigation)

Returning to the running example, we can view the Dubins' car as a reduced-order model used to enforce safety constraints on a complex mobile robot, such as a quadruped. Recall that the Dubins' car dynamics (4) take the form $\dot{q}_t = f_{\mathrm{rom}}(q_t) u_t$, where $q = (x_1, x_2, \theta)$ and $u = (u_1, u_2)$. Consider a barrier function defined on the Dubins' car dynamics, aimed at avoiding collisions with obstacles:

$$h(q) = d_0 - r - \kappa \cos(\theta - \theta_0)$$

where $d_0 = \| (x_1 - x_1^0, x_2 - x_2^0) \|$, with (x_1^0, x_2^0) the position of the obstacle, $\theta_0 = \arctan((x_2^0 - x_2) / (x_1^0 - x_1))$, and $\kappa > 0$ a tunable parameter.

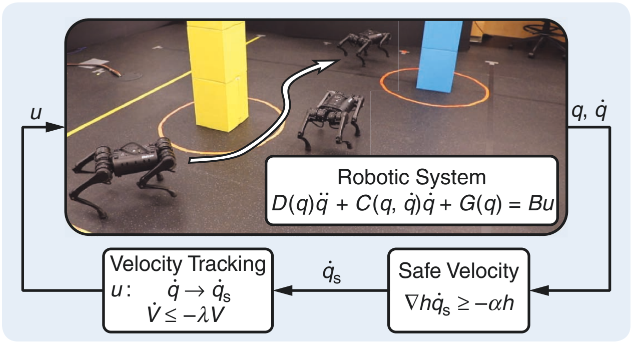
Let $u^d(q)$ be the feedback controller (20) synthesized in the previous section for tracking a nominal trajectory. Then, the safety filter (38) yields a QP on the Dubins' car:

$$\begin{aligned} u_{\mathrm{safe}}(q) = \; & \underset{u \in \mathbb{R}^2}{\operatorname{argmin}} \; \| u - u^d(q) \|_\Gamma^2 \\ & \text{subject to} \; \frac{\partial h}{\partial q} f_{\mathrm{rom}}(q) u \geq -\alpha h(q) \end{aligned} \tag{42}$$

where $\|u\|^2 = u^T \Gamma u$, with $\Gamma = \mathrm{diag}(1, R)$, where $R > 0$ is a control cost parameter. The result is a reference velocity $u_{\mathrm{safe}}(q) = (u_{1,\mathrm{safe}}(q), u_{2,\mathrm{safe}}(q))$ on the forward velocity and change in heading. These reference signals can be sent to a robot with more complex dynamics as though they were joystick commands. Theorem 1 guarantees safety for the more complex system under the assumption of good tracking.

Discussion

An underlying principle in designing LCAs for robotic systems is to synergistically leverage the strengths at each layer. For example, the lowest layer can handle



u
$q, \dot{q}$
Robotic System
$D(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = Bu$
Velocity Tracking
$u: \quad \dot{q} \to \dot{q}_s$
$\dot{V} \leq -\lambda V$
$\dot{q}_s$
Safe Velocity
$\nabla h \dot{q}_s \geq -\alpha h$

FIGURE 6 An LCA (from [54]) for controlling robotic systems via continuous reference signal generation. In this case, a safe reference velocity is generated that is tracked by a real-time controller realized on the robot.

Continuous-Time LCAs in Practice

To illustrate the practical consequences of Theorem 1, we highlight how a common reduced-order model can be used to achieve safety across a variety of robotic systems, including a drone, quadrupedal robot, manipulator, and full-scale automotive system. This diverse set of robotic systems has very different underlying dynamics, and yet, deploying a well-designed LCA does not require direct knowledge of these dynamics, rather, only "good" onboard tracking controllers that allow planning layers to operate on reduced-order models rather than on the underlying complex system dynamics (as illustrated in Figure 6).

DRONES AND QUADRUPEDS

We wish to enforce collision avoidance with obstacles in the environment. To begin, consider the "simplest" kinematic model of a robot, a single integrator:

$$\dot{q}_t = f_{\text{rom}}(q_t, v_t) = v_t \qquad \text{(S2)}$$

obtained by setting $y_t = q_t$. Collision avoidance is encoded by the safety constraint $\mathcal{S} = \{q \in \mathbb{R}^n : h(q) \geq 0\}$ for

$$h(q) = \|q - q_0\| - r.$$

Here, $q_0 \in \mathbb{R}^n$ is the centroid of the obstacle and r its radius. The safety filter (38) can be expressed as the QP

$$\begin{aligned} v_{\text{safe}}(q) = \underset{v \in \mathbb{R}^n}{\operatorname{argmin}} & \|v - v^d(q)\|^2 \\ \text{subject to } & \frac{(q-q_0)^T}{\|q-q_0\|} v \geq -\alpha(\|q-q_0\| - r) \end{aligned} \qquad \text{(S3)}$$

where $v^d(q) = -K_P(q - q_g)$ is a desired velocity that drives the system to a goal position $q_g \in \mathbb{R}^n$. In the case of planar collision avoidance $q, q_0, q_g \in \mathbb{R}^2$ (representing the spatial position in the plane), which is the case that will be considered in the context of the experimental implementation on a drone and quadruped.

The result of (S3) is a safe velocity $v_{\text{safe}}(q)$ that can be tracked with existing onboard tracking controllers. This was implemented on both drone and quadruped hardware platforms. We highlight that while these platforms have dramatically different underlying dynamics, by leveraging a well-designed LCA, the exact same safe reference velocity, $v_{\text{safe}}(q)$, can be used and tracked on both platforms. The results can be seen in Figure S4: safety is achieved [as certified by $h(q) \geq 0$]

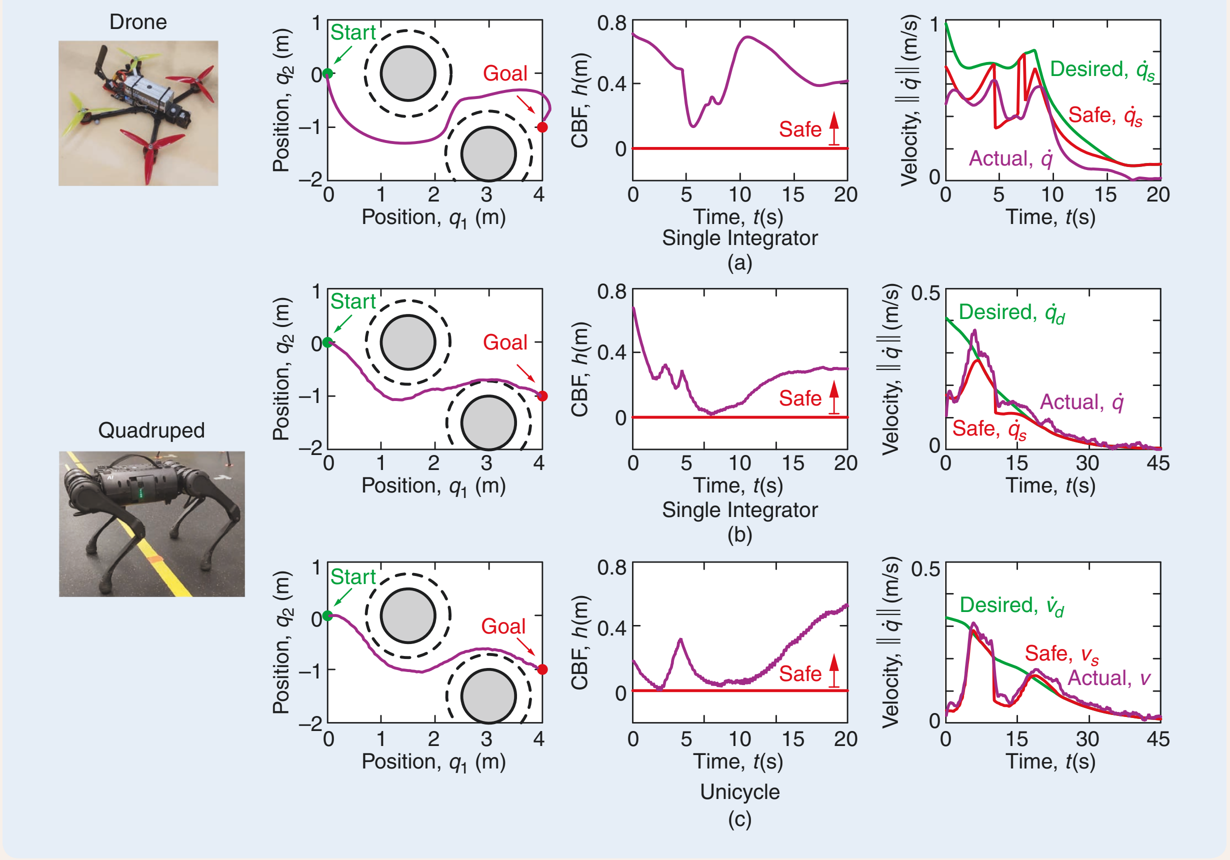


FIGURE S4 Experimental results on a drone and quadruped (from [54]). Both the (a) drone and (b) quadruped use a single-integrator (S2) reduced-order model and the corresponding QP (S3). Suitable integration into an LCA yields safe behavior. Additionally, a (c) unicycle (Dubins' car) model is used on the quadruped to achieve safe behavior that is less conservative.

for both the drone and quadruped tracking reference signals $v_{\text{safe}}(q)$ produced by (S3). For the quadruped, we can also use the Dubins' car as a reduced-order model instead of the single integrator, as described in the "Example 7 (Running Example: Robot Navigation)" section. In this case, the resulting safe velocity $u_{\text{safe}}(q)$ produced by the QP (38) is tracked as a reference signal. The resulting behavior is again safe but less conservative due to the Dubins' car being a better representation of the movement of the quadruped in the plane; that is, a better reduced-order model produces less conservative behavior while still maintaining safety.

MANIPULATORS

Consider a robot manipulator, as illustrated in Figure S5. The control task is to achieve collision-free behavior between the robot and environment while accomplishing a task (in this case, flipping a burger). Importantly, there is no access to the proprietary onboard real-time controllers of the commercial robot arm, and therefore, safety *must be achieved through an LCA*.

Let $A(q) \subset \mathbb{R}^3$ be the set of all points on the robot (which depends on the configuration of the robot $q \in \mathbb{R}^n$) and $B \subset \mathbb{R}^3$ be the set of all points in the environment. Collision-free behavior between the robot and environment, captured by $A(q) \cap B = \varnothing$ or $A(q) \subset \bar{B}$, with $\bar{B}$ the complement of B, is encoded by a barrier function $\mathcal{S} = \{q \in \mathbb{R}^n : \text{sd}_{AB}(q) \geq 0\}$ defined in terms of the *signed distance* ([S7]):

$$h(q) = \text{sd}_{AB}(q) := \underbrace{\inf_{\substack{p_A \in A(q) \\ p_B \in B}} \| p_A - p_B \|_2}_{\text{distance}(A(q), B)} - \underbrace{\inf_{\substack{p_A \in A(q) \\ p_B \in \bar{B}}} \| p_A - p_B \|_2}_{\text{penetration}(A(q), B)}. \tag{S4}$$

The advantage of using the signed distance, as opposed to the distance, is that the addition of the "penetration" term which gives a negative value when this occurs—as opposed to the distance which is strictly nonnegative. This negative value allows for convergence back to the safe set $\mathcal{S}$ per the fact that CBFs render $\mathcal{S}$ attractive.

The challenge with using the signed distance as a barrier function is that it is discontinuous on a set of measure zero [S8]. To accommodate for the discontinuities, consider

$$\frac{\partial h}{\partial q} = \frac{\partial \text{sd}_{AB}}{\partial q} = \frac{\partial \text{sd}_{AB}^{C^1}}{\partial q} + \delta(q) \tag{S5}$$

which decomposes $\text{sd}_{AB}(q)$ into its differentiable and nondifferentiable component, where the gradient of the nondifferentiable component, δ, is viewed as a disturbance that is nonsmooth on a set of measure zero; as a result, we can design a controller that is robust to adversarial disturbances of magnitude matching the essential supremum $\|\delta\|_\infty = \text{esssup}_{t \geq 0} \|\delta(q_t)\|$.

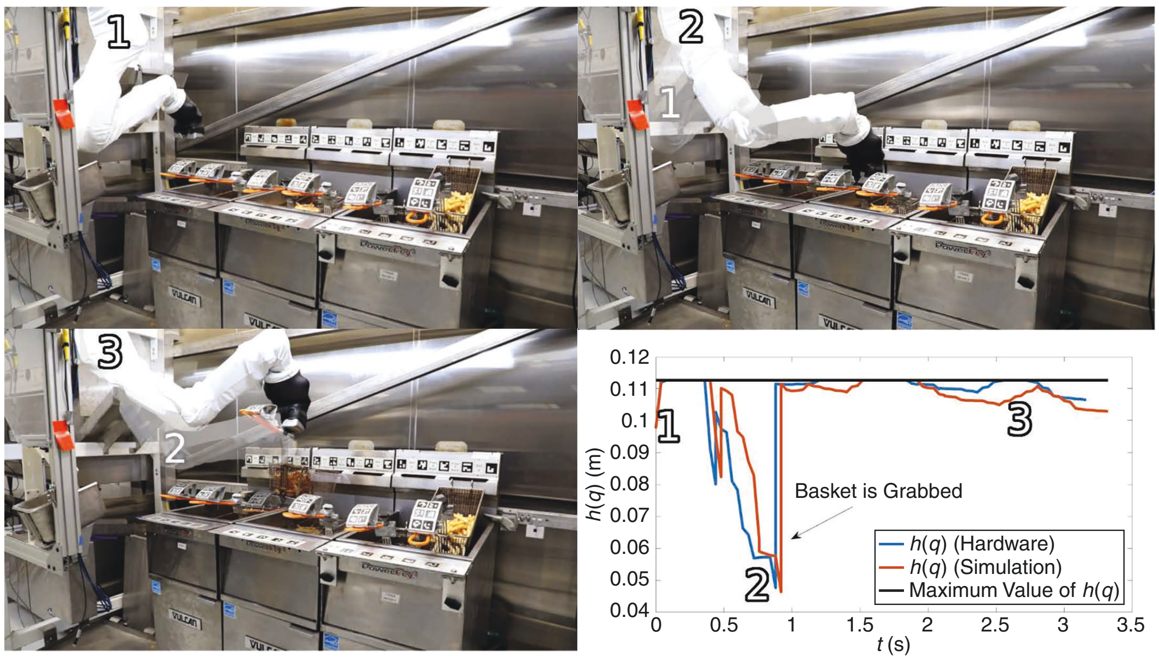


FIGURE S5 Achieving safety on a robot manipulator (from [55]). The manipulator executes a series of preplanned trajectories, and a safety filter is instantiated via a reduced-order model to prevent collisions with the environment. The value of the barrier function is shown, wherein nonnegative values imply collision-free behavior.

(Continued)

Continuous-Time LCAs in Practice *(Continued)*

For the reduced-order model, we consider a kinematic model of the robot arm (S2); that is, $\dot{q}_t = v_t$, with $q_t \in R^n$, for n, the number of degrees of freedom (in this case, $n = 6$). To enforce a safety filter on the reduced-order model, the goal is to leverage a QP of the form (38). Yet, in this case, due to the fact that the signed distance is not continuously differentiable, we leverage the decomposition in (S5) to obtain the QP

$$v_{\text{safe}}(q,t) = \underset{v \in \mathbb{R}^n}{\operatorname{argmin}} \| v - v^d(q) \|^2$$
$$\text{subject to} \quad \frac{\partial \text{sd}_{AB}^{C^1}}{\partial q} v \geq -\alpha(\text{sd}_{AB}(q)) + \|\delta\|_\infty \dot{q}_{\max} \tag{S6}$$

with $\dot{q}_{\max} = \|\dot{q}\|_\infty$ and $\|\delta\|_\infty$ defined as above. Here, $v_d(q)$ is obtained from a series of preplanned trajectories that must be executed while avoiding collisions; that is, $v_d(q) = K_P(q_d^i - q)$, with q_d^i the next waypoint (in time) of the preplanned trajectory.

The QP in (S6) was implemented experimentally on a FANUC robotic manipulator in a kitchen scenario [55]; that is, the robot was required to do a variety of cooking-related tasks while avoiding collisions with the environment. As illustrated in Figure S5, the robot was able to perform a variety of complex tasks while maintaining safety $h(q) = \text{sd}_{AB}(q) \geq 0$.

AUTOMOTIVE SYSTEMS

For complex real-world applications, domain-specific reduced models are needed. Additionally, as in the application to manipulators, real-world settings also require extended notions of safety to account for differences between the reduced and full-order dynamics.

To provide an example of this, consider adaptive cruise control (ACC), where the control objective is to achieve a desired speed subject to maintaining a safe distance from a lead car. In this setting, consider a reduced-order model (36) defined by a point mass model of a vehicle moving in a straight line:

$$\dot{y} = \underbrace{\begin{bmatrix} y_2 \\ -\frac{1}{m} F_r(y) \end{bmatrix}}_{f_{\text{rom}}(y)} + \underbrace{\begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix}}_{g_{\text{rom}}(y)} u, \qquad F_r(y) = c_0 + c_1 y_2 + c_2 y_2^2$$
$$\dot{z} = v_0 - y_2$$

where y_1 (in meters) is the position, $y_2 = \dot{y}_1$ (in meters per second) is the velocity, m is the mass of the car (in kilograms), the input u (in newtons) represents the wheel force F_w, and F_r is the rolling resistance; typically, c_0, c_1, and c_2 are determined empirically. Finally, z is the distance between the vehicles, where it is assumed that the lead vehicle is traveling at a constant speed v_0.

The key safety constraint is *keep a safe distance from the car in front of you*. This is generally encoded by the "half the speedometer" rule, which states that $D \geq v/2$ (with D in meters and v in kilometers per hour); that is, the distance between two vehicles should be at least half the current speed. Converting this to m and s results in the safety constraint

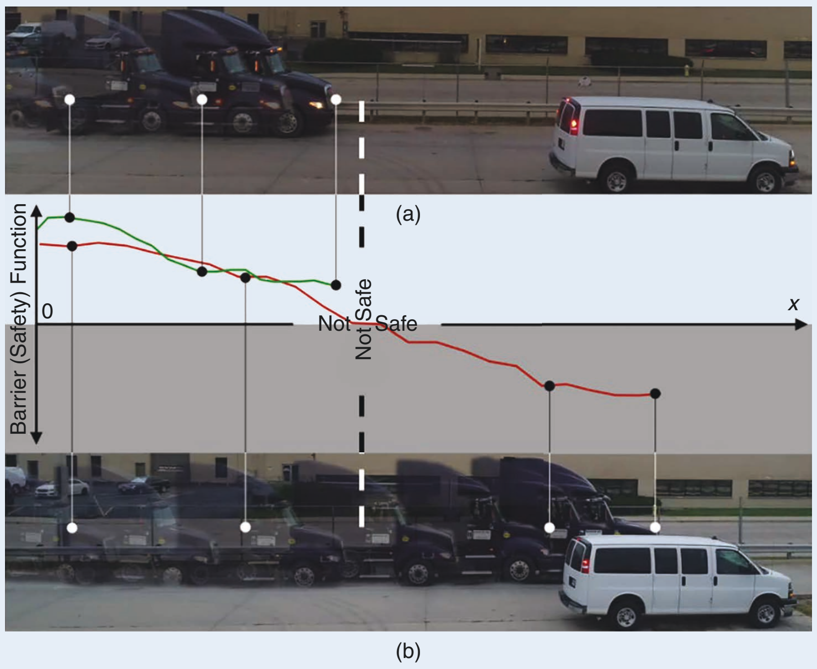


FIGURE S6 A safety filter implemented on a full-scale truck [S12]. (a) When the safety filter (S7) is implemented, safety is achieved. (b) The nominal controller v^d violates the safety condition.

(Continued)

Continuous-Time LCAs in Practice (*Continued*)

$z \geq 1.8y_2$, which can be translated to a barrier function $h(y, z) = z - 1.8y_2 \geq 0$. It is easy to verify that this is a valid CBF and can be implemented in practice [S9], but we consider the generalization

$$h(y, z) = z - (a_0 + a_1 y_2 + a_2 v_0 + a_3 y_2^2 + a_4 y_2 v_0 + a_5 v_0^2)$$

for which the parameters, a_i, can be determined such that $z \geq 1.8y_2$ is satisfied while allowing for actuation limits and other practical considerations to be enforced.

Let $v^d(y)$ be the "nominal" ACC system, that is, the current algorithm on the vehicle, which drives the velocity $y_2 \to v_g$. We can then instantiate a safety filter in the form of a QP:

$$v_{\text{safe}}(y) = \underset{v \in \mathbb{R}^s}{\operatorname{argmin}} \| v - v^d(y) \|^2$$
$$\text{subject to} \quad \dot{h}(y, v) \geq -\alpha h(y) + \frac{\left\| \frac{\partial h}{\partial y} g_{\text{rom}}(y) \right\|^2}{\epsilon(h(y))}, \quad \frac{\partial \epsilon}{\partial h} \geq 0. \tag{S7}$$

The added term $\epsilon(h(y))$ is a "tunable" term that enforces a generalization of input-to-state safety [S10] termed tunable input-to-state safety [S11]. Here, ϵ is a function that can be tuned and must have a positive derivative; we pick $\epsilon(h(y)) = \epsilon_0 e^{\beta h(y)}$. The safety filter (S7) was implemented on a class 8 truck without a trailer [S12]. As shown in Figure S6, the nominal ACC controller v^d results in a safety violation and, in fact, a collision. Using the safety filter on this nominal controller results in safe system behavior.

REFERENCES

[S7] J. Schulman et al., "Motion planning with sequential convex optimization and convex collision checking," *Int. J. Robot. Res.*, vol. 33, no. 9, pp. 1251–1270, 2014, doi: 10.1177/0278364914528132.

[S8] T. Sakai, "On Riemannian manifolds admitting a function whose gradient is of constant norm," *Kodai Math. J.*, vol. 19, no. 1, pp. 39–51, 1996, doi: 10.2996/kmj/1138043545.

[S9] A. Mehra, W.-L. Ma, F. Berg, P. Tabuada, J. W. Grizzle, and A. D. Ames, "Adaptive cruise control: Experimental validation of advanced controllers on scale-model cars," in *Proc. Amer. Control Conf. (ACC)*, Piscataway, NJ, USA: IEEE, 2015, pp. 1411–1418, doi: 10.1109/ACC.2015.7170931.

[S10] S. Kolathaya and A. D. Ames, "Input-to-state safety with control barrier functions," *IEEE Contr. Syst. Lett.*, vol. 3, no. 1, pp. 108–113, Jan. 2019, doi: 10.1109/LCSYS.2018.2853698.

[S11] A. Alan, A. J. Taylor, C. R. He, G. Orosz, and A. D. Ames, "Safe controller synthesis with tunable input-to-state safe control barrier functions," *IEEE Contr. Syst. Lett.*, vol. 6, pp. 908–913, 2022, doi: 10.1109/LCSYS.2021.3087443.

[S12] A. Alan, A. J. Taylor, C. R. He, A. D. Ames, and G. Orosz, "Control barrier functions and input-to-state safety with application to automated vehicles," 2022, *arXiv:2206.03568*.

high-dimensional nonlinear systems (for example, via Lyapunov and barrier functions), yet model uncertainty and "looking ahead" in nonlinear systems is challenging. Adding a reference signal generation layer that uses continuous-time reduced-order models mitigates model uncertainty while still yielding formal guarantees, such as on safety. Adding a discrete planning layer above the reference signal generating layer allows for longer horizon planning, for example, via MPC with a discrete-time linear reduced-order model. Combining these together mitigates the weaknesses at each layer while enjoying their strengths. This use of diverse models, timescales, and control approaches was highlighted through experimental demonstration on a wide variety of robotic systems. We return to the idea of diversity across layers enabling behavior that cannot be achieved by any single layer in the "Architecture Design as Multicriterion Optimization" section, where we introduce a quantitative notion of a *diversity-enabled sweet spot* (*DeSS*) in LCAs.

Final Remarks

The success of LCAs in robotic systems, and the ability to add and remove layers as needed, points to the power of these methods. It also conveys their complexity: different models at different layers, and the interfacing among these models, result in complex and notationally intensive mathematical models, and establishing formal guarantees becomes daunting. Yet, the fact that these approaches work in practice, and are widely understood as the "way to control robots," points to the value in formalizing and analyzing LCAs. It can be argued that this is a central challenge for the control community moving forward: going beyond homogeneous system models and analyzing heterogeneous models interacting within an LCA.

ARCHITECTURE DESIGN AS MULTICRITERION OPTIMIZATION

The previous sections illustrate how an LCA can be naturally derived from a global decision and control problem, and they provide a concrete instantiation of these ideas in the context of robotic systems. These results highlight both the power of LCAs and the art and complexity involved in designing them. We highlight that many idealized assumptions were made in the first part of the article: we assumed that the control system hardware was already fixed, that we knew how many layers were needed and what each layer should do, and how layers should interact within the LCA. In this section, which marks the start of the second half of the article, we try to address some of these idealized assumptions and propose a framework rooted in multicriterion optimization for quantitative reasoning about architecture design choices, such those described in the previous two sections. A key theme that we explore in this section is that while each layer may be subject to specific constraints and tradeoffs, by leveraging *diversity across layers*, these tradeoffs can be mitigated to yield high-performing LCAs, such as those highlighted in the previous sections.

We begin with a familiar illustrative example: long-distance travel. We consider three possible "travel layers," namely, air travel (implemented via aircraft and airports), public transit (implemented via buses and bus stops), and walking (implemented via human sensorimotor control). Each of these travel layers is subject to speed–accuracy tradeoffs (SATs), which are themselves a function of architectural design choices (but we will not focus on these here): air travel is fast but inaccurate since we can fly only between airports; public transit is moderately fast and moderately accurate, as we are limited to bus stops; and walking is slow but extremely accurate. These travel layers can be placed in a speed–accuracy plot, as in Figure 7.

However, as we all know, when traveling long distances, it is most efficient to appropriately combine these travel layers: we walk to the bus stop, take the bus to the airport, fly to the airport nearest our destination, take the bus to the stop nearest our destination, and then walk to our destination. Although not usually thought of in this way, this is an LCA for travel, with air travel serving as a fast but inaccurate layer, public transit serving as an intermediate layer, and walking as a slow but accurate layer. The resulting LCA, which implements diverse layers using diverse components, is nearly as fast as flying and just as accurate as walking. We call such an LCA that leads to minimal tradeoffs between speed and accuracy an architectural *sweet spot*.

It is our claim that such sweet spots are ubiquitously enabled through *diverse layers* being appropriately combined in LCAs. (Diverse layers typically require diverse hardware, or levels. We discuss levels in more detail in the "Key Concepts in Control Architecture" section.) Indeed, we see comparable diversity in sensorimotor control, robotics, computer networks, and biology, in order to mitigate what appears to be a universal constraint on individual layers, namely that the lower the layer in the "stack," the faster the layer must operate but the more limited it is in its capabilities. Nevertheless, by appropriately combining slow decision making with moderate speed trajectory generation and fast feedback control, we are able to design autonomous systems that are as flexible as the decision-making layer and as accurate and fast as the feedback layer. In the remainder of this section, we propose a quantitative framework for reasoning about such *DeSSs*.

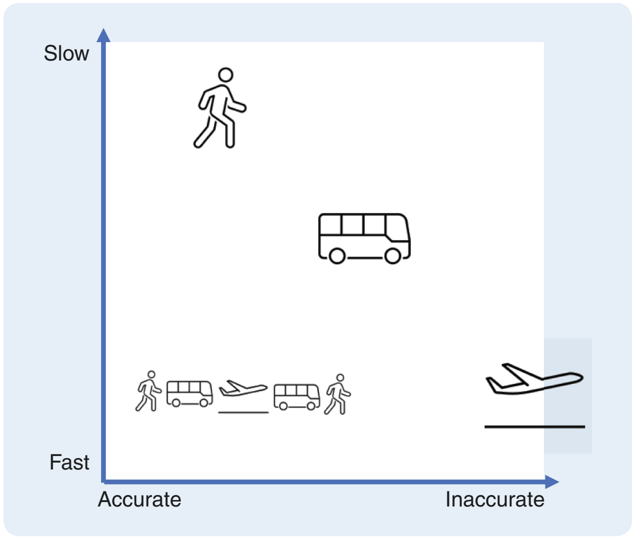


FIGURE 7 Each individual "travel layer" is subject to SATs, but combining them appropriately in an LCA enables an overall transportation system with minimal tradeoffs in either speed or accuracy.

Pareto Surfaces and Pareto Minimax Points

Our goal is to both characterize the fundamental tradeoffs that different control architectures induce and to determine whether a control architecture enjoys a (diversity-enabled) sweet spot. To formalize these concepts, we turn to multicriterion optimization.

Multicriterion Optimization

Multicriterion optimization problems seek to minimize a *vector-valued objective function*. Following [56, Ch. 4], we consider a vector optimization problem that seeks to minimize the vector-valued objective

$$C(x) = (C_1(x), \ldots, C_d(x))$$

with respect to the positive orthant $\mathbb{R}^d_+$. Such an optimization problem should be interpreted as having d different objectives C_i, each of which we would like to make small.

In contrast to scalar-valued objectives, we must take care in defining appropriate notions of optimality. In particular, we may define both optimal and Pareto-optimal points. A feasible point x^* is *optimal* if it is *unambiguously better* than any other feasible point, where better is defined in terms of the partial order induced by the positive orthant; that is, a feasible x^* is optimal if for any other feasible y, $C(x^*) \preceq C(y)$, that is, if $C_i(x^*) \le C_i(y)$ for all $i = 1, \ldots, d$. Most engineering design problems are subject to fundamental tradeoffs between optimization criteria C_i, and such an optimal point typically does not exist. Instead, a family of *Pareto-optimal* points can be defined, wherein a feasible point x^{po} is Pareto optimal if for any feasible y, if $C(y) \preceq C(x^{\text{po}})$, then $C(y) = C(x^{\text{po}})$; that is, a feasible point x^{po} is Pareto optimal if no other point exists that is unambiguously better. Indeed, the existence of multiple Pareto-optimal points implies that there is a fundamental tradeoff between the different objectives.

The standard approach to solving such a multicriterion optimization problem is via *scalarization*. A common approach to scalarization is to take a weighted sum of the objectives; that is, for $\lambda \in \mathbb{R}^d_{++}$, define the scalarized objective $C_\lambda(x) = \lambda^T C(x) = \Sigma_{i=1}^d \lambda_i C_i(x)$. By sweeping over weighting parameters $\lambda \succ 0$, we obtain a family of Pareto-optimal points $x^{\text{po}}(\lambda)$, which in turn defines a Pareto surface $(C_1(x^{\text{po}}(\lambda)), \ldots, C_d(x^{\text{po}}(\lambda))) \subset \mathbb{R}^d$. (Up to boundary points, such an approach is guaranteed to recover all Pareto-optimal points if the objective functions C_i are convex in x; see [56, Ch. 4].) An alternative, but also important, scalarization approach is to consider minimizing the maximum of the objectives; that is, $C_{\max}(x) = \max\{C_1(x), \ldots, C_q(x)\}$. The resulting solution x^{mm} is called the minimax Pareto-optimal point.

A familiar example of bicriterion optimization in control is LQR optimal control. Indeed, defining the vector-valued objective $\left(\Sigma_{k=0}^N \|x(k)\|_2^2, \Sigma_{k=0}^{N-1} \|u(k)\|_2^2\right)$, we recognize the LQR objective $\Sigma_{k=0}^{N-1}\|x(k)\|_2^2+\rho\|u(k)\|_2^2+\|x(N)\|_2^2$ as a scalarization of the competing small state and control cost objectives. An alternative, albeit less common, scalarization would be to consider the maximum objective $\max\left\{\Sigma_{k=0}^{N}\|x(k)\|_2^2, \Sigma_{k=0}^{N-1}\|u(k)\|_2^2\right\}$. See Figure 8 for an example of a typical Pareto curve for an LQR problem.

Sweet Spots Are Nearly Optimal Points

We now have the required concepts to formally define a *sweet spot*. Intuitively, a sweet spot is a point on the Pareto surface that is *nearly optimal*. We quantify this notion of near optimality by defining a σ *sweet spot* to be a minimax Pareto-optimal point that is σ away from being an optimal point in the following sense:

$$\sigma := \sup_{\lambda \succ 0}\max\{C_1(x^{\mathrm{mm}})-C_1(x^{\mathrm{po}}(\lambda)), \ldots C_d(x^{\mathrm{mm}})-C_d(x^{\mathrm{po}}(\lambda))\}. \tag{43}$$

In words, the measure σ characterizes the biggest loss in optimality *in any of the criterion* C_i of a minimax Pareto-optimal point relative to *any other Pareto-optimal point*. Note that if there exists an optimal point x^*, then $\sigma = 0$ and that σ increases as the tradeoff between objectives becomes more severe. See Figure 9 for a qualitative illustration of when σ is small or large as a function of the geometry of the Pareto surface.

Diversity Enables σ Sweet Spots

One of our key claims, which is broadly supported by examples in engineering, science, and biology, is that diversity enables nearly optimal sweet spots despite individual layers being subject to strict and at times severe tradeoffs. We begin with a simple stylized example for which the suboptimality measure σ can be computed exactly. We then explore a case study in sensorimotor control in the next section.

Illustrative Example: Bicriterion Least Squares

We study the bicriterion least-squares problem

$$\text{minimize}_x \ (\text{with respect to } \mathbb{R}_+^2)\,(\|A_1x-b_1\|_2^2, \|A_2x-b_2\|_2^2) \tag{44}$$

through the lens of DeSSs. We assume that $b_1, b_2 \in \mathbb{R}^m$ $A_1, A_2 \in \mathbb{R}^{m\times 2m}$, and $x \in \mathbb{R}^{2m}$. Our stylized architecture design problem is to design the matrices A_1 and A_2 by selecting their rows, possibly with replacement, from a palette of $2m$ linearly independent rows $\mathcal{V} = \{v_1, \ldots, v_{2m}\} \subset \mathbb{R}^{2m}$. Our goal is to quantify how *diversity in the row spaces of* A_1 *and* A_2 *affects the resulting* σ *sweet spot* of the bicriterion problem. We begin with some simple observations:

- If we assume that we design A_1 and A_2 to respectively have full row rank, then it is clear that each individual objective can be made zero. We make this assumption going forward, and hence, we have $\sigma = \max\left\{\|A_1x^{\mathrm{mm}}-b_1\|_2^2, \|A_2x^{\mathrm{mm}}-b_2\|_2^2\right\}$.

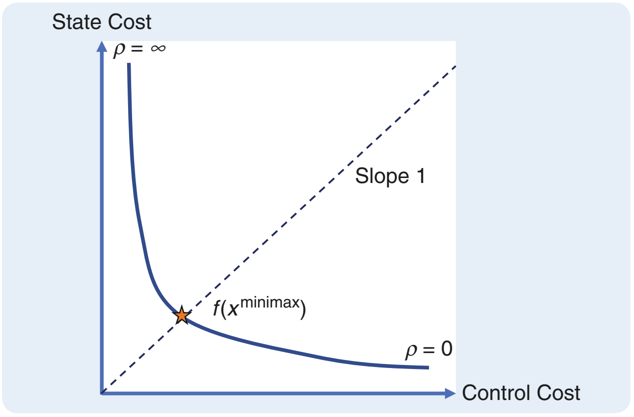


FIGURE 8 A Pareto surface and minimax Pareto-optimal point for an LQR. By varying the weight ρ on the control cost, the Pareto surface is traced out.

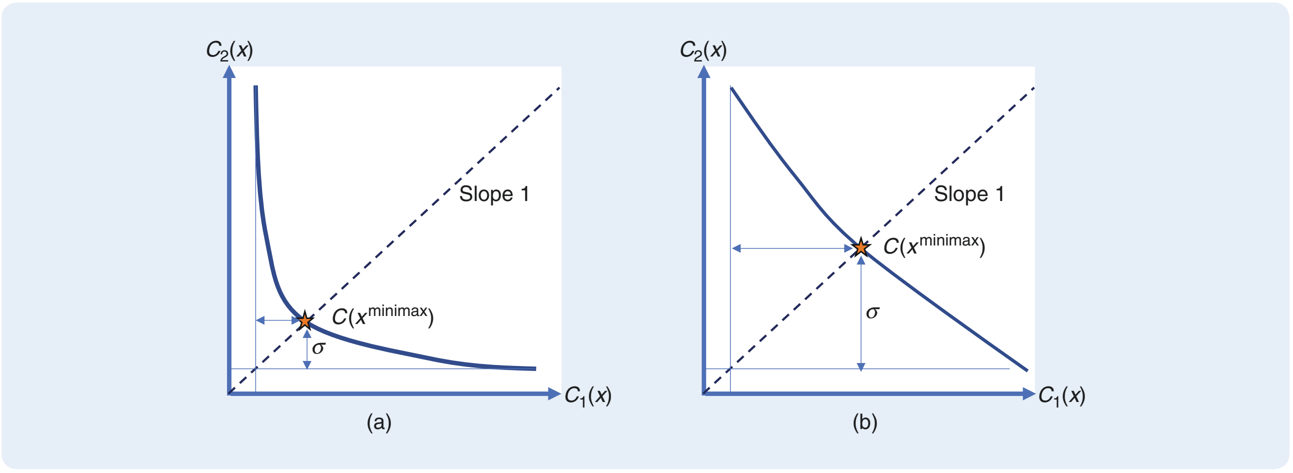


FIGURE 9 The measure σ of a minimax Pareto point quantifies how much of a tradeoff there is between competing objectives by comparing the minimax Pareto-optimal point to all other Pareto-optimal points. An (a) small σ and (b) large σ A small σ indicates that there exists a point that is *nearly optimal*.

Diversity enables nearly optimal sweet spots despite individual layers being subject to strict and at times severe tradeoffs.

» If we further assume that the stacked matrix $\bar{A} = [A_1^T, A_2^T]^T$ has full row rank, that is, that A_1 and A_2 do not share any rows selected from $\mathcal{V}$, then $\sigma = 0$. This is easily verified by setting $x^{\text{mm}} = \bar{A}^{-1}\bar{b}$, with $\bar{b} = (b_1, b_2)$.

Thus, our remaining task is to characterize the σ sweet spot for optimization problem (44) when A_1 and A_2 share a common row space. Toward that end, we consider the minimax scalarization (44)

$$\text{minimize}_x \ \max\{\|A_1x - b_1\|_2^2, \|A_2x - b_2\|_2^2\} \tag{45}$$

and its dual (see "Bicriterion Least Squares: Additional Details" for details):

$$\begin{aligned} \text{maximize}_{\lambda_1,\lambda_2,\mu_1,\mu_2} \quad & 2\mu_1^T b_1 - \frac{\|\mu_1\|_2^2}{\lambda_1} - 2\mu_2^T b_2 - \frac{\|\mu_2\|_2^2}{\lambda_2} \\ \text{subject to} \quad & A_1^T\mu_1 = A_2^T\mu_2 \\ & \lambda_1 + \lambda_2 = 1,\ \lambda_1, \lambda_2 > 0. \end{aligned} \tag{46}$$

This allows us to immediately reconfirm that $\sigma = 0$ if A_1 and A_2 do not share any rows, as in this case, any dual feasible solution has $\mu_1 = \mu_2 = 0$. Similarly, when $A_1 = A_2$, a simple argument shows that $\sigma = 1/4\|b_1 - b_2\|_2^2$. A generalization of this argument is presented in Theorem 2 and proved in "Bicriterion Least Squares: Additional Details," which allows us to characterize the solution when A_1 and A_2 share k rows.

Theorem 2

Consider the bicriterion least-squares problem (44). Suppose that A_1 and A_2 are both full row rank, and assume, without loss of generality, reordering rows in A_i and elements in b_i if necessary, that A_1 and A_2 share their first k rows. Then, the minimax solution x^{mm} to the scalarized problem (45) defines a $\left((1/4)\|E_k^T(b_1 - b_2)\|_2^2\right)$ sweet spot, as defined in (43). Here, $E_k = [e_1, \dots, e_k]$, with $e_i \in \mathbb{R}^m$ the standard basis elements.

Theorem 2 makes clear that the more diverse the matrices A_1 and A_2, that is, the smaller the number of shared rows k, the less severe the tradeoff; similarly, the less diverse the matrices A_1 and A_2, that is, the larger the number of shared rows k, the more severe the tradeoff. We compute a family of the resulting Pareto curves and minimax-optimal points for $m = 5$ in Figure 10(a) and plot the evolution of the suboptimality measure σ as a function of the number of

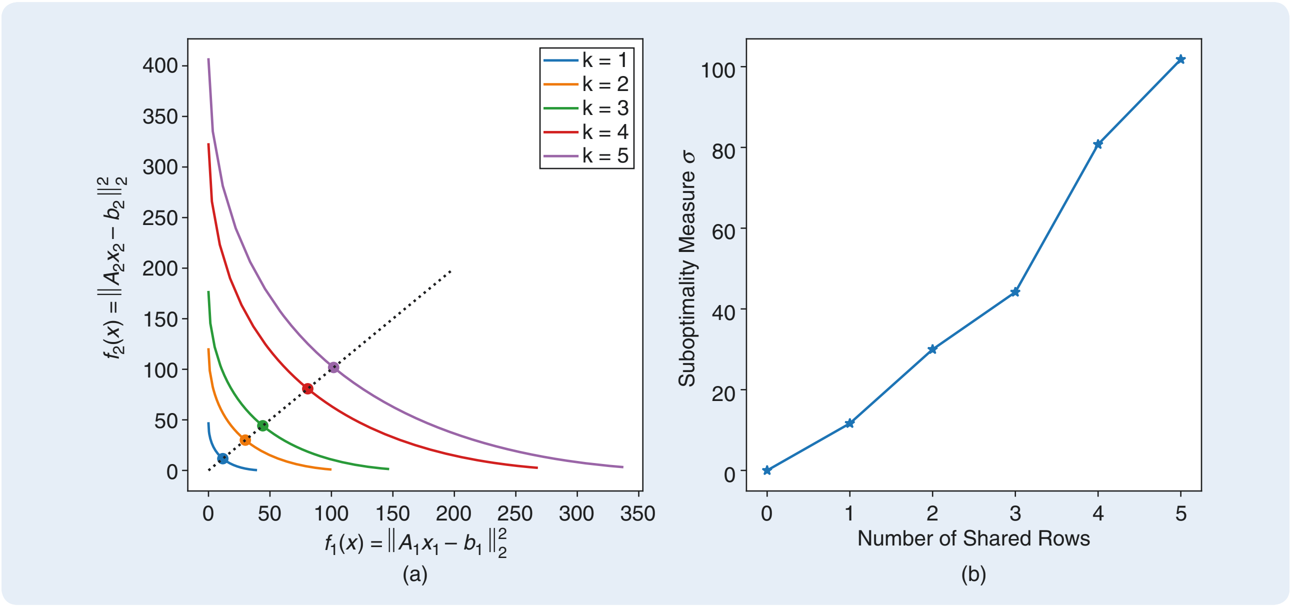


FIGURE 10 Increased diversity in row spaces provably leads to less severe tradeoffs, as quantified by a smaller suboptimality measure σ, in the bicriterion least-squares problem (44). (a) We observe how the Pareto surface for the bicriterion least-squares problem (44) becomes increasingly unfavorable as we decrease the diversity across A_1 and A_2. This is true both in terms of the overall Pareto surface and the suboptimality measure σ. (b) We plot the suboptimality measure σ as a function of shared rows across A_1 and A_2 for the bicriterion least-squares problem (44). We observe that σ deteriorates as we decrease diversity across A_1 and A_2.

shared rows in Figure 10(b). Parameters are randomly generated so as to ensure the requisite linear independence conditions and such that $|(b_1 - b_2)_i|$ is approximately even for all i. Details of how the parameters are generated can be found in "Bicriterion Least Squares: Additional Details," and the code used to create these plots can be found at https://colab.research.google.com/drive/1jK0fJbSzxBb78yHcru8EsRle9V-aZYYx?usp=sharing.

To further gain insight into the LCA design problem, let us view A_1 and A_2 as defining two layers, with layer i aimed at addressing control subtask b_i. This analogy reinforces that diversity is not enough to ensure a small suboptimality measure σ. The control subtasks, here characterized by b_1 and b_2, must themselves also be compatible with system diversity (or lack thereof). For example, even if $k=1$, a very large $(e_1^T(b_1-b_2))^2$ will nevertheless lead to a severe tradeoff between optimizing the two objectives, resulting in a large σ. Conversely, diversity is needed in A_1 and A_2 only if the control subtasks b_1 and b_2 are also diverse; if $b_1 = b_2$, then $A_1 = A_2$ will still yield $\sigma = 0$. Connecting this back to the travel example, if a destination is just a block away, then diversity is not required, and just walking is an optimal travel LCA. Conversely, if the destination is extremely remote, then the three layers of commercial air travel, public transit, and walking will still be very slow. Thus, this simple example hints at an explanation as to why diverse layers are needed by systems that must accomplish diverse tasks across diverse environments at diverse spatiotemporal resolutions. We explore a (still stylized) control problem in the next section that further reinforces this concept.

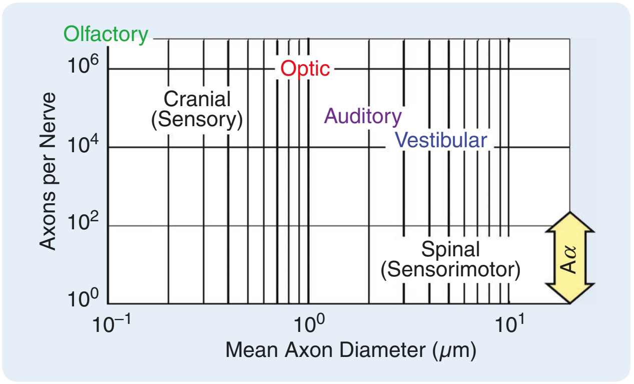


FIGURE 11 The axons per nerve ($\propto$ resolution) versus the mean axon diameter ($\propto$ speed) for four key cranial nerves and the largest ($A\alpha$) sensorimotor axon that occurs in spinal and peripheral nerves (in copies from one to hundreds).

Bicriterion Least Squares: Additional Details

DUAL TO PROBLEM (45)

Recall that the minimax least-squares problem is given by

$$\text{minimize}_x \ \max\{\|A_1x-b_1\|_2^2, \|A_2x-b_2\|_2^2\}. \tag{S8}$$

To derive an interesting dual problem, we consider the following equivalent problem:

$$\begin{aligned} \text{minimize}_{x_1,x_2,t} \quad & t \\ \text{subject to} \quad & \|A_1x_1-b_1\|_2^2 \le t \\ & \|A_2x_2-b_2\|_2^2 \le t \\ & x_1 = x_2. \end{aligned} \tag{S9}$$

The corresponding Lagrangian is given by

$$\begin{aligned} L(t,x_1,x_2,\lambda_1,\lambda_2,\nu) = t + \lambda_1(\|A_1x_1-b_1\|_2^2 - t) + \lambda_2(\|A_2x_2-b_2\|_2^2 - t) \\ + 2\nu^T(x_1-x_2) \end{aligned} \tag{S10}$$

for $\lambda_1, \lambda_2 > 0$.

We rewrite the Lagrangian in the more suggestive form

$$\begin{aligned} L(t,x_1,x_2,\lambda_1,\lambda_2,\nu) = t(1-\lambda_1-\lambda_2) + \lambda_1\|A_1x_1-b_1\|_2^2 \\ + 2\nu^Tx_1 + \lambda_2\|A_2x_2-b_2\|_2^2 - 2\nu^Tx_2. \end{aligned} \tag{S11}$$

Recalling that the dual function is defined as

$$g(\lambda_1,\lambda_2,\nu) = \inf_{t,x_1,x_2} L(t,x_1,x_2,\lambda_1,\lambda_2,\nu)$$

and that the dual problem is given by

$$\text{maximize}_{\lambda_1,\lambda_2>0,\nu} \ g(\lambda_1,\lambda_2,\nu)$$

we immediately conclude that $g(\lambda_1,\lambda_2,\nu)$ is bounded below if and only if $\lambda_1+\lambda_2 = 1$, $\nu \perp \ker(A_1) \Leftrightarrow \nu = A_1^T\mu_1$ for unconstrained μ_1, and $\nu \perp \ker(A_2) \Leftrightarrow \nu = A_2^T\mu_2$ for unconstrained μ_2. Under these conditions, the minimizers are $x_i^* = -A_i^\dagger((\mu_i/2\lambda_i)+b_i)$, which, after simplification and collecting like terms, results in the dual problem (46).

PROOF OF THEOREM 2

Under the assumptions of Theorem 2, the set of feasible μ_1 and μ_2 in the dual problem (46) is given by $\mu_1 = \mu_2 = E_k\alpha$ for any $\alpha \in \mathbb{R}^k$. It follows from the Karush–Kuhn–Tucker conditions of the dual problem (46) that the optimal solution is $(\mu_1^*, \mu_2^*, \lambda_1^*, \lambda_2^*) = (E_kE_k^T(b_1-b_2)/4, E_kE_k^T(b_1-b_2)/4, 1/2, 1/2)$ and that the corresponding σ sweet spot of the problem satisfies

$$\sigma = \frac{1}{4}\|E_k^T(b_1-b_2)\|_2^2.$$

EXPERIMENTAL DETAILS FOR FIGURE 10

We set $m=5$ and $n=10$ and draw the entries of A_1 and A_2 independent identically distributed (i.i.d.) according to a standard normal distribution (duplicating shared rows across A_1 and A_2 as necessary). This ensures the linear independence of rows with a probability of one. We draw the entries of b_1 i.i.d. according to a standard normal and set $b_2 = b_1 + \Delta$, where $\Delta \sim \mathcal{N}(0, 100I_m)$.

A CASE STUDY IN SENSORIMOTOR CONTROL

We adapt the following from Nakahira et al. [57] and Nakahira et al. [58]. Our goal in this section is to highlight how diverse layers, and the diverse hardware used to implement them, in the human sensorimotor LCA (see Figure 11) enable astonishingly efficient DeSSs despite severe SATs. To that end, we first derive robust performance limits for a simplified model of sensorimotor control subject to communication that is delayed and quantized due to its implementation using physiological hardware composed of axons. We then identify a simple layered architecture composed of delayed but accurate vision (planning) and fast but inaccurate reflex control (feedback) layers and show that this architecture is optimal for the aforementioned sensorimotor control model and leads to a DeSS. Finally, we show that despite the simplicity of the model and analysis, it is shockingly predictive of real-world behavior, as confirmed in "Experimental Validation in a Biking Simulator."

A Simplified Model

Consider an initial minimal model with discrete-time dynamics

$$\begin{aligned} x(k+1) &= ax(k) + w(k-T_w) + Q(u(k-T_u)) \\ u(k) &= K(x(0:k), w(0:k), u(0:k-1)) \end{aligned} \tag{47}$$

where $x(k) \in \mathbb{R}$ is the state, $w(k) \in \mathbb{R}$ is the disturbance, $u(k) \in \mathbb{R}$ is the control action generated by the controller K, and $Q: \mathbb{R} \to \mathscr{S}_R$, for $\mathscr{S}_R \subset \mathbb{R}$, a finite set of cardinality 2^R, is a quantizer that limits communication between the controller and the actuator to R bits/sampling interval. The form of the control law in system (47) implies that the controller is *full information*, as the control signal $u(k)$ is allowed to depend on all current and past states $x(0:k)$, current and past disturbances $w(0:k)$, and past control actions $u(0:k-1)$.

A schematic for this model appears in Figure 12, where we use P to denote the plant defined by (47). The control signal $\boldsymbol{u}$ is transmitted to the actuator (colocated with the physical plant P) via the communication channel C, which is defined by the composition of the quantizer Q with the delay block T_u. This delay block implies that the controller command $u(k)$ takes $T_u(\geq 0)$ sampling intervals to reach and be executed by the actuators; that is, $u(k)$ affects the plant $T_u + 1$ sampling intervals only later. (We assume that the channel C is memoryless and stationary with rate R, allowing us to restrict the quantizer Q to be memoryless and static as well. Generalizations that lift this assumption can be found in [57].) Note that because Q and the delay block commute the dynamics (47) and Figure 12 are indeed consistent. We assume $\|w(k)\|_\infty \leq 1$ and $x(0) = 0$. The disturbance is known to the controller, with an advance warning of $T_w(\geq 0)$ sampling intervals; that is, the controller has access to $w(0:k)$ even though the disturbance affects the plant $T_w + 1$ sampling intervals only later.

The robust control problem can then be posed as

$$\begin{aligned} &\underset{(K,Q)\in \mathcal{Q}_R}{\text{minimize}} \quad \sup_{k\geq 0, \|w(k)\|_\infty \leq 1} \|x(k)\|_\infty \\ &\text{subject to} \quad \text{dynamics (54)} \end{aligned} \tag{48}$$

where $\mathcal{Q}_R$ is the space of control laws defined by the pair of mappings (K, Q), with Q constrained to be a static memoryless quantizer of rate R; that is, $Q: \mathbb{R} \to \mathscr{S}_R$. This cost function is standard in L_1 robust control [59], except that a communication channel C, composed of a quantizer Q and a delay T_u, is inserted into the feedback loop. Perhaps surprisingly, this problem formulation still allows for a simple and intuitive analytic solution. Indeed, without quantization or delay, the control law

$$u(k) = -ax(k) - w(k)$$

ensures that $x(k+1) = 0$. Thus, any errors in the state are a direct consequence of quantization and/or delay or to saturation of the control signal u.

Fundamental Limits Due to Delay and Quantization

In this section, we provide an exact solution to the robust control problem (48) for fixed advance warning T_w and actuation delay T_u. In particular, we show that the worst-case state deviation can be expressed as a function of the plant pole a, the channel rate R, and the *net delay* of the system $T := T_u - T_w$. The achievable performance takes a different form depending on the net delay regime that the system is operating under. When the net delay T is positive ($T > 0$), this corresponds to a system in which the control action $u(k)$ can affect the plant T sampling intervals only *after* the disturbance $w(k)$ affects the state. Conversely, when the net delay T is nonpositive ($T \leq 0$), this corresponds to a system in which there is *advance warning* of the disturbance, allowing the controller to act in advance. These two qualitatively different cases are treated separately. We then use these insights in the next section to pose an LCA design problem that seeks to identify an appropriate combination of fast but inaccurate and slow but accurate neural signaling to enable a DeSS.

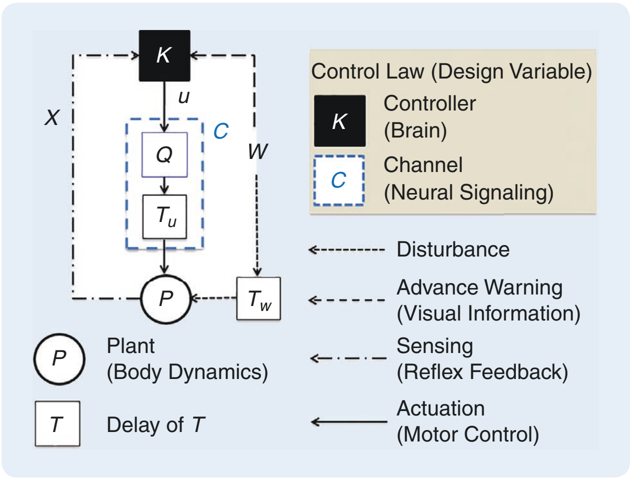

K
u
X
C
Q
W
T_u
P
T_w
Control Law (Design Variable)
K
Controller (Brain)
C
Channel (Neural Signaling)
Disturbance
Advance Warning (Visual Information)
P
Plant (Body Dynamics)
Sensing (Reflex Feedback)
T
Delay of T
Actuation (Motor Control)

FIGURE 12 The feedback system model for sensorimotor control.

Theorem 3

Suppose that $|a| < 2^R$. Then, the minimal state deviation achievable in robust control problem (48) is

$$\begin{array}{ll} \sum_{i=1}^{T} |a^{i-1}| + |a^T|(2^R - |a|)^{-1} & \text{if } T > 0 \\ (2^R - |a|)^{-1} & \text{if } T \leq 0. \end{array} \quad (49)$$

Conversely, if $|a| \geq 2^R$, then the system cannot be stabilized, and the optimal value to optimization problem (48) is infinite.

The performance limits (49) are remarkably simple and intuitive. The net warning case ($T \leq 0$) has only one term due to quantization, with the stabilizability condition $|a| < 2^R$ well known from the networked control system literature [60]. With no dynamics ($a = 0$), this reduces to a trivial rate distortion theorem with error 2^{-R}. The net delayed case ($T < 0$) is more interesting, with the first term due to the delay alone and the second term an additional contribution due to quantization. As expected, both grow rapidly with increased net delay T and unstable $a > 1$, for reasons familiar and intuitive.

SATs in Neural Signaling

We now add a tradeoff between temporal and spatial resolution in neural signaling to our model via the net delay T and data rate R. We believe this is the first important constraint in explaining the extreme heterogeneity found in the nervous system and is analogous to the SAT highlighted in the travel example above. The nervous system communicates between components and the body with a variety of nerves, which are bundles of axons. Axons are the wiring by which spiking neurons communicate long-range using action potentials, and it is possible to derive some rough tradeoffs from well-known physiology. Figure 11 examines some of the tremendous diversity of axon numbers and sizes among the cranial and peripheral nerves. We argue that much of this arises due to hard constraints on speed versus accuracy.

We suppose that our channel C (see Figure 12) is a single nerve with uniform signaling delay T_s and assume that the total delay T_u is the sum $T_u = T_s + T_c$ with an additional fixed delay T_c due to gray matter computation and other communications. Initially, we assume that T_c is fixed and given and that T_s is variable and depends on the nerve composition, as in Figure 11. Following the arguments provided in [57] and [58], we use the physiologically plausible yet remarkably simple relationship between data rate R and signaling delay T_s:

$$R = \lambda_\alpha T_s \quad (50)$$

where λ_α is a resource measure that scales with the axon area α.

Next, we explore the surprisingly rich consequences of the constraint $R = \lambda_\alpha T_s$ on our minimal model of sensorimotor control using Theorem 3. For simplicity, we write λ from now on, as the resource dependence is understood. One can verify that if $R = \lambda T_s$ and $T_u := T_s + T_c$, then the optimal performance specified in Theorem 3 becomes

$$\begin{array}{ll} \sum_{i=1}^{T} |a^{i-1}| + |a^T|(2^{\lambda T_s} - |a|)^{-1} & \text{if } T > 0 \\ (2^{\lambda T_s} - |a|)^{-1} & \text{if } T \leq 0. \end{array}$$

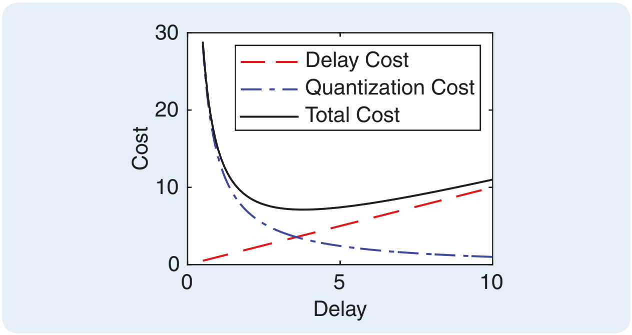


FIGURE 13 The impact of speed versus accuracy. The cost of delay $\sup_{\|w\|_\infty \leq 1} \|x_d\|_\infty$, the cost of quantization $\sup_{\|w\|_\infty \leq 1} \|x_q\|_\infty$, and the total cost $\sup_{\|w\|_\infty \leq 1} \|x_d\|_\infty$ are shown with varying delay T_s when $\lambda = 0.1$, $a = 1$.

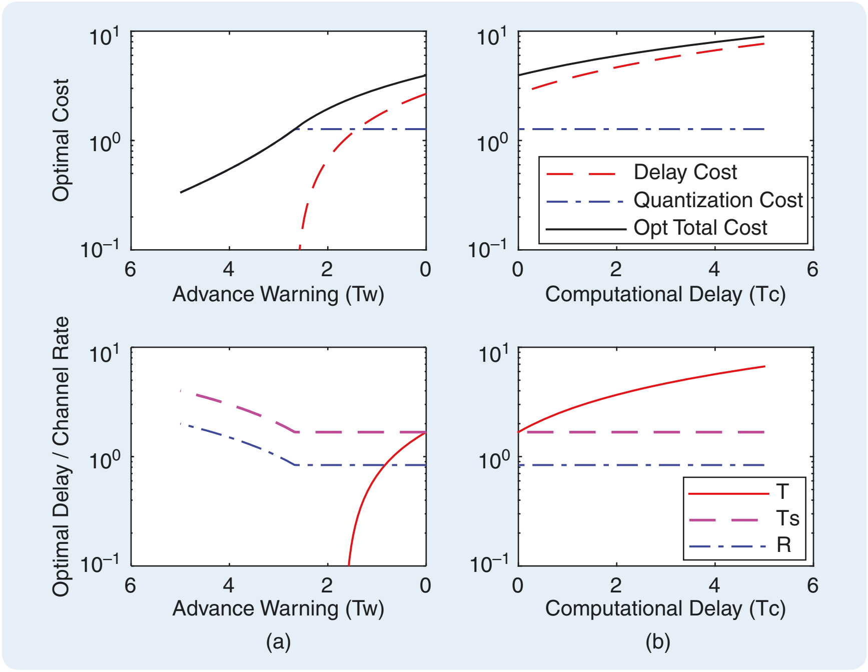


FIGURE 14 (a) A warned ($T_c = 0, T_w > 0$) system versus (b) a delayed ($T_w = 0, T_c > 0$) system.

Figure 13 describes the system performance when varying delay T_s (and, thus, channel rate R) for $T_c = T_w = 0$ and a fixed resource level α. Increased delay increases the delay error term $\sup_{\|w\|_\infty \le 1} \|x_d\|_\infty := \Sigma_{i=1}^{T} |a^{i-1}|$ but reduces the quantization error term $\sup_{\|w\|_\infty \le 1} \|x_q\|_\infty := (2^{\lambda T_s} - |a|)^{-1}$. Consequently, the optimal system-level performance is achieved at intermediate levels of delay and channel rate. Because of the exponential dependence, there is no analytic formula for the optimum, but the error is convex, and the minimum is easily found numerically. Next, we consider in more detail the consequences of these formulas by varying the additional delays and plotting the resulting optimal errors, bits, and delay.

Figure 14 gives the optimal delays T_s (and resulting net delay T) and channel rate $R = \lambda T_s$ that achieve the minimum total error when varying $T_w \ge 0$ and $T_c \ge 0$ separately in the two special cases: 1) $T = T_u - T_w \le 0$ (warned) and 2) $T = T_s + T_c > 0$ (delayed). What results are clearly two distinct regimes with distinct physiology. When the computation delay T_c is greater than zero, the system has a net delay T, and the delay cost increasingly dominates the total cost, leading to both the data rate R and signaling delay T_s becoming constant (that is, suggesting axons of a large and constant radius ρ) independently of T_c. This corresponds to the reflexes on the right half of Figure 11, with nerves having relatively few large axons—these are the physiological analogs to aircraft and airports from our travel example. The total error, due mostly to delay, can be much larger than the disturbance. Concretely, in running or cycling on rough terrain or through heavy traffic, a relatively small but well-placed perturbation to the foot or wheel can be amplified into a crash, even a fatal one; this effect gets worse at high speeds, when the delay is relatively larger. Our nervous system invests in large nerves, axons, and muscles to avoid such crashes, consistent with the theory.

With increasing advance warning $T_w > 0$, the net delay T becomes nonpositive, and in this case, the errors due to quantization increasingly dominate the total cost. Further, this total cost goes to zero as T_w increases, exactly the opposite of the delayed case. Further, as the advance warning T_w increases, so does the data rate R, and consequently, the axon radius ρ decreases (as $\alpha \approx \pi R \rho^2$ is fixed). This corresponds to the left side of Figure 11, with many relatively small axons—these are the physiological analogs to walking in our travel example. In running or cycling, we can start with huge errors to remotely located objects and given enough time, drive them to zero. Here, we are limited largely by the resolution of our vision in accurately locating the object, again consistent with the theory.

Thus, we have an extremely simple model that connects the high-layer requirements of advance warning and planning (for example, as enabled by vision) to the low-layer control implemented by fast reflexes. In the sequel, we explore further aspects of this model and introduce additional constraints and generalizations.

A Minimal LCA

One of the most important features of a visual system is its distributed nature, in which sensors, actuators, and computational components are interconnected via sparse communication. Figure 15 sketches a minimal model of this kind that is composed of two copies of each component in Figure 12. The plant dynamics are given by $x(k+1) = ax(k) + u(k) + w(k)$, except the disturbance is now composed of two terms $w(k) = v(k) + r(k - T_r)$, as is the control action $u(k) = u_L(k - T_L) + u_H(k - T_H)$, each generated by its own sensors, computing, and communication components. Visual trajectory planning is done through the control loop involving Q_H, which is responsible for tracking, via the control signal $u_H(k)$, a visual target whose change in position is captured by r. We assume a very simplified view of vision whereby remote (in space) sensing means that $r(k)$ is seen, but it takes T_r for the disturbance to arrive, effectively creating an advance warning of T_r, though the physical details are all causal.

On the other hand, local (reflex) compensation is done through the control loop involving Q_L. Disturbances, such as those caused by body and head motion, are captured by v and are sensed directly by the vestibular–ocular reflex, which computes a control action $u_L(k)$ to compensate. The control commands $(u_H(k), u_L(k))$ from both loops are sent to the plant through different signaling pathways, modeled by channels with rates R_H and R_L and

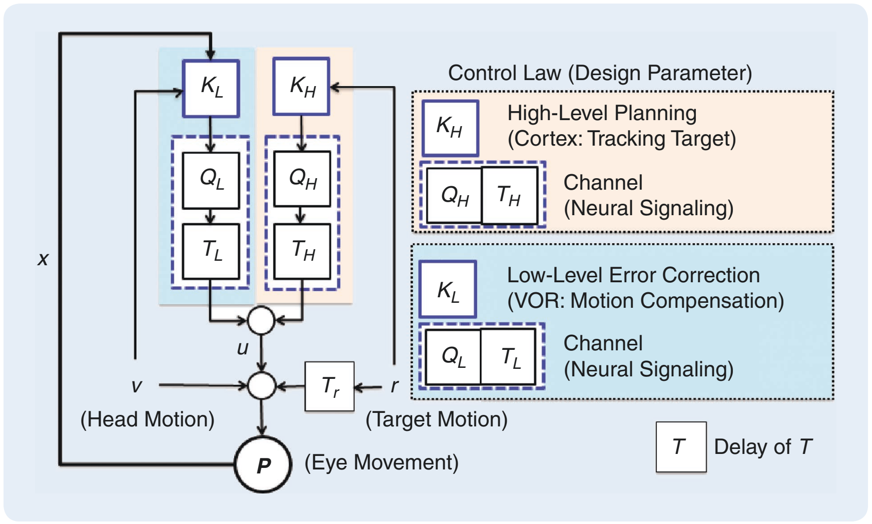


FIGURE 15 Modeling visual processes as an LCA. VOR: vestibular–ocular reflex.

delays T_H and T_L, respectively, after which their gains are summed to produce the final previously described control action $u(k) = u_L(k - T_L) + u_H(k - T_H)$. Connecting this LCA back to the formalism introduced in the "LCAs via Optimal Control Decomposition" and "LCAs for Robotic Systems" sections, we immediately recognize u_H as a *feedforward* control term computed at the *planning layer*, which provides advance warning of the coming reference position $\boldsymbol{r}$ and u_L as a *feedback* control term computed at the *feedback control layer* and executing in near real time to compensate for unforeseen disturbances $\boldsymbol{v}$.

Using the tradeoff (50) in both signaling pathways, and bounding $\|\boldsymbol{v}\|_\infty$ and $\|\boldsymbol{r}\|_\infty$ from above by one and δ, respectively, the optimal performance is then given by

$$\left\{\sum_{i=1}^{T_L}|a^{i-1}|+|a^{T_L}|(2^{R_L}-|a|)^{-1}\right\}+\delta(2^{R_H}-|a|)^{-1}.$$

This result follows by noting that the total system can be decomposed into two independent subsystems, corresponding to the Q_H and Q_L loops, and thus, so can its performance. The first subsystem is a delayed system driven by $\boldsymbol{v}$ and controlled by $\boldsymbol{u}_L$, while the second subsystem is a warned system driven by $\boldsymbol{r}$ and controlled by $\boldsymbol{u}_H$. From our previous analysis, it is expected that the first system achieves better performance when its nerves are composed of a few large and fast axons, whereas the second system achieves better performance when its nerves are composed of many small and slow axons. These phenomena can indeed be observed in real visual systems [61]. Specifically, the optic nerve has approximately 1 million axons of mean diameter 0.64 μm, with a conductive velocity (CV) of 0.46 μm, while the 20,000 vestibular axons have a mean diameter of 2.88 μm, with a CV of 0.41 μm, significantly larger and less numerous and slightly less variable.

We conclude by emphasizing that a key enabler for DeSSs is diversity in the hardware used to implement diverse layers to address diverse system tasks. For example, in the biking example discussed in "Experimental Validation in a Biking Simulator," if the trail planning layer had

Experimental Validation in a Biking Simulator

Sensorimotor control was studied in the context of the multisensory task of mountain bike riding, using a video game as the experimental platform [58]. The game captures tunable requirements on player performance that require layered architectures in the nervous system to create DeSSs, due to the constraints imposed by physiology; see Figure 15. Naively, success in the biking task seems to require speed and accuracy that the raw hardware lacks, making nonlayered solutions infeasible. The layered nervous system breaks the overall biking problem into a high *trails* (trajectory planning) layer of slow but accurate vision with trail look ahead for advance warning and a low *bumps* (feedback control) layer that uses fast but inaccurate muscle spindles and proprioception to sense and reject bump disturbances. The motor commands from these two control loops to the muscles simply add in the optimal case as

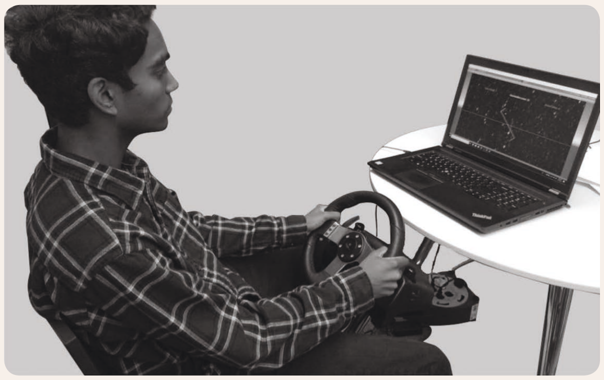


FIGURE S8 Bumps are added using motor torque in the wheel. Experiments can be done with bumps only, trails only, or both together; with varying trail speed and/or advance warning; and with additional quantization and/or time delay in the map from the wheel position to the players' actual position.

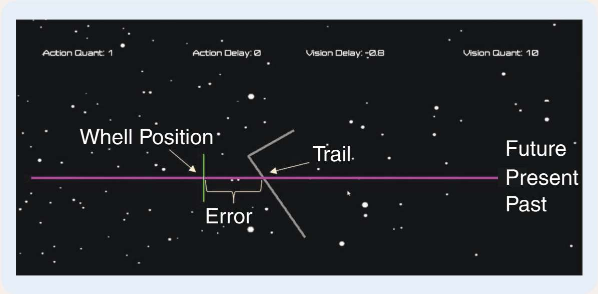


FIGURE S7 Players see a winding trail scrolling down the screen at a fixed speed, with a fixed advance warning (the visible trial ahead), both of which can be varied widely. The players aim to minimize the error between the desired trajectory and their actual position, using a gaming steering wheel.

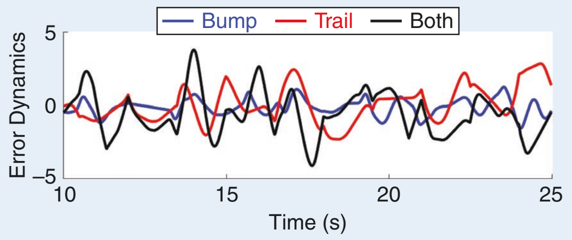


FIGURE S9 Errors in the case of bump only, trail only, and both.

(Continued)

Experimental Validation in a Biking Simulator *(Continued)*

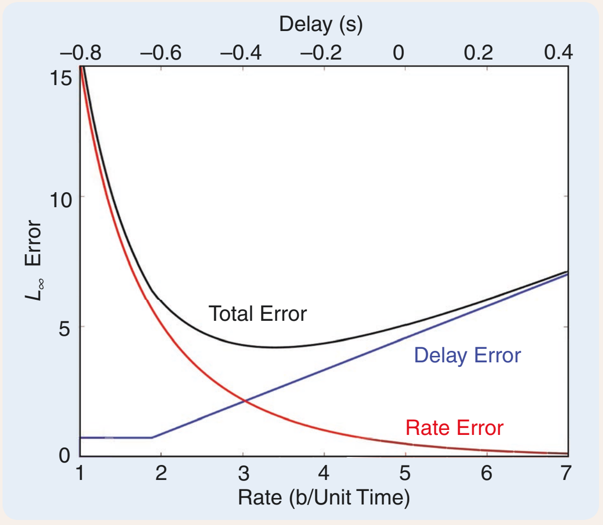


FIGURE S10 The error under added delay (blue), the error under added quantization (red), and the error under added delay plus quantization (black). In the last case, the added delay T and quantization rate R are subject to the component constraint $T = (R - 5)/20$. The dot shows the averaged error of four subjects, and the shadowed area indicates the standard error of the mean for these subjects.

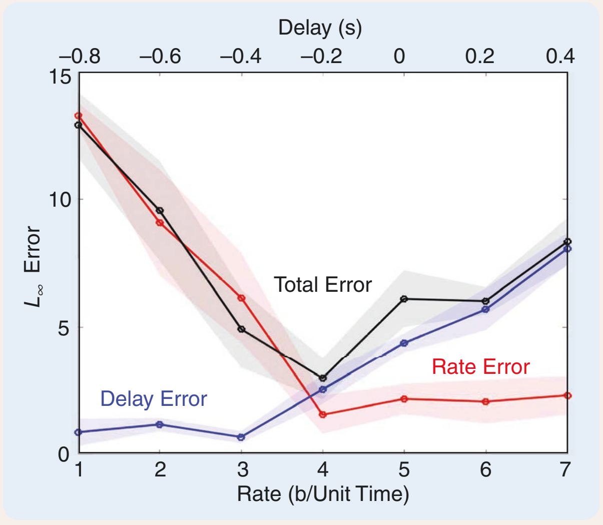


FIGURE S11 The delay error max(0, T) (blue), rate error $(2^R - 1)^{-1}$ (red), and total error max(0, T) + $(2^R - 1)^{-1}$ (black), with varying component signaling delay T_s and rate R subject to the component constraint $T = (R - 5)/20$.

well as in experiments [S13], [58], though muscles have their own constraints, as demonstrated by Fitts' law [62].

Nakahira et al. [58] developed experimental tasks and corresponding sensorimotor control models that mimicked three aspects of mountain biking: compensation by the spinal cord for the random shaking coming down the trail, the anticipation of turns in the trail by the visual system, and the stabilization of images on the retina by the oculomotor system to compensate bouncing. Two driving experiments were performed. The first was to test the interactions among layers, and the second was to test the errors caused by delays and rate limits in control within a layer. In the two experiments, subjects followed the trail on a computer screen and controlled a cursor with a wheel to stay on the trail. The goal of the subjects was to minimize errors between desired and actual trajectories shown on a computer monitor by moving the steering wheel (see Figures S7 and S8).

In the first experiment, the higher layer and the lower layer are coordinated, and the authors compared how the subjects' control behaviors and the resulting errors differed in three settings: 1) when there were random force disturbances to the steering wheel due to bumps on the ground (denoted as "bump only"), 2) when the trail trajectory was curved and changed direction (denoted as "trail only"), and 3) when both existed (denoted as "both"). Rejection of bump disturbance in the first and last settings is likely to be performed at the lower-layer reflex, while trajectory following in the second and last settings is likely to be performed at the higher-layer planning.

The experimental results are shown in Figure S9. The observed error in setting 3 (with both bumps and trail curvature) positively correlated with the sum of the errors from the first two settings, with either bumps or trail curvature (Pearson correlation coefficient = 0.57), suggesting that the two signals tended to have a consistent sign and amplitude. Moreover, the two signals showed no significant difference in the two-side t-test analysis. The results suggest that the two layers could be analyzed separately. This separability motivates the modeling of each layer separately and to further decompose the errors into those caused by neural signaling delays or rate limits in the control loop.

The impact of neurophysiological limits was studied in the second experiment. We observed changes in lateral control error in three settings: when external delays were added in the display, when external quantizers were added in the actuation effect of the steering wheel, and when both were added. These manipulations served as noninvasive probes for how component constraints affect system behavior. The lateral errors in the three settings are provided in Figure S11, and their corresponding theoretical prediction is available in Figure S10 (see the modeling details in the "A Case Study in Sensorimotor Control" section). The bridge between the constraints at the two levels highlights the benefits of the heterogeneity observed in nerves (Figure 11) and the advantages of layering in sensorimotor control (as in Figure 15).

REFERENCE

[S13] Q. Liu et al., "Experimental and educational platforms for studying architecture and tradeoffs in human sensorimotor control," in *Proc. IEEE Amer. Control Conf.,* 2019, pp. 483–488, doi: 10.23919/ACC.2019.8814470.

Developing a general quantitative design framework for multirate LCAs that enable DeSSs is arguably the most important open problem in engineering today.

to update the nominal trajectory faster than the vision could handle, the LCA would fail to enable a DeSS. It is this *multirate* nature of control tasks, characterized by local fast corrections and global slow updates, and which seems to be ubiquitous across engineered and natural complex systems, that allows for corresponding multirate LCAs to be designed that enable DeSSs. Developing a general quantitative design framework for multirate LCAs that enable DeSSs is arguably the most important open problem in engineering today and one that control theorists are particularly well suited to tackle.

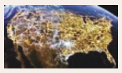
KEY CONCEPTS IN CONTROL ARCHITECTURE

In the first and second parts of the article, we introduced two concepts core to LCAs, namely, layers and DeSSs, and proposed quantitative frameworks for their analysis and design. In this final part, which should be viewed as a glossary of LCA terminology, we highlight that these are but a subset of the components that can be universally found in LCAs across domains. Although we do not have quantitative techniques for reasoning about them, we present qualitative descriptions and illustrate their importance using various case studies.

Table 1 reports concepts we believe are essential to the study of universal control architectures in the context of three familiar examples: clothing (see "Clothing as an LCA"), sensorimotor control, and the power grid. We also indulge in a more fanciful digression in "Lego as an LCA." These were introduced and developed in [58] and [63] and conceptually underpin much of the previous discussion.

- *Levels*: Conceptually, levels can be thought of as the (usually physical) substrates or components used to implement a system. All complex systems have many levels or scales; for example, in biology, levels range from molecules to synapses, cells, circuits, systems, and organisms. Analogous levels can be identified in familiar engineered systems. For example, in circuits, levels range from atoms to wires, resistors, capacitors, and transistors as well as integrated circuits and printed circuit boards. Deducing the levels experimentally is often necessary for understanding (reverse engineering) the design of existing control architectures found in nature and legacy engineered systems.
- *Layers*: Layers are complementary to levels and conceptually describe a functional decomposition of the overall behavior of a system. LCAs typically decompose across complexity and spatiotemporal scales, with more complex functionality implemented in higher global layers at a slower frequency and more rigid/structured functionality implemented in lower local layers at a higher frequency;

TABLE 1 Key concepts in control architectures are present across all engineered systems. Here, we illustrate these concepts using clothing, the human sensorimotor control system, and the power grid.

	Levels	**Layers**	**Laws**	**DeSSs**
	Physical	Functional	Pareto surface	Near-optimal Pareto point
Clothing	Garment Fabric Fibers Thread	Inner (soft, comfort) Middle (insulation) Outer (windproof)	Warm versus waterproof versus soft	Inner + middle + outer = waterproof, warm, soft
Sensorimotor	Nerve, muscle, axon, muscle fiber	Goals Planning Reflex	Speed versus accuracy	Vision + reflex = fast accurate motion
Power grid 	Grid Local distribution Transmission lines, substations	Economic dispatch Secondary frequency control Primary frequency control	Sustainable versus resilient versus efficient	Traditional + renewable + active control = sustainable, resilient, efficient power

Clothing as an LCA

Clothing is a familiar example that surprisingly highlights many universal concepts of control architecture [63]. The levels are familiar, ranging from thread and fibers to fabric, garments, and outfits, and we focus on the latter, with garment/outfit notation to denote levels. The layers for making clothing for harsh conditions are the outer/middle/inner garments. The outer layers provide waterproofing and wind proofing, the middle layers are insulating, and the inner layers are compatible (soft) for interfacing with skin. So, layers and levels are orthogonal decompositions of outfits, and both can have further decompositions. This architecture of clothing creates a DeSS so that outfits are weatherproof, warm, and soft when no individual garment or part provides all these features. Of note, skin and the rest of the body contain major evolved controls for thermoregulation so that clothing can be considered an extension on top of the skin of the complex feedback controls involved in the exquisitely tight control of the central temperature characteristic of healthy humans. Adding layers in this way is an important consequence of layered architectures.

Though clothing layering is usually purely passive, the outer layer provides a barrier function against wind and rain, the middle layer provides a barrier to heat loss, and the inner layer provides a soft barrier between the possibly rough outer layers and skin. It may seem strange to think of these as layers of passive control, but there is no other discipline that can integrate such passive mechanisms (which abound in engineering) into a full-stack theory of active/passive/lossless control layers.

A basic concept shown in the clothing example for understanding control architecture is "barriers." We naturally think of active controllers as creating barriers in the state space of controller/plant feedback interconnection, and the theories of Lyapunov and barrier functions and robust control extensions are explicitly aimed to make this rigorous, useful, and scalable [S14], [S15], [S16], [18]. What barriers in this sense allow for is showing that the set of possible controlled trajectories in state space robustly avoids "bad" regions. But if we want a more "full-stack" theory of architecture, where the higher levels and layers are typically active control, it will be necessary to include lower-layer control that is passive or even lossless. "Barriers" are already familiar in studying passive controllers but as post hoc analysis and less for design [S17]. We should probably think of active/passive/lossless as one example where there are both layers (for example, in a car that has active steering and braking, has passive nonslip tires, and is designed to be as lossless as can be in drag and friction) and levels (for example, in a car, active control is implemented in passive components plus power supplies, and physics tells us everything is microscopically lossless, which can be made rigorous using control theory [S18]).

Even the simplified proximal levels, layers, stages of dressing, and DeSSs described here are minimal essentials to creating functional outfits, and nothing simpler will work in a harsh environment. In particular, random piles of garments are vanishingly unlikely to make an outfit. Concretely, consider a small 30-garment wardrobe, with 10 each of garments for shell/warm/soft layers. Layering allows potentially 10^3 diverse but functional outfits, which is a much larger n^3 outfits versus $3n$ garments. But there are exponentially more (2^{30} =1e9) possible piles of garments, and the piles/outfits ratio of $2^n/n^3$ obviously grows exponentially with n garments in each layer.

One near universal in architectures is that they select functional but extremely thin and sparse subsets within the set of all possible "piles." These thin sparse subsets are even more extreme in the levels and layers below the garments level. Baking is another familiar example, with visible levels of ingredients and layers, such as cake, frosting, crust, filling, and so on. The levels and stages of baking are explicit in a recipe, but the supply chains that provide the ingredients are typically hidden behind convenient consumer interfaces. Random piles of ingredients and random stages of baking are extremely unlikely to produce anything even edible.

There are myriad tradeoffs and laws throughout the layers, levels, and stages that constrain what is possible, most obviously in the physical constraints on lower-level materials and the high-level users of the clothing architecture but also on all the stages of supply chains. But many constraints are evolved or designed as part of the architecture, such as the fabric/garment levels and outfit/garment layers, which were presumably not part of the earliest clothing using animal skins, even though all must obey physical laws. These added constraints in higher layers and levels are "constraints that deconstrain" ([64]) in that they are essential to creating the DeSS that is the very goal of architecture. The result is that a limited repertoire of fibers can create enormously diverse garments, which are functional due only to the constraints imposed by the universal architecture used by designers, manufacturers, and users. Baking has completely different details but is architecturally essentially the same.

This clothing architecture in harsh environments might be greatly simplified in others. Outfits in some tropical settings have one or even no layers, and garments can have a fabric made of plastics with low-level polymers but no threads or fibers. And so on. So, diversity among architectures is as universal as the diversity that any one architecture enables, and once the centrality of this diversity is recognized, both diversities motivate an integrated theory to design and upgrade all important architectures. But this is new and confusing even among experts, which we also hope to change.

(Continued)

Clothing as an LCA *(Continued)*

REFERENCES

[S14] S. Prajna and A. Jadbabaie, "Safety verification of hybrid systems using barrier certificates," in *Proc. Int. Workshop Hybrid Syst. Comput. Control*, Springer-Verlag, 2004, pp. 477–492, doi: 10.1007/978-3-540-24743-2_32.

[S15] M. Vidyasagar, *Nonlinear Systems Analysis*. Philadelphia, PA, USA: SIAM, 2002.

[S16] X. Xu, P. Tabuada, J. W. Grizzle, and A. D. Ames, "Robustness of control barrier functions for safety critical control," *IFAC-PapersOnLine*, vol. 48, no. 27, pp. 54–61, 2015, doi: 10.1016/j.ifacol.2015.11.152.

[S17] A. V. D. Schaft and D. Jeltsema, "Port-Hamiltonian systems theory: An introductory overview," *Found. Trends Syst. Control*, vol. 1, no. 2, pp. 173–378, 2014, doi: 10.1561/2600000002.

[S18] H. Sandberg, J.-C. Delvenne, and J. C. Doyle, "On lossless approximations, the fluctuation-dissipation theorem, and limitations of measurements," *IEEE Trans. Autom. Control*, vol. 56, no. 2, pp. 293–308, Feb. 2011, doi: 10.1109/TAC.2010.2056450.

Lego as an LCA

Lego is a simple and convenient toy system that illustrates many essentials of architecture, uses conventional digital control, and has transparent processes for the supply chain to (dis)assemble toys [S19]. Consider a familiar scenario where a child is repeatedly assembling, operating, and disassembling Lego robots to build a new one and, further, focus on the building of one robot from a box full of old partial robots and isolated basic parts. There are roughly 4,000 diverse standard Lego parts, which are produced by a manufacturing supply chain that is hidden (virtualized) from the child. There is an infinite variety of possible robots, which are nevertheless a vanishingly small subset of all nonfunctional Lego assemblies.

Focusing on building one robot, the minimal levels would be parts/robots consisting of the lower-level parts that then make up an assembled robot, though additional levels could include various functional subassemblies. The simplest stages would be disassembly/parts/assembly, which form a bow tie with a large but relatively thin knot of parts compared to the infinite variety of robots and assemblies as inputs and outputs. This depends on a universal snap protocol to make both disassembly and assembly easy. Building a Lego toy is a minimal example of the classic thin knot consisting of a set of parts plus the protocols specifying how the parts can be assembled. The most basic Lego has just one snap protocol and thousands of parts in its "knot." Most architectures have many more of both, totals that are still tiny compared to the variety of systems with a shared architecture.

This proximal part of the Lego supply chain would also have a control hourglass, where a child builder would take instructions and convert them into step-by-step assembly via the snap protocol. The thin middle waist layer would include the universal snap, here controlled repeatedly to control the overall assembly. The top layer would be the infinite possible instructions to assemble working robots, and the bottom layer would be the huge variety of supply chain steps that these instructions would control and the robots and subassemblies this produces.

A universal feature this illustrates but that can be a source of confusion is that the snap protocol is necessarily central to both the bow tie assembly knot and the control hourglass waist. In the bow tie knot, it is the physical mechanism that holds parts together and allows robots and their parts to be easily (dis)assembled. This bow tie alone would be useless, however, without an additional hourglass *control* of the snap process in each step of the (dis)assembly of a robot. The bow tie has essentially infinite diversity in the input of old robots or partial assemblies and the output of new robots. The hourglass also has infinite diversity in the top layer of instructions and the low layer of physical assembly steps, with a thin middle-layer waist that performs snap by snap (dis)assembly according to instructions.

The Lego snap is a sweet spot in the space of alternative connection and control protocols [S19]. One alternative is smooth bricks with no snap, which would be easier to assemble but would not be able to make robots. Another would be adding glue, which would make the robot more robust to trauma but make reuse difficult. The snap protocol is highly efficient, reusable, and robust but fragile to finely targeted attacks, such as removing imperceptibly thin and small bits of plastic just at the interface so that the snap would not hold. The process and the built robot, however, would be largely robust to similar removals away from the snaps, except in the computers controlling the robot. This extreme "robust yet fragile" feature is ubiquitous in real architectures [S20], with one aspect captured in Bode's integral formula.

The snap also makes it easy to manufacture new Lego parts that work with existing parts and architecture. The knot and waist utilizing the snap protocol form the "core" of the architecture "crux" for the control of assembly, which here is done by a child infinitely more complex than any Lego robot. This process could in principle be replaced by special-purpose assembly machines not greatly more complex than the robots they build, but attempts to build a

(Continued)

Lego as an LCA *(Continued)*

truly self-replicating universal Lego robot or machine have proved challenging.

In addition to the parts/robot levels, the functioning robot has, minimally, layers of computer/(sense and actuate)/plant, where the "plant" here would be the uncontrolled raw robot. This control system is distinct from the one doing assembly, and the computer would have sublayers of software/hardware, making this a toy version of a standard digital control system. Note that the software would typically be vastly more complex than the rest of the robot, and computer hardware would introduce vastly more levels, including microscopic components like transistors. A robot toy without sensors, actuators, and computers would be infinitely simpler, with only minimal functionality, but would still have some important architectural features. The complexity of the design process for a new robot would also be dominated by the control software, which would then be easily added in the assembly process. This assumes that the complex computer hardware is designed and manufactured separately and arrives as a completed brick component. The design and manufacture of the computer hardware would be vastly more complex than most robots using it as a component.

These minimal starting points illustrate the most essential universal architectural features beyond stages, levels, and layers, including DeSSs and virtualization, in both assembly and control. There is obviously large diversity in the parts and huge diversity in the possible toys, but the integrated functionality of a built robot with a digital controller illustrates how diverse parts enable this functionality but require the specific architectural layers, level, and stages to realize this functionality. An essential element of the DeSS is the use of virtualization in both the assembly and control of a Lego robot. The simplest is how the snap protocol is largely hidden in the assembled robot. It has not disappeared completely, as disassembly would reveal, but it is completely hidden in normal operation. This virtualizes both the parts and the assembly process so that the real-time control of the robot can ignore them. The control layers of software/hardware/(sense and act)/plant also have virtualization by every layer. The sense-and-act layer virtualizes the plant into an input-output system amenable to control, and the computer hardware virtualizes these details so that control can become a highly virtualized software design problem. These hardware layers severely constrain what control is possible but, if well designed, greatly facilitate both the design and implementation of sophisticated control. This creates a DeSS, where the resulting system has the flexibility and evolvability of software but speed and accuracy in the sensing and actuation hardware of the robot plant. Creating a DeSS is the most essential reason why an architecture and virtualization are used at all, when no individual component alone could make up the robot or its control.

The big advantage of Legos as a case study is that for simple assemblies, the process is transparent and doable by children, yet it illustrates many essential and universal features of architecture more generally. It also illustrates that when active control is added via specialized parts for sensing, computing, and actuation, the complexity explodes so that essentially all the design challenges are dominated by control and software, and then the remaining physical parts only enable that control. Together, all these architectural universals create a highly virtualized system with a DeSS far beyond what any level or layer could provide by itself. These kinds of efficiency, robustness, and evolvability tradeoffs addressed by virtualization dominate the design of most architectures in biology and technology and necessarily lead successful architectures to adopt some nearly universal features. Most are minimally present in toy Lego robots, and even more are present in the myriad cruxes in bacteria of replication, transcription, translation, metabolism, transport, and signal transduction, where all the cores have extremely conserved protocols for billions of years and even mostly conserved molecular machines.

With an explicit inclusion of control, the gap between the complexity of Lego and free-living bacteria is enormous, where the latter make not only all the parts and do self-replication but control allostasis and homeostasis in ways that typically do not arise in robots with external supplies of parts and energy. Nevertheless, they share striking universals, from levels, stages, and layers to cruxes of bow tie stages and hourglass controls; knot, waist, and core protocols; and virtualization and DeSSs. While there is no comparable universal terminology, we are proposing one here that is aimed to be maximally consistent with those specialized domains that do explicitly consider architecture. Bacteria and Legos surprisingly illustrate the most essential universals, but a large variety of other less familiar domains could as well. Particularly for experts in many domains of biological, medical, neurological, and technical systems, there are equally rich if less accessible examples of universal architectures.

Bacteria, however, are the original from which all else has evolved and remain arguably the most perfect. Their robustness and evolvability are due to the universal architectures that they share with all lineages descending from them, but their fragilities to hijacking are also devastatingly universal.

REFERENCES

[S19] M. E. Csete and J. C. Doyle, "Reverse engineering of biological complexity," *Science*, vol. 295, no. 5560, pp. 1664–1669, 2002, doi: 10.1126/science.1069981.
[S20] J. M. Carlson and J. C. Doyle, "Complexity and robustness," *Proc. Nat. Acad. Sci.*, vol. 99, no. suppl_1, pp. 2538–2546, 2002, doi: 10.1073/pnas.012582499.

There is a viable path toward a quantitative and universal theory of LCAs that the controls community is particularly well suited to pursue.

see, for example, Figure 2. Layers are the main architectural mechanism for taming complexity by breaking down a complex overall task into tractable subtasks (see the "LCAs via Optimal Control Decomposition" section) and that enable DeSSs [58], [62] by matching the spatiotemporal resolution of each layer with a corresponding control subtask (see the "Architecture Design as Multicriterion Optimization" section).

» *Laws*: Almost universally, we observe that hardware components have SATs, which impose a law on the low-level hardware that can then lead to high-level laws or constraints on optimal controllers. In neuroscience, vision is slower and more accurate than reflexes and proprioception. In immunology, adaptive immune responses take several days longer to mount than innate immune responses, but adaptive responses are more specific to the disease-causing pathogen. In computers, different storage components (for example, registers, cache, random-access memory, and disks) have extremely different speed, size, and cost. Typically, there are low-level hardware laws from physics that can directly impact higher levels as well as entirely new laws that arise at higher layers that have no parallel in physics and are associated with names like Turing, Shannon, and Bode. Developing an integrated theory of laws across layers and levels is essential to a theory of architecture.

» *DeSSs*: In engineering, complex system functionality requires diverse hardware, and most hardware is involved in diverse functions. If built out of homogeneous components, the SATs imposed by lower levels would make robust control impossible. However, these SATs allow for extreme diversity in the hardware, which can be leveraged with the right architectures to provide diverse functionality. Highly diverse hardware-level components (which are constrained by SATs) enable performance sweet spots that largely overcome the severe hardware-level SATs of individual components. In computers, such sweet spots include virtual memory management systems. In neuroscience, extreme diversity in axon sizes, receptors, and neurotransmitters is abundant [57], [58] but largely hidden. By itself, diversity of components only enables sweet spots of function; to achieve these functional sweet spots requires specific architectures to maximize the utility of diverse components, which we call DeSSs. We proposed a quantitative theory of DeSSs by viewing architecture design as multicriterion optimization and provide examples of these concepts at play.

Bow Ties, Hourglasses, Virtualization, and Abstraction

Fortunately, some features of LCAs are very familiar, particularly universal *bow ties* and *hourglasses* that appear in complex highly evolved systems at every scale and context. In both bow ties and hourglasses, two outer deconstrained stages and layers, with very diverse components that are evolvable and even swappable, are linked in the middle via a narrow highly constrained knot/waist with little diversity or evolvability. We call this *constraints that deconstrain* (as in [64]). The terminology of bow ties and hourglasses is not standard and can be confusing, but the distinction between them is useful and important. For a biologically motivated case study, see "Bow Ties and Hourglasses in Bacterial Metabolism." Both the bow tie and hourglass enable virtualization via universal shared interfaces, like operating systems (OSs), adenosine triphosphate (ATP), wall plugs, this text, ribosomes and translation, HTML, TCP/IP, high-definition media interfaces, membrane potentials, faucets, dashboards, and so on:

» *Bow tie*: Diversity is the aspect of architectures that is most familiar and easiest to discuss in the stages making up supply chains. Diverse proteins are produced by highly conserved translation "knot" protocols with amino acid inputs and controlled by a transcription hourglass. In metabolism, diverse carbon sources and molecules are linked via a thin "knot" of a few metabolic carriers and precursors. Diverse electric power sources and user appliances are linked in a bow tie via standard knot protocols (for example, 110 V and 60 Hz) in power grids. These examples all involve the flow of materials and energy through various stages and, with respect to diversity, have a bow tie shape, with diverse sources and products at the edges and highly conserved and less diverse "knots" in the middle. This enables independent and, thus, rapid evolution on both ends of the bow tie.

» *Hourglass*: An hourglass is used to describe the shape of layered communication and computing systems required to control bow ties. Diverse software runs on diverse hardware in an hourglass linked via less diverse "waist" OS in computers and their networks. Humans have diverse skills and memes and diverse tools, linked in an hourglass by shared languages and a poorly understood brain OS. Genes, apps, memes, words, technologies, and tools are highly modular and swappable, massively accelerating evolvability beyond what is possible with only the slow accumulation of small innovations.

» *Virtualization*: Hourglasses rely on virtualization to enable diversity both above and below the hourglass "waist." For example, OSs in computers act as *protocols that virtualize* the wildly diverse hardware and computer networks in modern computing systems, which in turn has led to the incredible progress and diversity of software and data. In decision and control systems, low-level unstable dynamics are virtualized by the feedback control layer, allowing the planning layer to use simple, reduced-order, and stable models for trajectory generation. Indeed, a commonly used model for trajectory generation in robotics across a wide variety of platforms (for example, quadrupeds, quadrotors, and mobile robots) is the Dubins' car, or unicycle, model—we expounded on this particular example of virtualization in robotics in previous sections [see Figure S1 in "Multirate LCAs in Practice" and the "Example 6: (Running Example: Robot Navigation)" section]. Here, the reference trajectory serves as the protocol between diverse planning and control layers, where each can be constructed using a diversity of algorithms, abstractions (see below), hardware, and software.

» *Abstraction*: Whereas the implementation of LCAs is enabled by bow ties, hourglasses, and virtualization, the *design* of layered architectures would be impossible without *abstractions*. For example, when writing computer software, engineers abstract OS/hardware

Bow Ties and Hourglasses in Bacterial Metabolism

An example that naturally embodies the structural features of layered architectures is the organization in the bacterial cell. A bacterial cell performs diverse types of complex functions on many timescales, from digesting nutrients and synthesizing macromolecules to adapting to environmental disturbances, cell cycling and decision making, and long-term evolution. This wide range of functions is fundamentally enabled by the layered architecture with bow ties and hourglasses (see Figures S12 and S13).

The lowest *layer*, or plant, consists of metabolites connected by reactions and summarized in a stoichiometry matrix. The overall organization has a *bow tie* with very diverse input and output stages and a thin low-diversity knot stage of precursors and carriers (ATP, nicotinamide adenine dinucleotide plus hydrogen, and so on) that are then cofactors throughout. Catabolic pathways convert input nutrients to knots, and then anabolic pathways make output products. We can also crudely view a bacterial cell as having two layers, with a low layer of metabolism and a high layer of gene expression, and then add structural features within a layer and between layers in the bacterial cell. Namely, each layer has a bow tie shape with a small knot (of carriers and precursors for metabolism) that connects diverse inputs and outputs on both sides. The high layer, viewed as a controller of the low layer, has an hourglass shape, with a thin universal waist (OS-like of transcription and translation) controlling and connecting diverse high-layer genes to diverse low-layer actuation by proteins. Bow ties and hourglasses are further universal features of complex architectures but are hidden and cryptic in normal operation, enabling efficiency and flexibility but potentially hiding fragilities.

A bacterial cell's metabolism layer obtains energy and materials from nutrients in the environment by using enzymes to catalyze reactions. The stoichiometry captures the structure but not rates of these reactions, and it has a bow tie shape; see Figure S12.

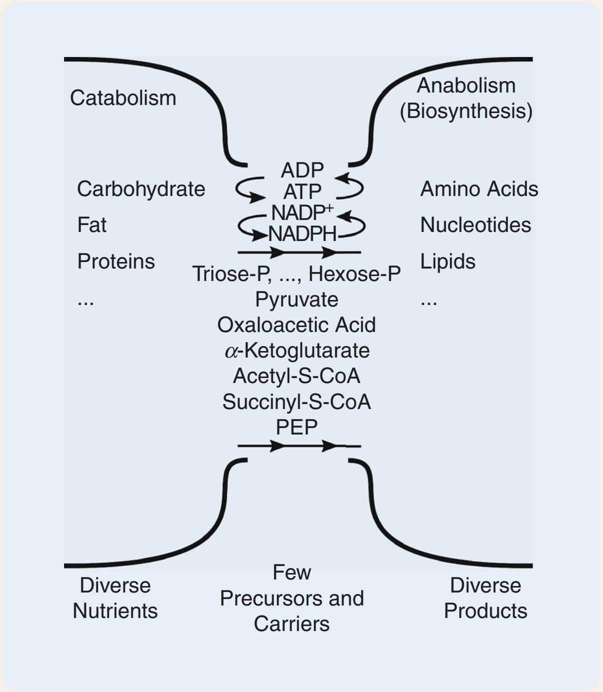


FIGURE S12 A bow tie in the low layer of metabolism stoichiometry in bacterial cells. ADP: adenosine diphosphate. CoA: coenzyme A; PEP: phosphoenolpyruvate.

(Continued)

Bow Ties and Hourglasses in Bacterial Metabolism *(Continued)*

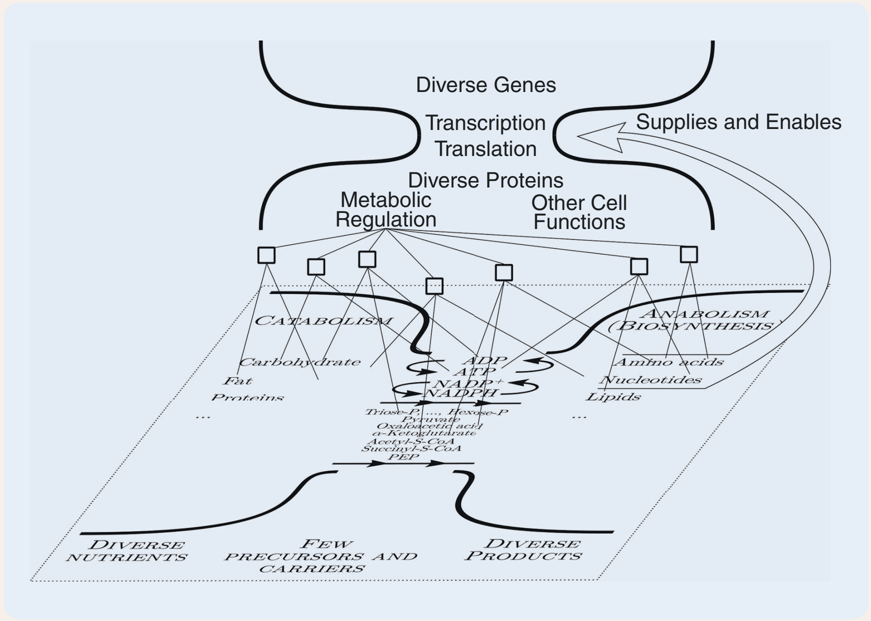


FIGURE S13 An hourglass in the gene expression layer to control the metabolism stoichiometry layer. Squares represent enzymes that locally regulate some metabolic reactions. The low layer is the stoichiometry of bacterial metabolism, with a bow tie shape. The metabolic reactions are locally regulated by enzymatic binding reactions, such as allostery (squares), which is in turn regulated by the high layer of gene expression. The gene expression layer, viewed as a controller for the low stoichiometry layer, has an hourglass shape, connecting diverse genes with diverse proteins via a thin waist of transcription and translation machinery. The gene expression layer regulates the low metabolism stoichiometry layer but also takes supply from and is enabled by the metabolism stoichiometry.

Diverse nutrient molecules are digested in the catabolism stage, and diverse macromolecules are synthesized in the anabolism stage, but the intermediate "knot" is a very thin stage of a few precursors and carriers. The ATP/adenosine diphosphate pair is the carrier for energy from catabolism to use for anabolism.

As metabolism happens on a fast timescale that is intrinsically unstable, regulation of these rapid reactions is needed locally because of delays in diffusion. Local enzymatic regulations through binding reactions serve as local actuators to be further controlled by higher layers. With these local enzymatic regulations stably maintaining a steady state of the cell's metabolism, this establishes a supply chain of molecules for energy, redox potential, and molecular building blocks used to perform tasks at a higher layer. This higher layer then can perform dynamics that take this supply chain as given and focus on goals with a virtualized molecular supply chain. For example, gene expression is one such layer. Here, building blocks, such as nucleic acids and amino acids, are used to build up large molecules, such as ribonucleic acids (RNAs) and proteins. The dynamics of gene expression can then focus on which RNAs and proteins are produced when and where, without worrying about the supply chain of building blocks or energy for synthesis. This "digital layer" is in contrast to those focusing on energy, redox potential, and molecular concentrations in the metabolism layer.

BOW TIES AND HOURGLASSES IN BACTERIAL METABOLISM

While the higher gene expression layer is fundamentally supplied and enabled by the lower metabolism layer, this higher layer regulates the lower metabolism layer on a slower timescale. While metabolic reactions tend to happen faster than seconds, gene expressions tend to happen in tens of minutes. The gene regulatory network can make more complex decisions and change the enzyme compositions precisely to actuate and coordinate at a global scale on the metabolism layer. For example, while rapid fluctuations in the ATP concentration need to be stabilized by local enzymatic feedback, a shift in the nutrient source requires the coordination on the gene expression level to stop expressing enzymes for old nutrients and start expressing enzymes to digest new nutrients.

To implement the gene expression layer's complex and diverse control of the metabolism layer, the cell organizes the gene expression controller in an hourglass shape; see Figure S13. Diverse signals in the form of combinatorial gene activation are mapped to diverse actions in expressed enzymes and other regulatory proteins via a thin waist that is the universal protocol of transcription–translation machinery. This hourglass structure is essential for the gene expression layer's control actions to scale up and facilitate diversity, namely, coping with diverse and complex disturbances and performing diverse and complex actions on and via enzymes.

as memory and compute, often ignoring, for example, device-level drivers and timing constraints. Note, however, that as software approaches the limits of what the underlying hardware can implement, these abstractions may no longer be valid, hence the need for, for example, real-time programming languages for embedded systems that directly access hardware resources. In decision and control

The impact in both the theory and application of nascent versions of these concepts has already been astounding both within and outside our community.

systems, abstractions abound. At the feedback control layer, the plant and controller are abstracted as mathematical operators operating on continuous- or discrete-time signals. At the trajectory planning layer, the potentially complex low-level closed-loop control system is abstracted using a simple dynamics model, such as a unicycle. This abstraction is valid thanks to the virtualization enabled by the feedback control layer below, but it also breaks down if the planned trajectories extend beyond the tracking capabilities of the closed-loop system, again showing that abstractions are useful only within operating ranges where virtualization can be reliably enforced. Because virtualization greatly enables the use of effective abstractions, these two distinct concepts are often confused.

CONCLUSIONS

We introduced a lexicon for key concepts in layered (control) architectures—levels, layers, stages, laws, DeSSs, hourglasses, bow ties, virtualization, and abstraction—and instantiated them in familiar and diverse examples, such as clothing, bacteria, GNC, robotics, and human sensorimotor control. These concepts are mostly familiar but are referred to using different terms across domains; thus, one primary goal of this article was to establish a common language to unify the study of architecture. Furthermore, for certain concepts, we also proposed quantitative frameworks for the analysis and synthesis of LCAs, grounded in robotics and sensorimotor applications.

We are very much aware that this article poses more questions than it answers and is likely to confuse (and perhaps even anger) applied and theoretical researchers alike. Nevertheless, we believe that underneath the cumbersome jargon and mathematical notation needed to convey our message, there is a viable path toward a quantitative and universal theory of LCAs that the controls community is particularly well suited to pursue. With that in mind, we hope that if the reader leaves this article with but one core message, it is that complex systems are composed of diverse levels and layers and that their analysis and design fall squarely within the skill set and expertise of the controls community. Indeed, the impact in both the theory and application of nascent versions of these concepts has already been astounding both within and outside our community, and we are excited about the potential future impact that this emerging field of study will have.

ACKNOWLEDGMENT

The work of Nikolai Matni was supported in part by National Science Foundation (NSF) Awards CPS-2038873, EECS-2231349, and SLES-2331880 and NSF CAREER Award ECCS-2045834. The work of Aaron D. Ames is supported in part by NSF Award CPS-1932091.

AUTHOR INFORMATION

Nikolai Matni (nmatni@seas.upenn.edu) is an assistant professor in the Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA 19104 USA, where he is also a member of the Department of Computer and Information Sciences (by courtesy); the General Robotics, Automation, Sensing, and Perception Lab; the Penn Research in Embedded Computing and Integrated Systems Engineering Center; and the Applied Mathematics and Computational Science graduate group. He has held positions as a visiting faculty researcher at Google Brain Robotics, New York, NY, USA; as a postdoctoral scholar in electrical engineering and computer science at the University of California, Berkeley, Berkeley, CA, USA; and as a postdoctoral scholar in the computing and mathematical sciences at the California Institute of Technology (Caltech), Pasadena, CA, USA. He received the Ph.D. degree in control and dynamical systems from Caltech in June 2016, and he received the B.A.Sc. and M.A.Sc. degrees in electrical engineering from the University of British Columbia, Vancouver, BC, Canada. His research interests broadly encompass the use of learning, optimization, and control in the design and analysis of autonomous systems. He is a recipient of the Air Force Office of Scientific Research Young Investigator Program Award (2024), National Science Foundation CAREER Award (2021), Google Research Scholar Award (2021), IEEE Control Systems Society George S. Axelby Award (2021), and 2013 IEEE Conference on Decision and Control (CDC) Best Student Paper Award. He was also a coauthor of papers that won the 2022 CDC Best Student Paper Award and the 2017 American Control Conference Best Student Paper Award. He is a Senior Member of IEEE.

Aaron D. Ames received the B.S. degree in mechanical engineering and the B.A. degree in mathematics from the University of St. Thomas, Saint Paul, MN, USA, in 2001, and the M.A. degree in mathematics and the Ph.D. degree in electrical engineering and computer sciences from the University of California, Berkeley, Berkeley, CA, USA, in 2006. From 2006 to 2008, he served as a postdoctoral scholar in control and dynamical systems with the California

Institute of Technology (Caltech). In 2008, he began his faculty career at Texas A&M University, College Station, TX, USA. He was an associate professor with the Woodruff School of Mechanical Engineering and the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA. Since 2017, he has been a Bren Professor of Mechanical and Civil Engineering and Control and Dynamical Systems, Caltech, Pasadena, CA 91125 USA. His research interests include the areas of robotic, nonlinear, safety-critical control, and hybrid systems, with a special focus on applications to dynamic robots, both formally and through experimental validation. He was a recipient of the 2005 Leon O. Chua Award for Achievement in Nonlinear Science and the 2006 Bernard Friedman Memorial Prize in Applied Mathematics from the University of California, Berkeley. He received the National Science Foundation CAREER award in 2010, the 2015 Donald P. Eckman Award, and the 2019 IEEE Control Systems Society Antonio Ruberti Young Researcher Prize. He is a Fellow of IEEE.

John C. Doyle received the B.S. and M.S. degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 1977 and the Ph.D. degree in mathematics from the University of California, Berkeley (UC Berkeley), Berkeley, CA, USA, in 1984. He is currently the Jean-Lou Chameau Professor of Control and Dynamical Systems, Electrical Engineering, and Bioengineering at the California Institute of Technology, Pasadena, CA 91125 USA. His research interests include mathematical foundations for complex networks, with applications in biology, technology, medicine, ecology, neuroscience, and multiscale physics that integrate theory from control, computation, communication, optimization, and statistics (for example, machine learning). He received the 1990 IEEE Baker Prize (for all IEEE publications) for work that was listed in the world top 10 most important papers in mathematics in 1981–1993, the IEEE Automatic Control Transactions Award (1988, 1989, and 2021), the 1994 American Control Conference O. Hugo Schuck Award, the 2004 Association for Computing Machinery Special Interest Group on Data Communication Paper Prize, and the 2016 Test of Time Award, and he was included in *Best Writing on Mathematics 2010*. His individual awards include the 1977 IEEE Power Hickernell Award, 1983 American Automatic Control Council Eckman Award, 1984 UC Berkeley Friedman Award, 1984 IEEE Centennial Outstanding Young Engineer Award (a one-time award for the IEEE 100th anniversary), 2004 IEEE Control Systems Field Award, and world records and championships in various sports. He is a Member of IEEE.

REFERENCES

[1] C. Draper et al., "Guidance and navigation," Massachusetts Inst. Technol., Cambridge, USA, 1965. [Online]. Available: https://www.ibiblio.org/apollo/hrst/archive/1713.pdf

[2] M. Bauer and J. C. Schlake, "Changes to the automation architecture: Impact of technology on control systems algorithms," in *Proc. 22nd IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, 2017, pp. 1–8, doi: 10.1109/ETFA.2017.8247697.

[3] T. Samad, P. McLaughlin, and J. Lu, "System architecture for process automation: Review and trends," *J. Process Control*, vol. 17, no. 3, pp. 191–201, 2007, doi: 10.1016/j.jprocont.2006.10.010.

[4] T. Samad and A. M. Annaswamy, "Controls for smart grids: Architectures and applications," *Proc. IEEE*, vol. 105, no. 11, pp. 2244–2261, Nov. 2017, doi: 10.1109/JPROC.2017.2707326.

[5] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," *Manuf. Lett.*, vol. 3, pp. 18–23, Jan. 2015, doi: 10.1016/j.mfglet.2014.12.001.

[6] M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle, "Layering as optimization decomposition: A mathematical theory of network architectures," *Proc. IEEE*, vol. 95, no. 1, pp. 255–312, Jan. 2007, doi: 10.1109/JPROC.2006.887322.

[7] D. P. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 8, pp. 1439–1451, Aug. 2006, doi: 10.1109/JSAC.2006.879350.

[8] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language*. San Mateo, CA, USA: Morgan Kaufmann, 2014.

[9] D. Pickem et al., "The Robotarium: A remotely accessible swarm robotics research testbed," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Piscataway, NJ, USA: IEEE, 2017, pp. 1699–1706, doi: 10.1109/ICRA.2017.7989200.

[10] M. J. Van Nieuwstadt and R. M. Murray, "Real-time trajectory generation for differentially flat systems," *Int. J. Robust Nonlinear Control: IFAC-Affiliated J.*, vol. 8, no. 11, pp. 995–1020, 1998, doi: 10.1002/(SICI)1099-1239(199809)8:11<995::AID-RNC373>3.0.CO;2-W.

[11] P. Kokotović, H. K. Khalil, and J. O'reilly, *Singular Perturbation Methods in Control: Analysis and Design*. Philadelphia, PA, USA: SIAM, 1999.

[12] Y. Zhang, D. Subbaram Naidu, C. Cai, and Y. Zou, "Singular perturbations and time scales in control theories and applications: An overview 2002–2012," *Int. J. Inf. Syst. Sci.*, vol. 9, no. 1, pp. 1–36, 2014.

[13] R. Alur and T. A. Henzinger, "Reactive modules," *Formal Methods Syst. Des.*, vol. 15, no. 1, pp. 7–48, 1999, doi: 10.1023/A:1008739929481.

[14] Y. Chen, J. Anderson, K. Kalsi, A. D. Ames, and S. H. Low, "Safety-critical control synthesis for network systems with control barrier functions and assume-guarantee contracts," *IEEE Trans. Control Netw. Syst.*, vol. 8, no. 1, pp. 487–499, Mar. 2021, doi: 10.1109/TCNS.2020.3029183.

[15] C. Zhao, U. Topcu, N. Li, and S. Low, "Design and stability of load-side primary frequency control in power systems," *IEEE Trans. Autom. Control*, vol. 59, no. 5, pp. 1177–1189, May 2014, doi: 10.1109/TAC.2014.2298140.

[16] D. Cai, E. Mallada, and A. Wierman, "Distributed optimization decomposition for joint economic dispatch and frequency regulation," *IEEE Trans. Power Syst.*, vol. 32, no. 6, pp. 4370–4385, Nov. 2017, doi: 10.1109/TPWRS.2017.2682235.

[17] A. D. Ames, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs with application to adaptive cruise control," in *Proc. 53rd IEEE Conf. Decis. Control*, Piscataway, NJ, USA: IEEE, 2014, pp. 6271–6278, doi: 10.1109/CDC.2014.7040372.

[18] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Trans. Autom. Control*, vol. 62, no. 8, pp. 3861–3876, Aug. 2017, doi: 10.1109/TAC.2016.2638961.

[19] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *Proc. 18th Eur. Control Conf.*, 2019, pp. 3420–3431, doi: 10.23919/ECC.2019.8796030.

[20] L. Lindemann and D. V. Dimarogonas, "Control barrier functions for signal temporal logic tasks," *IEEE Contr. Syst. Lett.*, vol. 3, no. 1, pp. 96–101, Jan. 2019, doi: 10.1109/LCSYS.2018.2853182.

[21] R. Dimitrova and R. Majumdar, "Deductive control synthesis for alternating-time logics," in *Proc. 14th Int. Conf. Embedded Softw.*, 2014, pp. 1–10, doi: 10.1145/2656045.2656054.

[22] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *Proc. 53rd IEEE Conf. Decis. Control*, Piscataway, NJ, USA: IEEE, 2014, pp. 81–87, doi: 10.1109/CDC.2014.7039363.

[23] S. Sadraddini and C. Belta, "Robust temporal logic model predictive control," in *Proc. 53rd Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Piscataway, NJ, USA: IEEE, 2015, pp. 772–779, doi: 10.1109/ALLERTON.2015.7447084.

[24] E. M. Wolff, U. Topcu, and R. M. Murray, "Optimization-based trajectory generation with linear temporal logic specifications," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Piscataway, NJ, USA: IEEE, 2014, pp. 5319–5325, doi: 10.1109/ICRA.2014.6907641.
[25] E. M. Wolff, U. Topcu, and R. M. Murray, "Robust control of uncertain Markov decision processes with temporal logic specifications," in *Proc. IEEE 51st IEEE Conf. Decis. Control (CDC)*, Piscataway, NJ, USA: IEEE, 2012, pp. 3372–3379, doi: 10.1109/CDC.2012.6426174.
[26] C. Fan, K. Miller, and S. Mitra, "Fast and guaranteed safe controller synthesis for nonlinear vehicle models," in *Proc. 32nd Int. Conf. Comput. Aided Verification (CAV)*, Los Angeles, CA, USA. Cham, Switzerland: Springer-Verlag, Jul. 21–24, 2020, pp. 629–652, doi: 10.1007/978-3-030-53288-8_31.
[27] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Trans. Robot.*, vol. 25, no. 6, pp. 1370–1381, Dec. 2009, doi: 10.1109/TRO.2009.2030225.
[28] P. Akella and A. D. Ames, "Disturbance bounds for signal temporal logic task satisfaction: A dynamics perspective," *IEEE Contr. Syst. Lett.*, vol. 6, pp. 2018–2023, 2022, doi: 10.1109/LCSYS.2021.3137267.
[29] M. Ahmadi, A. Singletary, J. W. Burdick, and A. D. Ames, "Barrier functions for multiagent-POMDPs with DTL specifications," in *Proc. 59th IEEE Conf. Decis. Control (CDC)*, Piscataway, NJ, USA: IEEE, 2020, pp. 1380–1385, doi: 10.1109/CDC42340.2020.9304266.
[30] M. Ahmadi, X. Xiong, and A. D. Ames, "Risk-averse control via CVaR barrier functions: Application to bipedal robot locomotion," *IEEE Contr. Syst. Lett.*, vol. 6, pp. 878–883, 2022, doi: 10.1109/LCSYS.2021.3086854.
[31] A. Singletary, M. Ahmadi, and A. D. Ames, "Safe control for nonlinear systems with stochastic uncertainty via risk control barrier functions," *IEEE Contr. Syst. Lett.*, vol. 7, pp. 349–354, 2023, doi: 10.1109/LCSYS.2022.3187458.
[32] A. Majumdar and M. Pavone, "How should a robot assess risk? towards an axiomatic theory of risk in robotics," in *Proc. 18th Int. Symp. Robot. Res. (ISRR)*, Cham, Switzerland: Springer-Verlag, 2020, pp. 75–84, doi: 10.1007/978-3-030-28619-4_10.
[33] S. Singh, Y. Chow, A. Majumdar, and M. Pavone, "A framework for time-consistent, risk-sensitive model predictive control: Theory and algorithms," *IEEE Trans. Autom. Control*, vol. 64, no. 7, pp. 2905–2912, Jul. 2019, doi: 10.1109/TAC.2018.2874704.
[34] A. Hakobyan, G. C. Kim, and I. Yang, "Risk-aware motion planning and control using CVaR-constrained optimization," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 3924–3931, Oct. 2019, doi: 10.1109/LRA.2019.2929980.
[35] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Aug. 1996, doi: 10.1109/70.508439.
[36] T. Marcucci, M. Petersen, D. v. Wrangel, and R. Tedrake, "Motion planning around obstacles with convex optimization," 2022, *arXiv:2205.04422*.
[37] W. Ubellacker and A. Ames, "Robust locomotion on legged robots through planning on motion primitive graphs," 2022, *arXiv:2209.07503*.
[38] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *Int. J. Robot. Res.*, vol. 36, no. 8, pp. 947–982, 2017, doi: 10.1177/0278364917712421.
[39] H. Yin, M. Bujarbaruah, M. Arcak, and A. Packard, "Optimization based planner–Tracker design for safety guarantees," in *Proc. Amer. Control Conf. (ACC)*, Piscataway, NJ, USA: IEEE, 2020, pp. 5194–5200, doi: 10.23919/ACC45564.2020.9148013.
[40] A. Singletary, T. Gurriet, P. Nilsson, and A. D. Ames, "Safety-critical rapid aerial exploration of unknown environments," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Piscataway, NJ, USA: IEEE, 2020, pp. 10,270–10,276, doi: 10.1109/ICRA40945.2020.9197416.
[41] N. Csomay-Shanklin, A. J. Taylor, U. Rosolia, and A. D. Ames, "Multi-rate planning and control of uncertain nonlinear systems: Model predictive control and control Lyapunov functions," in *Proc. IEEE 61st Conf. Decis. Control (CDC)*, Piscataway, NJ, USA: IEEE, 2022, pp. 3732–3739, doi: 10.1109/CDC51059.2022.9992902.
[42] S. Singh, A. Majumdar, J.-J. Slotine, and M. Pavone, "Robust online motion planning via contraction theory and convex optimization," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Piscataway, NJ, USA: IEEE, 2017, pp. 5883–5890, doi: 10.1109/ICRA.2017.7989693.
[43] S. Singh, H. Tsukamoto, B. T. Lopez, S.-J. Chung, and J.-J. Slotine, "Safe motion planning with tubes and contraction metrics," in *Proc. 60th IEEE Conf. Decis. Control (CDC)*, Piscataway, NJ, USA: IEEE, 2021, pp. 2943–2948, doi: 10.1109/CDC45484.2021.9682865.
[44] D. Sun, S. Jha, and C. Fan, "Learning certified control using contraction metric," in *Proc. Conf. Robot Learn.*, PMLR, 2021, pp. 1519–1539.
[45] M. H. Cohen and C. Belta, "Approximate optimal control for safety-critical systems with control barrier functions," in *Proc. 59th IEEE Conf. Decis. Control (CDC)*, Piscataway, NJ, USA: IEEE, 2020, pp. 2062–2067, doi: 10.1109/CDC42340.2020.9303896.
[46] S. L. Herbert, M. Chen, S. Han, S. Bansal, J. F. Fisac, and C. J. Tomlin, "Fastrack: A modular framework for fast and guaranteed safe motion planning," in *Proc. IEEE 56th Annu. Conf. Decis. Control (CDC)*, Piscataway, NJ, USA: IEEE, 2017, pp. 1517–1522, doi: 10.1109/CDC.2017.8263867.
[47] N. Matni and J. C. Doyle, "A theory of dynamics, control and optimization in layered architectures," in *Proc. Amer. Control Conf. (ACC)*, Piscataway, NJ, USA: IEEE, 2016, pp. 2886–2893, doi: 10.1109/ACC.2016.7525357.
[48] A. Srikanthan, V. Kumar, and N. Matni, "Augmented lagrangian methods as layered control architectures," 2023, *arXiv:2311.06404*.
[49] A. Srikanthan, F. Yang, I. Spasojevic, D. Thakur, V. Kumar, and N. Matni, "A data-driven approach to synthesizing dynamics-aware trajectories for underactuated robotic systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Piscataway, NJ, USA: IEEE, 2023, pp. 8215–8222, doi: 10.1109/IROS55552.2023.10341651.
[50] H. Zhang, A. Srikanthan, S. Folk, V. Kumar, and N. Matni, "Why change your controller when you can change your planner: Drag-aware trajectory generation for quadrotor systems," 2024, *arXiv:2401.04960*.
[51] U. Rosolia, A. Singletary, and A. D. Ames, "Unified multirate control: From low-level actuation to high-level planning," *IEEE Trans. Autom. Control*, vol. 67, no. 12, pp. 6627–6640, Dec. 2022, doi: 10.1109/TAC.2022.3184664.
[52] U. Rosolia and A. D. Ames, "Multi-rate control design leveraging control barrier functions and model predictive control policies," *IEEE Contr. Syst. Lett.*, vol. 5, no. 3, pp. 1007–1012, Jul. 2021, doi: 10.1109/LCSYS.2020.3008326.
[53] K. Garg, R. K. Cosner, U. Rosolia, A. D. Ames, and D. Panagou, "Multi-rate control design under input constraints via fixed-time barrier functions," *IEEE Contr. Syst. Lett.*, vol. 6, pp. 608–613, 2022, doi: 10.1109/LCSYS.2021.3084322.
[54] T. G. Molnar, R. K. Cosner, A. W. Singletary, W. Ubellacker, and A. D. Ames, "Model-free safety-critical control for robotic systems," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 944–951, Apr. 2022, doi: 10.1109/LRA.2021.3135569.
[55] A. Singletary, W. Guffey, T. G. Molnar, R. Sinnet, and A. D. Ames, "Safety-critical manipulation for collision-free food preparation," *IEEE Robot. Autom. Lett.*, vol. 7, no. 4, pp. 10,954–10,961, Oct. 2022, doi: 10.1109/LRA.2022.3192634.
[56] S. P. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[57] Y. Nakahira, N. Matni, and J. C. Doyle, "Hard limits on robust control over delayed and quantized communication channels with applications to sensorimotor control," in *Proc. 54th IEEE Conf. Decis. Control (CDC)*, Piscataway, NJ, USA: IEEE, 2015, pp. 7522–7529, doi: 10.1109/CDC.2015.7403407.
[58] Y. Nakahira, Q. Liu, T. J. Sejnowski, and J. C. Doyle, "Diversity-enabled sweet spots in layered architectures and speed–accuracy trade-offs in sensorimotor control," *Proc. Nat. Acad. Sci. USA*, vol. 118, no. 22, pp. 1–11, 2021, doi: 10.1073/pnas.1916367118.
[59] M. A. Dahleh and I. J. Diaz-Bobillo, *Control of Uncertain Systems: A Linear Programming Approach*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1994.
[60] G. N. Nair, F. Fagnani, S. Zampieri, and R. J. Evans, "Feedback control under data rate constraints: An overview," *Proc. IEEE*, vol. 95, no. 1, pp. 108–137, Jan. 2007, doi: 10.1109/JPROC.2006.887294.
[61] P. Sterling and S. Laughlin, *Principles of Neural Design*. Cambridge, MA, USA: MIT Press, 2015.
[62] Y. Nakahira, Q. Liu, T. J. Sejnowski, and J. C. Doyle, "Fitts' Law for speed-accuracy trade-off describes a diversity-enabled sweet spot in sensorimotor control," 2019. [Online]. Available: https://arxiv.org/abs/1906.00905
[63] J. C. Doyle and M. Csete, "Architecture, constraints, and behavior," *Proc. Nat. Acad. Sci.*, vol. 108, no. supplement_3, pp. 15,624–15,630, 2011, doi: 10.1073/pnas.1103557108.
[64] J. Gerhart and M. Kirschner, "The theory of facilitated variation," *Proc. Nat. Acad. Sci.*, vol. 104, no. suppl_1, pp. 8582–8589, 2007, doi: 10.1073/pnas.0701035104.