Sampling and Controlling Faster than the Computational Delay

Dominic Buchstaller, Eric C. Kerrigan and George A. Constantinides*

August 31, 2011

Abstract

For a sampled-data control system, we propose to choose the time between samples to be shorter than the computational delay involved in computing the control signal, an approach we call intra-delay sampling. It is shown that, utilising a parallel computing architecture, this is indeed feasible and that intra-delay sampling schemes yield better performance than their slower sampling counterparts.

1 Introduction

One may argue that, over the last century, developments in control theory and applications have mainly been fueled by two major shifts in technology: the invention of the operational amplifier and later, the concept of digital electronics and the microprocessor. The feedback amplifier sparked classical feedback control theory, which in turn formed the foundation of robust control theory (H_∞, etc.) in the 1980s. Later, the microprocessor gave rise to discrete-time analysis, which very much simplified the implementation of control algorithms, but also allowed the construction and realisation of 'digital only' algorithms such as model predictive control (MPC) or optimisation-based multiple model control [4]. With the classical microprocessor revolution and its ever-increasing clock frequencies and power requirements coming to an end, the next major shift in technology appears to be just around the corner: from single-thread/single-core processors to (massively) parallel computational architectures, such as multi/many-core processors, graphics cards, Field Programmable Gate Arrays (FPGAs) or Application-Specific Integrated Circuits (ASICs) [7]. This shift agrees with the change in the conventional wisdom that "power is free, but transistors are expensive" to "power is expensive, but transistors are free" [2]. As with previous innovations in technology, these developments naturally give rise to new control algorithms that were previously not thought feasible.

*D. Buchstaller is with Siemens Corporate Technology, Guenther-Scharowsky-Str. 1, 91058 Erlangen, Germany, dominic.buchstaller@siemens.com, E. C. Kerrigan is with the Department of Electrical and Electronic Engineering as well as with the Department of Aeronautics at Imperial College London, Exhibition Road, SW7 2AZ, UK, e.kerrigan@imperial.ac.uk and G. A. Constantinides is with the Department of Electrical and Electronic Engineering at Imperial College London, Exhibition Road, SW7 2AZ, UK, g.constantinides@imperial.ac.uk

Consider Figure 1. Here a continuous-time plant P is controlled by a digital controller C, where the computation of the control signal requires T_c seconds to terminate. Furthermore, the plant output is sampled and the control input is implemented with a (sampling) period of T_s seconds. Typically the digital controller is then realized as follows: the plant output is sampled and the computation of the control signal is initiated. Once the computation of the control signal has terminated (after T_c seconds) the control signal is implemented. At the same time, a new sample is taken and a further computation of the control signal is initiated, as depicted in Figure 2(a). We observe that T_s is bounded by T_c from below (in fact in this case $T_c = T_s$), since one has to 'wait' for the computation to terminate before one can sample again and initiate a further computation.

If parallel hardware is available, a standard approach to speed up the computation of the control signal is to perform parallelisable operations in parallel. For example, all rows of a state feedback controller $u(t) = Kx(t)$ can be implemented in parallel. Also, the underlying dot product can be accelerated utilising parallel hardware, e.g. see [8] where the authors make exhaustive use of parallel structures to accelerate the computation of a dot product. Such obvious exploitation of parallelism within the control algorithm leads to smaller computational delays T_c and therefore allows a higher sampling frequency $1/T_s$; however, we still face the potential limitation that $T_c \leq T_s$. Hence, the smallest achievable T_c, and therefore T_s, is a function of the exploitable parallelism in the control algorithm, which may be limited.

The core message of this paper is that one can find controller implementations that allow $T_s < T_c$, i.e. one samples faster than the computational delay, where T_s is only bounded from below by the amount of available parallel resource and a constant that depends on the geometry of the system. More precisely, we will show that the sampling time T_s is a function of the number of inputs, whereas the computational delay T_c is a function of the number of states.

How this is conducted can be observed from Figure 2(b): The plant output (or a measurement thereof) is sampled and the computation of the control signal is initiated. Then, after T_s seconds, the plant output is sampled again and a new computation of the control signal is initiated before the previous computation has terminated, hence $T_s < T_c$. We term this intuitive principle intra-delay sampling. Observe that intra-delay sampling fundamentally depends on the availability of parallel computational units, as the number of diagonal lines that intersect any given vertical line in Figure 2(b) is greater than one. Also note that intra-delay sampling is a single-rate control technique and therefore distinct from multi-rate control.

By utilising the proposed technique we are able to speed up the sampling and control signal update rate; spare parallel computational resources may be transformed directly into sampling speed-ups. The expected gains from the intra-delay sampling scheme are:

1. By increasing the sampling frequency, the controller may cope with higher bandwidth disturbances.
2. It reduces the controller reaction time to disturbances, promising better disturbance rejection.
3. We expect a smoother overall control signal.

Note that increasing the sampling frequency of the intra-delay sampling scheme alone does not necessarily translate into better disturbance rejection properties. For many implementations of the algorithm the computational delay will scale with the inverse of the sampling period, leading to increasingly poor disturbance rejection properties for increasingly small sampling periods, e.g. see Section 3.1. To prove existence of implementations that achieve a decoupling of computational delay and sampling period is one of the key results of the paper.

We stress that the theoretical bounds derived here are conceptual and proof of principle only: Linear state feedback is only considered for its simplicity and to illustrate the point rather than lending itself to actual implementations. The conducted analysis is a necessary first step towards the handling of more complicated control laws that require far more computational power than simple linear controllers. This is the first control paper that clearly shows how one may be able to decouple the sampling period from the computational delay while obtaining explicit formulas for computing T_s and T_c.

The paper is organized as follows. We will introduce the principle of intra-delay sampling with the help of a concrete control algorithm (linear state feedback) and discuss two possible implementations of a controller of this type: one where the computational delay T_c is a function of the ratio $\frac{T_c}{T_s}$, i.e. the sampling speed-up, and one where T_c is a constant. While the former does not allow for a straightforward analysis, the latter allows a clear reasoning why intra-delay sampling is useful in practice. We then give bounds on achievable sampling rates and computational delays for a specific hardware architecture (a top-of-the-line FPGA). Finally, we will show by simulation that intra-delay sampling has the potential to outperform standard sampling schemes.

2 Intra-delay sampling controller design

Suppose the time required to compute the control signal, i.e. the computational delay T_c, is known; we will see in Section 3 how T_c may be determined explicitly. As a measure of the sampling speed-up define

$$h := \frac{T_c}{T_s},$$

where we assume w.l.o.g. that $h \in \mathbb{N},\ h \geq 1$. To ensure that for $h \to \infty$, $T_c \to 0$ the closed-loop system remains meaningful and tends towards its continuous-time counterpart, we employ the theory of sampled-data systems (e.g. see [3]).

Throughout this section we will analyse three different plants, depicted in Figure 3: the continuous-time plant P, the sampled-data discrete-time plant $\tilde{P}$ and the sampled-data discrete-time plant incorporating the computational delay $\tilde{P}_h$. Here z^{-h} represents a discrete-time delay of h samples, hence $hT_s = T_c$ seconds. Consequently we will design a sampled-data controller for the plant $\tilde{P}_h$, i.e. a controller that is aware of its own computational delay.

2.1 Example: LQR control

The design challenge for a sampled-data LQR controller with a computational delay of $T_c = hT_s$ seconds can be formulated as:

$$\min_{u:[hT_s,\infty)\to\mathbb{R}^m} \int_0^\infty \left[x(t)^\top Q x(t) + u(t)^\top R u(t)\right] dt \tag{1a}$$

where $u : [0, hT_s) \to \mathbb{R}^m$ is given, subject to

$$P : \dot{x}(t) = Ax(t) + Bu(t),\ x(0) = x_0 \tag{1b}$$

and the zero-order-hold constraint

$$u(t) = u(kT_s),\ \forall t \in [kT_s, (k+1)T_s),\ \forall k \in \mathbb{N}, \tag{1c}$$

where $A \in \mathbb{R}^{n\times n},\ B \in \mathbb{R}^{n\times m},\ Q \in \mathbb{R}^{n\times n}, R \in \mathbb{R}^{m\times m}$ and $Q \geq 0, R > 0$ with $(Q^{1/2}, A)$ detectable, (A, B) stabilizable. Similar design challenges can be posed, e.g. for H_∞, etc. A digital controller that solves such a control problem can be derived in a number of ways, e.g. moving the controller delay into the plant, followed by Artstein's model reduction approach [1] to obtain a plant without input delay in continuous-time, followed by discretization and controller design. Here we give an exemplar derivation that is based on state augmentation in discrete-time to deal with the delay. We would like to stress that although these derivations are more or less standard, e.g. see [3], they are nevertheless central to the argument, since they motivate the intra-delay sampling controller structure that will be exploited subsequently.

From (1b) and (1c) we have that

$$\begin{aligned} x((k+1)T_s) &= e^{AT_s}x(kT_s) + \int_{kT_s}^{(k+1)T_s} e^{A((k+1)T_s-\theta)}Bu(\theta)d\theta \\ &= e^{AT_s}x(kT_s) + \int_{kT_s}^{(k+1)T_s} e^{A((k+1)T_s-\theta)}d\theta B u(kT_s) \\ &= e^{AT_s}x(kT_s) + \int_0^{T_s} e^{A\theta}d\theta B u(kT_s). \end{aligned}$$

With $h = \frac{T_c}{Ts} \in \mathbb{N}$, let $x[k] := x(kT_s)$ and $u[k] := u(kT_s)$. This leads to the discrete-time plant without delay $\tilde{P} : x[k+1] = \Phi x[k] + \Gamma u[k]$, where $\Phi := e^{AT_s} \in \mathbb{R}^{n\times n}$ and $\Gamma := \int_0^{T_s} e^{A\theta}d\theta B \in \mathbb{R}^{n\times m}$. We now construct the discrete-time plant with input delay $\tilde{P}_h$ by augmenting the state with past control inputs. We have with $\tilde{u}[k] := u[k+h]$ and $\tilde{x}[k+1] = \begin{bmatrix} x^\top[k+1] & u^\top[k+1] & \cdots & u^\top[k+h] \end{bmatrix}^\top$ that

$$\tilde{P}_h : \tilde{x}[k+1] = \underbrace{\begin{bmatrix} \Phi & \Gamma & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{bmatrix}}_{\tilde{A}} \tilde{x}[k] + \underbrace{\begin{bmatrix} 0 \\ I \end{bmatrix}}_{\tilde{B}} \tilde{u}[k],$$

where $\tilde{x}[k] \in \mathbb{R}^{(n+hm)},\ \tilde{A} \in \mathbb{R}^{(n+hm)\times(n+hm)},\ \tilde{B} \in \mathbb{R}^{(n+hm)\times m}$. We can now use standard design tools to derive a corresponding discrete-time LQR controller that generates a control signal u that minimizes the

continuous-time cost function in (1a) (the sampled-data control problem) subject to the computational delay T_c. For that purpose we have to determine the equivalent discrete-time weighting matrices $\tilde{Q}, \tilde{R}$ and $\tilde{S}$, which we will do next. Since $u(t) = u(kT_s) \in \mathbb{R}^m,\ \forall t \in [kT_s, (k+1)T_s),\ \forall k \in \mathbb{N}$ we have with (1b) and $\hat{x} = \begin{bmatrix} x^\top(t) & u^\top(t) & u^\top(t+T_s) & \cdots & u^\top(t+hT_s) \end{bmatrix}^\top$ that

$$\dot{\hat{x}} = \underbrace{\begin{bmatrix} A & B & 0 \\ & & \\ 0 & 0 & 0 \end{bmatrix}}_{\bar{A}} \hat{x},\ \forall t \in [kT_s, (k+1)T_s),\ \forall k \in \mathbb{N}.$$

So on the interval $t \in [kT_s, (k+1)T_s)$ we have $\hat{x} = e^{\bar{A}(t-kT_s)} \begin{bmatrix} \tilde{x}^\top[k] & u^\top[k+h] \end{bmatrix}^\top$. Rewriting the continuous-time cost function in (1a) leads with $\hat{A} = \begin{bmatrix} Q & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & R \end{bmatrix}$ to

$$\begin{aligned} J &= \int_0^\infty \hat{x}^\top \hat{A} \hat{x} dt = \sum_{k=0}^{\infty} \int_{kT_s}^{(k+1)T_s} \hat{x}^\top \hat{A} \hat{x} dt \\ &= \sum_{k=0}^{\infty} \int_{kT_s}^{(k+1)T_s} \begin{bmatrix} \tilde{x}[k] \\ u[k+h] \end{bmatrix}^\top e^{\bar{A}^\top (t-kT_s)} \hat{A} e^{\bar{A}(t-kT_s)} \begin{bmatrix} \tilde{x}[k] \\ u[k+h] \end{bmatrix} dt \\ &= \sum_{k=0}^{\infty} \begin{bmatrix} \tilde{x}[k] \\ \tilde{u}[k] \end{bmatrix}^\top \begin{bmatrix} \tilde{Q} & \tilde{S}^\top \\ \tilde{S} & \tilde{R} \end{bmatrix} \begin{bmatrix} \tilde{x}[k] \\ \tilde{u}[k] \end{bmatrix}, \end{aligned}$$

where the discrete-time sampled-data weighting matrices can be computed as $\begin{bmatrix} \tilde{Q} & \tilde{S}^\top \\ \tilde{S} & \tilde{R} \end{bmatrix} = \int_0^{T_s} e^{\bar{A}^\top t} \hat{A} e^{\bar{A}t} dt$. From [5] we then have that the optimal (sampled-data) controller gain matrix $\tilde{K}$ for the augmented system is given by $\tilde{K} = (\tilde{R} + \tilde{B}^\top \Sigma \tilde{B})^{-1}(\tilde{B}^\top \Sigma \tilde{A} + \tilde{S}^\top)$, where the discrete-time algebraic Riccati equation is $\Sigma = \tilde{A}^\top \Sigma \tilde{A} - (\tilde{A}^\top \Sigma \tilde{B} + \tilde{S})(\tilde{B}^\top \Sigma \tilde{B} + \tilde{R})^{-1}(\tilde{B}^\top \Sigma A + \tilde{S}^\top) + \tilde{Q}$. The minimizing control signal can be computed as

$$C : \tilde{u}[k] = \tilde{K} \tilde{x}[k],\ \forall k \in \mathbb{N}. \tag{2}$$

Note that there are more efficient ways to derive Σ than the direct computation via the given Riccati equation, e.g. [11], however, since these computations are performed off-line, we merely note that computable solutions exist for large h.

3 Controller implementation

The intra-delay sampling controller is now implemented as follows: at every sampling time $kT_s,\ k \in \mathbb{N}$, we compute (2) and implement the corresponding control signal $u(t),\ \forall t \in [kT_s, (k+1)T_s)$. Note that since we assume that each such computation requires T_c seconds to terminate and since $T_s < T_c$ this implies that we have to initiate h parallel computations, i.e. the number of diagonal lines that intersect every given vertical line in Figure 2(b). This first resource-related challenge is therefore that the utilised

hardware architecture is required to deal with at least h parallel processes. Furthermore, since for a fixed T_c the sampling speed-up $h = \frac{T_c}{T_s}$, or the minimum number of parallel processes, is an increasing function of the sampling frequency $1/T_s$, the amount of required (parallel) computational resources scales with h.

The second challenge that we face is posed by the computation of (2) itself. The main concern here is that the computational complexity scales with the sampling speed-up h, since $\tilde{x}[k] \in \mathbb{R}^{n+hm}$, and therefore the number of operations to compute each $\tilde{K}\tilde{x}$ scales with h. This implies that, for a direct implementation of the matrix-by-vector product, the computational delay T_c is in fact an increasing function of h and not a constant, which is inconvenient and potentially problematic. We will now discuss these issues in detail.

3.1 A direct implementation

Although today's General Purpose Processors (GPPs) usually incorporate a small number of cores, other architectures provide parallelism on a much larger scale. For example, top of the line graphics cards implement hundreds of parallel processing units. FPGAs or custom-built ASICs also allow the implementation of hundreds or thousands of parallel processing units. Future technologies will push these boundaries even further. To be able to argue about as many as possible of these very different technologies, we will consider two at opposite ends of the spectrum: a custom implementation of the controller, as would be found in FPGAs and ASICs, and an implementation of the controller on single-core/single-thread GPPs, where we will assume the availability of a large number of them.

The first challenge, that the number of parallel computations of the control signal is a function of h, is easily met by the properties of massively parallel architectures. The second challenge, that the complexity of the matrix-by-vector product grows with h, cannot be met in such a straight-forward fashion. A direct implementation of $\tilde{K}\tilde{x}[k]$ has the property that the computational delay T_c is an increasing function of h, even if the dot product is implemented by exhaustive use of parallel structures. To see this, observe that the computation of $\tilde{K}\tilde{x}[k]$ in (2) reduces to the computation of a simple matrix-by-vector product. Also observe that since we can write $\tilde{u}[k] = \tilde{K}\tilde{x}[k] = \begin{bmatrix}(\tilde{K}^1\tilde{x}[k])^\top & (\tilde{K}^2\tilde{x}[k])^\top & \cdots & (\tilde{K}^m\tilde{x}[k])^\top\end{bmatrix}^\top$ where each $K^q,\ q \in \{1, 2, ..., m\}$, is a row of $\tilde{K}$, we can compute the matrix-by-vector product $\tilde{K}\tilde{x}$ by computing m dot products in parallel. One possible implementation of a dot product is given in Figure 4, where every box represents a real, physical implementation of either a scalar multiplication or a scalar addition. We can then compute $\tilde{K}\tilde{x}$ by letting $v := \tilde{K}^q,\ q \in \{1, 2, ..., m\},\ w := \tilde{x}(kT_s),\ l := n + mh$ in Figure 4.

Custom implementation:

A typical custom hardware implementation of the dot product would utilize parallel hardware to compute every column in Figure 4. This is usually achieved by employing a standard technique in digital hardware design called 'pipelining' [6].

A pipelined implementation of the dot product can be visualized as in Figure 5, where each block represents the parallel hardware that implements one column of Figure 4. Observe that for $v := \tilde{K}^q, w := \tilde{x}(kT_s),\ q \in \{1, 2, ..., m\},\ l := (n + mh)$ the design in Figure 5 allows us to initiate a new computation of the control signal $\tilde{u}[k] = \tilde{K}\tilde{x}[k]$ every T_s seconds: new vectors are fed into the pipeline from the left, while previous computations are still filtering through to the right. This characteristic is precisely in line with the computational challenge posed by the intra-delay sampling scheme (see Figure 2(b)). The total delay for the computation of $v^\top w = \tilde{K}^q\tilde{x}$ is then given by

$$T_c = T_m + T_a\lceil \log_2(n + hm)\rceil \tag{3}$$

where the ceiling function $\lceil a\rceil := \min\{b \in \mathbb{Z} \mid b \geq a\},\ a \in \mathbb{R}$ and T_m and T_a are the times that the hardware requires to compute a scalar multiplication and addition, respectively. The computational delay T_c includes a $\log_2$ since the number of required addition stages scales logarithmically with the number of inputs, whereas the ceiling function ensures that the number of addition stages is an integer. Note that since we need to implement m such pipelines in parallel to compute $\tilde{K}\tilde{x}(kT_s)$, the utilisation, i.e. the required number of computational units, is given by

$$U = m\left[(n + hm)\text{MUL} + (n + hm - 1)\text{ADD}\right], \tag{4}$$

where MUL and ADD represents a single scalar multiplier and adder, respectively. Pipelining has the effect that the sampling time T_s is now bounded from below by the slowest element in the pipeline, i.e. in our case $T_s \geq \max\{T_m, T_a\}$. However, we note that deeper pipelining or a fully parallel implementation of the dot product allows a further reduction of T_s. For example, for a fully parallel implementation, T_s can be chosen as small as we can physically sample $\tilde{x}$, typically in the order of the clock-period of the device. In the interest of simplicity, we only consider one likely implementation: the scalar product is pipelined, adders and multipliers are not pipelined, hence $T_s \geq \max\{T_m, T_a\}$.

GPP implementation:

For single-core/single-thread GPPs the exploitation of parallelism on the level demonstrated for the custom implementation is usually not feasible, since inter-GPP communication delays are likely to negate the expected speed-up. This is one typical issue of multi-core GPP processors which often complicates a straightforward parallel implementation of algorithms. To take this effect into account, we will only compute all $\tilde{K}^q\tilde{x},\ q \in \{1, 2, ..., m\}$, in parallel utilizing multiple GPPs, whereas the dot products for $v := \tilde{K}^q, w := \tilde{x}(kT_s),\ l := (n + mh)$ in Figure 4 are computed sequentially. This reduces the inter-GPP communication considerably while posing a reasonably large atomic problem to every GPP. Please note that there naturally exist infinitely many possible GPP implementations where we only consider one likely one. The computational delay for such an implementation is given by

$$T_c = (n + hm)T_m + (n + hm - 1)T_a. \tag{5}$$

Assuming the availability of a sufficient number of parallel GPPs we can initiate a new computation whenever we like. Therefore, T_s can be chosen as small as we can physically sample $\tilde{x}$. The utilisation is then given by

$$U = hm\text{GPP}, \tag{6}$$

where GPP represents a computational unit that is able to perform scalar multiplications and additions.

Note that in (3) and (5) T_c is an increasing function of h. Although the intra-delay sampling controller may still show better disturbance rejection properties to the standard sampling controller, e.g. when the system dynamics are slow, the dependence of T_c on h complicates the analysis significantly. For example, we may substitute hT_s for T_c in (3) and (5). Together with the lower bounds on T_s, this gives the set of allowable pairs (h, T_s) as functions of T_m and T_a. We would then have to analyse if there exists allowable pairs $(h,\ T_s)$ that promise better (disturbance rejection) properties of the intra-delay sampling controller over the standard sampling controller ($T_c = T_s$ and $h = 1$). We will now highlight the relationship between intra-delay sampling and IIR filtering and give a controller implementation that decouples T_c and T_s.

3.2 A filter implementation of the intra-delay sampling controller where T_c is independent of h

We observed in the previous section that the computational delay T_c for a direct implementation of $\tilde{K}\tilde{x}$ is a function of h, since at every sampling time kT_s we have to compute a matrix-by-vector product for which the computational delay T_c scales with h. Now observe that we can rewrite (2) as follows. For $k \in \mathbb{N}$:

$$u[k+h] = \tilde{u}[k] = \tilde{K}\tilde{x}[k] = Kx[k] + \sum_{i=1}^{h} L_i u[k+h-i], \tag{7}$$

where $\tilde{K} =: \begin{bmatrix} K & L_1 & L_2 & \cdots & L_h \end{bmatrix} \in \mathbb{R}^{m\times(n+hm)}$. Replacing k with $k-h$ throughout leads to

$$u[k] = Kx[k-h] + \sum_{i=1}^{h} L_i u[k-i]. \tag{8}$$

Note that the controller described by (8) has the structure of an Infinite Impulse Response (IIR) filter. Furthermore, (8) with $u[k] = u(kT_s)$ leads to:

$$u(kT_s) = Kx(kT_s - hT_s) + \sum_{i=1}^{h} L_i u(kT_s - iT_s). \tag{9}$$

The key observation now is that, in order to compute $u(kT_s),\ k \in \mathbb{N}$, it is not necessary to evaluate the complete sum in (9) at once, but one can compute every step i whenever a new u becomes available. This implies that at every sampling time kT_s we initiate a new computation of the type Lu, as depicted in Figure 6 for the case where $h = 2$. We denote the time these computations require to terminate T_l. We then define the smallest achievable sampling time $T_s := T_l + 2T_a$.

In parallel to the computation of the sum, we also initiate the computation of Kx. Finally, when the computation of the sum is complete, we add the result of the computation of Kx. This gives u and prevents a build-up of addition latencies. Note that we will assume $n > m$ throughout, which is often the case. For $n \leq m$, although possible implementations exist, the computational timing will be different. Please note that we also assumed w.l.o.g. that T_s is an integer multiple of T_c, i.e. $h \in \mathbb{N}$.

One possible implementation of the algorithm is now given in Figure 7, where $\times$ represents an implementation of the dot product and $+$ represents a scalar addition. This is a so-called transpose implementation of the IIR filter, which functions as follows: All computations of the type Lu are initiated synchronously, where the required delay is implemented at the output, i.e. the result from the computation $L_h u[k]$ needs to propagate through h delay stages where the summation is performed along the way. Finally, when all summations of the type Lu are complete, the result from the earlier initiated computation Kx is added. This completes the construction of the control signal.

Custom implementation: The lowest achievable sampling time for a custom implementation of the IIR filter in Figure 7 with pipelined dot products is given by

$$T_s \geq T_l + 2T_a = T_m + T_a\lceil \log_2 m \rceil + 2T_a. \tag{10}$$

The computational delay for the proposed implementation is given by

$$T_c \geq T_m + T_a\lceil \log_2 n \rceil + T_a. \tag{11}$$

We note that the sampling time T_s is a function of the input dimension m, whereas the computational delay T_c is a function of the state dimension n. The maximum achievable sampling speed-up is therefore given by

$$h \leq \left\lceil \frac{T_m + T_a\lceil \log_2 n \rceil + T_a}{T_m + T_a\lceil \log_2 m \rceil + 2T_a} \right\rceil. \tag{12}$$

The ceiling function is a result of the possibility that initially $\frac{T_c}{T_s} \notin \mathbb{N}$, where we then simply delay T_c sufficiently long such that $\frac{T_c}{T_s} \in \mathbb{N}$, i.e. we let $T_c = hT_s$ where T_s is the lower bound from (10) and h is the largest $h \in \mathbb{N}$ that satisfies (12).

From Figure 2(b) we have that the intra-delay sampling scheme requires that a new control input is available every T_s seconds. Since we assume that all dot products are pipelined, the implementation in Figure 7 has this property. However, recall that Figure 7 only computes one row of u, hence we require m parallel implementations. This leads for a pipelined implementation of the dot product to the utilization

$$U \quad = \quad m\left[(hm+n)\text{MUL} + (hm+n)\text{ADD}\right] \tag{13}$$

where registers are assumed to require a negligible amount of resources.

GPP implementation: As before, the algorithm would be implemented differently on single-core/single-thread GPPs due to the specifics of the hardware. In this case we would compute all dot products

sequentially on a single GPP. The minimum achievable sampling time is then given by

$$T_s \geq T_m m + T_a(m-1) + 2T_a \tag{14}$$

and the computation delay is given by

$$T_c \geq T_m n + T_a(n-1) + T_a, \tag{15}$$

hence the maximum achievable sampling speed-up is given by

$$h \leq \left\lceil \frac{T_m n + T_a(n-1) + T_a}{T_m m + T_a(m-1) + 2T_a} \right\rceil. \tag{16}$$

For the utilisation we have that each computation of the type Lu requires one GPP, where the computation of the type Kx requires h GPPs, e.g. for $h = 2$ every vertical line in Figure 6 intersects h blocks of the type Lu and h blocks of the type Kx. The utilization is therefore given by

$$U = mh(n+m)\text{GPP}. \tag{17}$$

We can now observe that the lower bound on the computational delay T_c, as given in (11) and (15) is in fact independent of h and only depends on the number of states n, whereas the lowest achievable sampling time T_s depends on the number of inputs m. However, note that the achievable sampling speed-up h is not determined by T_c and T_s alone. The sampling speed-up is also bounded from above by the availability of parallel resources, i.e. h in (13) and (17) is bounded from above by the availability of adders and multipliers or GPPs, respectively. However, assuming a sufficient amount of parallel resources, the above analysis shows that the implementation of intra-delay sampling, i.e. $T_s < T_c$, is feasible. We will discuss the implications of analytical and resource constraints in the following section with the help of a concrete example.

4 Example: Intra-delay sampling on an FPGA

A top of the line FPGA (Xilinx Virtex 5, XC5VTX240T) contains approximately $L = 150 \times 10^3$ Look-Up Tables (LUTs) [10]. LUTs are the basic building blocks in FPGAs and represent the limiting factor in terms of implementability of a design. Running at a clock frequency of $f = 110$MHz, a minimum latency implementation of a scalar single-precision floating point adder requires approximately $U_a = 0.4 \times 10^3$ LUTs and a multiplier $U_m = 0.7 \times 10^3$ LUTs. The time to perform a single precision floating point addition and multiplication are given by $T_m \approx T_a \approx 2/f = 18.1ns$ [9]. For simplicity assume that $m = 1$.

We then have from (12) that the achievable sampling speed-up h is bounded from above by

$$\left\lceil \frac{T_m + T_a\lceil \log_2 n \rceil + T_a}{T_m + T_a\lceil \log_2 m \rceil + 2T_a} \right\rceil = \left\lceil \frac{2 + \lceil \log_2 n \rceil}{3} \right\rceil. \tag{18}$$

Furthermore, from (13) and the fact that the required resource U for a particular design is bounded from above by the available resource L, i.e. $U = mh\left[(m+n)U_m + (m+n)U_a\right] = mh(m+n)(U_m + U_a) \leq L$,

we obtain a further hardware-specific constraint on the sampling speed-up h as a function of the state dimension n. Both these upper bounds are now plotted in Figure 8, where we observe that the largest implementable speed-up $h = 3$, for $L = 150 \times 10^3$ Look-Up Tables, is achieved for a system with up to $n = 132$ states.

We then have from (10) and (11) that for $n = 132$ the smallest sampling time is given by $T_s = T_m + 2T_a = 54.3$ns and the computational delay is given by $T_c = hT_s = 217.2\text{ns} \geq T_m + T_a\lceil \log_2(n) \rceil + T_a = 181$ns.

Note that T_c is a function of the number of states of n and the clock frequency f. For example, in low power applications it may be necessary to reduce f significantly. This will lead to a larger T_c and will make the intra-delay sampling scheme interesting for applications with slower dynamics. We also expect the availability of much larger devices in the future. For example, the next version of Xilinx FPGAs will incorporate approximately $L = 500 \times 10^3$ Look-Up Tables (Virtex 6, XC6VLX760) [10]. By Figure 8, larger devices will allow the implementation of larger speed-ups for larger systems in the future.

Finally, note that this simple example has been chosen to demonstrate the fundamental mechanics and (resource) constraints of the intra-delay sampling principle. Sampling speed-ups may be very different for other implementations or algorithms. For example, see (16) where, for a GPP implementation of the intra-delay sampling controller with $T_a = T_m$ and $m = 1$, the upper bound on h scales linearly with n, giving rise to much larger speed-ups.

5 Example: Performance of the closed-loop system

In this section we discuss the closed-loop behaviour of the intra-delay sampling state feedback controller with the help of a simple example. We stress that such simple control problems will in most cases not require a intra-delay sampling implementation of the linear state feedback controller since standard sampling schemes are more than sufficient to provide reasonable sampling rates, even on very weak hardware. The purpose of the example is rather to highlight two key points: 1) the effect of the system geometry on the achievable sampling speed-ups and 2) better disturbance rejection properties if implemented on identical hardware (with identical computational delays).

Let P be given by $P : \dot{x}(t) = Ax(t) + B(u(t) + w(t)),\ \forall t \geq 0$, where

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -0.196 & 0.016 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -0.054 & 2.7 & 0 \end{bmatrix},\ B = \begin{bmatrix} 0 \\ 1.96 \\ 0 \\ 0.54 \end{bmatrix},\ x = \begin{bmatrix} \bar{x} \\ \dot{\bar{x}} \\ \phi \\ \dot{\phi} \end{bmatrix}$$

and w is defined below. This is the model of an inverted pendulum on a cart, where $\bar{x}$ is the position of the cart, ϕ is the angle between the pendulum and the vertical and the control objective is to apply a force $u(t)$ to the cart in order to balance the pendulum in an upright position, i.e. $\phi \to 0$.

The intra-delay sampling, sampled-data state feedback controller for the following simulations has been constructed as in Section 2.1 with continuous-time weights $Q = \begin{bmatrix} I & 0 \\ 0 & 20I \end{bmatrix}$, $R = 60$ in (1a). For simplicity, assume that $T_m = T_a$. We then have with $m = 1$ and $n = 4$ for a GPP implementation of the intra-delay sampling controller from (16) that $h = \left\lceil \frac{T_m n + T_a(n-1) + T_a}{T_m m + T_a(m-1) + 2T_a} \right\rceil = \left\lceil \frac{4+3+1}{1+2} \right\rceil = 3$, hence we can sample three times faster than the computational delay T_c. Assume that for a given architecture $T_c = 0.20$s. This gives us a minimum sampling time of $T_s = \frac{T_c}{h} = 0.0667$s. To evaluate the disturbance rejection properties of the plant let $w(t) = \begin{cases} 0 & 0 \leq t < 1.0001 \\ 20 & 1.0001 \leq t < 1.0401 \\ 0 & 1.0401 \leq t \end{cases}$, $\forall t \geq 0$, which is a force disturbance of 20N acting on the cart for a duration of 0.04s. In Figure 9(a) we plotted the system response (the angle ϕ) for three different controllers: a standard sampling controller with $h = 1$, a intra-delay sampling controller with $h = 3$ and the continuous-time LQR controller without delay ($T_c = 0$). Observe that the intra-delay sampling controller with $h = 3$ is able to suppress the disturbance with a 23% smaller response in comparison to the standard sampling controller with $h = 1$. This behaviour is intuitively quite acceptable: in the standard sampling scheme ($h = 1$), a sample is only taken every 0.20s, where the computational delay is also 0.20s, i.e. $T_c = T_s = 0.20$s. Since the disturbance is entering at $t = 1.0001$s, a counteracting control signal will not be applied until $t = 1 + T_s + T_c = 1.4$s, as depicted in Figure 9(b). In contrast, the intra-delay sampling controller for $h = 3$ has a sampling time of $T_s = 0.0667$s and a computation delay of $T_c = 0.20$s. It will therefore be able to react at time $t = 1 + T_s + T_c = 1.2667\text{s} < 1.4\text{s}$. Hence, intra-delay sampling effectively reduces the (maximum) total time to react to a disturbance, i.e. $T_c + T_s$, and therefore promises better performance in the presence of disturbances. In fact, the disturbance $w(t)$ has been chosen to illustrate this behaviour.

The total maximum time to react to a disturbance $T_c + T_s$ is now plotted in Figure 10 for different sampling speed-ups h. Note that for increasing h, $T_s + T_c$ must tend towards T_c since $T_s = \frac{T_c}{h} \to 0$ for $h \to \infty$ and T_c is constant. In the limit, the corresponding closed-loop system then consists of a continuous-time plant with a pure input delay T_c, and a corresponding continuous-time controller. The second curve in Figure 10 is the root mean square error of the output ϕ (closed-loop simulated for 15s) where now w is a Gaussian process with mean $\mu = 0$, variance $\sigma^2 = 1$ and a sample period of 0.001s. We observe the curves are qualitatively very similar. This shows that the maximum time to react $T_c + T_s$ provides a good indication for the disturbance rejection properties of the closed-loop system, where the intra-delay sampling controller performs increasingly well for increasingly large sampling speed-ups h. Finally, we observe that, apart from where the disturbance forces discontinuities, the control signal in Figure 9(b) is smoother for $h > 1$ than for $h = 1$, reducing the burden on the actuation hardware.

6 Conclusion

The shift in technology, away from single-core/single-thread processors to massively parallel architectures, opens up the opportunity for new control techniques that were previously not thought feasible.

In this paper we present a novel digital control technique, which we call intra-delay sampling, that enables a digital controller to sample faster than the computational delay involved in computing the control signal. By using a parallel computing infrastructure, we analytically showed 1) that intra-delay sampling is indeed feasible from an implementation point of view and that 2) the sampling time T_s is a function of the number of inputs, whereas the computational delay T_c is a function of the number of states. Furthermore, we demonstrated the advantages over standard sampling techniques in terms of disturbance rejection and highlighted the basic trade-offs that come with the approach. Although we used specific examples and controller architectures to illustrate the point, the principle of intra-delay sampling applies in a more general context, i.e. it is potentially applicable to other (digital) control techniques than the state feedback LQR case considered here. Especially computationally intensive algorithms, such as model predictive and multiple model control, may benefit from intra-delay sampling type schemes. The derivation of suitable intra-delay sampling implementations of these algorithms, however, will require significantly more research.

Although we did not consider actual hardware implementations of the analysed intra-delay sampling sate feed back controller in this paper, we can expect similar qualitative results in practice since the resulting controller structure (an IIR filter) is well studied and there exist efficient hardware implementations (at least on FPGAs) that preserve the discussed timing-properties.

6.1 Acknowledgements

The authors would like to thank Juan Jerez Fullana for asking the right questions that ultimately lead to the concept of intra-delay sampling. This work was funded by the EPSRC under grant number EP/G031576/1, with support from Xilinx, the European Space Agency (ESA) and Mathworks inc.

References

[1] Z Artstein. Linear systems with delayed controls: A reduction. *IEEE Transactions on Automatic Control*, 27(4):868–879, 1982.

[2] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Kreuzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. Williams, and K. A. Yelick. The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.

[3] K. J. Åström and B. Wittenmark. *Computer-Controlled Systems: Theory and Design*. Prentice Hall, 3rd edition, 1997.

[4] D. Buchstaller. *Robust Stability and Performance for Multiple Model Switched Adaptive Control*. PhD thesis, University of Southampton, School of Electronics and Computer Science (ECS), 2010.

[5] P. Dorado and A. Levis. Optimal linear regulators: The discrete-time case. *IEEE Transactions on Automatic Control*, 16(6):613–620, 1971.

[6] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 6th edition, 2006.

[7] ITRS. Semiconductor Industry Association (SIA), International technology roadmap for semiconductors (ITRS), http://www.itrs.net, 2009.

[8] A. R. Lopes and G. A. Constantinides. A fused hybrid floating-point and fixed-point dot-product for FPGAs. In *Proc. International Symposium on Applied Reconfigurable Computing, Bangkok, Thailand*, pages 157–168, 2010.

[9] Xilinx. Xilinx, inc., *Core Generator*, version 9.0.2, http://www.xilinx.com., 2010.

[10] Xilinx. Xilinx, inc., Virtex FPGA Family, http://www.xilinx.com/products/devices.htm, 2010.

[11] H. Zhang, G. Duan, and L. Xie. Linear quadratic regulation for linear time-varying systems with multiple input delays. *Automatica*, 42(9):1465–1476, 2006.

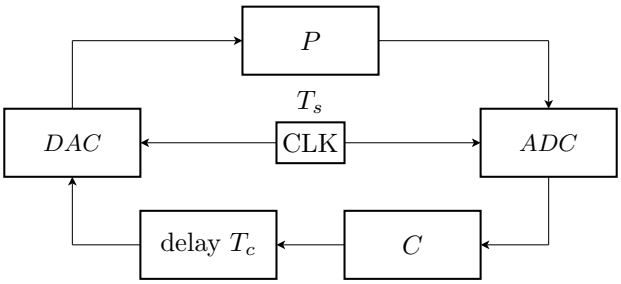
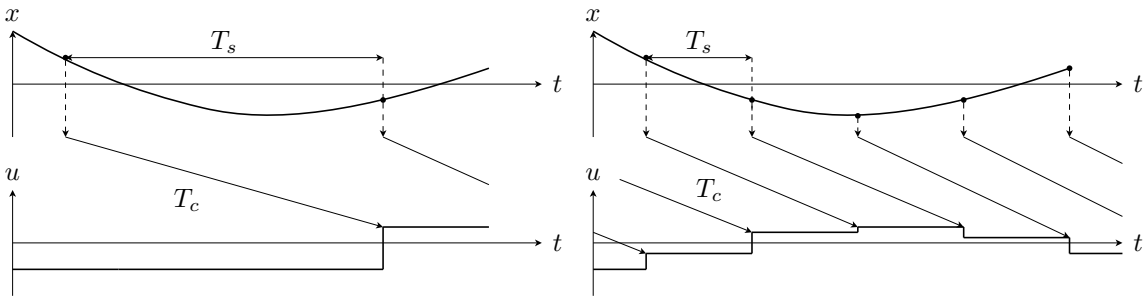


Figure 1: Sampled-data control loop, computational delay T_c, Digital to Analog Converter (DAC), Analog to Digital Converter (ADC).



(a) Standard sampling

(b) Intra-delay sampling

Figure 2: Illustration of standard and intra-delay sampling: sampling time T_s, computational delay T_c.

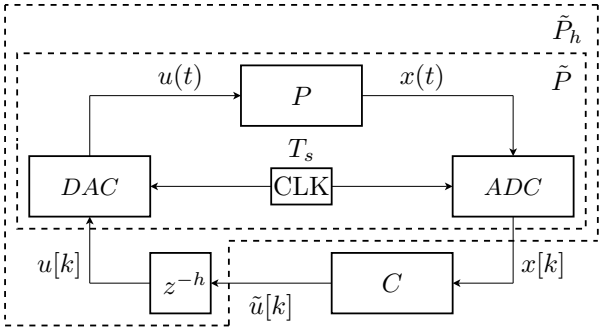


Figure 3: Controller output delay vs. plant input delay.

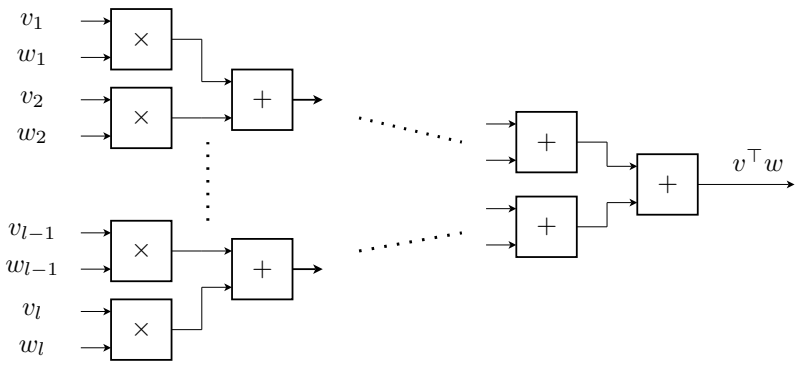


Figure 4: An implementation of the dot product between vectors $v, w \in \mathbb{R}^l$.

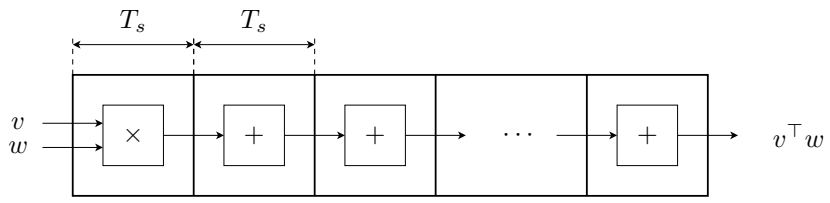


Figure 5: Parallel computation of the dot product in a pipeline.

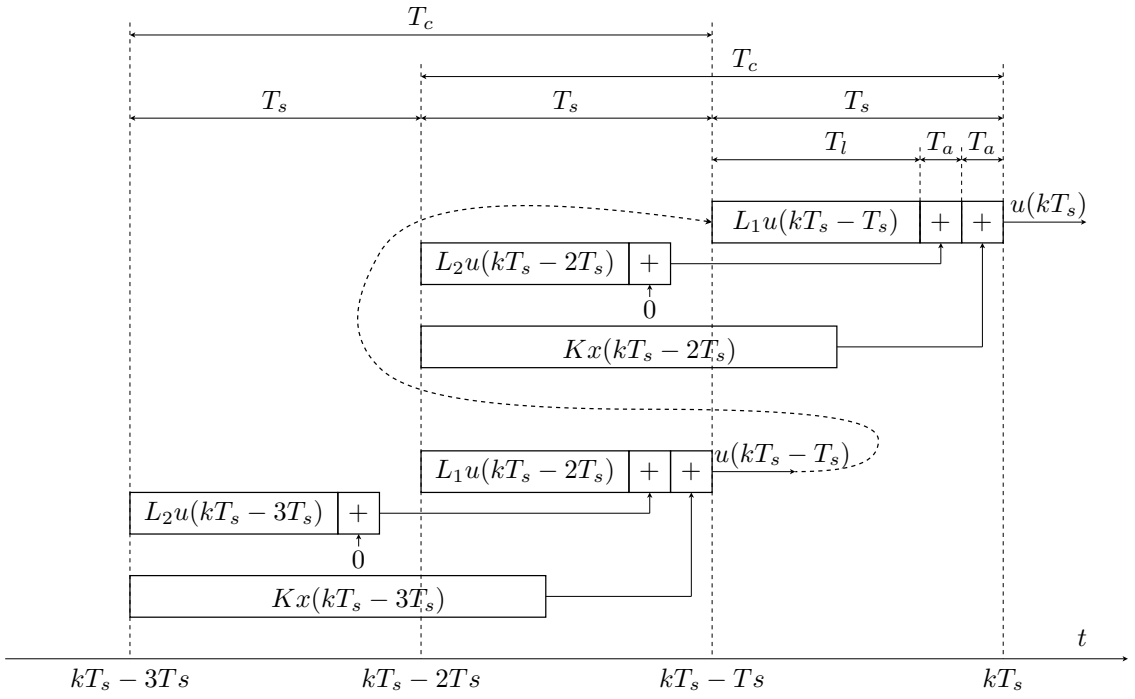


Figure 6: Timing of the computation of the control signal for $h = 2$, $n > m$.

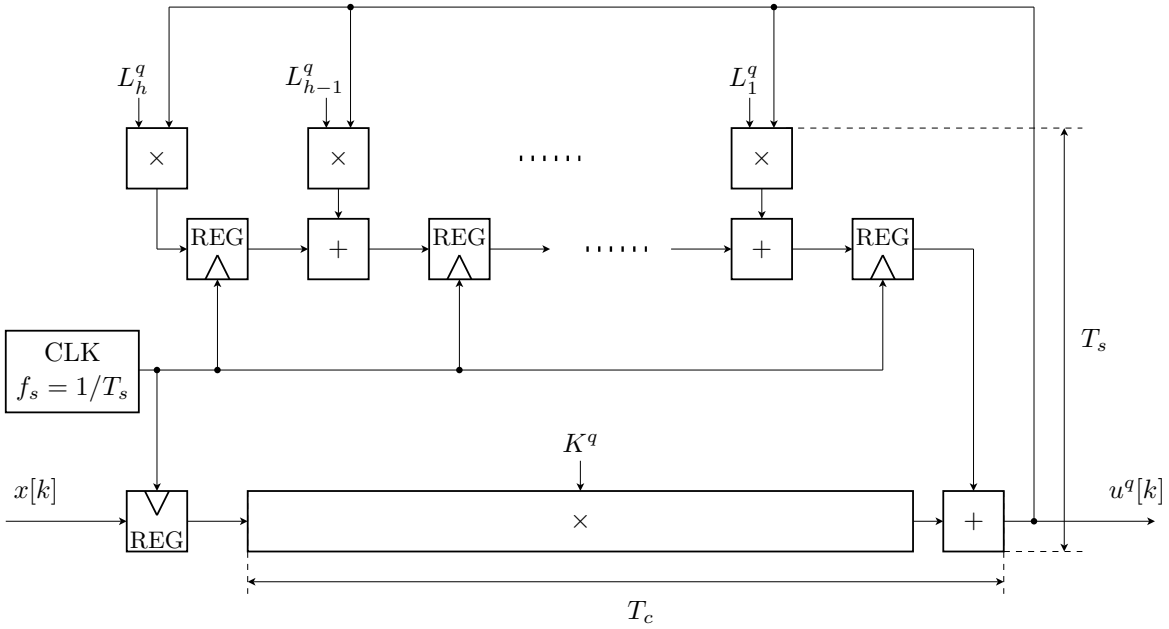


Figure 7: Transpose IIR implementation of (8).

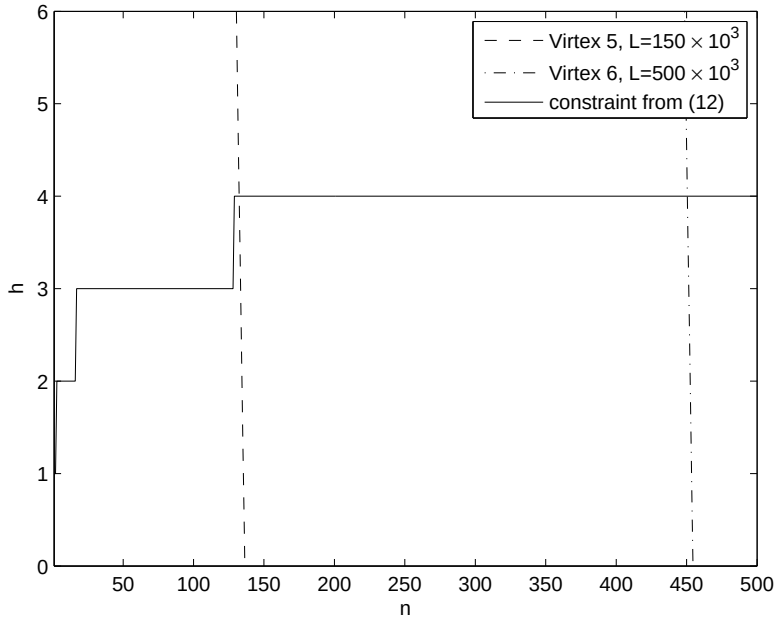
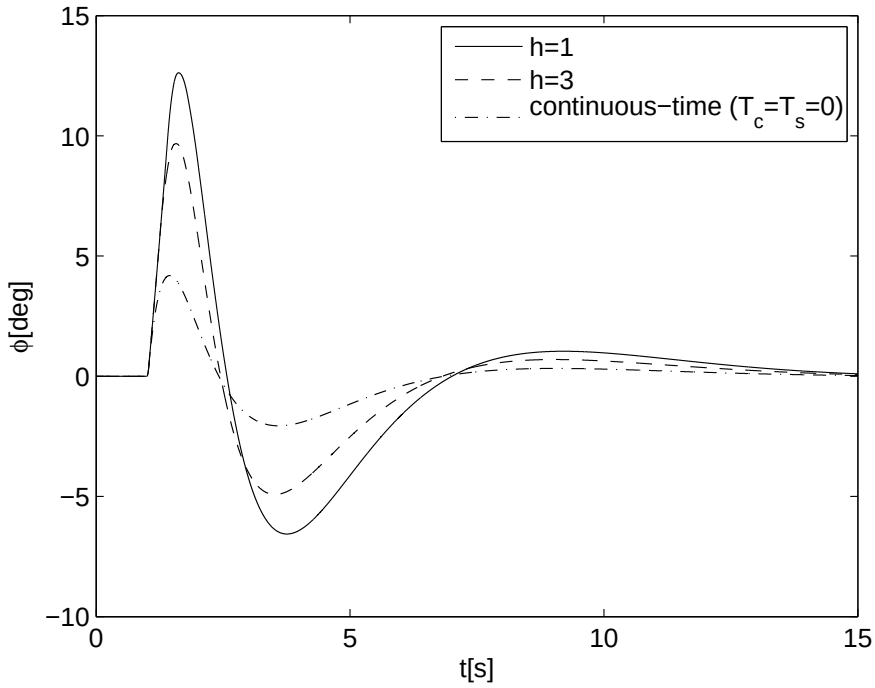
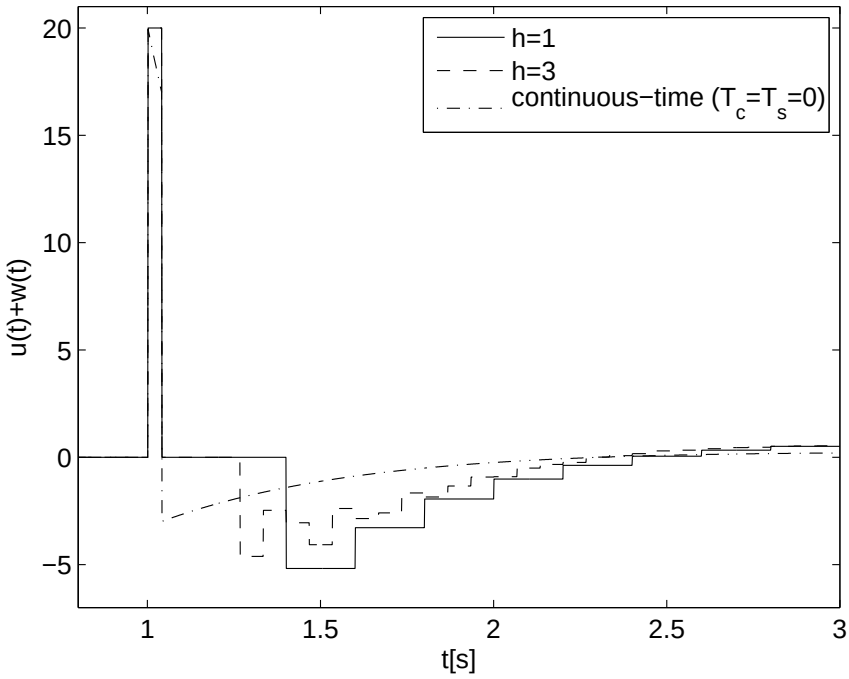


Figure 8: Achievable sampling speed-ups given analytical and resource constraints for $m = 1$.



(a) Response of P to a disturbance at $t = 1.0001$s.



(b) Control input to P for a disturbance at $t = 1.0001s$.

Figure 9: Disturbance rejection, $T_c = 0.20$s

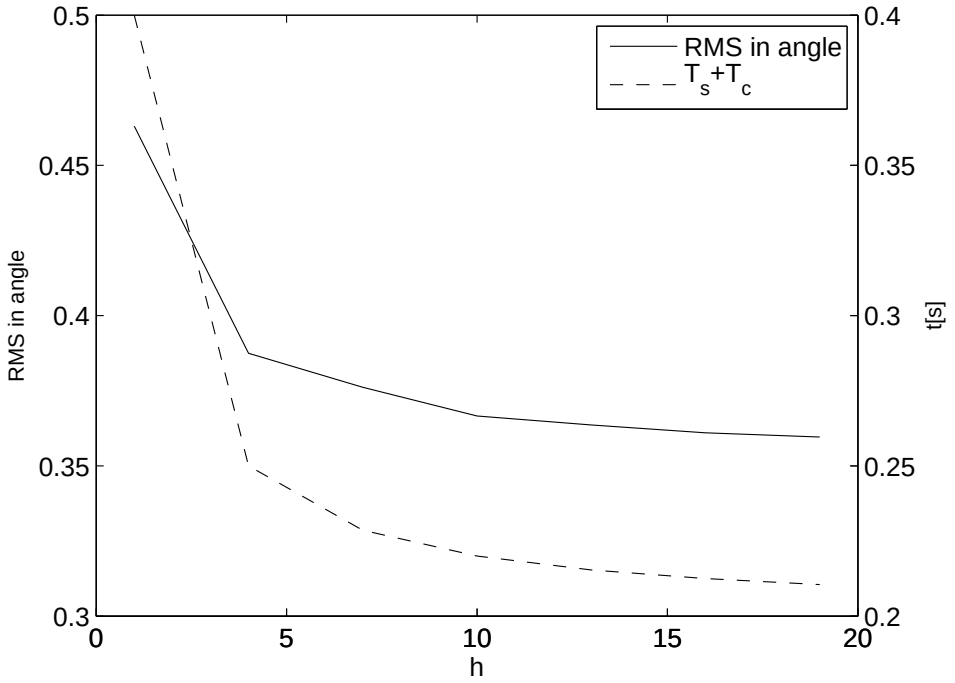


Figure 10: RMS error in angle and total time to react ($T_s + T_c$) to a random disturbance.