Intelligent Automation And Soft Computing,
2020 Vol. 26, no. 3, 631–639
DOI: 10.32604/iasc.2020.012549

A Progressive Output Strategy for Real-time Feedback Control Systems

Qiming Zou[1], Ling Wang[1, *], Jie Liu[1] and Yingtao Jiang[2]

1Department of Computer Science and Technology, Harbin Institute of Technology, Harbin, 150001, China
2Department of Electrical and Computer Engineering, University of Nevada, Las Vegas, 89154, USA,

ABSTRACT

The real-time requirements imposed on a feedback control system are often hard to be met, as the controller spends a disproportionately large amount of time waiting for a control cycle to reach its final state. When such a final state is established, multiple tasks have to be prioritized and launched altogether simultaneously, and the system is given an extremely short time window to generate its output. This huge gap between the wait and action times, perceived as a load unbalancing problem, hinders a control decision to be made in real time. To address this challenging problem, in this paper, we present a progressive output strategy that divides a control cycle into a few fine-grained control intervals, and the entire workload is scheduled across these control intervals. Dubbed as Progressive Output Strategy (PROS), this approach actively requests intermediate states be created between adjacent control cycles in an adaptive manner. Specifically, as the sensing information is arriving, a system that adopts PROS can generate a series of intermediate solutions that eventually converge to the final optimal control signal. This way, the controller will no longer waste its time idling while waiting for the arrival of all the data for one-shot decision-making. Rather the system actually cuts down the waiting time and is able to act on the intermediate data/states throughout the entire control cycle. Experimental results have confirmed that adopting the PROS in a feedback control loop can evenly distribute the workload over a control cycle, and thus, the time delay is reduced by as much as two orders of magnitude, which is essential to meet the most stringent timing requirements.

KEYWORDS: Real-time feedback control; online optimization; adaptive sampling.

1 INTRODUCTION

A system employs a feedback control loop is able to tightly integrate sensing, computing, and actuating components to handle uncertainties and dynamically adapt its behavior (Lindberg M. et al 2010), (Jacob R. et al 2016). In a feedback loop, as physical processes evolve over time, the computation time is tightly linked to the timing in the physical domain, giving rise to the real-time requirement (Jacob R. et al 2016). Up to date, a great deal of research has been focusing on how to speed up the computation in the feedback control loop. These works have centered around four themes: (1) making the system model numerically easy to compute (Shu L. et al 2012), (2) decomposing the optimization into sub-problems and solving them in a distributive manner (Boyd S. et al 2011), (3) generating solution close to optimal at each control step until the final solution is reached (Wang Y. et al 2010), (Bak S. et al 2017), (Li H. et al 2018), and (4) producing an explicit control law (e.g. Explicit Model Predictive Control (EMPC)) (Bemporad A. et al 2002), (Oberdieck R. et al 2017).

With all these advances in speeding up the computation for fast feedback loop control, it is still hard for a control system to meet the real-time requirement by focusing only on the feedback loop itself. Rather the workload unbalancing problem that was ignored in previous works needs to be carefully addressed. This problem can be particularly important when all the data have finally arrived and multiple high priority tasks yet to be launched altogether, pressuring the system to output its results as soon as possible, as explained in a motivating example shown in Fig. 1. Here a traffic control system is put in place to help alleviate traffic congestion by optimizing the traffic signal operation strategy. Observed traffic states are utilized as the feedback information, and the controller needs to adjust the traffic signals in response to the changes in traffic flow. The traffic information is received every one or several control cycles that can last as long as a few minutes; during the time, the controller has to sit idle, waiting for the

CONTACT Ling Wang wangling@hit.edu.cn

arrival of traffic data. However, when traffic data finally come, the controller is expected to generate the optimal signal strategy within just a few milliseconds. This huge gap between wait and action times, perceived as a load unbalancing problem that prevents the control system from achieving its real time goals, is addressed in this paper.

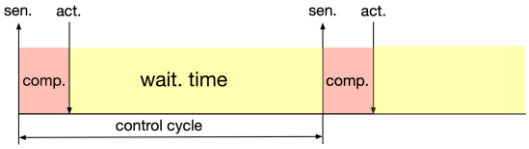

Figure 1: An illustration of the load unbalancing problem. The computation of control signal for actuation (act.) has to wait for a long period of time (referred as wait.time), until the sensors (sen.) sample the current state of the physical system. This mechanism forces the workload to be completed over an extremely short time (referred as comp. time).

A straightforward approach to solve this load unbalancing problem is to get the workload spread over the entire control interval. To be more specific, instead of having the controller waste its time idling while waiting for the arrival of all the data to make a control decision, the system actually cuts down the waiting time and acts on the intermediate data/states that are available at some specific intervals. Along this process, the controller continues to generate sub-optimal solutions and continues to refine them in the next control interval. By the time when all the data needed for making a decision in a control cycle are finally available, the controller can have a "warm-start". As a result, the controller, based on the sub-optimal solution thus far obtained, is now able to quickly complete its computation to generate the actual control signal, as shown in Fig. 2. The sampling rate, which relates to the number of sampling intervals in a control cycle, needs to be carefully selected, as it has serious implications on sampling complexity and control performance (Haimovich H. et al 2013), (Miskowicz M. et al 2014), (Sahoo A. et al 2015), (Wu L. et al 2017), (Tzoumas V. et al 2018). To our best knowledge, there is no open literature that deals with the sampling issues pertaining to workload balancing.

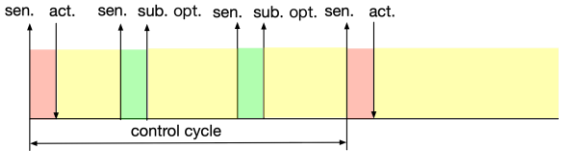

Figure 2: An illustration of the load spreading strategy. The computing component quickly adapts to the change of the sampled state and outputs sub-optimal solutions (sub.opt.). In this way, the workload to be performed over computing periods (red blocks) is distributed to several sample intervals (green blocks) in waiting periods (yellow blocks).

Based on the idea to use the idle time within a control cycle to generate sub-optimal output solutions, we present the Progressive Output Strategy (PROS) (see Fig. 3). PROS has two components: the sampling and the computing components. In each control cycle, the sampling component adaptively samples the state of the plant, after which it triggers computation based on the partial information it has already received. The computing component updates the estimation of the optimal solution and outputs an optimal solution at the end of a current control cycle. During the wait time, an intermediate solution can be generated along with the state changes in the physical processes. In this way, instead of completely staying idle, the sampling component requests sensing information and activates the computing process. By spreading the workload throughout the entire control cycle, the amount of time needs to be spent on computation at each sampling time tends to be small. As a result, one may use a low-end microcontroller, as opposed to a more expensive high-end machine, to achieve the same real-time performance. Although there are works dedicated to consider both sampling and computing at the same time (Tarbouriech S. et al 2016), (Pan Y V. et al 2015), (Hans J F. et al 2014), as far as we know, this paper is the first attempt to reduce time delay by adopting a strategy to spread the workload.

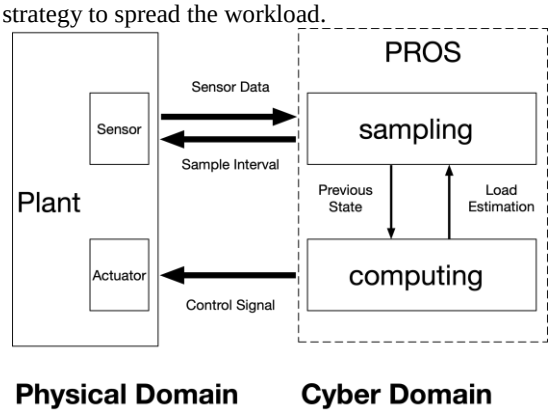

Figure 3: The framework of PROS. At every sample interval, the sampling component contacts the sensors to collect measured data. The length of the sample interval is determined by the load estimation provided by the computing component. The computing component exploits the relationship between the previous state and the current state to figure out how to warm-start a computation.

Our specific contributions are summarized as follows:

- A novel problem on how to add load balancing into real-time feedback control is formulated.
- We present the PROS strategy that can progressively obtain the optimal control signal in a control cycle.
- Empirical experiments show that our approach can reduce the time delay by two orders of magnitude on average with a low sampling complexity.

2 PROBLEM FORMULATION

A physical process can be mathematically represented as a seven-tuple P=(X,U,ρ,X_0,X_g,T,T_c), where

X is the set of states, X∈R^n.

U is the set of feasible control signals, U∈R^m.

ρ :X×U→X is the transition function (or dynamic function) of the physical system. ρ can be approximated by a linear model; that is, x_(t+1)=Ax_t+Bu_t for state vector x_t and input u_t at the t^th control cycle.

- X_0 is the set of initial states, and X_0⊂X.
- X_g is the set of target states, and X_g⊂X.
- T is the control horizon defined as the total number of control cycles.
- T_c is the time span of one control cycle.

The goal is to determine the best feedback loop based on the following criteria:

- Minimize the performance loss J_P, where the performance is defined as a weighted distance between the target state x_g∈X_g and the current state x_t∈X. That is,

$$_P = \min_{u_k \in \mathcal{U}} \sum_{k=t}^{T} (x_k - x_g)' Q (x_k - x_g) + u_k' R u_k \quad (1)$$

where $Q \in \mathbb{R}^{n \times n}$ is the penalizing matrix of the state deviation from the reference trajectory, and $R \in \mathbb{R}^{m \times m}$ is the cost matrix for the control signals.

- Minimize the total time delay J_T measured as the total computation time needed to obtain all the control signals. That is,

$$_T = \min_{u_k \in \mathcal{U}_{k=t}} \sum_{k=t}^{T} \Delta t_k^{act} \quad (2)$$

where Δt_k^{act} is time delay of the control signal during the $k^{th} = 1, \cdots, T$ control cycle.

According to the classic control theory, a controller can be expressed as a quadratic programming if the physical system can be approximated by a linear model and the cost function is quadratic (e. g. J_P). The controller needs to minimize the performance loss, taking into account of some constraints (e. g., safety) applicable to the control signals. That is, the controller can be expressed as:

$$^*(x_t) = \min_U \frac{1}{2} U' H U + U' g(x_t) \quad (3a)$$

$$GU \le b(x_t) \quad (3b)$$

where $U \triangleq [u_t, \ldots, u_{t+T}] \in \mathbb{R}^{m \times T}$ is the optimization vector; $g(x_t) = F' x_t$, $b(x_t) = b + E x_t$; and $H \succ 0$, $F \succ 0$ (positive definite) and they are obtained by incorporating $x_{k+1} = Ax_k + Bu_k$ into J_P in Eq. (1).

As indicated in Eq. (3), the controller needs to receive the measured state x_t of the system at the end of each control cycle. An approximate linear model will be applied to predict future states $x_{t+1}, \cdots, x_{t+T}$ based on x_t, and the controller will minimize the cost function that accounts for both the current and predicted states. The controller's output is an optimal control sequence over the time span of $[tT_c, (t+T)T_c]$. Only the optimal strategy within the interval of $[tT_c, (t+1)T_c]$ will be implemented in the system. At time step $t+1$, the states will be measured again and the data collected will be used to calculate the optimal solution for the next interval, $[(t+1)T_c, (t+T+1)T_c]$, in a rolling horizon fashion.

Although the rolling horizon style introduced above makes the controller more robust against external disturbances, it also leads to a load unbalancing problem. According to the formulation of the feedback loop, the state is measured at the end of control cycle tT_c, and the control signal is also expected to be obtained at the same time, which imposes a heavy workload to the control system.

3 PROS: PROGRESSIVE OUTPUT STRATEGY

The PROS framework controls the sampling interval, warm-starts the computation, and imposes a low workload for the computation of the control signal. As shown in Fig. 3, PROS is composed of a computing component and a sampling component. The computing component outputs intermediate and final solutions (the control signal) based on the current sampled state. In PROS, computing component needs to meet three requirements, namely real-time, robustness to interruption, and capability of providing workload information to the sampling component. Robustness for interruption is imperative because there is a possibility that computing may be interrupted by the sampling process.

The sampling component of PROS adaptively sets an interval for collecting sensing information from the sensors. This component shall be responsible for distributing the load during the wait time with a minimum number of sampling intervals. To achieve this goal, this paper proposes an urgency-triggered sampling strategy. Here urgency is defined based on the load information provided by the computing component and the remaining time in the current control cycle.

3.1 State-oriented Computing Strategy

The main idea of PROS is to adjust the controller's sampling interval and the computing component's needs to quickly adapt itself to the state changes. In light of the parametric active-set algorithm presented in qpOASES (Hans J F. et al 2014), which is an effective method to solve QP sequentially by exploiting the geometrical property (Hans J F. et al 2014), we herein present the State-Oriented Computing Strategy (SOCS).

To obtain QP(x_t) at current instance x_t, SOCS iteratively searches a straight line within the state space from its preceding value given as QP(x_(t-1)). Since the state of the physical system is viewed as one parameter of Eq. (3), the state space shall be treated the same as the parameter space. Fig. 4 shows the parameter space of a two-dimensional example. According to (Hans J F. et al 2014), the parameter

space can be divided into multiple regions. In each region, the optimal solution to Eq. (3) has an identical active set, which is a set of indices of the active constraints. It also means that if both the previous state and the current state fall into the same region, their corresponding optimal solutions have the same active set, and thus, no iteration is needed. On the other hand, if the two states are located in different state regions, several active set changes (iterations) are necessary to move from the previous solution to the current one. As shown in Fig. 4, we use a trajectory to indicate the dynamic change of states. Note that once the trajectory crosses the boundary of the state regions, it means the active set has been updated, which corresponds to one iteration.

Note that SOCS generates sub-optimal control signals along with the change of states (in a straight line originated from x_(t-1)), and only the latest solution (the path reaches the state x_t) is applied to the real physical system.

$$x_0 = x_{t-1} \tag{4a}$$

$$x_0^{new} = x_t \tag{4b}$$

$$\Delta x = x_0^{new} - x_0 \tag{4c}$$

$$\Delta g := g(x_0^{new}) - g(x_0) = F^T \Delta x \tag{4d}$$

$$\Delta b := b(x_0^{new}) - b(x_0) = E\Delta x \tag{4e}$$

The state and constraint vectors are given as follows:

$$x_0(\tau) := x_0 + \tau\Delta x \tag{5a}$$

$$g(\tau) := g(x_0) + \tau\Delta g \tag{5b}$$

$$b(\tau) := b(x_0) + \tau\Delta b \tag{5c}$$

where the parameter τ is the step length along the parameter change direction, which is determined by Eq. (10). We assume that the starting point is a known optimal solution U_0^* and λ_0^* (and their corresponding optimal active set is A_0^*) of the last QP(x_0) and we intend to obtain QP(x_0^new). The basic idea of qpOASES is to move from x_0 towards x_0^new, and thus from (U_0^*,λ_0^*) towards (U_new^*,λ_new^*), while keeping primal and dual feasibility, i.e. optimality, for all the intermediate points.

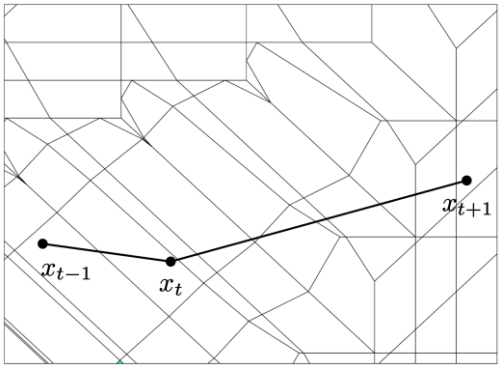

Figure 4: Paths from one QP to the next across multiple state regions: the solid line is the path created by the controller. Sub-optimal solutions are generated along the path, where x_t denotes the sampled state within the t^th control cycle.

Theorem 1: Assume the active set $\mathbb{A}$ stays unchanged during the state transition from x_0 to $x_0(\tau)$, i. e. $\mathbb{A} = \mathbb{A}(0) = \mathbb{A}(\tau)$. Let ΔU^* be the primal step direction and $\Delta\lambda_{\mathbb{A}^*}$ be the dual step direction; that is,

$$\tau\Delta U = U^*(\tau) - U^*(0) \tag{6a}$$

$$\tau\Delta\lambda_{\mathbb{A}} = \lambda_{\mathbb{A}}^*(\tau) - \lambda_{\mathbb{A}}^*(0) \tag{6b}$$

Then we shall have

$$\begin{pmatrix} H & G_{\mathbb{A}}^T \\ G_{\mathbb{A}} & 0 \end{pmatrix}\begin{pmatrix} \tau\Delta U^* \\ -\tau\Delta\lambda_{\mathbb{A}}^* \end{pmatrix} = \begin{pmatrix} -\Delta g \\ \Delta b_{\mathbb{A}} \end{pmatrix} \tag{7a}$$

Proof: At every intermediate point, the primal variable $U^*(\tau)$ and the dual variable $\lambda_{\mathbb{A}}^*(\tau)$ have to satisfy the optimality conditions, *i. e.,* KKT condition; therefore,

$$\begin{pmatrix} H & G_{\mathbb{A}}^T \\ G_{\mathbb{A}} & 0 \end{pmatrix}\begin{pmatrix} U^*(\tau) \\ -\lambda_{\mathbb{A}}^*(\tau) \end{pmatrix} = \begin{pmatrix} -g(\tau) \\ b_{\mathbb{A}}(\tau) \end{pmatrix} \tag{7b}$$

This condition is met at $\tau = 0$, as the solution starts from the previous optimal solution. Hence, we have

$$\begin{pmatrix} H & G_{\mathbb{A}}^T \\ G_{\mathbb{A}} & 0 \end{pmatrix}\begin{pmatrix} U^*(\tau) - U^*(0) \\ -\lambda_{\mathbb{A}}^*(\tau) - \lambda_{\mathbb{A}}^*(0) \end{pmatrix} = \begin{pmatrix} g(0) - g(\tau) \\ b_{\mathbb{A}}(\tau) - b_{\mathbb{A}}(0) \end{pmatrix} \tag{7c}$$

That is,

$$\begin{pmatrix} H & G_{\mathbb{A}}^T \\ G_{\mathbb{A}} & 0 \end{pmatrix}\begin{pmatrix} \tau\Delta U^* \\ -\tau\Delta\lambda_{\mathbb{A}}^* \end{pmatrix} = \begin{pmatrix} -\Delta g \\ \Delta b_{\mathbb{A}} \end{pmatrix} \tag{7d}$$

The active set stays unchanged as long as no previously inactive constraint becomes active. That is,

$$G_j^T(U^* + \tau\Delta U^*) = b_j(x_0) + \tau\Delta b_j \tag{8}$$

for some $j \in \mathbb{I}$, and no previously active constraint becomes inactive for some $j \in \mathbb{A}$. That is,

$$\lambda_j^* + \tau\Delta\lambda_j = 0 \tag{9}$$

Correspondingly, the maximum possible step length τ_{max} is determined as follows:

$$\tau_{max}^{prim} := \min_{j\in\mathbb{I}, G_j^T\Delta U^* < \Delta b_j} \frac{b_j(x_0) - G_j^T U^*}{G_j^T\Delta U^* - \Delta b_j} \tag{10a}$$

$$\tau_{max}^{dual} := \min_{j\in\mathbb{A}, \Delta\lambda_j < 0} -\frac{(\lambda^*)_j}{\Delta\lambda_j} \tag{10b}$$

$$\tau_{max} = min\{1, \tau_{max}^{prim}, \tau_{max}^{dual}\} \tag{10c}$$

In the case that τ_max equals one, the new state x_0^new has been reached, and at the same, the solution of the new quadratic program QP(x_0^new) has been found. In other cases, a constraint needs to be relaxed or added to active set A, which limits τ_max from reaching 1. Once the active set is updated, the procedure repeats, and a new step direction and size are obtained. This iteration stops until τ_max is equal to one, indicating the solution of QP (x_0^new) has been found.

The detailed algorithm is summarized in Algorithm 1. The input is the optimal solution (U_(i-1)^*,λ_(i-1)^*) of the problem at previous control cycle QP(x_(i-1)), and x_(i-1) is the system state (i.e. problem parameter). According to the optimal solution, we can obtain the corresponding optimal active set A_(i-1)^* which is a set of the indexes of active constraints (constraints that are satisfied with equality). The output of the algorithm is the optimal solution and the corresponding active set of current control cycle.

As SOCS produces a sequence of optimal solutions for QPs along the path, it is possible to break from this sequence any time and initiate a new path from the current iteration towards the next QP.

Algorithm 1 Online State-Oriented Controller (SOCS).

Input: Solution $(U^*_{i-1}, \lambda^*_{i-1})$ of $QP(x_{i-1})$, corresponding optimal active set $\mathbb{A}^*_{i-1}$, new parameter x_i

Output: Solution (U^*_i, λ^*_i) of $QP(x_i)$, corresponding optimal active set $\mathbb{A}^*_i$, step length sequence τ.

$\tau_{max} = 0,\ x_0 = x_{i-1},\ U^* = U^*_{i-1},\ \lambda^* = \lambda^*_{i-1},\ \tau = [\,]$.
while $\tau_{max}\ != 1$ **do**
  Calculate Δx, Δg and Δb via (4).
  Calculate primal and dual step directions ΔU^* and $\Delta\lambda^*$ via (7).
  Determine maximum step length τ_{max} from (10).
  Obtain optimal solution of $QP(x_0)$:
    $U^*(\tau_{max}) = U^* + \tau_{max}\Delta U^*$
    $\lambda^*(\tau_{max}) = \lambda^* + \tau_{max}\Delta\lambda^*$
    $x_0(\tau_{max}) = x_0 + \tau_{max}\Delta x$
  if $\tau_{max} == \tau^{dual}_{max}$ **then**
    $\mathbb{A} \leftarrow \mathbb{A} \setminus \{j\}(\tau^{dual}_{max} = -\frac{(\lambda^*)_j}{\Delta\lambda^*_j})$
  else if $\tau_{max} == \tau^{prim}_{max}$ **then**
    $\mathbb{A} \leftarrow \mathbb{A} \cup \{j\}(\tau^{prim}_{max} = \frac{b_j(x_0) - G_j^T U^*}{G_j^T \Delta U^* - \Delta b_j})$
  end if
  τ.append(τ_{max}).
  Set $x_0 \leftarrow x_0(\tau_{max}), U^* \leftarrow U^*(\tau_{max}), \lambda^* \leftarrow \lambda^*_{\mathbb{A}}(\tau_{max})$
end while
$U^*_i \leftarrow U^*,\ \lambda^*_i \leftarrow \lambda^*,\ \mathbb{A}^*_i \leftarrow \mathbb{A}$.

Algorithm 2 Coordination framework of SOCS and fixed sampling strategy

Input: Solution $(U^*_{t-1}, \lambda^*_{t-1})$ of $QP(x_{t-1})$, corresponding optimal active set $\mathbb{A}^*_{t-1}$, current state x_t

Output: Solution (U^*_t, λ^*_t) of $QP(x_t)$, corresponding optimal active set $\mathbb{A}^*_t$

$\mathbb{A}^0 \leftarrow \mathbb{A}^*_{t-1}$, $U^*_{0|t-1} \leftarrow U^*_{t-1}$, $\lambda^*_{0|t-1} \leftarrow \lambda^*_{t-1}$.
for $i = 1; i \le n_{iter}; i++$ **do**
  Wait for $wait.time$ (seconds) determined by (13)
  Obtain the new state $x_{i|t-1}$
  $(U^*_{i|t-1}, \lambda^*_{i|t-1}), \mathbb{A}^i \leftarrow SOCS(U^*_{i-1|t-1}, \lambda^*_{i-1|t-1}, x_{i|t-1}, \mathbb{A}^{i-1})$
end for
$(U^*_t, \lambda^*_t) \leftarrow (U^*_{n_{iter}|t-1}, \lambda^*_{n_{iter}|t-1}),\ \mathbb{A}^*_t \leftarrow \mathbb{A}^{n_{iter}}$

3.2 Urgency-triggered Sampling Strategy

This framework allows the workload of a control cycle to distributed to several sampling intervals, and it contains the following aspects:

1) The system samples the system and collects the data.

2) The system then breaks the cycle time into a number of sampling cycles (intervals) that each lasts for wait.time.

3) During each sampling cycle (interval), computing work can be performed, and the intermediate solution, as an approximation of the final output, shall be generated at the end of every interval.

This framework allows the workload of a control cycle to distributed to several sampling intervals.

Let the execution time of the computing component be less than or equal to the sample interval length, i.e.,exec.time≤wait.time. The intermediate solution would keep to be primal and it maintains dual feasibility, as τ_max converges to 1. According to Eq. (10), the primal feasibility is given as:

$$\min_{j\in\mathbb{I}, G_j^T\Delta U^* < \Delta b_j} \frac{b_j(x_0) - G_j^T U^*}{G_j^T \Delta U^* - \Delta b_j} \le 1$$
$$\rightarrow \frac{b(x_0) - G^T U^*}{G^T \Delta U^* - \Delta b} \le 1 \quad (11)$$
$$\rightarrow G^T(U^* + \Delta U^*) \ge (b(x_0) + \Delta b)$$

and the dual feasibility is given as:

$$\min_{j\in\mathbb{A}, \Delta\lambda_j < 0} -\frac{(\lambda^*)_j}{\Delta\lambda_j} \le 1$$
$$\rightarrow -\frac{(\lambda^*)}{\Delta\lambda} \le 1 \quad (12)$$
$$\rightarrow \lambda^* + \Delta\lambda \ge 0$$

As a result, the most important part of this framework is to choose the sample interval length wait.time. One simple approach to determine wait.time would be directly dividing T_c into a fixed number of intervals. To spread the most of the workload across the idle time, one idea is to make wait.time as short as possible, and avoid a collision between the operations of sampling and computing. Actually, by setting wait.time to be

$$wait.time = \max\ exec.time \quad (13)$$

any potential conflict can be avoided. A coordinated sampling and control scheme is summarized in Algorithm 2. We first initialize the current active set $\mathbb{A}^0$ with the working set $\mathbb{A}^*_{t-1}$ obtained from the previous control cycle. Then we run n_{iter} iterations of the proposed sampling and computing cooperation strategy n_{iter} times. Here n_{iter} , the number of iterations, is a constant determined offline, $i.e.$,

$$n_{iter} = \frac{T_c}{wait.time} \quad (14)$$

At every iteration, the sensors do not need to continuously measure the system, or during every control cycle; instead they measure states x_(i|t-1) for every wait.time (seconds). After the sensor information x_(i|t-1) is obtained, the suboptimal control signal U_(i|t-1)^* is derived by running SOCS as explained in the previous section.

In the fixed sampling strategy algorithm, the length of wait.time is determined off-line. A two-dimensional example is illustrated in Fig. 5. In the applications where frequent sampling is allowed, the fixed sampling strategy is expected to significantly reduce the number of iterations to generate the optimal control signal, and thus help improve the real-time performance. However, in some cases, sampling, like perception-based sampling, can be expensive or time-consuming. In these applications, the design objective is two-fold: minimizing the sampling times, and spreading most of the workload across the idle time. As shown in Fig. 4, the regions on a state space are

geometrically irregular, indicating the active set changes may happen regularly. A fixed wait.time, a control cycle is divided into equal intervals, would fail to achieve both objectives. Rather, an adaptive sampling strategy is proposed that the sample interval is determined by

$$wait.time = f(est.load) \times T_r \tag{15}$$

where f is a strictly monotonically decreasing function of est.load, and T_r is the remaining time of the current control cycle. One can see from Algorithm 1 that in the path from current sampled state x_(i-1|t) to next one x_(i|t), SOCS will generate a step length sequence

$$\tau = \{\tau_1, \tau_2, \cdots, \tau_K\} \tag{16}$$

where τ_k,k=1,2,⋯,K, is the step size from one state region to the next. Since the number of iterations K can be considered as a workload estimation, a reasonable choice of function f would be

$$f(est.load) := \frac{\alpha}{K} \tag{17}$$

where α is the weight of f.

Algorithm 3 Coordination framework of SOCS and adaptive sampling strategy

Input: Solution $(U^*_{t-1}, \lambda^*_{t-1})$ of $QP(x_{t-1})$, corresponding optimal active set $\mathbb{A}^*_{t-1}$, current state x_t
Output: Solution (U^*_t, λ^*_t) of $QP(x_t)$, corresponding optimal active set $\mathbb{A}^*_t$

$\mathbb{A}^0 \leftarrow \mathbb{A}^*_{t-1}$, $U^*_{0|t-1} \leftarrow U^*_{t-1}$, $\lambda^*_{0|t-1} \leftarrow \lambda^*_{t-1}$.
$i \leftarrow 0$, $cum.\ time \leftarrow 0$, $wait.\ time \leftarrow 0$
while $cum.\ time < T_c$ **do**
  $i = i + 1$.
  Wait for $wait.\ time$ (seconds)
  Obtain the new state $x_{i|t-1}$
  $(U^*_{i|t-1}\lambda^*_{i|t-1}), \quad \mathbb{A}^i, \quad \tau \leftarrow SOCS(U^*_{i-1|t-1}, \lambda^*_{i-1|t-1}, x_{i|t-1}, \mathbb{A}^{i-1})$
  $cum.\ time \leftarrow cum.\ time + wait.\ time + exec.\ time$
  if $T_c - cum.\ time > \delta$ **then**
    Determine $wait.\ time$ according to (15).
  else
    $wait.\ time \leftarrow T_c - cum.\ time$
  end if
end while
$(U^*_t, \lambda^*_t) \leftarrow (U^*_{i|t-1}, \lambda^*_{i|t-1})$, $\mathbb{A}^*_t \leftarrow \mathbb{A}^i$

According to Eq. (15), if remaining time T_r is short and the estimated value of future load, $.load$, is high, it means there is an urgency to complete the computation and cut down the $wait.time$ to be short. It is this reason that this strategy is named as urgency-triggered sampling strategy, as illustrated in Fig. 6, and the algorithm is detailed in Algorithm 3. Different from Algorithm 2, $wait.time$ is adjusted online. $cum.time$ is the cumulative spending time, which is the accumulated sum of the $wait.time$ and the execution time of SOCS. If the remaining time T_r is less than a predetermined threshold of δ, the sampling interval $wait.time$ would be set to be T_r.

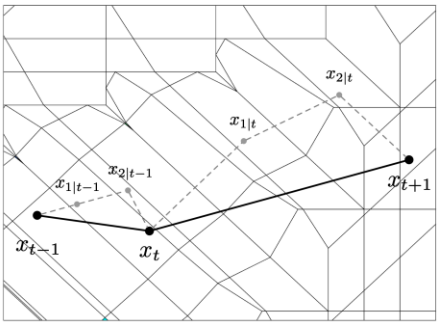

Figure 5: Paths from one QP to the next across multiple state regions. The dashed line is the path created by SOCS combined with the fixed sampling strategy, where $x_{i|t}$ denotes the i^{th} sampled state within the t^{th} control cycle. The sample interval is fixed to be $\frac{T_c}{3}$.

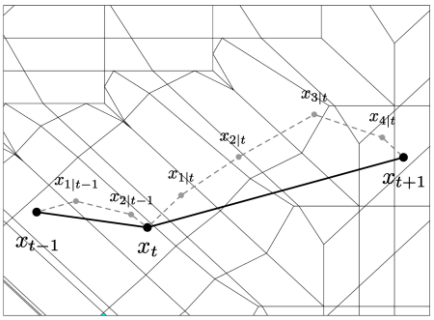

Figure 6: Paths from one QP to the next across multiple state regions. The dashed line shows the path created by applying SOCS combined with the urgency-triggered sampling strategy. Here $x_{i|t}$ denotes the i^{th} sampled state within the t^{th} control cycle.

4 SIMULATION

4.1 Experimental Setup

To evaluate our methodology on an industrial control application, we apply it to a DC servomechanism as reported in (Bemporad A. et al 1998). Details of the experimental setup are provided in Tab. 1.

Table 1. Experiment parameters

Parameters	Physical meanings	Values
T_c	Cycle time	0.1 s
T	Simulative control horizon	4 s (40 cycles)
N	Prediction horizon	0.3 s (3 cycles)
t_s	Simulation time step	0.01 s

The SOCS algorithm is implemented in Matlab using a qpOASS toolbox (Hans J F. et al 2014).

Several tests are conducted to investigate the behavior of three alternative computing strategies.

C1: traditional QP solver with an active set strategy.

C2: traditional QP solver initialized with the previous solution.

C3: state-oriented computing strategy presented in Section 3.

Next, we further combine the above computing strategies with different sampling strategies:

S1: fixed sampling strategy that all the sampling intervals are set to be the same.

S2: urgency-triggered sampling strategy, *i.e.*, sampling interval is adaptively determined by the estimation of the future load situation and the remaining time of the control cycle, as presented in Section 3.

For **S1**, the time of one control cycle is divided into equal intervals, and the duration of an interval is determined by the number of samples. According to Eq. (13), the sampling interval of **S1** should be set to the maximum execution time in the tests marked as **C1-3**, which in this case is determined to be 0.01 (seconds). Accordingly, the fixed sampling interval is 0.01 (seconds), which also means that the fixed sampling times of **S1** is 10 (times). Since **S2** requires the load information be provided in test **C3**, **S2** can only be combined with **C3**.

4.2 Simulation Results

The optimization algorithm solving the fundamental QP problems of **C1**, **C2**, **C3** is active-set method which can be easily warm-started. It is reasonable to use the number of iterations to profile the workload at different times. Since the real-time constraint applies at the end of each cycle (forced to output control signal), the focus is placed on the workload in the last sample interval. Fig. 7 depicts the number of iterations at each time step for different strategies within three randomly selected control cycles.

load unbalancing problem: Strategies **C1**, **C2** and **C3** only have one computing component, and their sampling interval spans the entire control cycle. As shown in Fig. 7, all the iterations complete in one time step, leading to a load unbalancing problem that may cause a large time delay. Although in the cases of **C2** and **C3,** warm-start from the previous solution is possible, their performance improvement is not obvious.

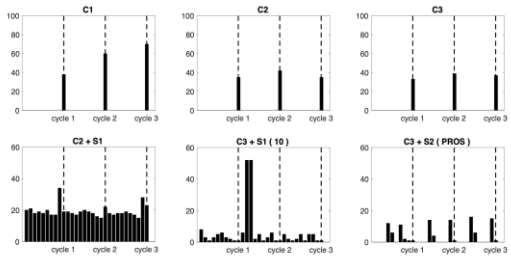

Figure 7: Number of iterations (y-axis) at each time step (x-axis) during three control cycles following different strategies. The dashed lines correspond to the last time step of each control cycle.

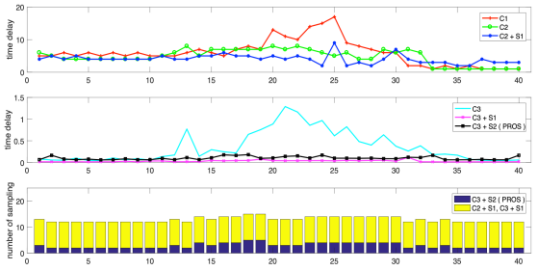

Figure 8: Number of samples and the amount of time delay (milliseconds) at each control cycle following different strategies.

load spreading: The load unbalancing problem is addressed by load spreading. To do this, computing components **C2** and **C3** are combined with different sampling strategies **S1** and **S2**. Different from simple combinations **C2+S1** and **C3+S1**, **C3+S2 (PROS)** proposed in this paper makes sampling and computing components cooperate, and adaptively control the sampling interval and computing complexity. As shown in Fig. 7, **C2+S1** cannot deliver load spreading, because its warm-start strategy is based on the previous solution which does not take advantage of the sampling strategies. In addition, due to the state-oriented warm-start strategy in **C3**, **C3+S1** and **C3+S2** are capable to spread the partial load to be completed during otherwise considered idle time. However, **C3+S1** relies on the frequent sampling, which may cause a large communication delay or add sampling complexity. This problem is further investigated in the following.

Table 2: Experiment results

Strategies	Avg. Time Delay (ms)	Avg. # Iteration	Avg. # Sampling
C1	8.76	61.92	1
C2	5.43	31.18	1
C3	0.54	28.31	1
C2+S1	5.23	21.43	10
C3+S1	0.07	2.11	10
C3+S2 (PROS)	0.19	3.87	2.78

Fig. 8 plots the time delay and the number of samplings obtained from applying different strategies over 40 control cycles. Here both **C3+S1** and **C3+S2** significantly outperform other baseline cases. Also, the performance gap between **C3+S1** and **C3+S2** is reasonably small, around 0.12 milliseconds on average. However, sampling times of **C3+S2** is much less than that of **C3+S1**, 72.25% less on average. This result confirms the proposed PROS may achieve near-

optimal performance with rather low sampling complexity.

The summary of experiment results over 40 control cycles are shown in Tab. 2. For *Avg. Time Delay* and *Avg. # Iteration*, we only evaluated the last sampling period of each control cycle. As one can see, C3 clearly outperforms other competing control algorithms in the term of average time delay. By combining with the S2, we further reduce the numbers of iterations and samplings. Therefore, C3+S2 (PROS) achieves small time delay with a reasonable sampling complexity.

5 CONCLUSION

Observing the impacts of the load unbalancing problem on the feedback control loop, we developed a cooperative framework, PROS, which coordinates the sampling and computing components of a controller to spread the load to the entire control cycle. The sampling component requests future load condition be estimated from the computing component, and it determines the sampling interval during one control cycle that is used to activate computation of an intermediate solution. In this way, the intermediate solution progressively approaches to the optimal control signal along with changes of the sampled state within a control cycle. Experimental results showed PROS outperformed the competing strategies by a wide margin. The proposed PROS can be extended to apply to the feedback control of nonlinear systems.

6 REFERENCES

A. Bemporad, M. Morari, V. Dua and E. N. Pistikopoulos, The explicit linear quadratic regulator for constrained systems. Automatica, 2002, 38(1): 3-20.

A. Sahoo, H. Xu and S. Jagannathan, Neural network-based event-triggered state feedback control of nonlinear continuous-time systems. IEEE Transactions on Neural Networks and Learning Systems, 2015, 27(3): 497-509.

A. Bemporad and E. Mosca, Fulfilling hard constraints in uncertain linear systems by reference managing. Automatica, 1998, 34(4): 451-461.

H. Li, J. Peng, J. He, R. Zhou and Z. Huang et al., A cooperative charging protocol for onboard supercapacitors of catenary-free trams. IEEE Transactions on Control Systems Technology, 2018, 26(4): 1219-1232.

H. Haimovich and E. N. Osella, On controller-driven varying-sampling-rate stabilization via lie-algebraic solvability. Nonlinear Analysis: Hybrid Systems, 2013, 7(1): 28-38.

J. F. Hans, K. Christian, P. Andreas, G. B. Hans and D. Moritz, Qpoases: a parametric active-set algorithm for quadratic programming. Mathematical Programming Computation, 2014, 6(4): 327-363.

L. Shu, B. D. Schutter, Y. G. Xi and H. Hellendoorn, Efficient network-wide model-based predictive control for urban traffic networks. Transportation Research Part C: Emerging Technologies, 2012(24): 122-140.

L. Wu, Y. Gao, J. Liu and H. Li, Event-triggered sliding mode control of stochastic systems via output feedback. Automatica, 2017(82): 79-92.

M. Miskowicz, Event-based sampling strategies in networked control systems. in Proc. WFCS, 2014: 1-10.

M. Lindberg and K. E. Arzén, Feedback control of cyber-physical systems with multi resource dependencies and model uncertainties. in *Proc. RTSS*, San Diego, USA, 2010: 85-94,.

R. Jacob, M. Zimmerling, P. Huang, J. Beutel and L. Thiele, End-to-end real-time guarantees in wireless cyber-physical systems. in *Proc. RTSS*, Porto, Portugal, 2016: 167-178.

R. Oberdieck, N. A. Diangelakis and E. N. Pistikopoulos. Explicit model predictive control: A connected-graph approach, Automatica, 2017(76): 103-112.

R. A. Borges, R. Oliveira, C. T. Abdallah and P. L. D Peres, Robust h∞ networked control for systems with uncertain sampling rates. IET Control Theory and Applications, 2010, 4(1): 50-60.

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, 2011, 3(1): 1-122.

S. Bak and P. S. D. Hylaa, A tool for computing simulation equivalent reachability for linear systems. in Proc. HSCC, 2017: 173-178.

S. Tarbouriech, A. Seuret, J. M. G. da Silva Jr and D. Sbarbaro, Observer-based event-triggered control co-design for linear systems. IET Control Theory and Applications, 2016, 10(18): 2466-2473.

V. Tzoumas, L. Carlone, G. J. Pappas and A. Jadbabaie, Control and sensing co-design. ArXiv e-prints: 1802.08376, 2018.

Y. Wang and S. Boyd, Fast model predictive control using online optimization. IEEE Transactions on Control Systems Technology, 2010, 18(2): 267-278.

Y. V. Pant, K. Mohta, H. Abbas, T. X. Nghiem and J. Deveitti et al., Co-design of anytime computation and robust control. in Proc. RTSS, 2015: 43-52.

7 DISCLOSURE STATEMENT

NO potential conflict of interest was reported by the authors.

8 NOTES ON CONTRIBUTORS

Q. M. Zou received the B.S. degree in Information Management and Information System, Anhui University of Technology, Maanshan, AH, China in July 2017, the M.S. degree in Computer Science, Harbin Institute of Technology, Harbin, HL, China in July 2019. Currently, he is a Ph.D. candidate in System Life Science, Kyushu University, Fukuoka, Japan. His research interest is optimal control and deep reinforcement learning.

L. Wang received the B.S. degree in Mathematics and the M.S. degree in Control Engineering from the Heilongjiang University, Harbin, HL, China, in 1992 and 1995, respectively, and the Ph.D. degree in Electrical Engineering from the University of Nevada, Las Vegas, NV, USA, in 2003. Currently, she is a Full Professor at Harbin Institute of Technology, Harbin, HL, China. Her research interests include AIoT, VLSI design, various aspects of computer-aided design, hardware-software codesign, high-level synthesis, and low-power system design.

Jie Liu is a Chair Professor at Harbin Institute of Technology (Shenzhen), China and the Dean of its AI Research Institute. Before that, he spent 18 years at Xerox PARC, Microsoft Research and Microsoft product teams. His research interests root in understanding and managing the physical properties of computing, such as energy, time, perception, and adaptation. He has published more than 120 peer-reviewed papers and has received 6 Best Paper Awards from top academic conferences (h-index = 62) in AI of Things, cyber-physical systems, mobile computing, and energy-efficient computing. He has filed more than 100 patents, with 60+ awarded. He has chaired a number of top-tier conferences in sensing and pervasive computing. Currently, he is the Steering Committee Chair for Cyber-Physical Systems and Internet of Things Week (CPS-IoT Week), Steering Committee Chair for ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN). He was an Associate Editor for IEEE Transactions on Mobile Computing and ACM Transactions on Sensor Networks. He is an IEEE Fellow and an ACM Distinguished Scientist.

Yingtao Jiang received the Ph.D. degree in Computer Science from University of Texas at Dallas, Richardson, TX, USA, in 2001. He joined the Department of Electrical and Computer Engineering, University of Nevada, Las Vegas, NV, USA, in August 2001. He has been a Full Professor since July 2013 at the same university. He was ECE Department Chair between 2015 and 2018, and now he is associate dean of the college of Engineering. His current research interests include VLSI circuits, wireless networks, computer architectures, machine learning, cloud computing, and nanotechnologies.