Frontiers in Scalable Distributed Control: SLS, MPC, and Beyond

Jing Shuang (Lisa) Li, Carmen Amo Alonso, and John C. Doyle

***Abstract*— The System Level Synthesis (SLS) approach facilitates distributed control of large cyberphysical networks in an easy-to-understand, computationally scalable way. We present an overview of the SLS approach and its associated extensions in nonlinear control, MPC, adaptive control, and learning for control. To illustrate the effectiveness of SLS-based methods, we present a case study motivated by the power grid, with communication constraints, actuator saturation, disturbances, and changing setpoints. This simple but challenging case study necessitates the use of model predictive control (MPC); however, standard MPC techniques often scales poorly to large systems and incurs heavy computational burden. To address this challenge, we combine two SLS-based controllers to form a layered MPC-like controller. Our controller has constant computational complexity with respect to the system size, gives a 20-fold reduction in online computation requirements, and still achieves performance that is within 3% of the *centralized* MPC controller.**

I. INTRODUCTION

The control of large cyberphysical systems is important to today's society. Relevant examples include power grids, traffic networks, and process plants. In each of these fields, emerging technology (e.g. renewables, autonomous vehicles) and increasingly complex architectures present new challenges, and theoretical frameworks and algorithms are needed to address them. Generally speaking, these widespread large-scale systems require a distributed control framework that offers scalable, structured solutions.

The recently introduced System Level Synthesis (SLS) framework [1] provides theory and algorithms to deal with large, complex, structured systems. SLS tackles common challenges in distributed control, including disturbance containment and communication constraints. Moreover, it enables distributed synthesis *and* implementation; the runtime of the synthesis algorithm is independent of the network size, and each subsystem can synthesize its own controller, bypassing the need for centrally coordinated controller synthesis and making this framework highly scalable.

Since its inception, many extensions of the SLS framework have been developed, including works on nonlinear plants [2], [3], model predictive control (MPC) [4], [5], adaptive control [6], [7], and learning [8]–[10]; the core SLS ideas are useful and applicable to a variety of settings. In this paper, we hope to familiarize more researchers and practitioners with this powerful and scalable framework.

One notable extension of the SLS framework is scalable MPC. MPC is ubiquitous in industry, but challenging for use in large networks due to scalability issues and high computational demand. Various distributed and hierarchical methods have been proposed to address this [11]; in this paper, we propose a novel SLS and MPC-based layered controller that is high-performing and uniquely scalable. We demonstrate our controller on an example system motivated by a power grid, which is subject to communication constraints, actuator saturation, disturbances, and setpoint changes that result from intermittently shifting optimal power flows (OPFs).

This paper introduces the basic mathematics of SLS (Section II), then presents a comprehensive overview of all SLS-based methods to date (Section III). These sections form a useful introductory reference for the system-level approach to controls. We follow up with a SLS-based case study (Section IV) in which we introduce a scalable distributed two-layered controller that successfully approximates centralized MPC performance while greatly reducing online computation.

II. THE SLS PARAMETRIZATION

We introduce the basic mathematics of SLS, adapted from §2 of [1]. For simplicity, we focus on the finite-horizon state feedback case; analogous results for infinite-horizon and output feedback can be found in §4 and §5 of [1]. We also briefly present the SLS-based MPC formulation.

A. Setup

Consider the discrete-time linear time varying (LTV) plant

$$x(t+1) = A_t x(t) + B_t u(t) + w(t), \tag{1}$$

where $x(t) \in \mathbb{R}^n$ is the state, $w(t) \in \mathbb{R}^n$ is an exogenous disturbance, and $u(t) \in \mathbb{R}^p$ is the control input. The control input is generated by a causal LTV state-feedback controller

$$u(t) = K_t(x(0), x(1), ..., x(t)) \tag{2}$$

where K_t is some linear map. Let Z be the block-downshift operator[1]. By defining the block matrices $\hat{A} := \text{blkdiag}(A_1, A_2, ..., A_T, 0)$ and $\hat{B} := \text{blkdiag}(B_1, B_2, ..., B_T, 0)$, the dynamics of system (1) over the time horizon $t = 0, 1, ..., T$ can be written as

$$\mathbf{x} = Z\hat{A}\mathbf{x} + Z\hat{B}\mathbf{u} + \mathbf{w} \tag{3}$$

where $\mathbf{x}$, $\mathbf{u}$, and $\mathbf{w}$ are the finite horizon signals corresponding to state, disturbance, and control input respectively. The controller (2) can be written as

$$\mathbf{u} = \mathbf{K}\mathbf{x} \tag{4}$$

Authors are with the Department of Computing and Mathematical Sciences, California Institute of Technology. {jsli, camoalon, doyle}@caltech.edu,

[1]Sparse matrix composed of identity matrices along its first block subdiagonal.

where $\mathbf{K}$ is the block-lower-triangular (BLT) matrix corresponding to the causal linear map K_t.

B. System Responses

Consider the closed-loop system responses $\{\mathbf{\Phi}_x, \mathbf{\Phi}_u\}$, which map disturbance to state and control input, i.e.

$$\mathbf{x} := \mathbf{\Phi}_x \mathbf{w} \tag{5a}$$

$$\mathbf{u} := \mathbf{\Phi}_u \mathbf{w} \tag{5b}$$

By combining (3) and (4), we easily see that

$$\mathbf{\Phi}_x = (I - Z(\hat{A} + \hat{B}\mathbf{K}))^{-1} \tag{6a}$$

$$\mathbf{\Phi}_u = \mathbf{K}(I - Z(\hat{A} + \hat{B}\mathbf{K}))^{-1} = \mathbf{K}\mathbf{\Phi}_x \tag{6b}$$

Definition 1. $\{\mathbf{\Phi}_x, \mathbf{\Phi}_u\}$ are *achievable* system responses if $\exists$ a block-lower-triangular matrix $\mathbf{K}$ (i.e. causal linear controller) such that $\mathbf{\Phi}_x$, $\mathbf{\Phi}_u$, and $\mathbf{K}$ satisfy (6). If such a $\mathbf{K}$ exists, we say that it *achieves* system responses $\{\mathbf{\Phi}_x, \mathbf{\Phi}_u\}$.

The SLS framework works with system responses $\{\mathbf{\Phi}_x, \mathbf{\Phi}_u\}$ directly. We use convex optimization to search over the set of achievable $\{\mathbf{\Phi}_x, \mathbf{\Phi}_u\}$, then implement the corresponding $\mathbf{K}$ using $\{\mathbf{\Phi}_x, \mathbf{\Phi}_u\}$. We can do this because the set of achievable system responses is fully parametrized by an affine subspace, as per the core SLS theorem:

Theorem 1. For plant (1) using the state-feedback policy $\mathbf{u} = \mathbf{K}\mathbf{x}$, where $\mathbf{K}$ is a BLT matrix, the following are true

1) The affine subspace of BLT $\{\mathbf{\Phi}_x, \mathbf{\Phi}_u\}$

$$\begin{bmatrix} I - Z\hat{A} & -Z\hat{B} \end{bmatrix} \begin{bmatrix} \mathbf{\Phi}_x \\ \mathbf{\Phi}_u \end{bmatrix} = I \tag{7}$$

   parametrizes all achievable system responses (6).
2) For any BLT matrices $\{\mathbf{\Phi}_x, \mathbf{\Phi}_u\}$ satisfying (7)), the controller $\mathbf{K} = \mathbf{\Phi}_u \mathbf{\Phi}_x^{-1}$ achieves the desired response (6) from $\mathbf{w} \mapsto (\mathbf{x}, \mathbf{u})$.

Proof. See Theorem 2.1 of [1]. □

Theorem 1 allows us to reformulate an optimal control problem over state and input pairs $(\mathbf{x}, \mathbf{u})$ as an equivalent problem over system responses $\{\mathbf{\Phi}_x, \mathbf{\Phi}_u\}$. As long as the system responses satisfy (7) (which can be interpreted as a generalization of controllability), part 2 of Theorem 1 guarantees that we will also have a controller $\mathbf{K}$ to achieve these system responses. Thus, a general optimal control problem can be formulated in the SLS framework as

$$\begin{aligned} \min_{\mathbf{\Phi}_x, \mathbf{\Phi}_u} \quad & f(\mathbf{\Phi}_x, \mathbf{\Phi}_u) \\ \text{s.t.} \quad & (7),\ \mathbf{\Phi}_x \in \mathcal{X}, \mathbf{\Phi}_u \in \mathcal{U}, \end{aligned} \tag{8}$$

where f is any convex objective function and $\mathcal{X}$ and $\mathcal{U}$ are convex sets. Details of how to choose f for several standard control problems is provided in §2 of [1]. Common specifications for distributed control, such as disturbance containment, communication delay, local communication, and actuation delay, can be enforced by sparsity patterns on $\mathcal{X}$ and $\mathcal{U}$. Suitable specifications are discussed at length in [1].

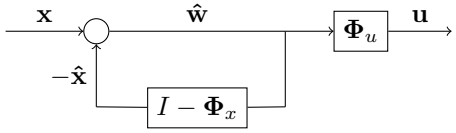

Fig. 1. Controller implementation

For many classes of f (e.g. $\mathcal{H}_2$ objective), (8) decomposes into smaller subproblems that can be solved in parallel. This allows for *distributed synthesis*; each subsystem in the system solves a local subproblem independently of other subsystems. Under localized communication constraints, the size of these subproblems are independent of total system size N, making the SLS formulation uniquely scalable. The size of these subproblems depend on d, the size of the local region; d-localized communication constrains each subsystem to only communicate with subsystems at most d-hops away. For large distributed systems, generally $d \ll N$, so scaling with d instead of N is highly beneficial.

C. Controller Implementation

Once we solve (8) and obtain the optimal system responses $\{\mathbf{\Phi}_x, \mathbf{\Phi}_u\}$, we can implement a controller $\mathbf{K}$ as per part 2 of Theorem 1. Instead of directly inverting $\mathbf{\Phi}_x$, we use the feedback structure shown in Fig. 1, which is described by

$$\mathbf{u} = \mathbf{\Phi}_u \hat{\mathbf{w}}, \quad \hat{\mathbf{x}} = (\mathbf{\Phi}_x - I)\hat{\mathbf{w}}, \quad \hat{\mathbf{w}} = \mathbf{x} - \hat{\mathbf{x}} \tag{9}$$

where $\hat{\mathbf{x}}$ can be interpreted as a nominal state trajectory, and $\hat{\mathbf{w}} = Z\mathbf{w}$ is a delayed reconstruction of the disturbance. This implementation is particularly useful because structural constraints (e.g. sparsity) imposed on the system responses $\{\mathbf{\Phi}_x, \mathbf{\Phi}_u\}$ translate directly to structure in the controller implementation. Thus, constraints on information sharing between controllers can be enforced as constraints on $\mathcal{X}$ and $\mathcal{U}$ in (8). Additionally, (9) allows for *distributed implementation*. As mentioned above, each subsystem solves a local subproblem to obtain the local sub-matrices of $\{\mathbf{\Phi}_x, \mathbf{\Phi}_u\}$; these sub-matrices are then used to locally implement a sub-controller, independently of other subsystems.

D. Model Predictive Control

SLS-based MPC consists of solving the following problem at each timestep t:

$$\begin{aligned} \min_{\mathbf{\Phi}_x, \mathbf{\Phi}_u} \quad & f(\mathbf{\Phi}_x, \mathbf{\Phi}_u, x(t)) \\ \text{s.t.} \quad & (7),\ \mathbf{\Phi}_x \in \mathcal{X}, \mathbf{\Phi}_u \in \mathcal{U} \\ & \mathbf{\Phi}_x x(t) \in \mathcal{P}_x \\ & \mathbf{\Phi}_u x(t) \in \mathcal{P}_u \end{aligned} \tag{10}$$

As with the core parametrization, f is any convex objective function (e.g. $\mathcal{H}_2$), and $\mathcal{X}$ and $\mathcal{U}$ are convex sets that may specify communication-motivated sparsity constraints. $\mathcal{P}_x$ and $\mathcal{P}_u$ are convex sets that specify state and input constraints on predicted states and inputs, such as upper and lower bounds; these translate to affine constraints on $\{\mathbf{\Phi}_x, \mathbf{\Phi}_u\}$.

Equation (10) can be broken down into smaller subproblems to be solved in parallel using the alternating direction

method of multipliers (ADMM) [12]. At each timestep, each subsystem accesses its own local subset of x_t, solves (10), and outputs a local control signal u_t, which is calculated by multiplying the first block-matrix of $\mathbf{\Phi}_u$ with x_t. Predicted values of x and u can be calculated by multiplying x_t by the appropriate block-matrices of $\mathbf{\Phi}_x$ and $\mathbf{\Phi}_u$, respectively. This formulation enjoys the same scalability benefits as the core SLS method: runtime is independent of total system size. For a more thorough analysis of runtime, subproblem partitioning, etc., see [4].

III. SLS-BASED TECHNIQUES & EXTENSIONS

We provide an overview of SLS-based works. The SLS parametrization provides a transparent approach to analyzing and synthesizing closed-loop controllers, and all methods described in this section exploit this fact. Some works focus on theoretical analyses, while others capitalize on the scalability provided by the SLS parametrization.

A. Standard and Robust SLS

The SLS parametrization was first introduced for state feedback in [13]. For a comprehensive review of standard SLS, we refer the interested reader to [1] and reference therein; this work describes state and output feedback and relevant robust formulations. Though theory for the infinite-horizon case is well-developed, implementation is generally limited to finite-horizon approximations; this is partially remedied in [14], which introduces an infinite-horizon implementation for the distributed state feedback $\mathcal{H}_2$ problem using SLS.

The main results on robust SLS are presented in [1]. Newer results on the robustness of the general SLS parametrization are found in [15]. Informally, robust SLS deals with cases in which the synthesized matrices $\{\mathbf{\Phi}_x, \mathbf{\Phi}_u\}$ do not describe the actual closed-loop system behavior, either by design or due to uncertainty. In these cases, robust SLS methods provide guarantees on the behavior of the closed-loop system. These guarantees are particularly useful for adaptation and learning, which are described later in this section.

In a setting with minimal uncertainty, robust SLS can be used when overly strict controller-motivated constraints on $\{\mathbf{\Phi}_x, \mathbf{\Phi}_u\}$ result in infeasibility during synthesis; this was the original goal of the robust SLS derivation. An alternative 'two-step' approach is presented in [16], whose results allow for separation of controller and closed-loop system constraints in the state-feedback setting.

B. SLS for Nonlinearities & Saturations

SLS for nonlinear systems with time-varying dynamics is presented in [2]. This work generalizes the notion of system responses to the nonlinear setting, in which they become nonlinear operators. No constraints are considered; instead, this work focuses on the relationship between achievable closed-loop maps and realizing state feedback controllers.

Nonlinear SLS can be applied to saturating linear systems [3] to provide distributed anti-windup controllers that accommodate state and input saturation constraints. An alternative approach to handling saturation is [17]. This work uses robust optimization techniques instead of nonlinear analysis. In the $\mathcal{L}_1$ case with non-coupled constraints, the nonlinear method performs better; however, the linear method handles more general cases including coupling, using a distributed primal-dual optimization approach.

C. Distributed Model Predictive Control

An SLS-based distributed and localized MPC algorithm is developed in [4], and briefly described in the previous section. This the first closed-loop MPC scheme that allows for distributed sensing *and* computation. Computation of this algorithm can be significantly accelerated via explicit solutions; so far, such solutions are available for the case of quadratic cost and saturation constraints [5]. Furthermore, this MPC algorithm readily extends to the robust setting – this is the subject of current work. In addition to forming the basis for this MPC algorithm, the SLS parametrization can also be used to perform robustness analysis and guarantees on the general MPC problem [18].

D. Adaptive Control & Machine Learning

A scalable adaptive SLS-based algorithm is presented in [6]. This work describes a framework for scalable robust adaptive controllers; at each timestep, measurements are collected to reduce uncertainty and boost performance while maintaining stability. This framework is applied in [19], which introduces SLS controllers that are robust to package dropouts. A different scalable adaptive SLS-based algorithm deals with networks that switch between topological configurations according to a finite-state Markov chain [7].

SLS is especially beneficial for machine learning because it directly relates model uncertainty with stability and performance suboptimality [8]. The SLS parametrization is used to analyze the linear quadratic regulator (LQR) problem in the case where dynamics are unknown [20], [21]. These works provide safe and robust learning algorithms with guarantees of sub-linear regret, and study the interplay between regret minimization and parameter estimation. Additionally, SLS forms the basis of a framework for constrained LQR with unknown dynamics, where system identification is performed through persistent excitation while guaranteeing the satisfaction of state and input constraints [9]. SLS is also used to provide complexity analysis and sharp end-to-end guarantees for the stability and performance of unknown sparse linear time invariant systems [22].

SLS-based analyses have been applied to the output feedback setting as well [10]. Motivated by vision-based control of autonomous vehicles, this work designs a safe and robust controller to solve the problem of controlling a known linear system for which partial state information is extracted from complex and nonlinear data. SLS also underpins the sample complexity analysis for Kalman filtering of an unknown system [23]. Additionally, SLS forms the basis for many ongoing works on data-driven control techniques.

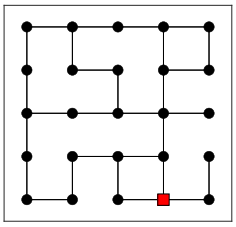

Fig. 2. Topology of our example system for the case study. We will plot the time trajectories of states, disturbances, and input for the red square node.

E. Additional works

SLS is extended to the spatially invariant setting in [24], [25]. In related work, [26] explores realizable structured controllers via SLS and discusses the limitations of SLS in the relative feedback setting. Additionally, several works focus on optimizing computation for SLS. Explicit solutions to the general SLS problem are described in [27], and [28] uses dynamic programming to solve for SLS synthesis problems 10 times faster than using a conventional solver. [29] describes deployment architecture for SLS, and is used in the construction of the SLSpy software framework [30]. In addition to this Python implementation, a MATLAB-based toolbox is also publicly available [31]. These toolboxes include implementations for standard and robust SLS, two-step SLS, nonlinear-saturation SLS, and distributed MPC.

IV. CASE STUDY: POWER GRID

We demonstrate the efficacy of a novel SLS-based controller on a power grid-like system with three key features:

1) Periodic setpoint changes, induced by changing optimal power flows (OPFs) in response to changing loads
2) Frequent, small[2] disturbances
3) Actuator saturation

This system is inspired by common challenges of power systems control, though it is by no means a high-fidelity representation of the grid. The purpose of this case study is to show good performance in the presence of these common challenges in a simple and accessible system.

A. System Setup

We start with a randomly generated connected graph over a 5x5 mesh network, shown in Fig. 2. Graph edges represent connections between buses. Interactions between neighboring buses are governed through the undamped linearized swing equations (11), similar to the example used in [1].

$$x_i(t+1) = A_{ii}x_i(t) + \sum_{j\in\mathcal{N}(i)} A_{ij}x_j(t) + \begin{bmatrix}0\\1\end{bmatrix}(w_i(t)+u_i(t)) \tag{11a}$$

$$A_{ii} = \begin{bmatrix}1 & \Delta t\\ -\frac{b_i}{m_i}\Delta t & 1\end{bmatrix}, \quad A_{ij} = \begin{bmatrix}0 & 0\\ \frac{b_{ij}}{m_i}\Delta t & 0\end{bmatrix} \tag{11b}$$

The state of bus i includes $x_i^{(1)}$, the phase angle relative to some setpoint, and $x_i^{(2)}$, the associated frequency, i.e.

[2]Relative to the size of the setpoint changes

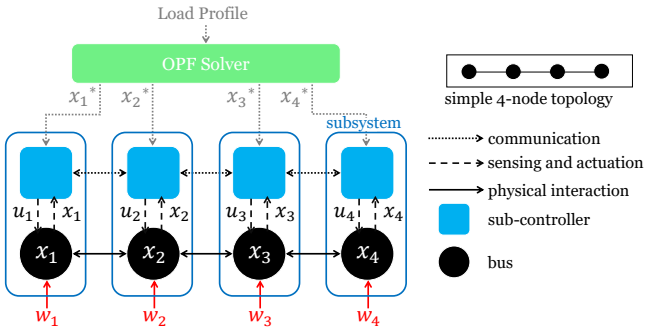

Fig. 3. Architecture of example system. For ease of visualization, we depict a simple 4-node topology instead of the 25-node mesh we'll be using. Grey dotted lines indicate periodic communications; the OPF solver sends x^* only on the timesteps when it receives a new load profile.

$x_i = [x_i^{(1)} \quad x_i^{(2)}]^\top$. m_i, w_i, and u_i are the inertia, external disturbance, and control action of the controllable load of bus i. b_{ij} represent the line susceptance between buses i and j; $b_i = \sum_{j\in\mathcal{N}(i)} b_{ij}$. b_{ij} and m_i^{-1} are randomly generated and uniformly distributed between [0.5, 1], and [0, 10], respectively. Large values of m_i^{-1} render the system unstable; the spectral radius of the A matrix is 1.5.

We periodically generate a new random load profile and solve a centralized DC-OPF problem to obtain the optimal setpoint x^*. We then send each sub-controller its individual optimal setpoint. Each subsystem reaches this setpoint in a distributed manner; sub-controllers are only allowed to communicate with their immediate neighbors and the neighbors of those neighbors (i.e. local region of size $d = 2$). In addition to tracking setpoint changes, controllers must also reject random disturbances. The setup is shown in Fig. 3.

We additionally enforce actuator saturation constraints of the form $|u_i(t)| \leq u_{max}$. Actuator saturation is a ubiquitous constraint in control systems.

B. Two-Layer Controller

The combination of large setpoint changes, small disturbances, and relatively tight saturation bounds is challenging for an offline controller. Optimal performance requires the use of distributed SLS-based MPC. However, MPC incurs significant online computational cost since an optimal control problem must be solved at every timestep. We propose a two-layer controller to reduce this computation.

We decompose the main problem into two subproblems – reacting to setpoint changes and rejecting disturbances – and assign each subproblem to a layer. The top layer uses MPC to plan trajectories in response to large setpoint changes, while the offline bottom layer tracks this trajectory and rejects disturbances. Our controller offers unique scalability benefits compared to standard layered MPC controllers [11], and offers an efficiency boost over the SLS-MPC controller.

The top layer of the controller uses SLS-based MPC to generate saturation-compliant trajectories. As described in section II-D, this layer can be implemented in a distributed manner, with runtime independent of total system size. Every T_{MPC} timesteps, the top layer solves (10) and generates a safe trajectory of states x and inputs u for the next T_{MPC}

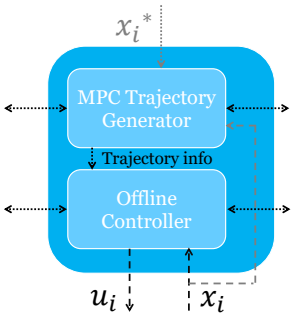

Fig. 4. Layered sub-controller for the i^{th} subsystem. Grey dotted line indicates periodic setpoint changes sent by the OPF solver. Grey dashed line indicates periodic sensing of local state x_i. Horizontal black dotted lines indicate communication with neighboring sub-controllers.

timesteps – this gives a T_{MPC}-fold reduction in computational cost compared to a single-layer MPC controller. To maximize performance, we time the top layer's trajectory generation to coincide with the periodic setpoint changes.

The trajectory generator alone is insufficient; since it only runs once every T_{MPC} timesteps, disturbances can persist or amplify between runs, severely compromising performance. To deal with disturbances and preserve performance, we use an offline SLS controller in the bottom layer. This controller receives trajectory information from the top layer and outputs a control signal u that tracks the desired trajectory.

The two-layer controller is shown in Fig. 4. This layered controller is fully distributed; each subsystem synthesizes and implements both layers of its own controller, largely independently of other subsystems. Synthesis of the two layers is independent of one another, although some synthesis parameters may be shared. Additionally, all cross-layer communication is local, i.e. a top layer controller will never communicate with a bottom layer controller from a different subsystem. Distributed implementation and synthesis allows our controller to enjoy a runtime that is independent of total system size, like its component SLS-based controllers. Thus, this layered MPC controller is uniquely scalable, and well suited for use in systems of arbitrary size.

We emphasize the contrast between our controller and the ideal controller – centralized MPC. First, centralized MPC requires instantaneous communication between all subsystems and some centralized unit, while our controller is distributed and subsystems may only communicate within a local set of neighboring subsystems. Second, centralized MPC requires online computation once every timestep, while our controller requires online computation only once every T_{MPC} timesteps. Third, centralized MPC's computational complexity scales quadratically with system size, which is impractical for large systems; our controller's complexity scales independently of system size. Despite this, our controller performs *nearly identically* to this ideal centralized MPC. We summarize the above comparisons in Table I.

C. Simulation Results

We compare the performance of the layered controller with $T_{MPC} = 20$ ('LocLayered') with the performance of the centralized MPC controller ('CenMPC'). For reference, we include the optimal linear controller with ('SatCenLin') and without ('UnsatCenLin') actuator saturation. The centralized controllers are free of communication constraints while the layered controller is limited to local communication.

TABLE I
CENTRALIZED MPC VS. LOCAL LAYERED MPC

	Centralized MPC	Local Layered MPC
Communication	Global	Local
Online computation	Every timestep	Every T_{MPC} timesteps
Complexity w.r.t. system size N	$O(N^2)$	$O(1)$
Performance	Optimal	Within 1-3% of optimal

TABLE II
LQR COSTS CORRESPONDING TO FIG. 5

Controller	Actuator Saturation	Global Communication	LQR Cost
UnsatCenLin	No	Yes	1.00
SatCenLin	Yes	Yes	3.97e13
CenMPC	Yes	Yes	1.90
LocLayered	Yes	No	1.93

The resulting LQR costs, normalized by the non-saturating optimal centralized cost, are shown in Table II. We plot trajectories from the red square node from Fig. 2 in Fig. 5, focusing on a window of time during which only one setpoint change occurs. The non-saturating controller is omitted.

For this setpoint change, saturation-induced windup effects result in oscillations of increasing size from the saturated linear controller, causing it to lose stability and incur astronomical costs. Windup effects are mitigated by both online controllers, which perform similarly despite drastically different computational requirements. As desired, the proposed layered controller achieves near-optimal performance.

To check general behavior, we re-run the simulation 30 times with different randomly generated grids, plant parameters, disturbances, and load profiles. The resulting LQR costs, normalized by the non-saturating optimal cost in each run, are shown in Table III. We show the mean cost over all 30 trials and the mean cost over the 26 trials in which the saturated linear controller maintained stability. Observations from the initial example hold; the layered controller consistently achieves near-optimal performance. The two online methods again demonstrate enormous improvement over the saturated linear controller, which often loses stability. Performance differences predominately arise from reactions to setpoint changes; when the saturated linear controller manages to maintain stability after a setpoint change, performance is similar across all methods. Lastly, we note that both the online methods' costs are within 17% of the centralized optimal cost *without* actuator saturation.

V. CONCLUSIONS

We reviewed SLS and derivative works, and demonstrated an effective combination of SLS-based methods in a novel layered controller. This controller performed similarly to centralized MPC and is superior in scalability, communication requirements, and computational cost.

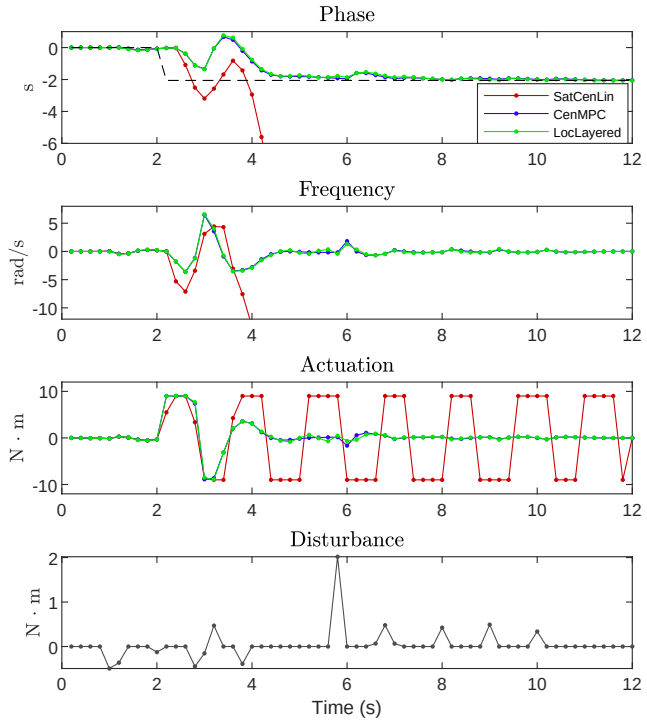

Fig. 5. Performance of two centralized strategies ('SatCenLin', 'CenMPC') and our layered controller ('LocLayered'). The linear saturating controller loses stability and gives extremely large oscillations in phase and frequency, which are omitted from the plot after around $t = 4$; the associated actuation engages in oscillations as well, which are shown on the plot.

TABLE III

LQR COSTS AVERAGED OVER 30 TRIALS

Controller	LQR cost	
	Total	SatCenLin stable
UnsatCenLin	1.00	1.00
SatCenLin	1.32e7	1.13
CenMPC	1.16	1.08
LocLayered	1.17	1.09

SLS-based methods are effective standalone controllers. However, as the systems we seek to control become increasingly complex and uncertain, standalone controllers are insufficient; techniques must be combined (i.e. via layering) to create a controller that is at once safe, scalable, efficient, and high-performing. SLS-based methods are excellent candidates for use in such layered controllers, as they readily interface with one another and with learning-based methods.

REFERENCES

[1] J. Anderson, J. C. Doyle, S. H. Low, and N. Matni, "System level synthesis," *Annual Reviews in Control*, vol. 47, pp. 364–393, 2019.

[2] D. Ho, "A System Level Approach to Discrete-Time Nonlinear Systems," in *Proc. IEEE ACC*, 2020, pp. 1625–1630.

[3] J. Yu and D. Ho, "Achieving Performance and Safety in Large Scale Systems with Saturation using a Nonlinear System Level Synthesis Approach," in *Proc. IEEE ACC*, 2020, pp. 968–973.

[4] C. Amo Alonso and N. Matni, "Distributed and Localized Model Predictive Control via System Level Synthesis," in *Proc. IEEE CDC*, 2020, pp. 5598–5605.

[5] C. Amo Alonso, N. Matni, and J. Anderson, "Explicit Distributed and Localized Model Predictive Control via System Level Synthesis," in *Proc. IEEE CDC*, 2020, pp. 5606–5613.

[6] D. Ho and J. C. Doyle, "Scalable Robust Adaptive Control from the System Level Perspective," in *Proc. IEEE ACC*, 2019, pp. 3683–3688.

[7] S. Han, "Localized Learning of Robust Controllers for Networked Systems with Dynamic Topology," in *Proceedings of Machine Learning Research*, 2020, pp. 687–696.

[8] N. Matni, A. Proutiere, A. Rantzer, and S. Tu, "From self-tuning regulators to reinforcement learning and back again," in *Proc. IEEE CDC*, 2019, pp. 3724–3740.

[9] S. Dean, S. Tu, N. Matni, and B. Recht, "Safely learning to control the constrained linear quadratic regulator," in *Proc. IEEE ACC*, 2019, pp. 5582–5588.

[10] S. Dean, N. Matni, B. Recht, and V. Ye, "Robust Guarantees for Perception-Based Control," in *Proceedings of Machine Learning Research*, vol. 120, 2020, pp. 350–360.

[11] R. Scattolini, "Architectures for distributed and hierarchical Model Predictive Control - A review," *Journal of Process Control*, vol. 19, no. 5, pp. 723–731, 2009.

[12] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122.

[13] Y.-S. Wang, N. Matni, S. You, and J. C. Doyle, "Localized Distributed State Feedback Control," in *Proc. IEEE ACC*. American Automatic Control Council, 2014, pp. 5478–5755.

[14] J. Yu, Y.-S. Wang, and J. Anderson, "Localized and Distributed H2 State Feedback Control," 2020. [Online]. Available: http://arxiv.org/abs/2010.02440

[15] N. Matni and A. Sarma, "Robust Performance Guarantees for System Level Synthesis," in *Proc. IEEE ACC*, 2020, pp. 779–786.

[16] J. S. Li and D. Ho, "Separating Controller Design from Closed-Loop Design: A New Perspective on System-Level Controller Synthesis," in *Proc. IEEE ACC*, 2020, pp. 3529–3534.

[17] Y. Chen and J. Anderson, "System Level Synthesis with State and Input Constraints," in *Proc. IEEE CDC*, no. 2, 2019, pp. 5258–5263.

[18] S. Chen, H. Wang, M. Morari, V. M. Preciado, and N. Matni, "Robust Closed-loop Model Predictive Control via System Level Synthesis," in *Proc. IEEE CDC*, 2020, pp. 2152–2159.

[19] C. Amo Alonso, D. Ho, and J. Maestre, "Distributed Linear Quadratic Regulator Robust to Communication Dropouts," 2020. [Online]. Available: https://arxiv.org/abs/2103.03967

[20] S. Dean, H. Mania, N. Matni, B. Recht, and S. Tu, "Regret bounds for robust adaptive control of the linear quadratic regulator," in *NeurIPS*, 2018, pp. 4188–4197.

[21] ——, "On the Sample Complexity of the Linear Quadratic Regulator," *Foundations of Computational Mathematics*, pp. 1–43, 2019.

[22] S. Fattahi, N. Matni, and S. Sojoudi, "Efficient Learning of Distribtued Linear-Quadratic Control Policies," *SIAM Journal on Control and Optimization*, vol. 58, no. 5, pp. 2927–2951, 2020.

[23] A. Tsiamis, N. Matni, and G. J. Pappas, "Sample Complexity of Kalman Filtering for Unknown Systems," in *Learning for Decision and Control*, 2020.

[24] E. Jensen and B. Bamieh, "Optimal Spatially-Invariant Controllers with Locality Constraints: A System Level Approach," in *Proc. IEEE ACC*, 2018, pp. 2053–2058.

[25] ——, "A Backstepping Approach to System Level Synthesis for Spatially-Invariant Systems," in *Proc. IEEE ACC*, 2020, pp. 5295–5300.

[26] ——, "On Structured-Closed-Loop versus Structured-Controller Design: the Case of Relative Measurement Feedback," 2020. [Online]. Available: http://arxiv.org/abs/2008.11291

[27] J. Anderson and N. Matni, "Structured state space realizations for SLS distributed controllers," in *IEEE 55th Annual Allerton Conference on Communication, Control, and Computing*, no. 2, 2017, pp. 982–987.

[28] S.-H. Tseng, C. Amo Alonso, and S. Han, "System Level Synthesis via Dynamic Programming," in *Proc. IEEE CDC*, 2020, pp. 1718–1725.

[29] S.-H. Tseng and J. Anderson, "Deployment Architectures for Cyber-Physical Control Systems," in *Proc. IEEE ACC*, 2020, pp. 5287–5294.

[30] S.-H. Tseng and J. S. Li, "SLSpy: Python-Based System-Level Controller Synthesis Framework," 2020. [Online]. Available: http://arxiv.org/abs/2004.12565

[31] J. S. Li, "SLS-MATLAB: Matlab toolbox for system level synthesis," 2019. [Online]. Available: https://github.com/sls-caltech/sls-code