Oversampling-Based Control with Implementations

Max Nyberg Carlsson

I. Introduction

Delays in closed-loop control systems tend to be undesirable. Even if not causing instability, the system performance may suffer and behave more oscillatory. The delays may not be inherent to the controlled plant, as sampling, control signal calculation, actuation, and communication all require varying amounts of time. Controllers in particular can be limited in usefulness by the computation time, for example solving the optimization problem of a nonlinear MPC every timestep. Compared to running control software on normal microcontrollers, cloud and edge computing can open up the use of more powerful hardware which may alleviate this problem at the expense of network delays and some overhead. Networked control systems has been a popular topic for the last twenty years, and is still highly relevant with the transition to industry 4.0 [1]–[3].

The sampling period h must be larger than the worst-case latency when using a single-core processor, otherwise the control loop is prone to missing deadlines. However, if the work can be delegated to multiple computation units, such as edge offloading, multiple control loops can execute in parallel in an interlaced manner. For example, with two controllers one can use measurements from $t = 0, h, 2h, 3h, \ldots$ whereas the other samples at $t = 0.5h, 1.5h, 2.5h, 3.5h, \ldots$. Both controllers have the same period and buffer the output, such that the latency always is h, but the interval between any sample or actuation is halved and the ensemble of controllers can react to disturbances more rapidly. As the number of controllers goes to infinity, the set of controllers approaches a continuous-time controller with deadtime h. We call this *oversampling-based control*, which is different from oversampling used in digital signal processing.

The idea behind the proposed approach is to use computational power as resources to obtain better control performance by replicating an existing controller. The objective is not to design an optimal controller.

II. Oversampling

The most important assumption is that the computational resources are fully utilized by the implemented control law, i.e., parallelization of the algorithm. It is also assumed that control signals are buffered by waiting after calculations are finished, illustrated in Figure 1 with three interlaced controllers. Besides ensuring each controller is periodic, control signals will be applied in the correct order.

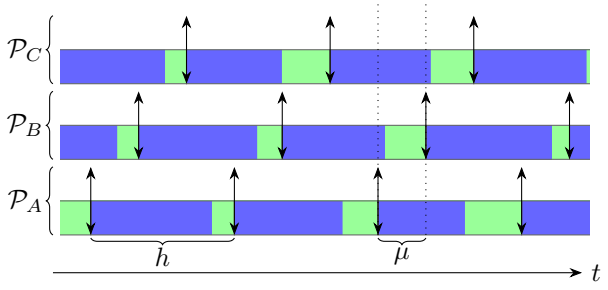

Fig. 1. Illustration of interlaced control utilizing three processors $\mathcal{P}_A$, $\mathcal{P}_B$ and $\mathcal{P}_C$. Sampling and actuation occurs simultaneously at the up respectively down arrows. Calculations are done during the blue segments, and the output is buffered until the green segments end. One specific controller interacts with the plant with period h; as there are multiple processes, the shorter period μ is how often the plant is interacted with by any of the controllers.

The basic idea is to sample faster than control signals are calculated by executing N controllers and applying their outputs in an interlaced manner. Interlacing control signals is best explained with Figure 1 where upward arrows denote sampling, and actuation is represented by downward arrows. The most recent signal is held as output by the actuator.

Let h denote the sampling interval for a single controller and $\mu = \frac{h}{N}$ the time between any sampling. The delay from a disturbing event occurring to when it is being measured is in the interval $[0, \mu]$, which shrinks to 0 as $N \to \infty$. In total, the delay from disturbance to correcting control signal is in $[h, h + \mu]$, and goes to h as $N \to \infty$, i.e., increasing N improves the worst case response time. With the delay lowered, disturbances may be suppressed more effectively. This delay can either be ignored or dealt with by estimating how the plant evolves from sampling until actuation time.

In order to execute multiple controllers in parallel and interlace their control signals, a set of computational units must be available for use as illustrated in Figure 2. This can, for example, be multiple processor cores or servers over a network. As of now, N is kept constant and dynamically changing it is left for future work.

As a side effect of distributing the calculations, there is an inherent robustness. There are still multiple backup controllers if one computation unit fails or is compromised and becomes a periodic disturbance. Investigating the effects of failing controllers is left to future work.

Utilizing Simulink [4] and the TrueTime toolbox [5], full control of simulation time is possible, allowing N to

This work was supported by: the ELLIIT Strategic Research Area, the Vinnova AORTA project (Dnr 2022-03039), and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

Email: max.nyberg_carlsson@control.lth.se

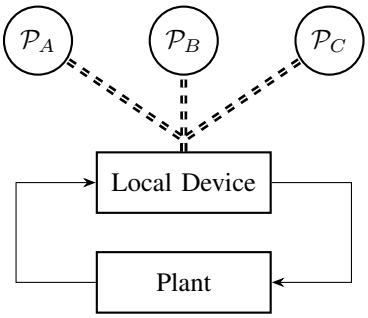

Fig. 2. The *plant* interactions are decided by the *client* that runs on a *local device*. Communicating with the client, controllers are running on $N = 3$ *processes*, here named $\mathcal{P}_A$–$\mathcal{P}_C$.

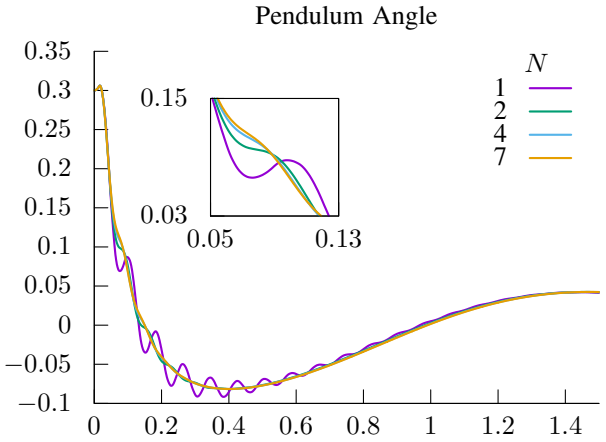

Fig. 3. The angle of a simulated Furuta pendulum, with an initial angle of 0.3 radians. The number of controllers, N, varies from one to seven. The more frequent control loop allow a more damped transient. Here $h = 0.017$ ms.

be easily changed. The effect of varying the number of controllers is demonstrated on a simulated Furuta pendulum in Figure 3. All controllers have $h = 17$ ms, and larger N makes the trajectory less oscillatory.

III. Implementation

Oversampling-based control was also used on a real plant, with a client-server architecture similar to Figure 2. A client application, running on a device physically close to the plant, communicates with N server applications, each setup to solve MPC [6] problems formulated using CVXGEN [7].

The server application receives state measurements and calculates the appropriate control signal, which is sent back to the client and gets enqueued for actuation. Containerized with Alpine Linux [8] as the base image, the server application is easily deployed and can run wherever there is an appropriate container runtime.

The client application on the local device oversees communication. It receives data from the sensors and outputs to the actuators. The client also sequentially delegates work to the server applications, and handles timings.

Server applications were deployed in three configurations: on the local device on different CPU cores, across different computers and on a private edge. The pendulum angle variances are presented in Tables I to III. Either no disturbance is applied, or periodically there is an impulse torque directly before or after sampling.

TABLE I

Angle variance when servers run locally.

	N			
	1	2	4	7
no disturbance	0.00246	0.00175	0.00139	0.00139
before sampling	0.00254	0.00188	0.00159	0.00145
after sampling	0.00256	0.00178	0.00173	0.00152

TABLE II

Angle variance when servers run on separate computers.

	N			
	1	2	4	7
no disturbance	0.00195	0.00145	0.00132	0.00125
before sampling	0.00222	0.00169	0.00154	0.00167
after sampling	0.00228	0.00192	0.00172	0.00165

TABLE III

Angle variance when servers run on the edge. Due to overruns and lost packets, the controllers run with larger h than in Tables I and II. With only one process in use, the controller fails to stabilize.

	N			
	1	2	4	7
no disturbance	†	0.00603	0.00361	0.00463
before sampling	†	0.00325	0.00245	0.00227
after sampling	†	0.00560	0.00393	0.00398

References

[1] F.-Y. Wang and D. Liu, *Networked Control Systems: Theory and Applications*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[2] P. Skarin, W. Tärneberg, K.-E. Årzén, and M. Kihl, "Control-over-the-cloud: A performance study for cloud-native, critical control systems," in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, 2020, pp. 57–66.

[3] "5G Alliance for Connected Industries and Automation." [Online]. Available: https://5g-acia.org/

[4] "Simulink version 10.2 (R2020b)," Natick, Massachusetts, United States, 2022. [Online]. Available: https://www.mathworks.com/products/simulink.html

[5] D. Henriksson, A. Cervin, and K.-E. Årzén, "Truetime: Real-time control system simulation with matlab/simulink," in *Proceedings of the Nordic MATLAB Conference*, 2003.

[6] J. Rawlings, D. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017.

[7] J. Mattingley and S. Boyd, "Cvxgen: A code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, pp. 1–27, 2012.

[8] "Alpine linux," Alpine Linux Development Team. [Online]. Available: https://www.alpinelinux.org/