Sampling and Controlling Faster than the Computational Delay

Dominic Buchstaller ∗ Eric C. Kerrigan ∗∗
George A. Constantinides ∗∗∗

Abstract: For a sampled-data control system, the sampling period is chosen smaller than the computational delay, an approach we call intra-delay sampling. Utilising parallel computing architectures, it is shown that intra-delay sampling schemes are feasible and that they yield better performance than their slower sampling counterparts.

Keywords: Sampled-data control, digital control, implementation.

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 ($H_\infty$, 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 (see Buchstaller (2010)).

With the classical microprocessor revolution and its ever-increasing clock frequencies coming to an end, the next major shift in technology appears to be just around the corner: from single-thread/single-core general purpose processors (GPPs) to (massively) parallel computational architectures, such as multi/many-core GPPs, graphics cards, Field Programmable Gate Arrays (FPGAs) or Application-Specific Integrated Circuits (ASICs) (see ITRS (2009)) that give rise to new control algorithms previously not thought feasible.

Consider Figure 1. Here a continuous-time plant $P$ is controlled by a digital controller $C$ with a sampling period $T_s$, where the computation of the control signal requires $T_c$ seconds to terminate. Typically, the plant output is sampled, the computation of the control signal is initiated and once this computation has terminated 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 $T_s = T_c$), since one has to ‘wait’ for the computation to terminate before one can sample again and initiate a further computation.

A standard approach to speed up the computation 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 software can be accelerated utilising parallel hardware, e.g. see Lopes and Constantinides (2010); however, we still face the potential limitation that $T_s \leq T_c$. Hence, the smallest achievable $T_s$, and therefore $T_c$, 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 $T_s < T_c$ where $T_s$ bounded from below by the amount of available parallel resource and a constant that depends on the geometry.

* 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.
of the system. How this is conducted can be observed from Figure 2(b): The plant output is sampled and the computation of the control signal is initiated. After \( T_s \) seconds, the plant output is sampled again and a new computation 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. Note that intra-delay sampling is a single-rate control technique and therefore distinct from multi-rate control.

Intra-delay sampling draws its main motivation from the fact that for modern control algorithms, such as MPC, the computational delay \( T_c \) is often too large. Even if the available parallelism is exploited in full, \( T_c \) may still not be sufficiently small to meet the (real-time) constraints of the application. 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.

The paper is organized as follows. We will introduce the principle of intra-delay sampling utilising a concrete control algorithm (LQR state feedback controller) and discuss two possible implementations of a controller of this type: one where the computational delay \( T_c \) is a function of the sampling speed-up \( \frac{T_s}{T_c} \) and one where \( T_c \) is a constant. While the former complicates a straightforward analysis, the latter allows a clear reasoning why intra-delay sampling is useful in practice. 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 \( T_c \) is known; we will see in Section 3 that for the considered implementations \( T_c \) may be determined explicitly. As a measure of the sampling speed-up define
\[
    h := \frac{T_s}{T_c}, \quad \text{where we assume w.l.o.g. that } h \in \mathbb{N}, \ h \geq 1.
\]
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 Feuer and Goodwin (1996)).

We will now 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:

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

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

\[
    \dot{x}(t) = Ax(t) + Bu(t), \ x(0) = x_0 \quad (1b)
\]

and the zero-order-hold constraint

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

where \( A \in \mathbb{R}^{n \times n}, \ B \in \mathbb{R}^{n \times m}, \ Q \in \mathbb{R}^{m \times m}, \ R \in \mathbb{R}^{m \times m} \) and \( Q \succeq 0, \ R > 0 \) with \( (Q^{1/2}, A) \) detectable, \( (A, B) \) stabilizable. Similar design challenges can be posed, e.g. for \( H_{\infty} \), etc. Here we give an exemplar derivation. Although these derivations are standard (see Feuer and Goodwin (1996) and the references therein), they are nevertheless central to the structural arguments that follow.

Let \( h = \frac{T_s}{T_c} \in \mathbb{N}, \ x[k] := x(kT_s) \) and \( u[k] := u(kT_s) \). From (1b) and (1c) we have that

\[
    \tilde{P} : x[k + 1] = \Phi x[k] + \Gamma u[k], \quad \text{where } \Phi := e^{AT_c} \in \mathbb{R}^{n \times n} \text{ and } \Gamma := \int_0^{T_s} e^{A\theta} dB \in \mathbb{R}^{n \times m},
\]
By augmenting the state with past control inputs we have \( \tilde{u}[k] := u[k + h] \) that \( \tilde{P}_h \) is defined by

\[
\begin{bmatrix}
    x[k + 1] \\
v[k + 1] \\
\vdots \\
v[k + h]
\end{bmatrix} =
\begin{bmatrix}
    \Phi & 0 & 0 & \cdots & 0 \\
    0 & I & 0 & \cdots & 0 \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    0 & 0 & 0 & \cdots & I \\
0 & 0 & 0 & \cdots & 0 \\
0 & 0 & 0 & \cdots & 0 \\
\end{bmatrix}
\begin{bmatrix}
    x[k] \\
v[k] \\
\vdots \\
v[k + h - 1] \\
\end{bmatrix} + \begin{bmatrix}
    \tilde{u}[k] \\
\end{bmatrix},
\]

where \( \tilde{x}[k] \in \mathbb{R}^{(n + mh)} \), \( \tilde{\mathbf{A}} \in \mathbb{R}^{(n + mh) \times (n + mh)} \), \( \tilde{\mathbf{B}} \in \mathbb{R}^{(n + mh) \times m} \). The equivalent discrete-time sampled-data weighting matrices \( \tilde{Q}, \tilde{R}, \tilde{S} \) can be determined as

\[
\begin{bmatrix}
    \tilde{Q} \\
    \tilde{S} \\
    \tilde{R}
\end{bmatrix} =
\begin{bmatrix}
    \int_0^{T_s} e^{A^T_t A e^{A^t_t}} dt \\
    \mathbf{A}^{T} \mathbf{A} \\
    \mathbf{A}^{T} \mathbf{B} \\
\end{bmatrix}
\]

where \( \tilde{T}_s \) is a scalar step size. From e.g. Dorado and Levis (1971), we then have that the optimal (sampled-data) control signal gain matrix \( \tilde{K} \) for the augmented system is given by:

\[
\tilde{K} = \begin{bmatrix}
    \tilde{R} + \tilde{B}^T \Sigma \tilde{B} - (\tilde{A}^T \Sigma \tilde{B} + \tilde{B}) \tilde{R} - \tilde{B} \tilde{A} \tilde{S} \tilde{A}^T + \tilde{Q} \end{bmatrix},
\]

where the discrete-time algebraic Riccati equation is \( \Sigma = \tilde{A}^T \Sigma \tilde{A} - (\tilde{A}^T \Sigma \tilde{B} + \tilde{B}) \tilde{S} \tilde{A}^T \tilde{B} + \tilde{R} \). Note that there are more efficient ways to derive \( \Sigma \), e.g. see Zhang et al. (2006), however, since these computations are performed off-line, we merely note that solutions exist for large \( h \). The minimizing control signal can be computed as

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

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 \( u(k) \), \( \forall k \in [kT_s, (k + 1)T_s) \). Note that we assume that each such computation requires \( T_s \) seconds to terminate and that \( T_s < T_c \), hence we have to initiate \( h \) parallel computations, i.e. the number of diagonal lines that intersect each given vertical line in Figure 2(b). The first challenge is therefore that the hardware architecture has to deal with at least \( h \) parallel processes. Furthermore, since for \( T_c \) fixed, \( h = \frac{T_c}{T_s} \), or the minimum number of parallel processes, is an increasing function of \( 1/T_c \), the amount of required 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 \( \tilde{x}[k] \in \mathbb{R}^{n + mh} \), 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

The first challenge is easily met by the properties of massively parallel architectures. The second challenge 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 \( \tilde{u}[k] = \tilde{K} \tilde{x}[k] \) with \( \tilde{K} = (\tilde{K}^1 \tilde{x}[k])^T \cdot (\tilde{K}^2 \tilde{x}[k])^T \cdots (\tilde{K}^m \tilde{x}[k])^T \) where each \( \tilde{K}^i \), \( q \in \{1, 2, ..., m\}, w := \tilde{x}(kT_s), l := n + mh \) 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’ (e.g. see Hennessy and Patterson (2006)). A pipelined implementation of the dot product can be visualized 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 \( u[k] = \tilde{K} \tilde{x}[k] \) every \( T_c \) seconds: new vectors are fed into the pipeline from the left, while previous computations are still filtering through to the right. The total delay for the computation of \( v^Tw = \tilde{K}^q \tilde{x} \) is then given by

\[
T_c = T_m + T_a \log_2(n + mh),
\]

where \( [a] := \min \{b \in \mathbb{Z} \mid b \geq a\} \), \( a \in \mathbb{R} \) and \( T_m \) and \( T_a \) are the times that the hardware requires to compute a scalar multiplication and addition, respectively. 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

\[
U = m [(n + mh)MUL + (n + mh - 1)ADD],
\]

Fig. 4. An implementation of the dot product.
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. for a pipelined implementation $T_s \geq \max\{T_m, T_a\}$.

We note that deeper pipelining or a fully parallel implementation of the dot product allows a further reduction of $T_s$. 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. We will therefore only compute all $K^q x$, $q \in \{1, 2, ..., m\}$ in parallel utilizing multiple GPPs, whereas the dot products for $v := K^q x, w := \bar{x}(K^q a)$ in Figure 4 are computed sequentially. We then have

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

Assuming a sufficient number of parallel GPPs we can initiate a new computation whenever we like. Therefore, $T_s$ can be chosen arbitrarily small, i.e. $T_s \geq 0$. The utilisation is then given by

$$U = hm\text{GPP},$$  \hspace{1cm} (6)

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

In either case, $T_s$ 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 $T_s$ on $h$ complicates the analysis significantly. We give a controller implementation that decouples $T_s$ and $T_a$ next.

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

We observed in Section 3.1 that the computational delay $T_c$ for a direct implementation of $K^q x$ is a function of $h$, since at every sampling time $kT_s$ we have to compute a matrix-by-vector product for which $T_c$ scales with $h$. Now observe that we can rewrite (2) as follows. For $k \in \mathbb{N}$:

$$u[k + h] = \bar{u}[k] = \bar{K} \bar{x}[k] = K x[k] + \sum_{i=1}^{h} L_i u[k + h - i],$$  \hspace{1cm} (7)

where $\bar{K} := [K L_1 L_2 \ldots L_h] \in \mathbb{R}^{m \times (n + hm)}$. Replacing $k$ with $k + h$ throughout this leads to

$$u[k] = K x[k + h] + \sum_{i=1}^{h} L_i u[k - i].$$  \hspace{1cm} (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:

$$u(kT_s) = K x(kT_s + hT_a) + \sum_{i=1}^{h} L_i u(kT_s - iT_s).$$  \hspace{1cm} (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) in one go, 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 $T_i$. The achievable sampling time is therefore bounded from below by $T_s \geq T_i + 2T_a$.

In parallel to the computation of the sum, we also initiate the computation of $K^q x$. Finally, when the computation of the sum is complete, we add the result of the computation of $K^q x$. This gives $u$ and prevents a build-up of addition latencies. Note that, as in Figure 6, we will assume $n > m$ throughout, which is usually 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 $T_a$ 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 $L_i u[k]$ needs to propagate through $h$ delay stages, implemented as edge-triggered registers (REG), 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

$$T_s \geq T_i + 2T_a = T_m + T_a \lceil \log_2 m \rceil + 2T_a$$  \hspace{1cm} (10)

whereas the computational delay is given by

$$T_c \geq T_m + T_a \lceil \log_2 m \rceil + T_a.$$  \hspace{1cm} (11)

We note that $T_s$ is a function of the input dimension $m$, whereas $T_c$ is a function of the state dimension $n$. The maximum achievable sampling speed-up is therefore given by $h \leq \left[ T_s + T_a \lceil \log_2 m \rceil + 2T_a \right] / [T_m + T_a \lceil \log_2 m \rceil + 2T_a].$ 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

$$U = m \left[ (hm + n)\text{MUL} + (hm + n)\text{ADD} \right]$$  \hspace{1cm} (12)

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

### GPP implementation:

Typically, we would compute all dot products sequentially on a single GPP. The minimum achievable sampling time is then bounded by

$$T_s \geq T_m + T_a \lceil \log_2 m \rceil + 2T_a$$  \hspace{1cm} (13)

and the computational delay is bounded by

$$T_c \geq T_m + T_a \lceil \log_2 m \rceil + 2T_a.$$  \hspace{1cm} (14)

hence the maximum achievable sampling speed-up is bounded by

7526
\[ T_s \left( kTs - T_s \right) + u(kTs) \]

\[ T_s \left( kTs - 2T_s \right) + u(kTs - T_s) \]

\[ T_s Kx \left( kTs - 2T_s \right) \]

\[ kTs - 2T_s kTs - T_s kTs \]

\[ u(kTs) \]

\[ kTs - 3T \]

\[ T_c \]

Fig. 6. Timing of the computation of the control signal for \( h = 2, n > m \).

\[ h \leq \left\lfloor \frac{T_m n + T_a(n - 1) + T_a}{T_m m + T_a(m - 1) + 2T_a} \right\rfloor. \] (15)

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

\[ U = mh(n + m)GPP. \] (16)

Observe that the lower bound on the computational delay \( T_c \) of this implementation, as given in (11) and (14) is in fact independent of \( h \) and only depends on the geometry of the system and the latency \( T_a \); the bound on \( T_c \) depends on the number of states \( n \), whereas the lowest achievable sampling time \( T_s \) depends on the number of inputs \( m \). 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.

4. EXAMPLE

Let \( P \) be given by

\[ \dot{x}(t) = Ax(t) + B(u(t) + w(t)), \quad \forall t \geq 0 \]

where

\[
\begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & -0.196 & 0.016 & 0 \\
0 & 0 & 0 & 1 \\
0 & -0.054 & 2.7 & 0 \\
\end{bmatrix}
\]

and \( w \) is defined below. This is the model of an inverted pendulum on a cart, where \( \dot{x} \) is the position of the cart, \( \phi \) is the angle between the pendulum and the vertical and the control objective is to construct \( u(t) \) such that \( \phi \to 0 \).

The intra-delay sampling, sampled-data state feedback controller is constructed as in Section 2.1 with continuous-
control signal will not be applied until \( t = 1 + T_a + T_c = 1.4s \), as depicted in Figure 8(b). In contrast, for the intra-delay sampling controller with \( h = 3 \) we have \( T_s = 0.0667s \) and \( T_c = 0.20s \). It will therefore be able to react at time \( t = 1 + T_s + T_c = 1.2667s < 1.4s \). 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 mechanism. For random disturbances it can equally be shown that higher intra-delay sampling speed-ups lead to smaller transients.

5. CONCLUSION

In this paper we present a novel digital control technique, termed intra-delay sampling, that enables a digital controller to sample faster than the computational delay. By using a parallel computing resources, we analytically showed 1) that intra-delay sampling is feasible and 2) the sampling time \( T_c \) is a function of the number of inputs, whereas the computational delay \( T_s \) is a function of the number of states. Furthermore, we demonstrated superior disturbance rejection properties over standard sampling techniques. 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 considered state feedback LQR case. Although we did not consider actual hardware implementations 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.

Future work could include a rigorous robustness analysis of the proposed intra-delay sampling scheme.

REFERENCES


