Effectively Utilizing Elastic Resources in Networked Control Systems

Michael Balszun*, Debayan Roy*, Licong Zhang*, Wanli Chang†, Samarjit Chakraborty*
*TU Munich, Germany, †Singapore Institute of Technology, Singapore
*Email:{michael.balszun, debayan.roy, licong.zhang, samarjit}@tum.de, †Email:wanli.chang@singaporetech.edu.sg

***Abstract*—The rapid growth in the size and complexity of modern Cyber-Physical Systems (CPS) imposes increasing demand for the embedded resources, especially the communication resources. As a result, resource-efficient CPS design has become an important issue. Towards the design of networked embedded control systems, a major branch of CPS, reliable and deterministic communication is able to achieve satisfactory control performance. However, the amount of this type of resource that can be provided by the embedded platform is often limited. On the other hand, it is difficult to guarantee the control performance with non-deterministic communication resources, due to their unpredictable behavior. In this paper, we propose a novel control scheme to efficiently utilize elastic communication resources. In general, the non-deterministic communication resources are flexibly deployed on top of the deterministic communication resources to achieve stability and good control performance. In the rare worst-case, when non-deterministic communication is completely unavailable, the deterministic communication resources are used to guarantee stability and the control performance satisfying the design requirement. The experimental results show that the performance of the control application is ensured to satisfy the design requirement in the worst case and that better control performance is achieved when non-deterministic resources are available.**

I. Introduction

Cyber-physical systems (CPS) are widely found in the automotive, industry automation and avionics domains. Embedded control systems form a major branch of CPS. With rapid development in wired and wireless communication systems, coupled with the prominence of modern sensors and actuators with integrated communication interfaces, networked control systems (NCS) have become very popular. In the NCS, a control loop is implemented over distributed components, such as multiple sensors, actuators, and processors, that are connected over one or more communication networks. Very often, multiple applications share the communication resources.

Many control applications are safety-critical and even when not, a malfunction could lead to significant costs in terms of damaged equipment, loss of production or degraded customer satisfaction [1]. Therefore, it is important to ensure that the control performance satisfies certain requirements. Deterministic communication, such as e.g. provided by time triggered protocols on wired networks, guarantees the timing of the closed-loop control messages transmission, and thus the control performance with an appropriately designed controller. However, wastage frequently occurs, making such deterministic communication resources expensive, especially for the cost-sensitive automotive industry. For example, a fixed time slot is periodically and exclusively assigned to an application, which may not need the entire duration for information transmission, if at all. There is also non-deterministic communication, such as e.g. over wireless networks where packet dropouts can occur at any moment or event-driven, best-effort scheduling schemes. Those are often more cost efficient and flexible but make it difficult to guarantee a certain control performance.

978-1-5386-1898-1/17/$31.00

In this paper, we consider a control system exposed to both deterministic and non-deterministic communication resources, as shown in Figure 1. The sensor is connected to the controller and the actuator via two kinds of communication channels. Examples are the static (deterministic) and dynamic (non-deterministic) segments of FlexRay, or wired (deterministic) and wireless (non-deterministic) networks. The sensor sends the states of the physical process to the controller every sampling period. The deterministic communication enables a reliable, but longer sampling period h_s, which is the interval between two neighboring deterministic samples. We design a controller for h_s (i.e., the safe controller) achieving stability and the control performance satisfying the requirement. When the non-deterministic communication resources are available, an additional shorter sampling period h_f, which is the interval between two neighboring non-deterministic samples, is possible. We make use of h_f (i.e., applying the fast controller), whenever it will improve the control performance and still ensure the stability. In the case that deploying non-deterministic communication resources (i.e., using h_f) might degrade the control performance or jeopardize the stability, we will fall back to the deterministic communication resources and stick to h_s.

In the literature on control systems with mixed communication, almost all the works that consider non-deterministic communication resources assume a certain degree of determinism. For example, the maximum rate of packet drops is assumed in [2], [3], [4]. The worst-case delay and jitter are assumed in [5]. Event-triggered control is discussed in [6], [7], [8], [9], where the communication resource is assumed to be available whenever required. In [10], [11], [12], the non-determinism comes from contention among various applications sharing the communication resources. That is, the communication channel itself is always assumed to be reliable. Besides, most previous works only allow a fixed usage pattern of the non-deterministic communication resources.

For instance, in [10], [11], [12], the system can only switch once from the non-deterministic event-triggered communication to the deterministic time-triggered communication.

The main contributions of this paper are as follows. First, we are able to use the non-deterministic communication more flexibly (unlike a fixed usage pattern) and the decision boils down to every sampling period. This potentially optimizes the utilization of the non-deterministic communication resources, which can be further investigated in the future. Second and more importantly, we are able to guarantee the control performance and stability for completely non-deterministic communication resources. That is, the system is able to handle the case when only the deterministic communication is available. The proposed approach can be generalized to address broader non-determinism in the control systems, beyond the communication perspective.

The rest of the paper is organized as follows. In Section II, we summarize the related works. The background on feedback control systems is provided in Section III. We formally describe the system model under consideration, the proposed control scheme and the corresponding implementation technique in Section IV. The advantages of our proposed scheme are demonstrated with experimental results from a case study in Section V. Section VI makes the concluding remarks and points out future research directions.

II. Related Work

In control theory, the main emphasis has been on designing controllers taking the network and computation uncertainties into account and analyzing the stability and worst-case performance of control loops in the presence of non-determinism. However, most of the design approaches, ensuring safety and worst-case performance, impose constraints on network determinism such as maximum rate of packet drops [2], [3], [4], worst-case delay and jitter [5]. Otherwise, if the communication timings are stochastic, then only probabilistic or approximate analysis is possible [13], [14], which is not acceptable for safety-critical systems. Here, it may be noted that better guarantees on closed-loop system properties can be provided by imposing stricter constraints on determinism of implementation platform.

On the other hand, embedded systems engineers have mainly focused on implementing the controllers on embedded platforms in such a way that certain timing guarantees can be provided. Consequently, a conventional approach is to implement the controller in a time-triggered fashion, where all the timing characteristics like closed-loop delay and sampling period can be calculated [15], [16]. Correspondingly, safe and performance-optimal controller can be designed. However, this often leads to resource over-provisioning and wastage. A more recent approach is event-triggered or self-triggered control, where the controller is triggered only when required [6], [7], [8], [9]. However, such an implementation loosely assumes that the resource is available when required, requiring a thorough schedulability analysis.

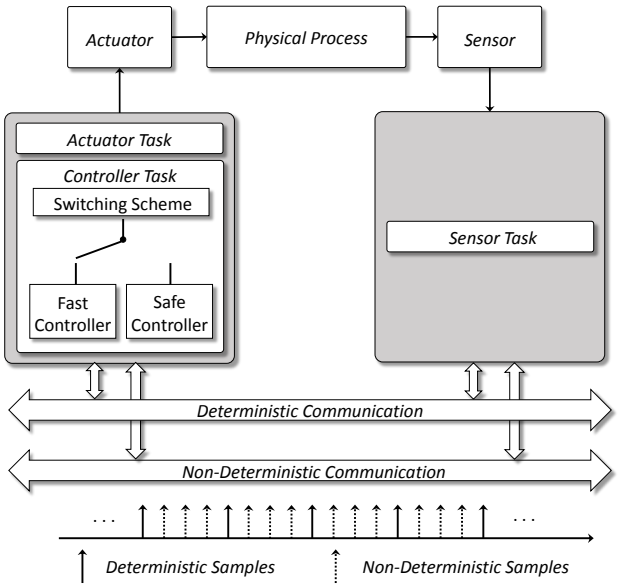


Figure 1: The proposed control scheme using elastic resources.

In the control literature, there is a well-established theory to determine the stability of switched control systems. In order for the overall system to be stable, it is insufficient to prove the closed-loop stability of each individual system. Tools to ensure the switching stability include the common quadratic Lyapunov functions (CQLF) and switched quadratic Lyapunov functions (SQLF) [17], [18]. However, this cannot be applied in a straightforward way for the problem under consideration, which involves one control system with non-determinism.

Furthermore, in the context of mode switching in controller implementations, a control scheme is proposed in [19], where the controller switches from a self-triggered mode to a time-triggered mode based on the current state of the closed-loop system. In the self-triggered mode, the next time instant when the control input must be computed and transmitted, namely, the trigger time, is decided in the current instance while in the time-triggered mode, the controller is triggered periodically. The trigger times for a certain discretized region of the state space are pre-computed and stored in the cache memory, so that they can be fetched based on the current state in the self-triggered mode. This is done to save the online computation time. When the current state does not fall into the pre-considered region, the trigger time computation task is dispatched with a low priority. If the task is not executed, the schedule switches to the time-triggered mode. The idea is to use self-triggered control as much as possible to improve the resource efficiency. However, the communication network to implement the controller is not considered.

Another group of works [10], [11], [12] consider exploiting hybrid communication protocol of FlexRay in the controller implementation. It is proposed in [10] that switching from the dynamic (flexible TDMA) to the static (TDMA) segment of FlexRay in the event of disturbance can lead to high resource efficiency while maintaining the performance of the time-

triggered control. These works consider that multiple control applications can share few TDMA slots. When a disturbance occurs, the application is allocated with a TDMA slot, if it has not been occupied by other applications. However, slot sharing among multiple control applications requires the underlying platform to be runtime reconfigurable, which is not the case for most time-triggered architectures like FlexRay, and thus presents an important implementation challenge to be addressed.

In [11] the minimum number of slots required to guarantee the worst-case settling time of each control application in a given set is computed with an assumption on the arrival rate of disturbance for the whole set. However, this analysis is restrictive in the sense that not all the control applications sharing one slot may be interdependent and several applications may experience disturbances at the same time. The authors lift the restriction in [12] on the disturbance arrival and correspondingly considered that a controller may have to wait to get a TDMA slot if it is not free. However, in order to guarantee the control performance, these works assume that the worst-case delay of a control application can be calculated when mapped onto the dynamic segment of FlexRay. Our approach requires no worst-case timing analysis, which is useless for the completely non-deterministic communication, since in the worst case, the non-deterministic communication resources are not available at all. Moreover, probabilistic guarantees can be provided over such contention-based protocols [20], [21]. However, they are not helpful in safety-critical systems.

Lastly, our approach is also inspired from the simplex architecture [22], where a complex unverifiable controller is used in combination with a simple controller. In this architecture, the complex controller is used for better performance while the worst-case stability is guaranteed by the simple controller. Here, the system is in the complex control mode within a bounded safe region in the state space and switches to the simple control mode as the system state crosses the boundary. This safe region is usually a subspace within the stable region of the simple controller. We adapt the simplex concept to be applied in the NCS, where the complex controller is fast, yet subject to packet drops while the simple controller is safe and robust. We further propose an implementation scheme to realize such a switched controller system.

III. Feedback Control Systems

This paper mainly addresses the problem of guaranteeing the worst-case and improving the expected performance of networked embedded control systems by taking into consideration the elastic nature of the platform resources on which the control applications are implemented. There are mainly two aspects of designing networked embedded controllers: (i) computing the control law and (ii) implementing the controllers. It is to be noted here that our contribution is on the implementation part. Correspondingly, we assume as a prerequisite for our proposed implementation scheme

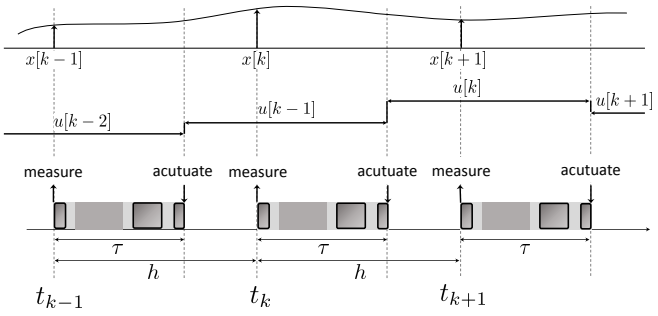


Figure 2: Distrete-time system with sensor-to-actuator delay.

that given the closed-loop timing properties like sampling period and delay, there exists a control law which satisfies the required control performance. In this regard, we describe in this section the basics of feedback control systems under consideration [23]. We also give an example of how one may compute a control law for the given performance requirement, sampling period and delay.

A. System Model

In this section, we discuss linear time-invariant (LTI) single-input systems for which the continuous-time dynamics can be represented mathematically as a set of differential equations given by

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t), \\ y(t) &= Cx(t). \end{aligned} \tag{1}$$

For an n-th order system with m outputs, $x(t) \in \mathbb{R}^{n\times 1}$, $u(t) \in \mathbb{R}$ and $y(t) \in \mathbb{R}^{m\times 1}$ represent respectively the system states, the control input and the outputs at time t. For LTI systems, the matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times 1}$ and $C \in \mathbb{R}^{m\times n}$, i.e., the state, input and output matrices respectively, are constant.

Now, in a networked embedded control system, a controller is implemented as a software running on the processor and the feedback and control signals are transmitted as data frames over a communication network. Correspondingly, we may say that the sensors read the feedback signals and the actuator applies the control input at discrete instants of time, and therefore, the closed-loop system can be naturally represented by a sampled-data model, as shown in Figure 2. Let us consider that the feedback signals are measured at time instants $\{t_k\}$ and are represented by $\{x[k]\}$, where, $x[k] = x(t_k)$. Traditionally, for the sake of simplicity, a controller is implemented according to a constant sampling period h, where, $t_{k+1} - t_k = h$. In addition, software execution and frame transmission take non-negligible time (particularly for highly constrained processing units and contention-based networks). Therefore, we must consider a delay τ between sampling and actuation. Correspondingly, the control input u[k] calculated based on the measurement $x[k]$ is applied at time instant $t_k+\tau$ and is held constant till the next input $u[k+1]$ is applied at

$t_{k+1}+\tau$, as shown in Figure 2. From this assumption, we may write that

$$u(t) = u[k], \;\; t_k + \tau \leq t < t_{k+1} + \tau. \tag{2}$$

Thus, the delayed sampled-data model [23] becomes

$$\begin{aligned} x[k+1] &= \phi x[k] + \Gamma_0 u[k - \left\lfloor \frac{\tau}{h} \right\rfloor] + \Gamma_1 u[k - \left\lceil \frac{\tau}{h} \right\rceil], \\ y[k] &= Cx[k], \end{aligned} \tag{3}$$

where ϕ, Γ_0 and Γ_1, for the continuous-time model presented above, sampling period h and closed-loop delay τ, are given by

$$\begin{aligned} \phi &= e^{Ah}, \\ \Gamma_0 &= \int_0^{h-\tau} (e^{At}dt) \cdot B, \\ \Gamma_1 &= \int_{h-\tau}^{h} (e^{At}dt) \cdot B. \end{aligned} \tag{4}$$

B. Stability

We recognize that closed-loop control systems are predominantly based on the theory of feedback. A state-feedback controller to close the control loop can be mathematically represented as

$$u[k] = -Kx[k], \tag{5}$$

where the control input is a linear function of the system states.

Here, we will give an example on how such a feedback controller can be designed, so that the closed-loop system is stable. For the sake of simplicity, let us consider a system where $0 < \tau < h$, and thus, (3) can be written as

$$x[k+1] = \phi x[k] + \Gamma_0 u[k] + \Gamma_1 u[k-1], \;\; y[k] = Cx[k]. \tag{6}$$

Now, for an augmented state vector $z[k] = \begin{bmatrix} x[k] & u[k-1] \end{bmatrix}^T$, this becomes

$$z[k+1] = \phi_z z[k] + \Gamma_z u[k], \quad y[k] = C_z z[k], \tag{7}$$

where ϕ_z, Γ_z and C_z are given by

$$\phi_z = \begin{bmatrix} \phi & \Gamma_1 \\ 0 & 0 \end{bmatrix}, \;\; \Gamma_z = \begin{bmatrix} \Gamma_0 \\ \mathbf{I} \end{bmatrix}, \;\; C_z = \begin{bmatrix} C & 0 \end{bmatrix}. \tag{8}$$

It is to be noted that for $\tau \geq h$, we can augment the state to $z[k] = \begin{bmatrix} x[k] & u[k-1] \cdots u[k - \lceil \frac{\tau}{h} \rceil] \end{bmatrix}^T$ and then represent the augmented system accordingly.

Now, substituting $u[k] = -K_z z[k]$ in (7), we get

$$z[k+1] = (\phi_z - \Gamma_z K_z) z[k]. \tag{9}$$

Considering $\phi_{cl} = \phi_z - \Gamma_z K_z$, (9) can be rewritten as

$$z[k+1] = \phi_{cl}^{(k+1)} z[0]. \tag{10}$$

Without loss of generality, let us assume $z_e = \mathbf{0}$ as the equilibrium state of the system, and therefore, a system is globally asymptotically stable (GAS) if

$$\lim_{k \to \infty} ||z[k]|| = \mathbf{0}. \tag{11}$$

Consequently, the system in (10) is GAS when

$$\lim_{k \to \infty} ||\phi_{cl}^k|| = 0. \tag{12}$$

This is only possible when the eigenvalues of ϕ_{cl}, i.e., λ_i $\forall i = 1, 2, ..., (n+1)$, satisfy

$$|\lambda_i| < 1. \tag{13}$$

Here, λ_i's also represent the closed-loop system poles.

Considering this stability condition, pole-placement can be used to design a stabilizing controller for LTI single-input systems using the Ackermann's formula

$$K_z = \begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix} \gamma_c^{-1} H(\phi_z), \tag{14}$$

where γ_c is the controllability matrix given by

$$\gamma_c = \begin{bmatrix} \Gamma_z & \phi_z \Gamma_z & \phi_z^2 \Gamma_z & \cdots & \phi_z^n \Gamma_z \end{bmatrix}, \tag{15}$$

and for the selected poles λ_i's satisfying (13), $H(\phi_z)$ is given by

$$H(\phi_z) = (\phi_z - \lambda_1 \mathbf{I})(\phi_z - \lambda_2 \mathbf{I}) \cdots (\phi_z - \lambda_{n+1} \mathbf{I}). \tag{16}$$

It is to be noted here that Ackermann's formula is only valid when the system is controllable, i.e., γ_c has a full rank or is invertible.

C. Performance-Aware Controller Design using PSO

Besides stability, the design of a controller often also need to consider closed-loop performance and physical constraints. Common performance metrics include settling time, control error, input energy, and so on. On the other hand, the physical constraints are input saturation, actuator bandwidth, etc. Correspondingly, in order to design a controller, we need to formulate an optimization problem with closed-loop poles λ_i's as variables satisfying (13) and physical constraints where the optimization objective is the closed-loop performance. However, the resulting problem is non-convex in nature, and therefore, in this paper we use Particle Swarm Optimization (PSO) technique [24], [25] to solve the problem as it is of polynomial time complexity.

In this technique, the process starts with an initial set of particles with certain position and velocity in the design space. Subsequently, the particles travel the design space in search of more optimal solution according to certain rule. For each particle, the trajectory of search is guided by two points, i.e., the local optimal, P_i^l, and the global optimal, P^g, points. P_i^l corresponds to the best point the i-th particle has come across in terms of optimality while P^g is the best among all P_i^l's. Now, the position update for each particle is given by

$$\begin{aligned} V_i^* &= \alpha_0 V_i^* + \alpha_1 R_i^1 (P_i^l - P_i^*) + \alpha_2 R_i^2 (P^g - P_i^*), \\ P_i^+ &= P_i^* + V_i^*, \end{aligned} \tag{17}$$

where V_i^*, P_i^* and P_i^+ represent respectively the current velocity, the current position and the next position, R_i^1 and R_i^2 are two uniformly distributed random numbers between 0 and 1, and α_0, α_1 and α_2 are parameters determined empirically. Moreover, any two points in the design space are compared

based on the objective value only if both or none satisfy all the constraints otherwise the feasible one is better. The algorithm is terminated once all particles have converged or the maximum number of iterations has been reached.

IV. Control Scheme for Elastic Resources

In the last section, we have described how to design a controller when the closed loop system dynamics are known. However, our goal in this paper is to design a control algorithm that elastically uses additional available resources to improve control performance, but does not rely on them to provide basic performance guarantees. Specifically, we are working under the assumption that those resources are available most of the time, but that the system can not guarantee their availability to our application. Such uncertainties might be rooted in sporadic congestion on event triggered communication channels, intermittent faults in the sensors or event triggered, high priority tasks that interrupt/prevent the execution of the control task itself. Even if none of the above play a role, the reality in most non-trivial networked control systems is that resource utilization can not be accurately predicted in an analytical fashion. So while simulations, test and approximations may show that the resources will most likely be available, an actual proof can often either not be given or at least only via high amounts of over provisioning. Actually we want to even encourage such a split into instances of deterministic and non-deterministic allocation of resources to reduce the burden to the system engineer, while still providing value to the application developer. Once such a separation of deterministic and non-deterministic time instances is provided, the control designer only has to know whether the control loop can be deterministically closed at a particular time instance t_i or not.

With that knowledge, he should then be able to achieve the following two goals:

(a) Guaranteeing a certain minimal performance – even in the worst case.
(b) Achieving good performance in the expected case, where the loop is closed (almost) every time.

As those two properties generally become relevant under different operating conditions our intuitive approach is to use one conservative/slow controller $\mathcal{S}$ that is able to guarantee (a) under any conditions, a second, aggressive/fast controller $\mathcal{F}$ that can provide (b) under expected conditions and from those build a mixed controller $\mathcal{M}$ that switches between $\mathcal{S}$ and $\mathcal{F}$ in a way that always preserves guarantee (a) and preserves (b) as good as possible.

In the remainder of this section, we will provide a more formal description of our model as well as the assumptions we make about the system and the control algorithms. Then we will describe the actual selection algorithm that merges $\mathcal{S}$ and $\mathcal{F}$ into $\mathcal{M}$ and prove that this algorithm indeed preserves (a) and (b).

A. Formal Description of System and Control Model

As explained, we model the actual availability of processing and communication resources used by the control loop as a sequence of time instances $\{t_k\}$ at which the control loop is closed. The exact sequence $\{t_k\}$ is unknown in advance, but the following relations always hold:

$$\begin{aligned} \{t_k\} &\subseteq \left\{t_k^F\right\} \\ \{t_k\} &\supseteq \left\{t_k^S\right\} \\ \{t_k\} &\sim \left\{t_k^F\right\} \end{aligned} \tag{18}$$

Where $\left\{t_k^S\right\}$ are the deterministic time instances (where the control loop is guaranteed to be closed) and $\left\{t_k^F\right\}$ the union of the deterministic and non-deterministic time instance. So for all instances $\left\{t_k^F\right\} \setminus \left\{t_k^S\right\}$ we expect the control loop to be closed but can not be sure of it. The last relation $\{t_k\} \sim \left\{t_k^F\right\}$ shows our expectation that the control loop will be closed at almost all non-deterministic time instances and while we obviously can not rely on this for the guaranteed base performance it is the assumption our algorithm will be optimized for. Finally, we require the system to be configured in a way that, if the control loop is not closed at a particular time instance $t_i \notin \{t_k\}$, then the control input u will remain the same as before (the system does not update the set point of the actuator) which is the most common behavior in practice.

We only consider sparse disturbances, where a disturbance means that the plant state is moved from an equilibrium region $Z_{\mathcal{E}}$ around the set-point to a disturbed state z_0 from one time instance to another. This can either be due to an external influence (e.g. an external force or a load change) or because the set-point was changed. In order to keep the equations and formulations simple however, whenever we talk about the system state z we actually mean the system state relative to the set-point. So the controller's goal is always to drive the system state back to zero. In many practical cases this transformation can actually be achieved via an additional feed-forward component in the controller. By sparse we mean that a new disturbance only arrives after the previous on was successfully rejected and the system state is back in $Z_{\mathcal{E}}$.

Regarding the modeling of the proposed controller, we first of all define a control strategy as the combination of the two functions $\mathcal{K}(z,t)$ and $\mathcal{G}(z,t)$, where $\mathcal{K}$ is used to calculate the new control input u and $\mathcal{G}$ decides if the control input should be updated at all. Hence, $\mathcal{G}$ can be seen as the ability of a controller to deliberately not close the control loop even when $t_i \in \{t_k\}$.

The mixed control strategy $\mathcal{M}$ we propose is based on the combination of two simpler, deterministic, stateless, state feedback control strategies (possibly with a feed-forward part) denoted $\mathcal{S}$ and $\mathcal{F}$. $\mathcal{S}$ was designed under the assumption that $\{t_k\} = \left\{t_k^S\right\}$ and $\mathcal{F}$ for the case $\{t_k\} = \left\{t_k^F\right\}$ with the property that – in the case of $\{t_k\} = \left\{t_k^F\right\}$ – $\mathcal{F}$ yields a much better control performance than $\mathcal{S}$. For $\mathcal{S}$ and $\mathcal{F}$ we require that

$$\mathcal{G}^S(\cdot, t_i) = \begin{cases} true & t_i \in \{t_k^S\} \\ false & otherwise \end{cases} \tag{19}$$

$$\mathcal{G}^F(\cdot, t_i) = \begin{cases} true & t_i \in \{t_k^F\} \\ false & otherwise \end{cases} \tag{20}$$

Note however that for our system any $t_i \in \{t_k\}$ will also be in $\{t_k^F\}$ and hence $\mathcal{G}^F(\cdot, t_i)$ will always return true.

An intuitive example that can be modeled in this way is for $\{t_k^S\}$ and $\{t_k^F\}$ to represent periodic sampling with two different periods h^S and h^F with $\frac{h^S}{h^F} = N > 1$ (h^S being an integral multiple of h^F). Conversely, $\mathcal{S}$ and $\mathcal{F}$ could be linear state feedback controllers described by control matrices K^S and K^F which are designed via pole placement for the respective sampling periods as described in sectionsec:background. While those are the cases we look at in our experimental evaluation, we want to point out however that the switching strategy proposed in the following does not depend on $\{t_k^S\}$ and $\{t_k^F\}$ being periodic nor on $\mathcal{S}$ and $\mathcal{F}$ being designed via pole placement (or even being linear at all).

For the scope of this paper, we define control performance in terms of the settling time T, which is the time a controller needs to drive the plant from a disturbed state z_0 back to a small equilibrium region $Z_\mathcal{E}$ around the set point. With the property that – once inside $Z_\mathcal{E}$ – $\mathcal{S}$ can ensure that the plant's state stays in that region.

The fundamental guarantee we want to provide with $\mathcal{M}$ is that – for any valid realization $\{t_k\}$ and initial disturbed state z_0, the following inequality holds true:

$$T^M(z_0) \leq T^S(z_0) \cdot \Delta \tag{21}$$

where T^M is the settling time of the mixed control strategy $\mathcal{M}$ and $\Delta > 1$ represents a design parameter that allows to trade performance guarantees under worst case conditions for better performance under the expected conditions (i.e., $\{t_k\} \sim \{t_k^F\}$). This trade-off is important, as we want to use the aggressive controller as much as possible, but in most cases the state trajectory determined by $\mathcal{F}$ will lead through an area, from which T^S would be even longer than from z_0. In addition, our objective for the design of $\mathcal{M}$ is to minimize the settling time $T^M(z_0)$ for the case $\{t_k\} = \{t_k^F\}$ (i.e. that resources are available at all non-deterministic time instances). A good heuristic seems to try to achieve the same control performance as $\mathcal{F}$ as that control strategy was designed for $\{t_k\} = \{t_k^F\}$ in the first place. Consequently we formulate our optimization goal as.

$$min\left(\frac{T^M(z_0)}{T^F(z_0)}\right) \quad | \ \{t_k\} = \{t_k^F\} \tag{22}$$

Finally we want to point out, that the condition (21) implies stability as long as $\mathcal{S}$ guarantees stability.

B. Proposed control algorithm

Given a (potentially augmented) plant model A (see III), a platform described by $\{t_k^S\}$ and $\{t_k^F\}$ and the two control strategies $\mathcal{S}$ and $\mathcal{F}$, the strategy $\mathcal{M}$ we propose to achieve the control objectives (21) and (22) is constructed by composing $\mathcal{S}$ and $\mathcal{F}$. More specifically it is characterized by a selection strategy $\Phi(t_i, z_i) \mapsto \mathcal{C} \in \{\mathcal{S}, \mathcal{F}\}$ that – for each $t_i \in \{t_k\}$ – decides to apply either $\mathcal{S}$ or $\mathcal{F}$. That strategy is formally described in Algorithm 1, which is described in the following:

Algorithm 1 Selection strategy $\Phi(t_i, z_i)$

Input: z, t_i
Parameters: $\mathcal{S}$, $\mathcal{F}$, Δ, $Z_\mathcal{E}$
Global Variables: $t_d := -1$

```
1:  if z ∈ Z_E then
2:      t_d := −1
3:      select S
4:  else
5:      let t_n := findNext(t_i ∈ {t_k^S})
6:
7:      //calculate deadline if disturbance is new
8:      if t_d = −1 then
9:          if t_i ∈ {t_k^S} then
10:             t_d := t_i + T^S(z) · Δ
11:         else
12:             t_d := t_n + T^S(z) · Δ
13:         end if
14:     end if
15:
16:     let u_F := K^F(z, t_i)
17:     let z_n := predictZn(z, t_i, u_F)
18:     let ft := T^S(z_n)
19:
20:     //apply F or fall back to S
21:     if t_d − t_n ≤ ft then
22:         select F
23:     else
24:         select S
25:     end if
26: end if
```

Lines 1 – 3: First of all, if the plant is already in the equilibrium region $Z_\mathcal{E}$, the slow control strategy $\mathcal{S}$ is picked and the deadline is set to a negative value to indicate the system is not in a transient state. As $\mathcal{S}$ does only need the deterministic time instances, this is a simple way to guarantee stability. Furthermore, depending on the system structure and implementation, this might save system resources, as the new control input hast to be computed and transmitted less often. It might even be possible to switch the sensors into a mode with a lower sampling frequency.

Lines 7 – 14: If however, the system is in a disturbed state, the first step is to determine if a new deadline has to be calculated (if the system just left the equilibrium state) or if the controller still tries to reject a previous state. As described

in section IV-A $T^S(z)$ represents the settling time from state z if only controller $\mathcal{S}$ was used. How exactly this time is predicted during runtime obviously depends on the plant and the controller. In cases where no closed form solution can be given, a simulation might be necessary (for linear systems this e.g. requires a sequence of matrix multiplications). In some cases it might also be beneficial to pre-compute those settling times and store them in a lookup table.

The reason, why the next deterministic time instance t_n serves as an offset if $t_i \notin \{t_k^S\}$ is that this would be the first time instant after the disturbance, where $\mathcal{S}$ would initiate a control update.

Lines 16 – 21: Whether the system just arrived in a disturbed state or is in the process of rejecting a previous disturbance, the next step is to determine if selecting the fast controller is possible without running the risk to arrive in a state from which (21) can no longer be achieved. To that end, the algorithm first predicts the system state at the next deterministic time instance t_n assuming that $\mathcal{F}$ would be picked now and the control loop is not closed at any of the non-deterministic instances till then. Then the fall back settling time $ft = T^S(z_n)$ is determined. As explained before this is the time the slow control algorithm would need to drive the system from state z_n to the equilibrium state (figure 3 gives an overview over some of the time related variables used in the algorithm). If this shows that applying u_F now, not updating the control input until t_n and only selecting $\mathcal{S}$ afterwards satisfies (21) $\mathcal{F}$ is selected. Otherwise the algorithm falls back to $\mathcal{S}$ for this time instance, which might result in not updating the control input at all.

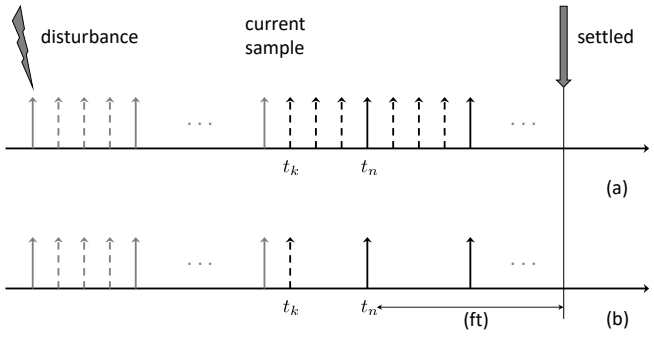


Figure 3: (a) shows the relationship between the disturbance, the current sample, the next guaranteed sample and the settling of the system. (b) shows the fallback time ft is obtained on the assumption that only deterministic time instances will be available from now on.

A detailed explanation of why this algorithm fulfills our performance criteria (21) and (22) will follow in the next section, but we want to mention already at this point that Line 21 presents a sufficient, but not a necessary condition for selecting $\mathcal{F}$ – it might be possible to choose $\mathcal{F}$ even if the fall-back time exceeds $t_d - t_n$. Also, choosing $\mathcal{F}$ when possible is not necessarily optimal but a heuristic. Essentially we are trading optimality for simplicity with the goal to reduce computation effort during runtime and make the algorithm easier to analyze and verify.

C. Analysis of Algorithm

An optimal selection strategy would – in each time step t_i – have to solve the optimization problem:

$$\min_{\phi_i} \left(E\left[T^M(z_i)\right]\right) \tag{23}$$

$$s.t.\ \ T^M(z_i) < t_d - t_i \ \ | \ \forall \{t_k\}_{k>i} \tag{24}$$

As this does not seem computationally feasible in the context of an embedded real-time application, our algorithm described in the previous section uses a much simpler heuristic approach, which still guarantees (21) but might result in slightly worse control performance.

The first step is to identify which choices are actually feasible in the first place. For either $\mathcal{S}$ or $\mathcal{F}$ to be viable we have to show that after making this choice at time instance t_i there exist at least one sequence of selections $\{\phi_k\}$ for any possible future $\{t_k\}$ that will drive the system state back to the equilibrium region $Z_\mathcal{E}$ before the deadline t_d as calculated after the initial detection of the disturbance. Please note that this does not have to be the same sequence that will actually be taken, but the fact that there is such a sequence at all makes the choice feasible.

Generally speaking, finding such a sequence can be considered a minimax search, where the algorithm first selects either $\mathcal{S}$ or $\mathcal{F}$ with the goal of minimizing the settling time and the system – as the adversary – selects the next time instance where the control loop will be closed with the goal of maximizing the settling time from the range of $[t_{i+1}, t_n]$. Performing an actual minimax search that would find the shortest possible settling time that can be guaranteed by the control algorithm even for a worst case realization of t_k seems complex at first but can be simplified in two ways:
First, the system can never make things worse by selecting a time instance $t_{i+x} \notin \{t_k^S\}$, because the controller has the ability to select $\mathcal{S}$ which effectively means not to close the control loop in this time instance, meaning the system can only prevent updating the control inputs but not enforce it. As a result, it is enough to find a valid selection sequence for the case of $\{t_k\} = \{t_k^S\}$ and the exact same trajectory and consequently settling time can be achieved for any other $\{t_k\}$. In our proposed algorithm we do not actually search for the sequence, resulting in the best settling time but instead only check what would happen if $\mathcal{S}$ is selected every time. The thus approximated settling time might exceed the true best settling time that is achievable for $\{t_k\} = \{t_k^S\}$. However, our assumption is that if there would be a much better solution for that case , then the control strategy $\mathcal{S}$ would have been designed accordingly instead.

To summarize, in order to determine if a choice ϕ_i is viable, we only determine the settling time for the case that $\{t_k\} = \{t_k^S\}$ and $\{\phi_k\}_{k>i} = \mathcal{S}$. The fact that the check assumes the usage of $\mathcal{S}$ for all future instances t_k allows to skip the check for $\phi_{i+1} = \mathcal{S}$ as this would be redundant.

Furthermore $\mathcal{S}$ is trivially acceptable for the first choice when the disturbance is initially detected, as the deadline is derived from this. As a result, using $\mathcal{S}$ is always a viable choice and does not have to be checked at all.

After determining if $\mathcal{F}$ is a viable choice at all, our algorithm always picks that strategy if possible, even though this might not actually be the optimal choice for that particular time instance. The intuitive reason, why this very simple (and hence fast) strategy works nonetheless is that our general assumption about the system is that $\{t_k\} = \{t_k^F\}$, which is the situation $\mathcal{F}$ was designed for in the first place. Furthermore, we expect the base controllers $\mathcal{S}$ and $\mathcal{F}$ to exhibit a certain robustness against noise anyway, such that small deviations from $\{t_k^F\}$ (meaning the control loop might not be closed at some non-deterministic time instances) should not significantly degrade control performance.

A final and very important observation is that providing the guarantee (21) and a good expected control performance in general (as expressed via (22)) depend on multiple assumptions detailed in section IV-A. More basic properties like stability however can be given under a much wider range of circumstances, because at some point before or equal to the deadline, the algorithm will permanently switch to $\mathcal{S}$ until the equilibrium state is reached. That means in addition to any properties discussed so far, the algorithm inherits any properties from $\mathcal{S}$, albeit in a possibly time delayed manner.

D. Computing settling time

The computationally heavy part of our algorithm is the calculation of $T^S(z)$. Even for a relatively simple state-feedback control algorithm, as used in this work, we are not aware of a closed form solution to compute the settling time for a discrete-time system. Instead, one has to simulate the system evolution for a (possibly high) number of steps to determine the settling time. Fortunately, the algorithm only has to consider the deterministic time instances and as we only need to know if the deadline t_d can be satisfied, this gives an upper bound on how far the state evolution has to be predicted. Still, depending on the dimensionality of the problem and the complexity of the control algorithm, this computation might be infeasible during runtime due to limited computational resources on the embedded platforms and even if the number of steps is bounded there might still be a high variability in the runtime of this algorithm. An alternative approach would be to rasterize the state-space, pre-compute the worst-case settling times for each slice off-line and store them in a lookup table. However, this approach might have a high memory requirement, especially as a coarser granularity introduces a higher over-approximation. The advantage is however, that the run-time computation is very small (only the calculation of z_n and a lookup are required). Also, for linear systems the function $T(z)$ usually has a logarithmic shape that – when rasterized with an adaptive granularity – can be approximated with comparatively few support points. For the case of simple state-feedback controllers, where the control law itself only involves a low-dimensional matrix multiplication, we expect however that it is more efficient to compute the settling time during runtime.

V. Experimental Results

A. Simulation Setup

For our evaluation we use a plant model of a DC motor taken from [26] with the following continuous state space representation:

$$\begin{aligned} A &= \begin{bmatrix} -10 & 1 \\ -0.020 & -2 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 2 \end{bmatrix} \\ C &= \begin{bmatrix} 10 \end{bmatrix}, \quad D = \begin{bmatrix} 0 \end{bmatrix} \end{aligned} \tag{25}$$

The sampling period and network delay parameters where chosen as follows:

$$\begin{aligned} h^f &= 0.04 \\ h^s &= h^f \cdot 3 = 0.12 \\ \tau &= h^f/2 = 0.02, \end{aligned} \tag{26}$$

where h^f, h^s and τ represent the sampling period for the fast/aggressive controller, the sampling period of the slow/safe controller and the sensor-to-actuator delay respectively.

For this system we designed two linear state feedback controllers via the PSO technique described in section III, yielding the following control gain values:

$$\begin{aligned} K^F &= \begin{bmatrix} -50.0 & -11.5 & -0.431 \end{bmatrix} \\ K^S &= \begin{bmatrix} -11.0 & -5.16 & -0.201 \end{bmatrix} \end{aligned} \tag{27}$$

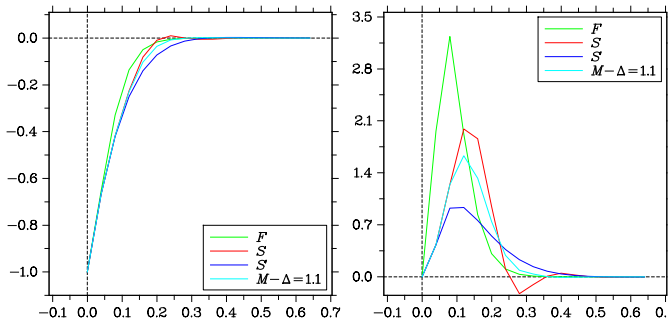
Note, that K has three dimensions as a network delay $\tau \leq h$ requires one additional state to keep track of the last control input. In addition, we also designed a second, instable plant by simply inverting the entry a_{11} in the matrix A. In that case, the control gain values found by PSO are

$$\begin{aligned} K^F &= \begin{bmatrix} -500 & -27.1 & -0.917 \end{bmatrix} \\ K^S &= \begin{bmatrix} -139 & -10.4 & -0.372 \end{bmatrix} \end{aligned} \tag{28}$$

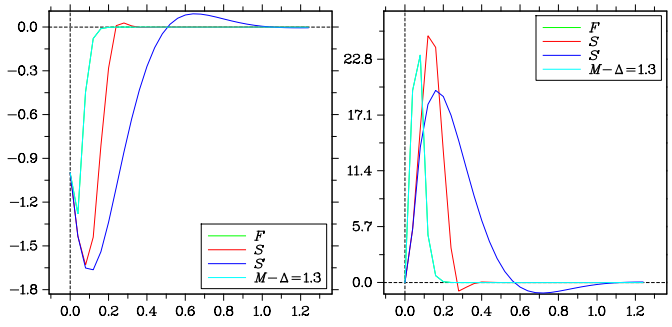
Figure 4 shows the evolution of individual state dimensions over time for each of the two plants with different control strategies $\mathcal{S}$, $\mathcal{F}$ and $\mathcal{M}$ for the case that $\{t_k\} = \{t_k^F\}$. In addition, we also evaluated control strategy $\mathcal{S}'$ that applies the conservative control law K^S at every time instance of $\{t_k\}$ and not just at instances $\{t_k^S\}$. In particular the case of plant 2 demonstrates that even for linear plants, applying a control law at additional time instances can actually degrade the control performance significantly.

As discussed before, given the two basic control strategies $\mathcal{S}$, $\mathcal{F}$, the remaining design parameter is the slack factor Δ (i.e., the factor by which we allow the worst case settling time of our mixed control strategy $\mathcal{M}$ to exceed that of $\mathcal{S}$). Figures 5 and 6 show how the performance of $\mathcal{M}$ depends on different values of Δ and on how often the control loop is closed at the non-deterministic time instances. For that purpose we generated random sequences $\{t_k\}$ for each $0 < p < 1$, s.t.

$$\begin{aligned} &\{t_k^S\} \subseteq \{t_k\} \\ &|\{t_k\}| = |\{t_k^S\}| + \left(|\{t_k^F\}| - |\{t_k^S\}|\right) \cdot p \end{aligned} \tag{29}$$



(a) Plant 1



(b) Plant 2

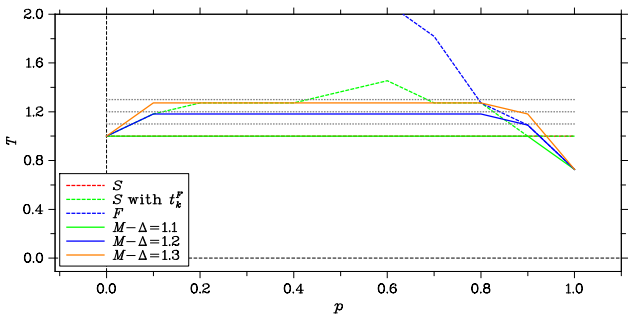
Figure 4: State evolution over time for different control strategies with $\{t_k\} = \{t_k^F\}$

In other words, a valid sequence $\{t_k\}$, where only a fraction p of all possible non-deterministic time instances where actually included. Then we simulated the system evolution for each of the sequences with different controllers $\mathcal{M}_\Delta$, where Δ was varied from 1.1 to 1.3. The observed average and worst case settling times can be seen in the respective sub-figures. They are normalized to the settling time of $\mathcal{S}$ and again, we also give the performance of $\mathcal{F}$, $\mathcal{S}$ and $\mathcal{S}'$ for comparison.

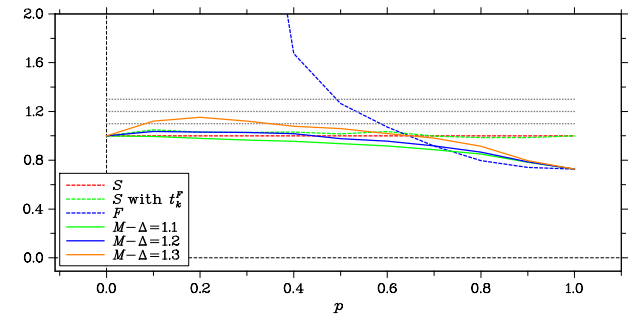
B. Analysis of simulation results

What the figures show is that the worst case settling time stays indeed below the set limit of $\Delta \cdot T^S$ (represented by the horizontal lines) in all cases and that for $\{t_k\} = \{t_k^F\}$ we indeed can achieve the same settling time as $\mathcal{F}$.

However, those two examples also show that finding the optimal value for Δ is a non-trivial problem: An upper limit for Δ can obviously be given by the safety requirements of the application (i.e., what is the worst case settling time we can tolerate), but contrary to what one might expect, a higher Δ does not always yield better results. In fact, Figure 5 shows that a Δ of 1.1 yields better worst *and* average case performance than the higher values. In particular, the settling times where always smaller or equal to T^S and not just $\Delta \cdot T^S$. On the other hand, the simulations for plant two show that in some cases, values of Δ that are too small can – even in the best case – result in worse settling times than just using $\mathcal{S}$: For values of 1.1 and 1.2 we see that T^M increases with the number of additional time instances, so allocating optional resources to this task would even be harmful. Only with $\Delta = 1.3$ do we get

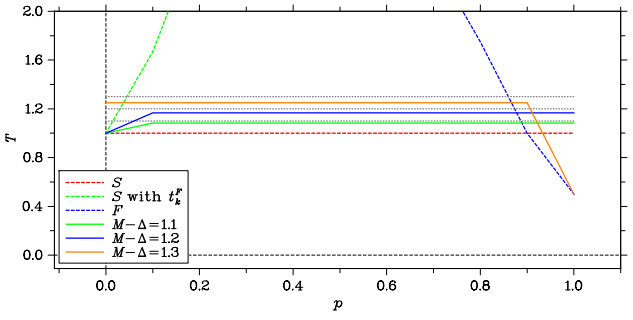


(a) Worst case settling time

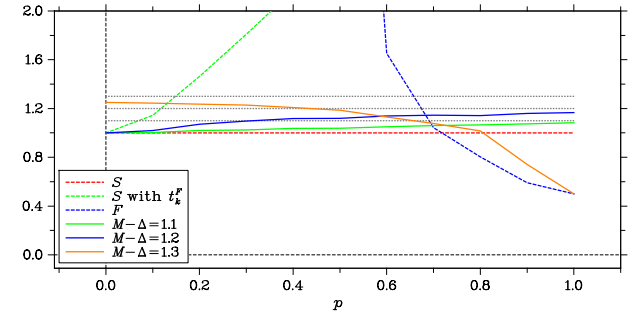


(b) Average settling time

Figure 5: Observed settling times for plant 1



(a) Worst case settling time



(b) Average settling time

Figure 6: Observed settling times for plant 2

the desired result of having a settling time that is roughly half of T^S, but consequently we do see a significant performance degradation for low values of p, although they do not exceed the set limit.

Still, in summary, our results show that the controller is able to exploit the non-deterministic time instances to achieve the same performance as $\mathcal{F}$ in the expected case, while guaranteeing that the performance does not fall below the predetermined threshold. However, they also show the importance of choosing a good Δ.

VI. Concluding Remarks

In this paper we presented a novel approach for the effective utilization of elastic resources in the domain of networked control systems. We provided a formalism that allows the system designer to describe non-deterministic resources in an abstract but usable manner to the control engineer and provided an initial design template for control algorithms that want to use those non-deterministic resources effectively without compromising on basic performance guarantees.

In particular, our approach does not require any knowledge about the availability of non-deterministic resources at the next time instant (i.e. we don't require the ability to reserve those resources for a certain number of consecutive instances to be of value to us). The limited resource can be the communication between sensor and controller, or the communication between the controller and the actuator, or even the processing time on one of the involved processors.

While our approach provides a strict guarantee about the worst-case behavior for the combination of any two given stateless control algorithms, methods for analyzing the expected behavior and choosing a suitable value for the slack factor Δ warrant more investigation. Another field worth investigating is the generalization of this approach beyond the case of rejecting a single disturbance to, e.g. general reference tracking and different kinds of performance metrics.

Acknowledgment

This work was partially supported by Deutsche Forschungsgemeinschaft (DFG) through the TUM International Graduate School of Science and Engineering (IGSSE).

References

[1] X. Liu, X. Chen, and F. Kong, "Utilization control and optimization of real-time embedded systems," *Foundations and Trends® in Electronic Design Automation*, vol. 9, no. 3, pp. 211–307, 2015.

[2] R. Majumdar, I. Saha, and M. Zamani, "Performance-aware scheduler synthesis for control systems," in *Proceedings of the ninth ACM international conference on Embedded software (EMSOFT)*, pp. 299–308, ACM, 2011.

[3] I. Saha, S. Baruah, and R. Majumdar, "Dynamic scheduling for networked control systems," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control (HSCC)*, pp. 98–107, ACM, 2015.

[4] D. Goswami, R. Schneider, and S. Chakraborty, "Relaxing signal delay constraints in distributed embedded controllers," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 6, pp. 2337–2345, 2014.

[5] A. Cervin, "Stability and worst-case performance analysis of sampled-data control systems with input and output jitter," in *2012 American Control Conference (ACC)*, pp. 3760–3765, IEEE, 2012.

[6] P. Tabuada, "Event-triggered real-time scheduling of stabilizing control tasks," *IEEE Transactions on Automatic Control*, vol. 52, no. 9, pp. 1680–1685, 2007.

[7] M. Abdelrahim, V. S. Dolk, and W. P. M. H. Heemels, "Input-to-state stabilizing event-triggered control for linear systems with output quantization," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 483–488, IEEE, 2016.

[8] A. Aminifar, P. Tabuada, P. Eles, and Z. Peng, "Self-triggered controllers and hard real-time guarantees," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 636–641, IEEE, 2016.

[9] S. Samii, P. Eles, Z. Peng, P. Tabuada, and A. Cervin, "Dynamic scheduling and control-quality optimization of self-triggered control applications," in *2010 31st IEEE Real-Time Systems Symposium (RTSS)*, pp. 95–104, IEEE, 2010.

[10] D. Goswami, R. Schneider, and S. Chakraborty, "Re-engineering cyber-physical control applications for hybrid communication protocols," in *2011 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, IEEE, 2011.

[11] A. Masrur, D. Goswami, R. Schneider, H. Voit, A. Annaswamy, and S. Chakraborty, "Schedulability analysis of distributed cyber-physical applications on mixed time-/event-triggered bus architectures with retransmissions," in *2011 6th IEEE International Symposium on Industrial and Embedded Systems*, pp. 266–273, IEEE, 2011.

[12] A. Masrur, D. Goswami, S. Chakraborty, J. Chen, A. Annaswamy, and A. Banerjee, "Timing analysis of cyber-physical applications for hybrid communication protocols," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1233–1238, IEEE, 2012.

[13] J. Nilsson, B. Bernhardsson, and B. Wittenmark, "Some topics in real-time control," in *1998 American Control Conference (ACC)*, pp. 2386–2390, IEEE, 1998.

[14] E. Boje, "Approximate models for continuous-time linear systems with sampling jitter," *Automatica (Journal of IFAC)*, vol. 41, no. 12, pp. 2091–2098, 2005.

[15] R. Schneider, D. Goswami, S. Zafar, M. Lukasiewycz, and S. Chakraborty, "Constraint-driven synthesis and tool-support for flexray-based automotive control systems," in *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp. 139–148, ACM, 2011.

[16] D. Roy, L. Zhang, W. Chang, D. Goswami, and S. Chakraborty, "Multi-objective co-optimization of flexray-based distributed control systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2016 IEEE*, pp. 1–12, IEEE, 2016.

[17] B. C. Kuo, *Digital Control Systems*. HRW Series in Electrical and Computer Engineering, New York: Holt, Rinehart and Winston, Inc., 1980.

[18] H. Lin and P. J. Antsaklis, "Stability and stabilizability of switched linear systems: A survey of recent results," *IEEE Transactions on Automatic Control*, vol. 54, no. 2, pp. 308–322, 2009.

[19] I. Saha and R. Majumdar, "Trigger memoization in self-triggered control," in *Proceedings of the tenth ACM international conference on Embedded software (EMSOFT)*, pp. 103–112, ACM, 2012.

[20] R. Blind and F. Allgower, "Analysis of networked event-based control with a shared communication medium: Part i - pure aloha and part ii - slotted aloha," in *2011 IFAC World Congress*, 2011.

[21] A. Molin and S. Hirche, "Optimal design of decentralized event-triggered controllers for large-scale systems with contention-based communication," in *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, pp. 4710–4716, IEEE, 2011.

[22] L. Sha, "Using simplicity to control complexity," *IEEE Software*, vol. 18, no. 4, pp. 20–28, 2001.

[23] K. J. Åström and B. Wittenmark, *Computer-controlled systems (3rd ed.)*. Prentice Hall Information and System Sciences, USA: Prentice-Hall, Inc., 1997.

[24] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks*, pp. 1942–1948, IEEE, 1995.

[25] W. Chang and S. Chakraborty, "Resource-aware automotive control systems design: A cyber-physical systems approach," *Foundations and Trends® in Electronic Design Automation*, vol. 10, no. 4, pp. 249–369, 2016.

[26] B. Messner and D. Tilbury, "Control tutorials for matlab and simulink."