Any Time Control[1]

Daniele Fontanelli, Luca Greco
Interdepartmental Research Center "E. Piaggio", Pisa University

March 31, 2006

[1] This research has been partially supported by EC contract HYCON and RUNES

Chapter 1

Any Time Control

It is widely accepted in the control community that the Computer Controlled Systems (briefly, CCSs), have a lot of advantages based primarily on the high reconfigurability of the controller platform and the ability of making complex yet fast decisions. Such positive features make the CCS a useful platform for multitasking control, i.e. using only one single-processor platform to control several plants. Within this framework, the computer has to share its computational time to solve several critical tasks, each one with its priority. Taking into account schedulability and real time operating systems problems, it is just a step forward to realize that tasks with low priority could be interrupted *any time*, with unpredictable distributions, and it is just a logical intuition that not all the tasks can be at the highest priority.

¿From a control point of view, each control task should compute a control input to the controlled system to prevent instability or to ensure the performances. This field seems to be at the border between the control community and the computer science community and it has not been widely investigated. Some work has been carried out by [1] introducing more general scheduling models and methods for control systems, where control design methodology takes the availability of computing resources into account and allows the trade–offs between control performance and computer resources utilization, and also introducing the idea of any time CCS without solutions. In [2], the Control Server is introduced, allowing the separation between scheduling and control design by treating the controllers as scalable real–time components. Jitter and latency is included in the model and cannot be taken over. Furthermore, the interrupt time, the I/O operations and the overrun handling are not taken into account, simply imposing only soft real time control tasks. Useful tools for the analysis of real time control performance are the *Jitterbug* ([3]) and the *True Time* ([4]) that analyze the control performance once the control tasks are implemented as soft real time tasks.

1.1 Basic Idea

The underlying idea of the *Any Time Control* paradigm is the ability to improve the computation accuracy as the available time increases. This concept has been widely studied in literature, both from computer scientists [5], introducing the concept of *imprecise computation*, and from filter design theory [6], improving the desired filter result as the number of polynomial terms is increased.

In what follows, each discrete time controller, implementable on a generic digital machine, is described and computed on its state space representation. Improving the computation accuracy as the available computation time increases implies the definition of a “scalable” controller. Albeit the concept of imprecise computation can be moved to the *imprecise control* and although a necessary minimum control calculation burden can be fixed as the minimum computational time guaranteeing the stability control, a fundamental question is what should be considered as a correct “accuracy improvement” in the control computation. Assuming that a given target, discrete controller $K(z)$ can be decomposed into a set of discrete controllers $K_i(z)$, with $i = 1, \dots, N$, the accuracy improvement can be addressed in two ways:

Model reduction. The approximation index for the accuracy improvement is the closed loop behavior as a whole. The discrete target controller $K(z)$ is approximated using model reductions:

- *Balanced reduction*: the state space model is reduced by weakly observable or controllable modes cancellations;
- *Modal reduction*: the state space model is reduced separating the modes set by their relative velocities;
- *Generic reduction*: the state space model is reduced following some generic reduction index, for instance the terms of the Partial Fraction Expansion.

The model reduction approach has two main drawbacks: its meaningful interpretation is in open loop, while the closed loop effects or the performance satisfaction are not so evident; the minimum controller obtained with this technique could be computationally (and dimensionally) more complex than the relative minimum controller obtained with a performance reduction.

Performance reduction. The approximation index for the accuracy improvement is the performance. The set of performances $\hat{P}$, that the target controller $K(z)$ satisfies, are separated into $N-1$ parts

$$\hat{P} = \left\{\hat{P}_1, \hat{P}_2, \dots, \hat{P}_{N-1}\right\},$$

that are incrementally implemented by each controller. $K_1(z)$ satisfies the most important requisite: the stability. For each $i = 2, \dots, N-1$, if $K_i(z)$ satisfies the performances $\{\hat{P}_1, \dots, \hat{P}_{i-1}\}$, the controller $K_{i+1}(z)$ satisfies $\{\hat{P}_1, \dots, \hat{P}_i\}$. It is straightforward that $K_N(z)$ satisfies the entire set $\hat{P}$, such that $K_N(z) = K(z)$. The most important feature of this architecture is that each controller is designed as a stand-alone controller and the controllers are related each other only by the performances. In fact, a target controller is not strictly necessary, remembering that only the definition of the set of performances $\hat{P}$ is needed.

Furthermore, the performance guided model reduction can be further extended adding a "scalability" constraint. If the controller $K_i(z)$ ensures the performances $\{\hat{P}_1, \dots, \hat{P}_{i-1}\}$ satisfaction and has the set of poles p_i and zeros z_i, it is possible to construct a controller $K_{i+1}(z)$ that satisfies the performances $\{\hat{P}_1, \dots, \hat{P}_i\}$ and has a set of poles $p_j = \{\bar{p}_j, p_i\}$ and zeros $z_j = \{\bar{z}_j, z_i\}$. In some sense that will be clearer in the subsequent sections, if the additional scalability constraint is satisfied, the controller $K_{i+1}(z)$ has to compute only the additional singularities effects, i.e. $\bar{p}_j$ and $\bar{z}_j$. However, there is a trade–off between the performance and the scalability constraint, both with respect to the controller design and the state space implementation. Nevertheless, in general applications, $K_N(z) \neq K(z)$ with the dimension of $K_N(z)$ greater than $K(z)$ (i.e. with a larger number of poles).

In order to explain the two main approaches, consider the continuous time transfer function of an unstable, non–minimum phase plant

$$G(s) = \frac{s-2}{(s+10)(s+5)(s+2)(s-1)}$$

and he resulting discrete transfer function, sampled with a frequency of $100Hz$

$$G(z) = \frac{1.5935\ 10^{-7}(z+3.568)(z-1.02)(z+0.2561)}{(z-1.01)(z-0.9802)(z-0.9512)(z-0.9048)}.$$

Using a systematic control approach to stabilize the system and to ensure an approximate first order step response (i.e.LQR design method) the controller

$$K(z) = \frac{-1136.1395(z-0.9827)(z-0.9509)(z-0.905)}{(z-0.7408)(z-0.7047)(z^2-1.719z+0.7408)}, \tag{1.1}$$

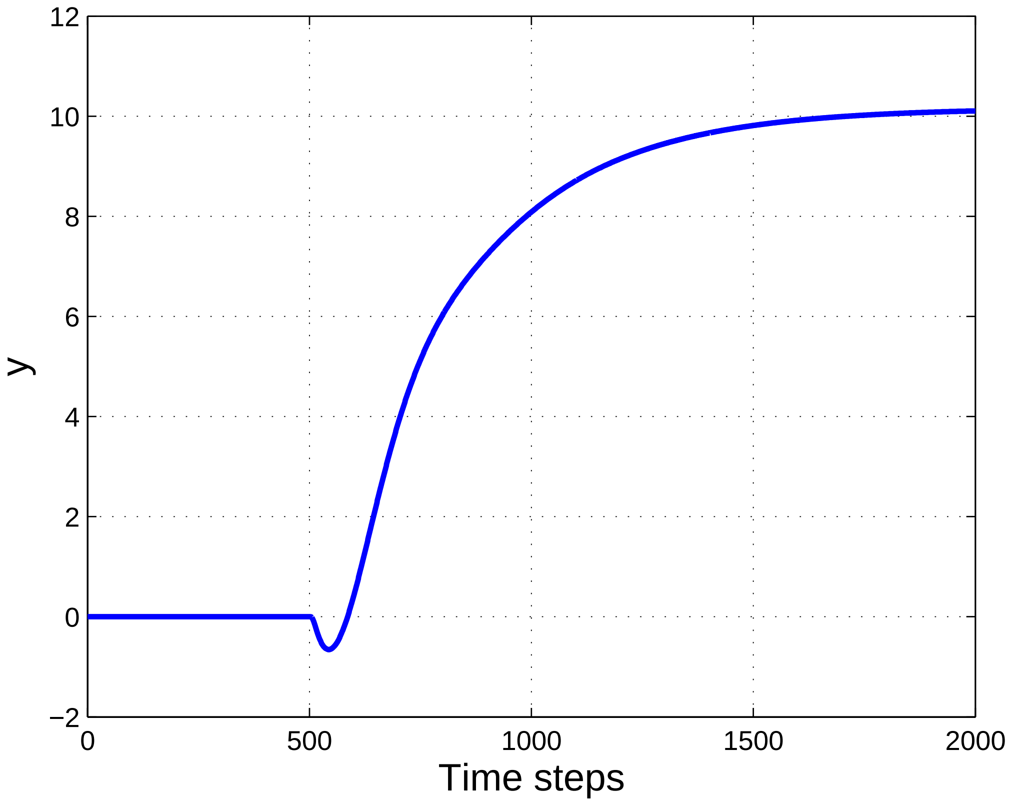


Figure 1.1: Step response of the system $G(z)$ closed in loop with the controller $K(z)$.

is determined (see figure 1.1).

A balanced state space representation of the controller $K(z)$ is

$$A_k = \begin{bmatrix} 0.4328 & 0.2974 & 0.1114 & 0.01632 \\ -0.2974 & 0.936 & -0.0354 & -0.004747 \\ 0.1114 & 0.0354 & 0.904 & -0.02812 \\ -0.01632 & -0.004747 & 0.02812 & 0.892 \end{bmatrix}, \; B_k = \begin{bmatrix} 33.88 \\ -4.615 \\ -3.108 \\ 0.3606 \end{bmatrix},$$

$$C_k = \begin{bmatrix} -33.88 & -4.615 & 3.108 & 0.3606 \end{bmatrix}, \; D_k = 0.$$

for which the state space $x = [x_1, x_2, x_3, x_4]^T$ is organized in descendent order of controllability and observability, i.e. x_4 is the weakly controllable and observable mode of the system.

Applying a reduction by suppressing x_4, amounts in removing the last column and row of A_k and related elements in the B_k and C_k arrays. This way, the second order step response, reported in figure 1.2, is obtained. The system has poor performances, but is still stable.

As shown by this simple SISO example, the model reduction acts on the system performances in an unpredictable way, because of the closed loop behavior. Moreover, if a further reduction is performed by removing the variable x_3, the resulting closed loop system turns out to be unstable. Therefore, the minimum controller order, using the balanced model reduction, is three.

To accomplish a modal reduction, the system is firstly transformed in Jordan form and then a modal removal is attempted. Unfortunately, the proposed example does not permit a modal reduction and the resulting closed loop system is unstable regardless of what x_i is removed.

It is evident that, while adding or removing stable poles has not impact on the open loop stability, this is not true for closed loop stability. This explain why minimum controllers produced by model

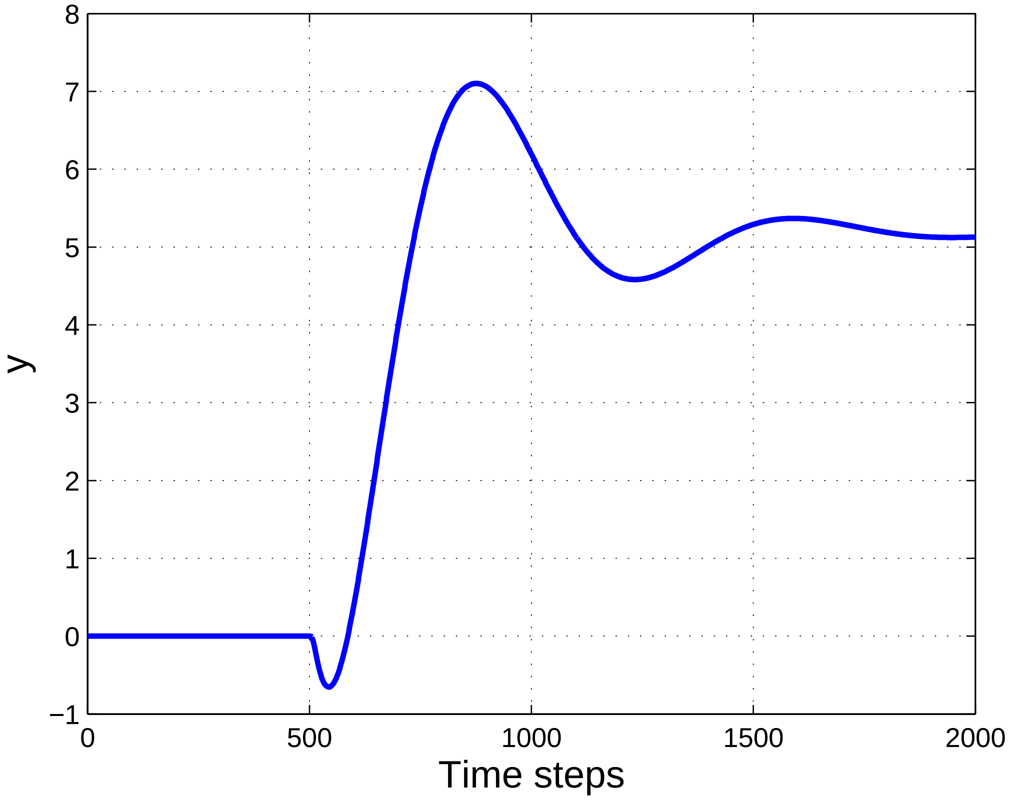


Figure 1.2: Step response of the system $G(z)$ closed in loop with the reduced controller obtained by suppressing the x_4 state space variable.

reduction may have large dimension.

Consider now the obvious requirement of stability and the additional performance $\hat{P}_1$ of first order step response. A minimum controller (of order one)

$$K_1(z) = -900\frac{z - 0.9827}{z - 0.7408}$$

can ensure the stability (see figure 1.3).

Let $C_i(z)$ be a generic controller and let $C_1(z) = K_1(z)$. Let now the controller

$$C_2(z) = 1.2624\frac{(z - 0.9509)(z - 0.905)}{(z - 0.7047)(z^2 - 1.719z + 0.7408)}$$

be such that the series (regardless of the order for SISO systems) $C_2(z)C_1(z) = K_2(z)$ ensures the performance $\hat{P}_1$. The controller $K_2(z)$ is exactly the target controller (1.1), with the step response depicted in figure 1.1.

It is worthwhile to note that with the performance reduction approach, the designer has much more degrees of freedom, may produce minimum controller of lesser dimension and it is sometimes able to satisfy the performance by preserving already placed poles.

1.2 Feasibility of the Any Time Controller

In a multitasking system a single Central Processor Unit (CPU) has to satisfy a set of different tasks. Within the executable set of tasks, control tasks are of major interest. The computational

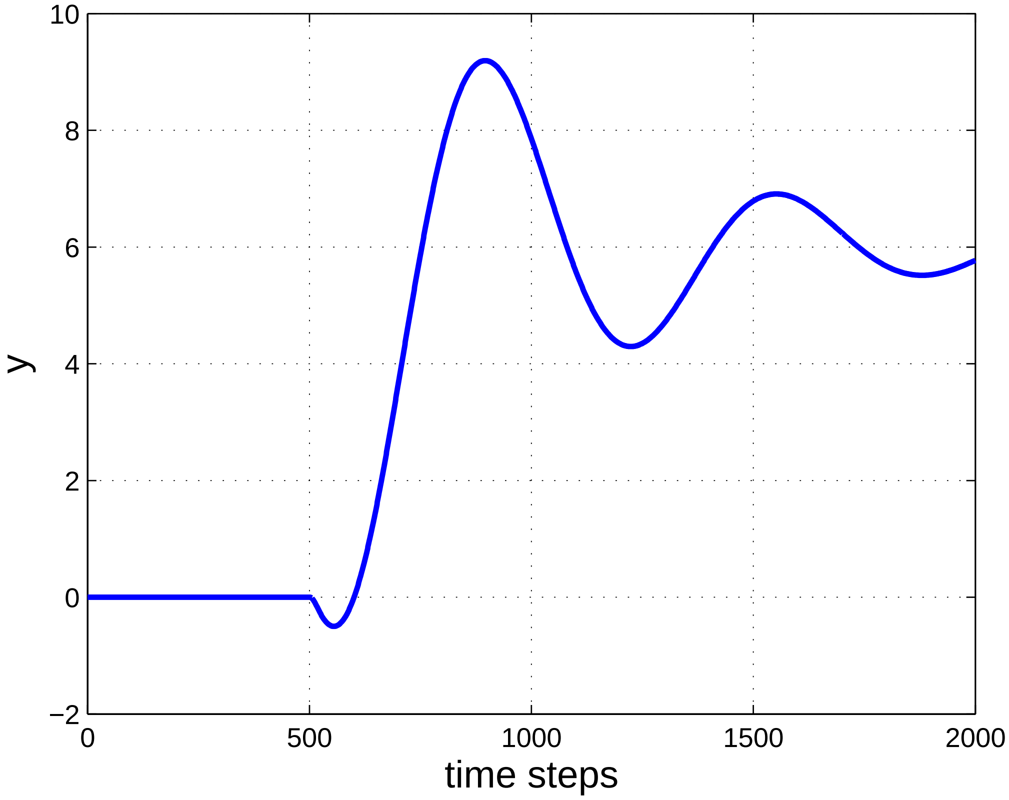


Figure 1.3: Step response of the system $G(z)$ closed in loop with the controller $K_1(z)$ (stability requirement).

environment is assumed to be *real time*, i.e. the correctness of the control tasks depends both on the logical computation and on the time by which the results are available.

According to real time operating systems literature and scheduler policies, tasks are separated into three major classes: *hard* real time, *soft* real time and *non* real time. The idea underlying this taxonomy is the time criticalness of each class of task: hard real time tasks *must* finish their work within their deadline, soft real time tasks *should* finish their work within their deadline, while non real time have no time constraint at all.

Tasks are also classified, according to their time constraints, as *periodic* or *aperiodic*. As defined in [7], a periodic task executes its computations or transmissions at fixed periodic intervals and may be both hard or soft real time, while an aperiodic task has irregular time intervals between two consecutive activations and it has soft deadlines, e.g. a human operator that interacts with the system. For completeness, a *sporadic* task is an aperiodic task with a hard deadline, e.g. critical fault tolerance system.

The digital controllers design for physical systems involves classical problems such as choosing the sampling time, ensuring stability and performance. On the other hand, any actual implementation of a set of controllers must fulfil further logical and temporal constraints dealing with no zero–time execution of the control law, shared resources (the most important of which is the CPU), dependencies, numerical sensitivity, etc.

Usually a CCS executes regularly a set, possibly time dependent, of periodic control tasks and is endowed with an interrupt mechanism to manage external events. In response to these events aperiodic or sporadic tasks are activated with respect to their hardness.

The first concern of a real time system is to guarantee that each hard real time task executes within its deadline. This is a strong requirement, significantly limiting the complexity of the implementable control laws. A way to recover partially the freedom in control design, still satisfying the time constraints, is to split any periodic control task in a *mandatory* part and one or more *optional* (thus skippable) parts. A sporadic task is considered now as being mandatory only and an aperiodic task as optional only. The criticalness of hard tasks is preserved ensuring that only the mandatory parts satisfy the time constraints. Strictly speaking, if all the mandatory parts of a set of tasks are schedulable for every time instant, the scheduler satisfies the *feasible mandatory constraint* [5].

This approach has been used for imprecise computation [5], where a complex algorithm is divided in a mandatory part and a certain number of optional parts progressively refining their results. If all the optional parts are completed, the exact result is provided. The previous method relies on the hypothesis of algorithm *monotonicity.* An algorithm is said *monotone* if the imprecise results, produced by terminating it before complete execution, grow in precision.

Any time controllers, hence, must be designed such that any termination of their execution produces a valid control law (at least a minimal one computed in the mandatory part) performing better than the previous intermediate laws.

In the proposed multitasking architecture, four main architectural factors are taken into account:

1. The periodic task T_i has a period t_i. The computational time left by mandatory parts is assigned dynamically to the optional parts according to the tasks' priorities p_i.

2. The period t_i of some tasks may be variable depending on external events. A change in the period can be managed as a change of task in the set (*time dependent set*).

3. The activation of aperiodic and sporadic tasks is unpredictable.

4. The scheduler satisfies the *feasible mandatory constraint.*

Two main scheduler scenarios can be considered depending on the a–priori knowledge of the available computational time for the next execution of each task:

A) *A–priori knowledge with high confidence level.* The mandatory part is executed and then the whole optional time is devoted to the computation of the controller as greater as possible for the available time.

B) *No a–priori knowledge.* The computation of the controller can be interrupted at *anytime* after the mandatory part, therefore scalable controllers are progressively computed in the optional time (pure any time).

It is worth noting that these two scenarios lead to very different implementations of the any time paradigm and convey different problems. The first scenario is closer to the idea of *multi–version tasks.* In fact, the optional parts can be considered as mutually exclusive versions of the same task, having different computational times and guaranteeing different performances. Once a version is chosen it cannot be changed in favor of a less demanding one and if it terminates prematurely, that is no more time can be given to its execution by the scheduler, the unique valid control law is given by the mandatory part.

The second scenario provides a greater flexibility as it allows a task to be interrupted anytime, but its implementation is more complex. The optional parts are executed in order of complexity and the computations performed previously (or at least a part of them) are "recycled" and constitute the basis for the computation of the following parts. Therefore, when the execution is terminated, the last optional part completed provides the valid control law. This approach, however, requires a special design of the optional parts. They cannot be designed independently as for the first scenario, since they should reuse the previously computed parts. The controller has, therefore, a nested structure where the optional parts are like "Chinese Boxes" each including the previous ones.

The Any Time Controller is an attempt to provide more flexibility to the multitasking control problem, but requires a more complex design phase as many trade–offs have to be taken into account. Summing up, the previous four assumptions on tasks and scheduler characteristics have to be fulfilled along with the following:

5) The controller is split into several elementary controllers in order to break the control computation into several phases. The first elementary piece must guarantee the stability of the plant (*mandatory part*). As the CPU has time, the control computation is further performed, better approximating the final desired control (*optional parts*).

6) *Monotonicity*: the performances must increase as the available time increases.

7) The minimal controller must be computationally simpler than the full controller. This hypothesis is relative to the implementation of the minimal controller as the mandatory part of an any time scalable controller compared with the full controller implemented as the classical, monolithic controller. Therefore, it is not an obvious requirement, because the method used to reduce the full controller may impose a state space representation for the minimal one which is more expensive to be computed than an ad–hoc representation for the full controller. This assertion will become more evident in section 1.3.

Remark 1 *In the literature of CCS, the problem of the "time–to–control" and its variability (* jitter*) is often taken into account. For instance, in [3] a dedicated "jitterbug" software is developed. It is customary the attempt of the designer to reduce the time-to-control, in order to provide the control as close as possible to the instant the inputs were read. In the anytime paradigm, the available computational time between the end of the mandatory part and the task deadline is used to compute the optional parts, consistently with the feasible mandatory constraint. It is, therefore, evident that each control is computed and furnished to the system near its deadline, thus maximizing the time–to–control. A way to reduce the delay between reading input and writing output is to fix a deadline d_i for the task T_i not coinciding with the task period t_i (see figure 1.4). The reading operation is supposed to be performed via hardware at each step of any new period. We will not go further into this problem in the rest of the paper, because it is not relevant for the analysis of the any time controller framework.*

1.3 Single controller digital implementation

The implementation of a digital controller on a PC, DSP or any dedicated hardware requires the designer to cope with many issues related to computational complexity and numerical reliability. Use of finite precision mathematics leads to problems of quantization, round–off and numerical sensitivity to coefficient perturbation. Moreover, controllers may be also ill–conditioned. All these problems may yield controllers significantly deviating from ideal behavior, with poor performances or even unstable. These problems have many solutions ranging from the design of an ad–hoc hardware to the choice of an appropriate representation for the controller (see for instance [8] and [9]).

However, when a certain precision has been chosen, (word length, fixed or floating or mixed fixed–floating point numbers, etc.) the designer has to trade between complexity and numerical reliability. Since the implementation of linear digital controllers is usually performed in state space form, in this paper we do not deal with quantization issues, focusing on the choice of an appropriate state space realization. In fact, it can affect significantly the complexity and the numerical sensitivity of the digital controller.

For our purpose, the complexity of a controller is related to the time required to compute a step and provide the output. A controller is considered more complex than another if it requires more time to be computed. This is not a rigorous definition as it depends on the hardware used to perform

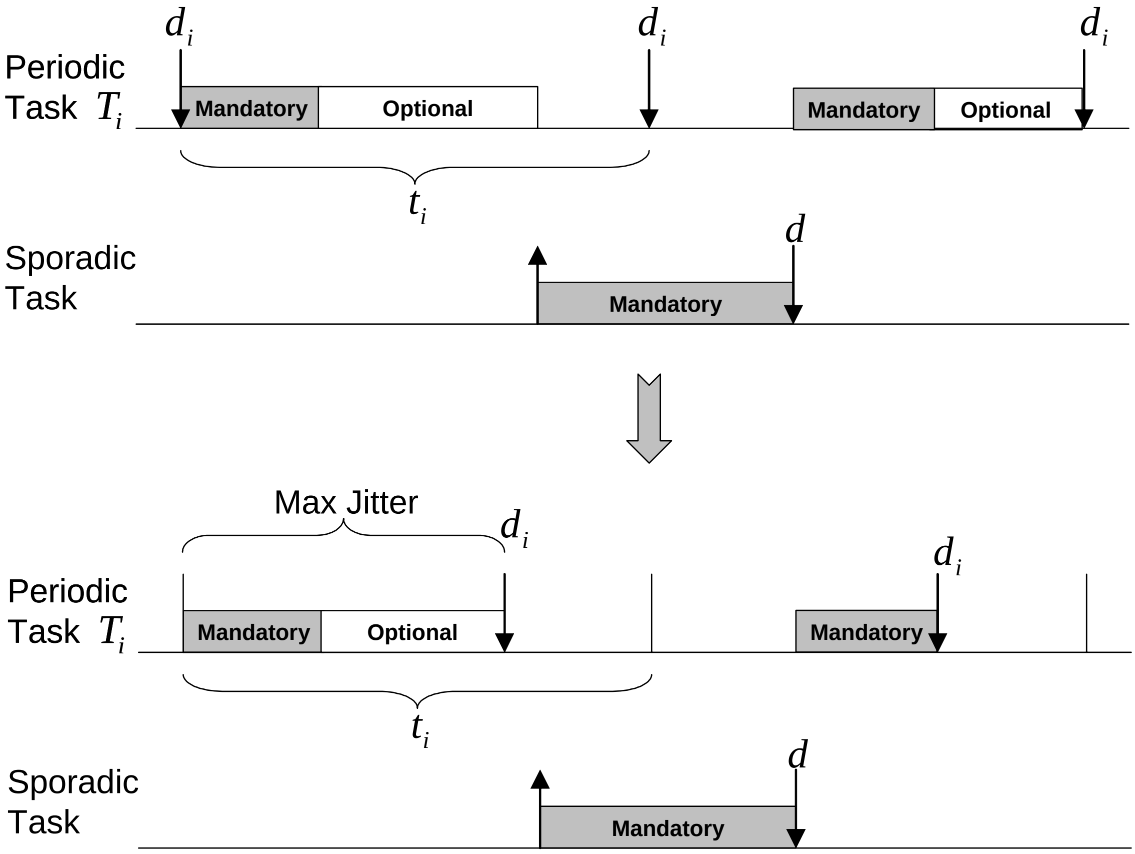


Figure 1.4: Example of two different scheduler execution, with both periodic and sporadic hard real time tasks. In the bottom part of the figure, a maximum allowed jitter constraint is added.

computations, but remarks our interest on the time needed to compute a controller, which is the most important parameter from an any time point of view.

Any mathematical operation requires a certain number of clocks to be executed, which varies for different operations and different hardware. Since real time systems do not use cache devices, the time spent to compute a controller does not vary on a fixed hardware. This allows us to relate complexity to the number of operations. As aforementioned, however, different operations require different number of clocks. Therefore, considering a generic hardware platform and recalling that it is widely accepted in literature that multiplications require much more clocks than additions, we assume that the complexity of a controller is given by the number of multiplications involved.

For comparison purpose we will analyze three state space realizations:

1. Full matrices;
2. Companion form (such as for instance Controllable form);
3. Jordan canonical form.

Let us start considering the SISO case.

We want to find an irreducible state space representation $\{A, b, c, d\}$ of the minimal proper transfer function $g(z)$. The feedthrough term d can be computed as:

$$d = g(\infty) = \lim_{z\to\infty} g(z),$$

then we let:

$$\hat{g}(z) = g(z) - g(\infty)$$

be the strictly proper part of $g(z)$. This allows us to focus only on the $\{A, b, c\}$ realization of a strictly proper transfer function $\hat{g}(z)$ of degree n given by:

$$\hat{g}(z) = \frac{n(z)}{d(z)} = \frac{\beta_{n-1}z^{n-1} + \beta_{n-2}z^{n-2} + \cdots + \beta_1 z + \beta_0}{z^n + \alpha_{n-1}z^{n-1} + \cdots + \alpha_1 z + \alpha_0},$$

where some coefficient β_i can be zero so that the numerator may have degree $m < n-1$. Recall that $\hat{g}(z)$ being minimal, $n(z)$ and $d(z)$ are coprime.

A state space realization of $\hat{g}(z)$ without any particular structure is given by a full $n \times n$ real matrix A and full $n \times 1$ and $1 \times n$ real vectors for inputs and outputs respectively. The numerical robustness of this representation depends on the particular choice of the matrices, therefore it requires a particular analysis for each individual case. Its complexity, instead, can be easily computed in terms of number of multiplications and is given by:

$$C_s^F = n^2 + n + n = n(n+2).$$

This is not a good realization as long as the designer has the freedom to choose the representation, but it can turn out to be a mandatory choice due to other constraints. It is the case, for instance, when the family of scalable controllers is produced by means of model reduction techniques. Indeed, the technique used to perform the model reduction may fix the state space realization.

When the realization of each controller can be chosen arbitrarily, then the Companion form is the most convenient, at least for SISO systems, from a complexity point of view. Consider the Standard Controllable form:

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & & 1 \\ -\alpha_0 & -\alpha_1 & -\alpha_2 & \cdots & \alpha_{n-1} \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

$$c = \begin{bmatrix} \beta_0 & \beta_1 & \beta_2 & \cdots & \beta_{n-1} \end{bmatrix}.$$

It can be built directly by the polynomials $n(z)$ and $d(z)$. Moreover, the number of multiplications to perform a step of this controller is very low as can be seen from the following piece of code:

```
y = beta0*x1 + beta1*x2 + ...  + betan_1*xn;
x1 = x2;
x2 = x3;
⋮
xn = -alpha0*x1 - alpha1*x2 - ...  - alphan_1*xn + u;
```

The output requires n multiplications, but the A matrix requires only n multiplications for the last row (the 1's on the superdiagonal amounts to a shift in the state variables) and the input needs only a sum. Therefore, the complexity of the Controllable form is:

$$C_s^C = n + n = 2n.$$

Unfortunately, this representation is affected by a great sensitivity to parameters variation as pointed out, for instance, by [8] and, hence, should be avoided.

Finally, we introduce the Jordan canonical form, representing a good compromise between numerical reliability and complexity.

The Jordan structure of the A matrix can be derived directly from the $\hat{g}(z)$, since its minimality implies that there can be only one block for each distinct eigenvalue. The vector b can be arbitrarily fixed (for instance with only 1 and 0 elements), provided that the full controllability is ensured, instead the output vector becomes fixed by the choice of b. Equivalently, we can choose c such that the full observability is ensured and derive b later. The matrices can be partitioned as follows:

$$A = \begin{bmatrix} J_1 & & & \bigcirc \\ & J_2 & & \\ & & \ddots & \\ \bigcirc & & & J_k \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix}$$
$$c = \begin{bmatrix} c_1 & c_2 & \cdots & c_k \end{bmatrix}.$$

The matrices J_i are $r_i \times r_i$ Jordan blocks and the b and c vectors are partitioned accordingly. This allows us to consider, for the purpose of complexity analysis, the single subsystem $\{J_i, b_i, c_i\}$ realizing the i-th transfer function $\hat{g}_i(z)$ such that $\hat{g}(z) = \sum_{i=1}^{k} \hat{g}_i(z)$.

If J_i is the block related to the real eigenvalue λ_i, then it has the well known structure:

$$J_i = \begin{bmatrix} \lambda_i & 1 & & \bigcirc \\ & \lambda_i & \ddots & \\ & & \ddots & 1 \\ \bigcirc & & & \lambda_i \end{bmatrix}$$

and b_i can be chosen simply as:

$$b_i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

to satisfy the Popov-Belevic-Hautus rank condition. The output vector c_i can be computed in many ways. For instance, it can be derived by the partial fraction expansion of $\hat{g}_i(z)$:

$$\hat{g}_i(z) = \frac{\delta_{i1}}{z - \lambda_i} + \frac{\delta_{i2}}{(z - \lambda_i)^2} + \cdots + \frac{\delta_{i,r_i}}{(z - \lambda_i)^{r_i}},$$

as:

$$c_i = \begin{bmatrix} \delta_{i,r_i} & \delta_{i,r_i - 1} & \cdots & \delta_{i1} \end{bmatrix};$$

or can be computed by means of the r_i Markov coefficients derived by $\hat{g}_i(z)$ and the Reachability matrix R_i as follows:

$$c_i = h^T R_i^{-1},$$

where $h^T = \begin{bmatrix} h_{i1} & h_{i2} & \cdots & h_{i,r_i} \end{bmatrix}$ is the vector of Markov coefficients. Recall that at this stage the matrix R_i is defined since J_i and b_i are fixed. Regardless of the method used to derive c_i, it is generally a full $1 \times r_i$ vector.

A piece of code implementing the subsystem $\{J_i, b_i, c_i\}$ related to a real eigenvalue, can be as follows:

```
yi = ci_1*xi_1 + ci_2*xi_2 + ...  + ci_ri*xi_ri;
xi_1 = lambdai*xi_1 + xi_2;
xi_2 = lambdai*xi_2 + xi_3;
⋮
xi_ri = lambdai*xi_ri + u;
```

The complexity of $\hat{g}_i(z)$ is then given by $r_i + r_i = 2r_i$.

If the block J_i is related to a couple of complex conjugate eigenvalues $\lambda_i = \sigma_i + j\omega_i$, $\bar{\lambda}_i = \sigma_i - j\omega_i$, then it can be written in real Jordan form as:

$$J_i = \begin{bmatrix} \sigma_i & \omega_i & 1 & 0 & & & \\ -\omega_i & \sigma_i & 0 & 1 & & \bigcirc & \\ & & \sigma_i & \omega_i & \ddots & & \\ & & -\omega_i & \sigma_i & & 1 & 0 \\ & & & & \ddots & 0 & 1 \\ & \bigcirc & & & & \sigma_i & \omega_i \\ & & & & & -\omega_i & \sigma_i \end{bmatrix}.$$

It can be proved that the vector:

$$b_i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

is capable of guaranteeing the full controllability of the couple (J_i, b_i). The vector c_i can be computed in a similar way as before and is generally of full dimension. If the block J_i has again dimension r_i (but now necessarily even) then the complexity of $\hat{g}_i(z)$ is given by $2r_i + r_i$.

The complexity of the full canonical Jordan realization of $\hat{g}(z)$ is given by the sum of the complexity of the various blocks. However, it is worth noting that the presence of 1's in the blocks and in the input vectors are immaterial for our complexity criterion. Therefore b does not take any part in the complexity evaluation, c requires n multiplications and each eigenvalue contributes with one multiplication if it is real or with two if it is complex, no matter on how the matrix is partitioned in Jordan blocks. Finally, we can consider the two extreme cases of blocks all related to real eigenvalues and blocks all related to complex conjugate eigenvalues and provide the following bounds for the complexity of the Jordan canonical form:

$$2n \leq C_s^J \leq 3n.$$

Now we can justify the requirement on the dimension of the minimal controller, implemented in the mandatory part, made in the previous section. If the reduction technique constrains the minimal controller to be in full matrices form, then it may have a greater complexity than that of the full classical controller implemented in Jordan form. In this case it is better to implement directly the classical controller and not the scalable version. Consider, for instance, a full SISO controller of dimension $n = 6$ implemented in Jordan form and a reduced controller of dimension $r = 3$ implemented in full matrices form. The complexity of the full controller is $C_n = 2n = 12$ less then the complexity of the reduced one $C_r = r(r+2) = 15$. Similar remarks hold also for MIMO systems.

The same need for numerical reliability leading the choice of a particular realization of a SISO controller, occurs also in the realization of a MIMO controller. This is essentially the reason why we focus on Jordan form.

It is a well known fact that realizing a MIMO system is a significantly more complex task than realizing a SISO one, mainly if we look for a minimal form. The dimension of the state space must be set equal to the McMillan degree of the Matrix Transfer Function (MTF), which is usually greater or at least equal to the degree of the least common denominator of the various transfer functions in the MTF. Moreover, the structure of the Jordan form is no longer obvious as for the SISO case, because more than one block can be associated to the same eigenvalue and the dimension of these blocks may be the same. Due to the minimality, hence to the full controllability and observability, the maximum number of blocks relative to the same eigenvalue can be equal to the minimum between the number of inputs and outputs.

Suppose we have a $p \times q$ proper rational MTF $G(z)$ and we want to find a realization $\{A, B, C, D\}$ with A in Jordan form. First of all we compute D as:

$$D = G(\infty) = \lim_{s \to \infty} G(z),$$

and we let:

$$\hat{G}(z) = G(z) - G(\infty)$$

be the strictly proper part of $G(z)$. The block structure of $\hat{G}(z)$ is an intrinsic property, therefore the irreducible Jordan form can be uniquely determined up to block permutations. The maximum number of blocks associated to the same eigenvalue can be at most equal to $\min(p, q)$.

In many works (as for instance [10]), the minimal Jordan form is produced by building a non minimal Jordan form and then reducing it (two-steps reduction). In [11] the determination of the McMillan degree and of the Jordan structure, namely the number and the dimension of the various blocks, is performed directly by computing the ranks of some special matrices which turn out to be the Hankel matrix and its submatrices.

A strictly proper MTF $\hat{G}(z)$ having m distinct eigenvalues can be factored as a sum of m MTFs containing only one pole with multiplicity r_i:

$$\hat{G}(z) = \sum_{i=1}^{m} \hat{G}_i(z).$$

Therefore, the full minimal Jordan realization $\{A, B, C\}$ of $\hat{G}(z)$ is derived by the minimal Jordan realizations $\{A_i, B_i, C_i\}$ of the $\hat{G}_i(z)$ as follows:

$$A = \begin{bmatrix} A_1 & & \bigcirc \\ & \ddots & \\ \bigcirc & & A_m \end{bmatrix},$$

$$B = \begin{bmatrix} B_1 \\ \vdots \\ B_m \end{bmatrix}, \quad C = \begin{bmatrix} C_1 & \cdots & C_m \end{bmatrix}.$$

Each Jordan realization of the $\hat{G}_i(z)$ can be found independently of the other and in the same way, hence let us consider only one $\hat{G}_i(z)$. It can be further factored as:

$$\hat{G}_i(z) = \sum_{j=1}^{r_i} \frac{1}{(z - \lambda_i)^j} Q_{j-1}$$

where the $p \times q$ matrix coefficients Q_j are used to build the Hankel matrices used to derive the Jordan structure. If n_j is the number of blocks of dimension j, $N_i = \sum_{j=1}^{r_i} n_j$ is the total number of blocks of $\hat{G}_i(z)$, then the Jordan canonical form of A_i is given by:

$$A_i = \begin{bmatrix} J_{1,r_i} & & \bigcirc \\ & \ddots & \\ \bigcirc & & J_{N_i,1} \end{bmatrix},$$

where

$$J_{k,j} = \begin{bmatrix} \lambda_i & 1 & & \bigcirc \\ & \lambda_i & \ddots & \\ & & \ddots & 1 \\ \bigcirc & & & \lambda_i \end{bmatrix}$$

is the k-th ($k = 1, \ldots, N_i$) Jordan block and has dimension $j \times j$.

As for SISO systems, the Jordan form of the matrix A_i is uniquely determined, and the forms of B_i and C_i are related in that, choosing one of them fixes the other. Unfortunately, in the MIMO case neither B_i nor C_i can be fixed arbitrarily, only taking into account the full controllability or observability constraint. The construction of the input and output matrices is more complex and cannot be carried out by simply filling matrices with 1's in appropriate positions. However, there are certain degrees of freedom to exploit to the aim of reducing the number of operations needed to compute it. For instance, one can try to fill one of these matrices with as many 0's as possible. We will not go into this issue thoroughly, anyway, and consider the worst case of full B_i and C_i matrices. For our purpose, we prefer a systematic and numerically robust algorithm to get the irreducible Jordan form as the one provided in [12]. This algorithm is capable to derive B_i and C_i matrices along with the minimal Jordan form of A_i involving only SVD computations. The SVD is known to be a numerically robust technique to deal with rank computation problems, as those required in the derivation of the Jordan form of A_i and can be used profitably also to compute directly B_i and C_i.

For the purpose of estimate the complexity of a MIMO controller, consider a realization with McMillan degree $n = \sum_{i=1}^{m} \sum_{j=1}^{r_i} j \cdot n_j$ and full $n \times q$ input and $p \times n$ output matrices. As in the SISO case the 1's in the above diagonal amounts only in doing some sums, and are therefore immaterial for our cost criterion. The total amount of multiplications to be performed to compute a step of this controller, hence its complexity, is equal to:

$$C_m^J = n + pn + nq = n(p + q + 1),$$

in the worst case. Due to the non special form of B_i and C_i matrices, this realization loses some of its convenience if compared to its analogous for SISO systems, but it retains the same numerical reliability properties. However, it remains one of the most convenient realizations for MIMO systems also from a complexity point of view. Indeed, it is obviously less expensive then a full realization, whose complexity is given by:

$$C_m^F = n(p + q + n),$$

but it can be also less complex then a companion form.

There are many companion forms for MIMO systems [13, 14] and they are all significantly more complex then their SISO counterparts. Consider the controllable companion form which makes use of all the input vectors [15]:

$$A = \begin{bmatrix}
0 & 1 & 0 & \cdots & 0 & & & & & & & & & & & \\
0 & 0 & 1 & & 0 & & & \bigcirc & & & & & & \bigcirc & & \\
\vdots & & & \ddots & \vdots & & & & & & \cdots & & & & & \\
0 & 0 & 0 & \cdots & 1 & & & & & & & & & & & \\
\star & \star & \star & \cdots & \star & \star & \star & \star & \star & \star & \cdots & \star & \star & \star & \star & \star \\
 & & & & & 0 & 1 & 0 & \cdots & 0 & & & & & & \\
 & & \bigcirc & & & 0 & 0 & 1 & & 0 & & & & \bigcirc & & \\
 & & & & & \vdots & & & \ddots & \vdots & \cdots & & & & & \\
 & & & & & 0 & 0 & 0 & \cdots & 1 & & & & & & \\
\star & \star & \star & \star & \star & \star & \star & \star & \cdots & \star & \cdots & \star & \star & \star & \star & \star \\
 & & \vdots & & & & & \vdots & & & \ddots & & & \vdots & & \\
 & & & & & & & & & & & 0 & 1 & 0 & \cdots & 0 \\
 & & \bigcirc & & & & & \bigcirc & & & & 0 & 0 & 1 & & 0 \\
 & & & & & & & & & & \cdots & \vdots & & & \ddots & \vdots \\
 & & & & & & & & & & & 0 & 0 & 0 & \cdots & 1 \\
\star & \star & \star & \star & \star & \star & \star & \star & \star & \star & \cdots & \star & \star & \star & \cdots & \star
\end{bmatrix}$$

$$B = \begin{bmatrix}
0 & 0 & 0 & \cdots & 0 \\
0 & & & & 0 \\
\vdots & & & & \vdots \\
1 & \star & \star & \cdots & \star \\
0 & 0 & 0 & \cdots & 0 \\
0 & 0 & & & 0 \\
\vdots & \vdots & & & \vdots \\
0 & 1 & \star & \cdots & \star \\
 & & \vdots & & \\
0 & 0 & 0 & \cdots & 0 \\
0 & 0 & 0 & & 0 \\
\vdots & \vdots & \vdots & & \vdots \\
0 & 0 & 0 & \cdots & 1
\end{bmatrix}$$

with C generally full matrix. In this form A has q matrices in companion form on the diagonal and B is partitioned accordingly. With the understanding that the $\star$'s in previous equations stand for not null elements, we can compute the complexity of this form as nq multiplications for the A matrix (as before we do not care of 1's), $\frac{q(q-1)}{2}$ multiplications for the B and pn for the C matrix. The complexity is therefore:

$$C_m^C = n(p+q) + \frac{q(q-1)}{2},$$

that can be considered not very different from the complexity of the Jordan form. It is worth noting, however, that any companion minimal form requires the MTF $\hat{G}(z)$ to be first factored in coprime factors, which is known to be a non simple task.

So far we have considered minimal realizations with the persuasion that aiming at minimality is the best choice, but it could not be true. Non minimal realizations may involve a lesser number of multiplications due to the more empty structure of the matrices they require. For instance, a direct realization of any $\hat{G}(z)$ can be obtained by juxtaposing the realizations of each transfer function in $\hat{G}(z)$. If each entry is realized in controllable form, then the A matrix has dimensions given by the sum of the degrees of each denominator in $\hat{G}(z)$ (let N be this number), while B and C have

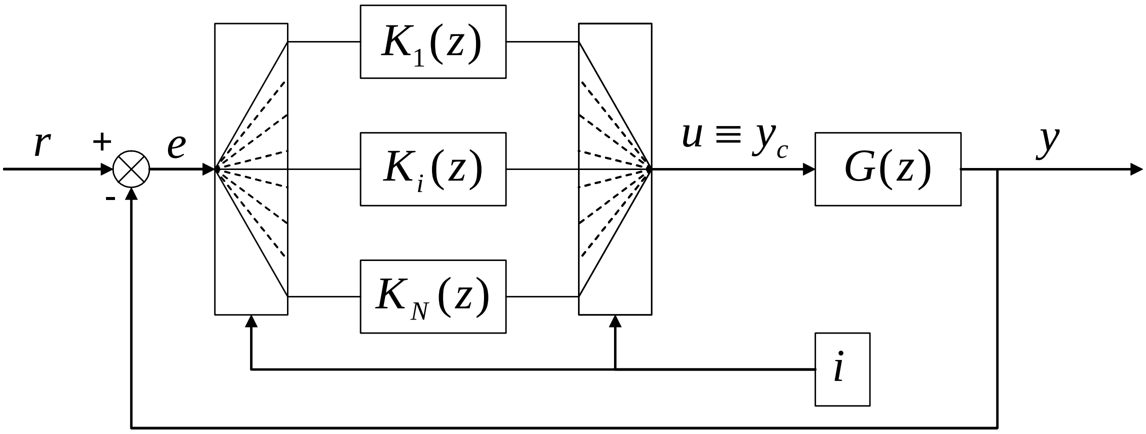


Figure 1.5: Mutually exclusive single loop connection.

dimensions $p \times N$ and $N \times q$ respectively. Although N is usually a much greater number then the McMillan degree n, the matrix A has only N numbers not 0 and not 1, B has few not null entries and are all set to 1, C, instead, is generally a full matrix. The total complexity is, therefore, equal to $N(1+q)$, which may be less than the complexity of any minimal realization. There are a lot of non minimal realizations and it is possible that at least one of them is better than any minimal one with respect to complexity. The designer has to decide wether the complexity reduction is obtained at the expense of a reasonable increase of the state dimension or not. Anyway, the dimension of a non minimal realization is not unique and its profitability depends on the single case. For future comparisons to make sense, hence, we will refer only to minimal realizations.

1.4 Closed loop models

A switching architecture with unitary feedback for the any time paradigm is reported in figure 1.5.As long as the computational time is available, the index i is increased, switching to the next, more complex controller. The bus selectors, leaded by the index i, exclusively connect the input error e to the controlled system input u through the indexed controller $K_i(z)$. The controllers from $K_{i+1}(z)$ to $K_N(z)$ are “frozen” or rather their state space representation is not updated. The states of these frozen controllers have been called *sleeping states*. Each controller $K_i(z)$ is made up of some components controllers $(C_1(z), \dots, C_{i-1}(z))$ which can realize any connection scheme. Since in the anytime paradigm the scalability must be intended from a code point of view only, the connection schemes have not a proper physical meaning, but a logical one. These logical connection schemes may help in gaining more intuition during the design phase.

Obviously some connections can be more suitable than others for the any time implementation, therefore this aspect deserves a deeper analysis. The set of feasible connections must be defined to take into account the controllers' scalability constraint and the further constraint of the implicit *execution order*. With some connection schemes, for the computation of the controller $K_i(z)$, the component controller $C_i(z)$ must be computed after the component controllers $C_1(z)$ to $C_{i-1}(z)$ (as it is evident in the series connection where the output of $C_j(z)$ is the input for $C_{j+1}(z)$). Moreover, the violation of the execution order leads to the lost of already performed computations since, if the input to the component controller $C_i(z)$ changes, then $C_i(z)$ must be computed again. Unfortunately, it can be an unavoidable characteristic of some connection schemes such as for instance, the nested loop scheme with unitary feedback.

Model reduction. Model reduction techniques have not their counterpart in the logical connections of controllers. Indeed, this is a straight consequence of the state space reduction that can only be converted into computational time preserving algorithms. In general, the effects on the fre-

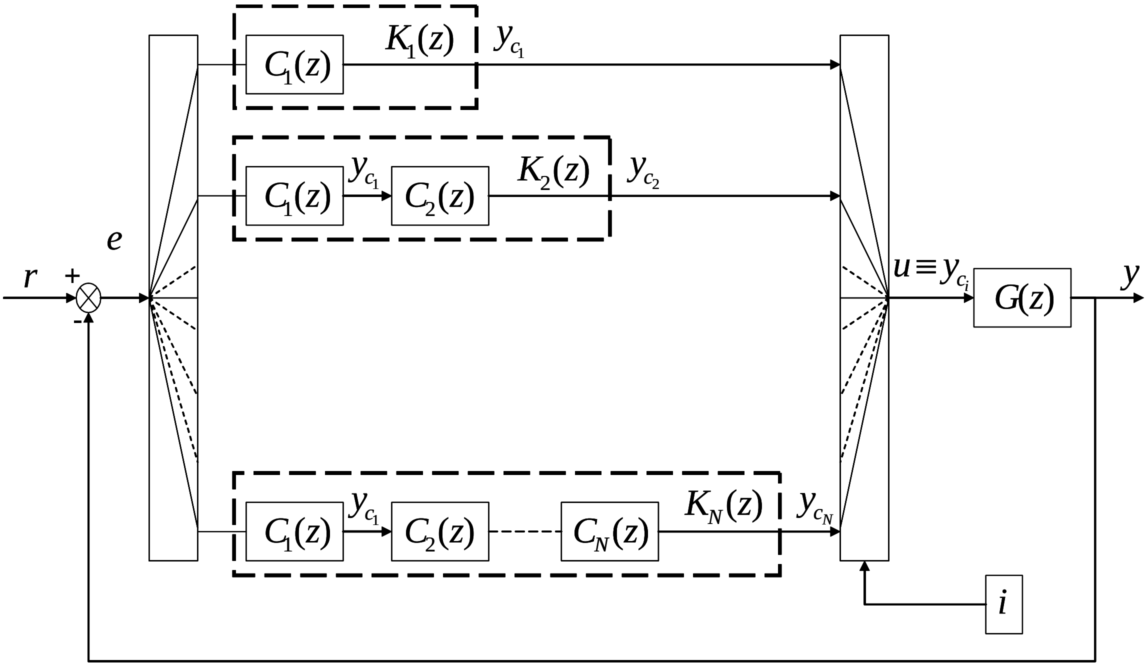


Figure 1.6: Series connection. The dashed boxes represent the controller $K_j(z)$, with $j = 1, \dots, N$. Refer to figure 1.5.

quency domain are not so easily identified, as well as the effects on the performances, since a state space model reduction has effects on the controller's singularities placements. This means that the controllers $K_i(z)$ are mutually exclusive, hence the logical connection is as in figure 1.5. However, the actual computational effort does not suffer of this connection drawbaks, as it is performed by hemming the dynamic matrices of the controller.

Series connection. The series connection of the various controllers comes directly from their implementation and their frequency domain decomposition. For instance, referring to the example in section 1.1, the proposed performance reduction is trivially mapped on the series connection. For SISO systems there are no limitations on the possible connection order among the controllers, but this order is imposed by the scalability constraint (execution order). Instead, for multi–input and/or multi–output systems, the order of connections is fundamental. Indeed, it is straightforward that commutativity is no longer valid for matrix products. Furthermore, if the transfer matrix $G(z)$ has p rows and m columns (i.e. p outputs and m inputs), the simplest controller $C_1(z)$ has m rows and p columns, while the controllers from $C_2(z)$ through $C_N(z)$ have m rows and m columns. For a visualization of the series connection refer to figure 1.6: again, in this architecture the bus selector chooses the computed controller series.

Parallel connection. Still very simple and useful for the any time paradigm, the parallel connection (see figure 1.7, where each switching line is controlled by the index i: if $K_i(z)$ is computed, than the switches from 2 to i are closed) can be regarded as a logical connection scheme. It is worthwhile to note from figure 1.7 that it can be easily interpreted with the main generic scheme in figure 1.5.

Scalability is traced in the decomposition of the controller $K_i(z)$ in a sum of elementary controllers $C_1(z)$ to $C_i(z)$. In such a way, the control input u of the plant $G(z)$ is the sum of the previously computed controllers

$$u = \sum_{j=1}^{i} y_{c_j} \ ,$$

where y_{c_j} is the output of the controller $C_i(z)$.

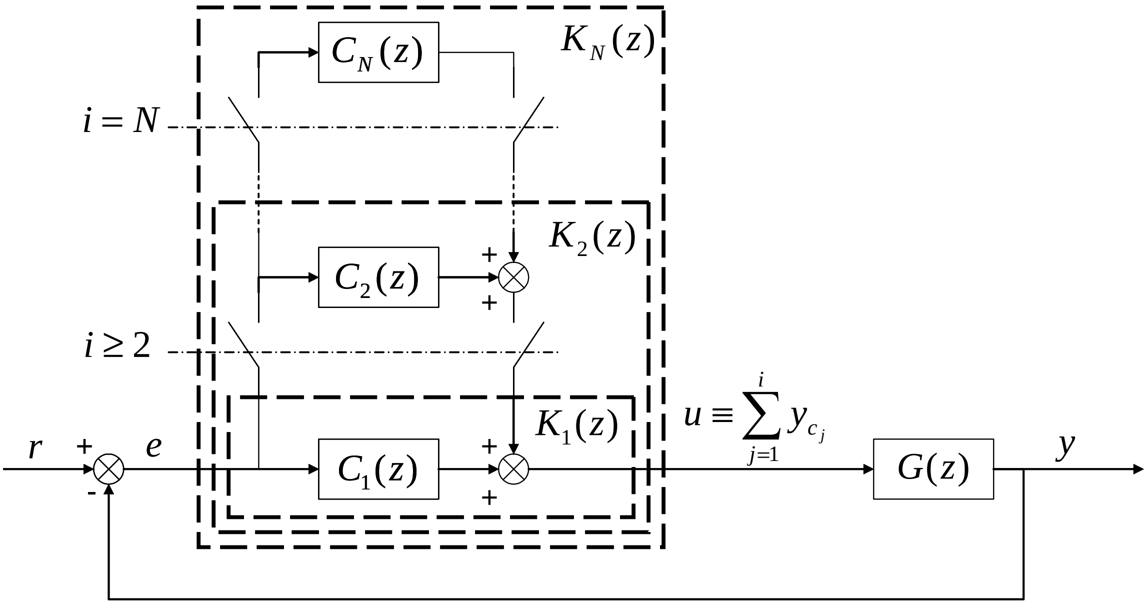


Figure 1.7: Parallel connection. The index i controls the switches' aperture. The system control u is the sum of the previously computed controllers.

The execution order constraint is easily verified in the parallel connection definition, since each controller receives the same error input $e = r - y$ at the same time instant.

Although difficult to read for closed loop stability or performance achievement, the simplest parallel form can be found by applying *Partial Fractions Decomposition* on the transfer function of the target controller $K(z)$.

Finally, the closed loop schemes presented are not the only representatives of the virtually infinite number of possible connections among a set of controllers, but they have been chosen for their simplicity (an obvious target for complexity reduction) and because any possible different connection can be reduced to one of the presented assemblages, adding at least some more block, e.g. a pre–filter and/or a feed–through block. Nevertheless, one more particular closed loop connection is briefly described, due to its positive features on the controller design.

Nested loop connection. The *nested loop* connection can be thought of as a single closed loop scheme with a plant, which is itself a closed loop scheme (e.g. multi–rate systems).

Consider a single closed loop (the variable z is omitted for the sake of brevity)

$$y = G_{cl_1} r = (I + G\, C_1)^{-1}\, G\, C_1\, F_1\, r\ ,$$

where $r \in \mathbb{R}^p$ is the external reference signal, C_1 is the minimal controller, F_1 is the associated pre–filter block (two degrees of freedom design) and $y \in \mathbb{R}^p$ is the regulated output (see figure 1.8, a)). Closing the loop with a second, more complex controller C_2 with an associated F_2 pre–filter block (figure 1.8, b)) yields

$$y = (I + G_{cl_1}\, C_2)^{-1}\, G_{cl_1}\, C_2\, F_2\, r\ = (I + G\, C_1\, \ (I + F_1\, C_2))^{-1}\, G\, C_1\, F_1\, C_2\, F_2\, r\ .$$

By imposing

$$y = (I + G\, C_{eq_2})^{-1}\, G\, C_{eq_2}\, F_{eq_2}\, r,$$

where C_{eq_2} and F_{eq_2} are respectively the controller and the pre–filter block equivalents to obtain a single closed loop scheme, the relations

$$\begin{cases} C_{eq_2} & = & C_1\ (I + F_1\, C_2) \\ F_{eq_2} & = & (I + F_1\, C_2)^{-1}\, F_1\, C_2\, F_2, \end{cases}$$

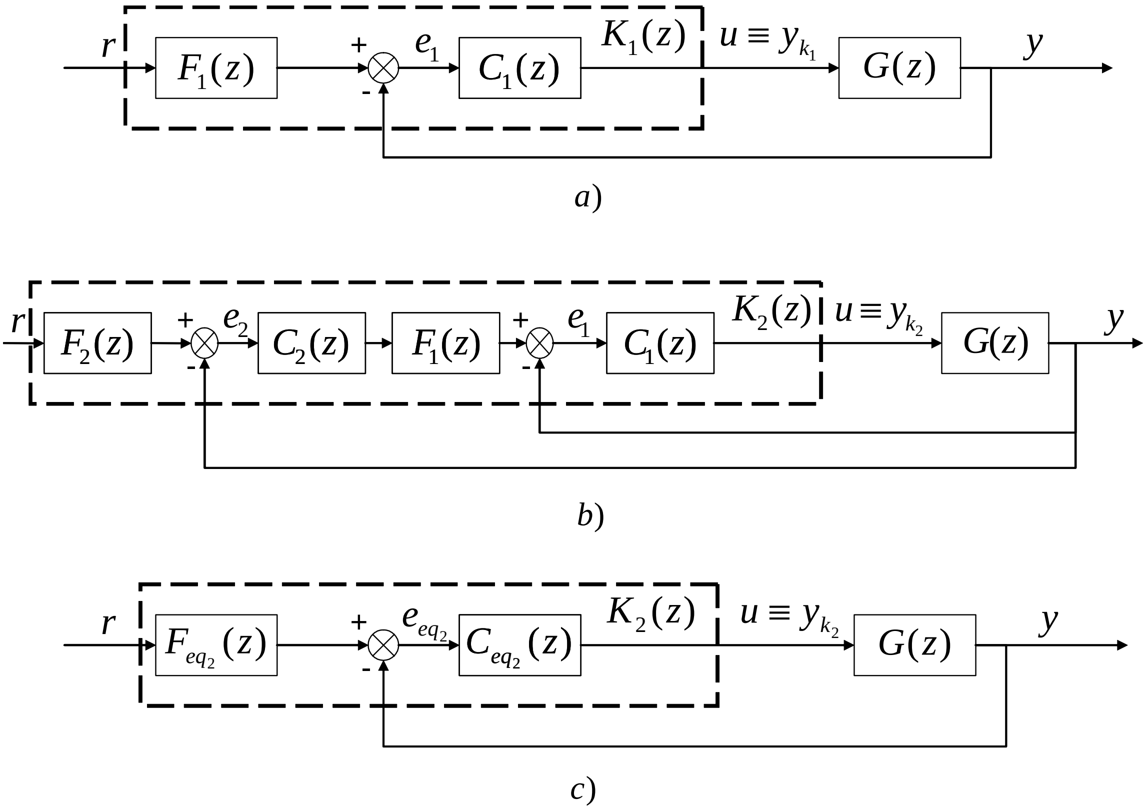


Figure 1.8: a) Single closed loop architecture with pre–filter block. b) Nested closed loops with two controllers of ascendent complexity. c) Single closed loop with pre–filter block equivalent to b).

are gained (see figure 1.8, c)). The procedure can be further iterated to obtain for the controller C_i the equivalents

$$\begin{cases} C_{eq_i} = C_{eq_{i-1}} \left(I + F_{eq_{i-1}}\, C_i\right) \\ F_{eq_i} = \left(I + F_{eq_{i-1}}\, C_i\right)^{-1} F_{eq_{i-1}}\, C_i\, F_i. \end{cases}$$

Although the designer helpful characteristics of the presented nested loop assemblage are evident, its efficiency for the any time controller approach is quite poor since some computation performed for the i–th controller are discarded to perform the $(i+1)$–th controller. This poor scalable property is due to the fact that the reference changes at each nested loop input. Indeed, the computation of the controller C_{i+1} affects the input u_{c_i} to the controller C_i, violating the execution order property (in figure 1.9, another single loop equivalent, explicitly dependent from original blocks C_1, C_2, F_1 and F_2, is reported). Little improvements in scalability are achieved by inverting the controllers, with the minimal controller C_1 as the most external (by inverting the index 2 with 1 in figure 1.8, b)). Nevertheless, with MIMO systems, dimensions have to be taken into account: if the controlled system G has p rows and m columns (i.e. p inputs and m outputs), with $p \neq m$ and $p > 1$ and/or $m > 1$, controllers C_i must all have m rows and p columns, implying that prefilters F_i, with $i = 2, \ldots, N$, are of the same dimensions of the system G. Furthermore, even if the controlled system G is squared or SISO, by inverting the controllers in figure 1.8 the design that ensures closed loop stability and performances is almost impossible, since each controller C_i, with $i = 1, \ldots, N-1$, should ensure the set of performances $\{\hat{P}_1, \ldots, \hat{P}_i\}$ with the system G, the closed loop of C_{i+1} and G, the closed loop of C_{i+1} and the closed loop of C_{i+2} and G, and so on.

Hence, the nested loop connection with unitary feedback presents many problems related to the execution order constraint. The nested loop scheme directly connected in the feedback path, still preserves the same appealing with respect to design simplicity than the previous one, but is more

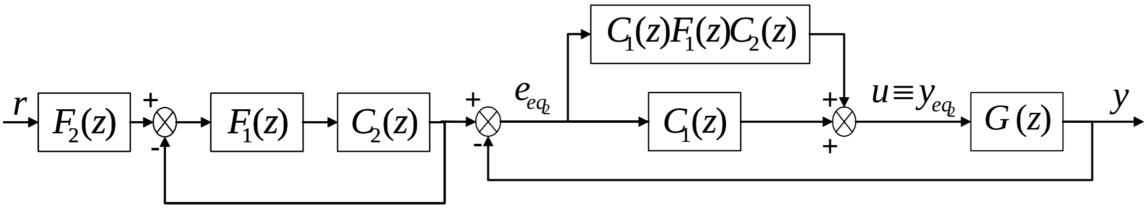


Figure 1.9: Equivalent closed loop connection expressed with respect to original controllers $C_1(z)$ and $C_2(z)$ and associated pre–filters $F_1(z)$ and $F_2(z)$.

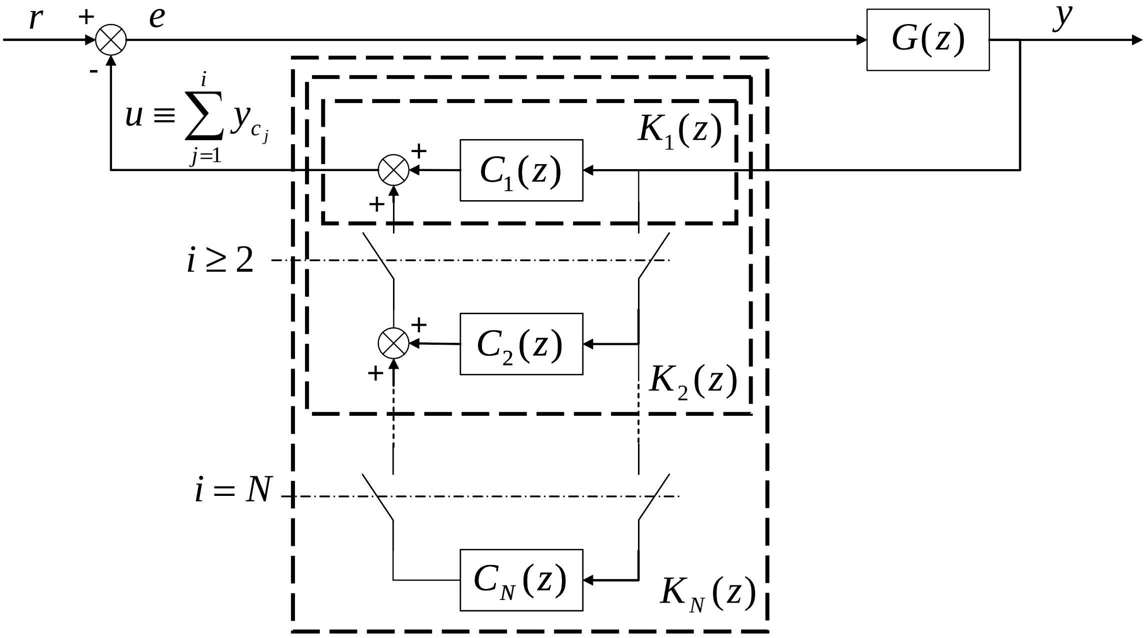


Figure 1.10: Closed loop with parallel connection on the feedback path.

suitable for scalability and verifies the execution order constraint). Indeed, it is easy to see that this connection is equivalent to the parallel one in the feedback loop (see figure 1.10).

The controller C_1 is connected in closed loop with the system G, the controller C_2 is connected in closed loop with the system $G_{cl1} = (I+GC_1)^{-1}G$ and so on. Therefore, each controller $K_i = \sum_{j=1}^{i} C_j$.

1.4.1 Realizations and code skeletons

Consider a generic system $G(z)$ to be controlled and its realization (A_s, B_s, C_s, D_s), whose state variable is x_s, the inputs are u and the outputs are y and where $A_s \in \mathbb{R}^{n_s \times n_s}$ is the dynamic matrix, $B_s \in \mathbb{R}^{n_s \times m_s}$ is the input matrix, $C_s \in \mathbb{R}^{p_s \times n_s}$ is the output matrix and $D_s \in \mathbb{R}^{p_s \times m_s}$ is the input direct feed–through matrix. The state space dynamics of the discrete time controlled system is trivially

$$\begin{cases} x_s^+ &= A_s\, x_s + B_s\, u \\ y &= C_s\, x_s + D_s\, u, \end{cases}$$

where x_s^+ is the state at the next iteration.

Furthermore, let $(A_{c_i}, B_{c_i}, C_{c_i}, D_{c_i})$ be a generic realization of the component controller $C_i(z)$, whose state variable is $x_{c_i} \in \mathbb{R}^{n_{c_i}}$, the inputs are $u_{c_i} \in \mathbb{R}^{m_{c_i}}$ and the outputs are $y_{c_i} \in \mathbb{R}^{p_{c_i}}$. Two assumptions are taken into account:

1. To avoid logical loops, at least one among the controllers $K_i(z)$ and the system under control $G(z)$ is chosen to be strictly proper, i.e. at least one between the minimal controller $C_1(z) \equiv K_1(z)$ and the plant $G(z)$. For simplicity's sake, we consider the system under control to be strictly proper ($D_s = 0$).

2. To reduce the computational complexity, the realizations are minimal. Non–minimal realizations are somewhat interesting considering other closed loop system properties, as switching stability [16].

Model reduction. For the minimum controller $C_1(z)$ (or equivalently $K_1(z)$), the model reduction has the updating equations

$$\left\{ \begin{array}{rcl} x_{c_1}^{+} & = & A_{c_1}\, x_{c_1} + B_{c_1}\, (r - C_s\, x_s) \\ y_{c_1} & = & C_{c_1}\, x_{c_1} + D_{c_1}\, (r - C_s\, x_s). \end{array} \right. \tag{1.2}$$

The main feature is that each dynamic matrix is "hemmed" with each suppressed part such that the controller $C_{i+1}(z)$ is given by

$$\left\{ \begin{array}{rcl} x_{c_{i+1}}^{+} = \begin{bmatrix} x_{c_i} \\ \bar{x}_{c_{i+1}} \end{bmatrix}^{+} & = & \begin{bmatrix} A_{c_i} & \alpha_{c_{i+1}} \\ \beta_{c_{i+1}} & \bar{A}_{c_{i+1}} \end{bmatrix} \begin{bmatrix} x_{c_i} \\ \bar{x}_{c_{i+1}} \end{bmatrix} + \begin{bmatrix} B_{c_i} \\ \bar{b}_{c_{i+1}} \end{bmatrix} (r - C_s\, x_s) \\ y_{c_{i+1}} & = & \begin{bmatrix} C_{c_i} & \bar{c}_{c_{i+1}} \end{bmatrix} \begin{bmatrix} x_{c_i} \\ \bar{x}_{c_{i+1}} \end{bmatrix} + D_{c_N}\, (r - C_s\, x_s), \end{array} \right.$$

where $\bar{A}_{c_{i+1}}$, $\bar{b}_{c_{i+1}}$ and $\bar{c}_{c_{i+i}}$ are respectively a square matrix of dimension $\bar{n}_{c_{i+1}} = n_{c_{i+1}} - n_{c_i}$, a matrix of dimension $\bar{n}_{c_{i+1}} \times p_s$ and a matrix of dimension $m_s \times \bar{n}_{c_{i+1}}$, while $\alpha_{c_{i+1}}$ ($\beta_{c_{i+1}}$) is a matrix of dimension $n_{c_i} \times \bar{n}_{c_{i+1}}$ ($\bar{n}_{c_{i+1}} \times n_{c_i}$).

The final closed loop state space dimension is then equal to $n_s + n_k$ (where n_k is the dimension of the (final) target controller), that is the smallest state space dimension among the different approaches. The closed loop system with the controller $K_i(z)$ has the following updating equations

$$\left\{ \begin{array}{rcl} \begin{bmatrix} x_s \\ x_{c_1} \\ \vdots \\ \bar{x}_{c_i} \\ \vdots \\ \bar{x}_{c_N} \end{bmatrix}^{+} & = & \begin{bmatrix} A_s - B_s\, D_{c_i}\, C_s & B_s\, C_{c_1} & \dots & B_s\, \bar{c}_{c_i} & \dots & 0 \\ -B_{c_1}\, C_s & A_{c_1} & \dots & \alpha_{c_i} & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ -\bar{b}_{c_i}\, C_s & \beta_{c_i} & & \bar{A}_{c_i} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} x_s \\ x_{c_1} \\ \vdots \\ \bar{x}_{c_i} \\ \vdots \\ \bar{x}_{c_N} \end{bmatrix} + \begin{bmatrix} B_s\, D_{c_N} \\ B_{c_1} \\ \vdots \\ \bar{b}_{c_i} \\ \vdots \\ 0 \end{bmatrix} r \\ y & = & \begin{bmatrix} C_s & 0 & \dots & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} x_s \\ x_{c_1} \\ \vdots \\ \bar{x}_{c_i} \\ \vdots \\ \bar{x}_{c_N} \end{bmatrix} \end{array} \right.$$

representing the whole state space and clarifying the existence of the *sleeping states.* For the controller $K_i(z)$, the *sleeping states* belong to a non computed part (the states of the controllers $C_{i+1}(z)$ through $C_N(z)$).

Let us now take a brief look on the code implementation. The first step for the any time controller feasibility is to compare all the proposed code implementations with the implementation of the target controller $K(z)$, with state space realization (A_k, B_k, C_k, D_k) of dimension n_k (see[8] for a deep discussion on digital controllers' implementation):

```
read inputs(r,y);                 }
yk := Ck*xk + Dk*(r - y);         } Mandatory part.        (1.3)
xk := Ak*xk + Bk*(r - y);         }
write outputs(yk);                }
```

Due to unpredictable events, but satisfying the feasibility of the mandatory part, the scheduler will execute one of the possible controller $K_i(z)$ at any task period iteration. An example of the code skeleton for the model reduction is

```
read inputs(r,y);
Old_xc[1] := xc[1];
yc[1] := Cc1*Old_xc[1] + Dc1*(r - y);
xc[1] := Ac1*Old_xc[1] + Bc1*(r - y);
write outputs(yc[1]);
```

Mandatory part

```
Old_xc[2] := bar_xc[2];
yc[2] := yc[1] + bar_Cc2*Old_xc[2];
xc[1] := xc[1] + alpha_c2*Old_xc[2];
bar_xc[2] := beta_c2*Old_xc[1] +
  + bar_Ac2*Old_xc[2] + bar_bc2*(r - y);
write outputs(yc[2]);
```

Optional part 1

```
       ⋮
Old_xc[i] := bar_xc[i];
yc[i] := yc[i-1] + bar_Cci*Old_xc[i];
xc[1] := xc[1] + alpha_ci[1]*Old_xc[i];
bar_xc[2] := bar_xc[2] + alpha_ci[2]*Old_xc[i];
  ⋮
bar_xc[i-1] := bar_xc[i-1] + alpha_ci[i-1]*Old_xc[i];
bar_xc[i] := beta_ci[1]*Old_xc[1] + ... +
  + beta_ci[i-1]*Old_xc[i-1] + bar_Aci*Old_xc[i] +
  + bar_bci*(r - y);
write outputs(yc[i]);
```

Optional part i

```
       ⋮
```

For ease of notation in each code skeleton throughout the paper, with `xc[i]` (`bar_xc[i]`) we intend the i–th vector component of length n_{c_i} ($\bar{n}_{c_i}$), while with `alpha_ci[1]` (`beta_ci[1]`) we mean the submatrices of the controller.

The code implementation highlights the discrepancy with the theoretical assumption on the state space dimension. Indeed, the state value at the instant k is stored to compute the scalable part at instant $k+1$ (in the variable `Old_xc[i]`). Notwithstanding the implementation memory consumption, this approach seems to be the more suitable for the any time paradigm since the model reduction itself is based on computational complexity scalability.

Series connection. Choosing a series connection, only the minimal controller $C_1(z)$ (or $K_1(z)$) has the updating equations as (1.2), while the generic controller $C_i(z)$ has

$$\left\{ \begin{array}{lcl} x_{c_i}^{+} & = & A_{c_i}\, x_{c_i} + B_{c_i}\, y_{c_{i-1}} \\ y_{c_i} & = & C_{c_i}\, x_{c_i} + D_{c_i}\, y_{c_{i-1}}, \end{array} \right.$$

which leads to the open loop any time controller $C_i(z)$

$$\left\{\begin{array}{lcl}
\begin{bmatrix} x_{c_1} \\ x_{c_2} \\ \vdots \\ x_{c_i} \\ \vdots \\ x_{c_N} \end{bmatrix}^{+} & = & \begin{bmatrix} A_{c_1} & 0 & \dots & \dots & \dots & 0 \\ B_{c_2}\, C_{c_1} & A_{c_2} & 0 & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ B_{c_i}(\prod_{j=i-1}^{2} D_{c_j})C_{c_1} & B_{c_i}(\prod_{j=i-1}^{3} D_{c_j})C_{c_2} & \dots & A_{c_i} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & \dots & \dots & \dots & 0 \end{bmatrix} \begin{bmatrix} x_{c_1} \\ x_{c_2} \\ \vdots \\ x_{c_i} \\ \vdots \\ x_{c_N} \end{bmatrix} + \\
 & & + \begin{bmatrix} B_{c_1} \\ B_{c_2}\, D_{c_1} \\ \vdots \\ B_{c_i}\, (\prod_{j=i-1}^{1} D_{c_j}) \\ \vdots \\ 0 \end{bmatrix} (r - C_s\, x_s) \\
 & & \\
y_{c_i} & = & \begin{bmatrix} (\prod_{j=i}^{2} D_{c_j})C_{c_1} & (\prod_{j=i}^{3} D_{c_j})C_{c_2} & \dots & C_{c_i} & \dots & 0 \end{bmatrix} \begin{bmatrix} x_{c_1} \\ x_{c_2} \\ \vdots \\ x_{c_i} \\ \vdots \\ x_{c_N} \end{bmatrix} + \\
 & & +(\prod_{j=i}^{1} D_{c_j})(r - C_s\, x_s)
\end{array}\right.$$

It is worth noting that the series connection has *sleeping states* too, that are a common feature of the any time paradigm.

Once each controller has been established, the closed loop state space dimension is

$$n_s + \sum_{i=1}^{N} n_{c_i}\ ,$$

that is equal to the model reduction as long as $\sum_{i=1}^{N} n_{c_i} = n_k$ (as in the lucky example in section 1.1).

The closed loop dynamics is then

$$\left\{\begin{array}{rcl}
\begin{bmatrix} x_s \\ x_{c_1} \\ \vdots \\ x_{c_i} \\ \vdots \\ x_{c_N} \end{bmatrix}^+ & = & \begin{bmatrix} A_s - B_s(\prod_{j=i}^{1} D_{c_j})C_s & B_s(\prod_{j=i}^{2} D_{c_j})C_{c_1} & \dots & B_s\,C_{c_i} & \dots & 0 \\ -B_{c_1}\,C_s & A_{c_1} & 0 & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -B_{c_i}(\prod_{j=i-1}^{1} D_{c_j})C_s & B_{c_i}(\prod_{j=i-1}^{2} D_{c_j})C_{c_1} & \dots & A_{c_i} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} x_s \\ x_{c_1} \\ \vdots \\ x_{c_i} \\ \vdots \\ x_{c_N} \end{bmatrix} \\
 & + & \begin{bmatrix} B_s(\prod_{j=i}^{1} D_{c_j}) \\ B_{c_1} \\ \vdots \\ B_{c_i}(\prod_{j=i-1}^{1} D_{c_j}) \\ \vdots \\ 0 \end{bmatrix} r \\
y & = & \begin{bmatrix} C_s & 0 & \dots & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} x_s \\ x_{c_1} \\ \vdots \\ x_{c_i} \\ \vdots \\ x_{c_N} \end{bmatrix}
\end{array}\right.$$

Going into the details of the implementation code

$$\begin{array}{l}
\left.\begin{array}{l}
\texttt{read inputs(r, y);} \\
\texttt{yc[1] := Cc1*xc[1] + Dc1*(r - y);} \\
\texttt{xc[1] := Ac1*xc[1] + Bc1*(r - y);} \\
\texttt{write outputs(yc[1]);}
\end{array}\right\}\text{Mandatory part} \\[1em]
\left.\begin{array}{l}
\texttt{yc[2] := Cc2*xc[2] + Dc2*yc[1];} \\
\texttt{xc[2] := Ac2*xc[2] + Bc2*yc[1];} \\
\texttt{write outputs(yc[2]);}
\end{array}\right\}\text{Optional part 1} \\
\quad\vdots \\
\left.\begin{array}{l}
\texttt{yc[i] := Cc2*xc[i] + Dc2*yc[i-1];} \\
\texttt{xc[i] := Ac2*xc[i] + Bc2*yc[i-1];} \\
\texttt{write outputs(yc[i]);}
\end{array}\right\}\text{Optional part } i \\
\quad\vdots
\end{array} \tag{1.4}$$

In this case (series connection), the scalability and the any time architecture of the implementation are directly obtained and easily traceable.

Parallel connection. On the other hand, the parallel connection has all the controllers computed with

$$\left\{\begin{array}{rcl} x_{c_i}^+ & = & A_{c_i}\,x_{c_i} + B_{c_i}\,(r - C_s\,x_s) \\ y_{c_i} & = & C_{c_i}\,x_{c_i} + D_{c_i}\,(r - C_s\,x_s) \end{array}\right. \tag{1.5}$$

(each controller receives the same error input) and the open loop any time controller $K_i(z)$ is computed with

$$\left\{\begin{array}{lcl} \begin{bmatrix} x_{c_1}\\ x_{c_2}\\ \vdots \\ x_{c_i}\\ \vdots\\ x_{c_N}\end{bmatrix}^+ & = & \begin{bmatrix} A_{c_1} & 0 & \dots & \dots & \dots & 0\\ 0 & A_{c_2} & 0 & \dots & \dots & 0\\ \vdots&\vdots&\vdots&\vdots&\vdots&\vdots\\ 0 & \dots & 0 & A_{c_i} & \dots & 0\\ \vdots&\vdots&\vdots&\vdots&\vdots&\vdots\\ 0&\dots&\dots&\dots&\dots&0\end{bmatrix} \begin{bmatrix} x_{c_1}\\ x_{c_2}\\ \vdots \\ x_{c_i}\\ \vdots\\ x_{c_N}\end{bmatrix} + \begin{bmatrix} B_{c_1}\\ B_{c_2}\\ \vdots\\ B_{c_i}\\ \vdots\\ 0\end{bmatrix}(r - C_s\, x_s)\\ y_{c_i} & = & \left[\begin{array}{cccccc} C_{c_1} & C_{c_2} & \dots & C_{c_i} & \dots & 0\end{array}\right] \begin{bmatrix} x_{c_1}\\ x_{c_2}\\ \vdots \\ x_{c_i}\\ \vdots\\ x_{c_N}\end{bmatrix} + (\sum_{j=i}^{1} D_{c_j})(r - C_s\, x_s) \end{array}\right. .$$

A closed loop state space representation (for the controller $K_i(z)$) is

$$\left\{\begin{array}{lcl} \begin{bmatrix} x_s\\ x_{c_1}\\ \vdots \\ x_{c_i}\\ \vdots\\ x_{c_N}\end{bmatrix}^+ & = & \begin{bmatrix} A_s - B_s(\sum_{j=i}^{1} D_{c_j})C_s & B_s\, C_{c_1} & \dots & B_s\, C_{c_i} & \dots & 0\\ -B_{c_1}\, C_s & A_{c_1} & 0 & \dots & \dots & 0\\ \vdots&\vdots&\vdots&\vdots&\vdots&\vdots\\ -B_{c_i}\, C_s & 0 & \dots & A_{c_i} & \dots & 0\\ \vdots&\vdots&\vdots&\vdots&\vdots&\vdots\\ 0&0&\dots&0&\dots&0\end{bmatrix} \begin{bmatrix} x_s\\ x_{c_1}\\ \vdots \\ x_{c_i}\\ \vdots\\ x_{c_N}\end{bmatrix} + \\ & & + \begin{bmatrix} B_s(\sum_{j=i}^{1} D_{c_j})\\ B_{c_1}\\ \vdots\\ B_{c_i}\\ \vdots\\ 0\end{bmatrix} r \\ y & = & \left[\begin{array}{cccccc} C_s & 0 & \dots & \dots & \dots & 0\end{array}\right] \begin{bmatrix} x_s\\ x_{c_1}\\ \vdots \\ x_{c_i}\\ \vdots\\ x_{c_N}\end{bmatrix} \end{array}\right.$$

with an implementation code

```
read inputs(r, y);                                 }
yc[1] := Cc1*xc[1] + Dc1*(r - y);                  } Mandatory part
xc[1] := Ac1*xc[1] + Bc1*(r - y);                  }
write outputs(yc[1]);                              }

yc[2] := Cc2*xc[2] + Dc2*(r - y) + yc[1];          }
xc[2] := Ac2*xc[2] + Bc2*(r - y);                  } Optional part 1
write outputs(yc[2]);                              }
   ⋮
yc[i] := Cc2*xc[i] + Dc2*(r - y) + yc[i-1];        }
xc[i] := Ac2*xc[i] + Bc2*(r - y);                  } Optional part i
write outputs(yc[i]);                              }
   ⋮
```

Nested loop connection. Regardless of the connection in the feedback rather than feedforward path, the open loop state space representation and the code skeleton of the parallel connection is the same (the difference is tha controller input that is y rather $r - y$). A closed loop state space representation (for the controller $K_i(z)$) is

$$\left\{\begin{array}{rcl}
\begin{bmatrix} x_s \\ x_{c_1} \\ \vdots \\ x_{c_i} \\ \vdots \\ x_{c_N} \end{bmatrix}^+ & = & \begin{bmatrix} A_s + B_s(\sum_{j=i}^{1} D_{cj})C_s & -B_s\, C_{c_1} & \dots & -B_s\, C_{c_i} & \dots & 0 \\ B_{c_1}\, C_s & A_{c_1} & 0 & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ B_{c_i}\, C_s & 0 & \dots & A_{c_i} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} x_s \\ x_{c_1} \\ \vdots \\ x_{c_i} \\ \vdots \\ x_{c_N} \end{bmatrix} + \\
 & & + \begin{bmatrix} B_s \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix} r \\
y & = & \begin{bmatrix} C_s & 0 & \dots & \dots & \dots & 0 \end{bmatrix} \begin{bmatrix} x_s \\ x_{c_1} \\ \vdots \\ x_{c_i} \\ \vdots \\ x_{c_N} \end{bmatrix}
\end{array}\right.$$

Comments. Finally, some comments should be carried out on the implementation code skeletons presented. First of all, each code segment should be thought as the core part of a controller procedure, activated by a real time clock (periodic task).

Moreover, the line code

```
read inputs(r, y);
```

represents a read operation from an I/O port (e.g. an A/D converter or a digital port), equal to an access to a memory buffer. It is worthwhile to note that only one read operation of the quantity $e = r - y$ could be performed at a time, but we want to emphasize that the process output y and the reference signal r are usually generated by different process and may be of different typology (e.g. a trajectory planner implemented as an algorithm on a digital machine).

For the output operations

```
write outputs(yc[i]);
```

the write command is regarded as a register recording (e.g. a saving operation on a D/A memory buffer) whose output is available for the process as the task deadline is reached. It is worthwhile to note that these read/write operations are a common, constant overhead for each controller computation. Furthermore, the $C_i(z)$ controller computation is effectively stored if the computation has been terminated. Strictly speaking, the $C_i(z)$ controller implementation code should be changed to

```
yc[i] := Cc2*xc[i] + Dc2*(r - y);
delta_xc[i] := Ac2*xc[i] + Bc2*(r - y);

write outputs(yc[i]);  }
xc[i] := delta_xc[i];  } atomic
```

where the last two operations are *atomic* (not difficult to ensure since the latter operations are memory assignment, virtually performed in zero time).

1.5 Different approaches to the any time paradigm

In section 1.4 the meaningful architectures for the any time controller framework are discussed both from logical connections and from code implementations. The approaches that fulfill the assumptions on the any time feasibility and maintain more degrees of freedom with respect to state space model are the series and parallel connections. Therefore, in what follows the logical connections are not considered anymore and series (or parallel) connections are given for granted.

Given a particular discrete time system $G(z)$ to control and a set of performances $\hat{P}$ to satisfy (or, equivalently, a target controller $K(z)$), the any time paradigm defines the control and implementation issues in order to obtain the system control inputs in any given computational time Δt (provided that Δt is equal to or greater than the mandatory computational time). For, the controller is split into a series of controllers $K_i(z)$, with $i = 1, \ldots, N$, where $K_1(z)$ is the simplest, less computational demanding controller that in general only stabilizes the system, and $K_N(z)$ ensures all the performances $\hat{P}$ (or, equivalently, is equal to or "similar" to the target controller $K(z)$). Each controller $K_i(z)$, with $i = 1, \ldots, N$, is made by a logical connection of elementary controllers $C_j(z)$, with $j = 1, \ldots, N_c$.

Full Exploitation Scalability (FES). As said in previous sections, to achieve scalability and computational time reduction it is desirable that some component controllers $C_j(z)$ belong to more than one controller $K_i(z)$, in order to preserve already performed calculations and improve time computational savings. In this sense, "optimal" scalability is achieved if:

1. $C_1(z) = K_1(z)$;

2. $N = N_c$;

3. For each $i = 2, \ldots, N$, if a general connection of component controllers $C_1(z)$ to $C_{i-1}(z)$ constitutes the controller $K_{i-1}(z)$, than the controller $K_i(z)$ is constituted by the same connection among the component controllers $C_1(z)$ to $C_i(z)$.

If all of the above properties are satisfied, the any time paradigm is perfectly satisfied and named *Full Exploitation Scalability* (FES) as it exploits all the previously performed computations. In order to analyze the main characteristics of FES, SISO systems with series connections are taken into account (refer to (1.4) for a code skeleton example). Nevertheless, extensions to MIMO systems and other logical connections can be easily achieved with minor adjustments. Hence, considering the FES approach with series connection for SISO systems, the main advantage of the proposed technique relies on the simplicity of the controller design as the use of systematic techniques is naturally supported.

On the other hand, the use of systematic design techniques (e.g. H_2, H_∞, LQR, LQG) for the implementation of each component controllers $C_i(z)$, ensuring at each step a new performance $\hat{P}_{i-1}$, may also represent a drawback. Indeed, the pole placement for each controller $K_i(z)$, that satisfy the performance set $(\hat{P}_1, \ldots, \hat{P}_{i-1})$, does not take into account that an additional component controller $C_{i+1}(z)$ will be designed and connected in series to obtain the controller $K_{i+1}(z)$, satisfying the additional performance $\hat{P}_i$. In this sense, the already placed singularities at step i are in general a drawback at step $i+1$ and force a more complex, with more control effort, controller $C_{i+1}(z)$. Therefore, also the final controller $K_N(z)$ is in general greater (i.e. with a larger number of singularities) and less efficient than the target controller $K(z)$, designed to satisfy all the performances at a time.

Taking into account the advantages and the drawbacks of the FES approach, three rather different scenarios could be realistically considered for a generic control design:

1. *Optimal Design*: the controllers $K_i(z)$, with $i = 1, \ldots, N$, are minimal and the final controller $K_N(z)$ has the same dimension of the target controller $K(z)$. The minimality is referred to the dimension of the controller, i.e. the number of singularities involved, guaranteeing the set of performances $(\hat{P}_1, \ldots, \hat{P}_{i-1})$.

2. *Worst Design*: all the poles of each component controller $C_i(z)$ obstruct the design of the subsequent component controller $C_{i+1}(z)$.

3. *Standard Design*: each controller $C_i(z)$ is minimal and has only a subset of poles that obstruct the design of the controller $C_{i+1}(z)$. The minimality is referred to the dimension of the i–th controller designed for the system augmented with the obstructing poles.

Unfortunately, using systematic control design techniques the *Worst Design* case is the more probable. With more efforts requested to the controller designer, *Standard Design* can be achieved. The *Optimal Design* is obtained in particularly lucky situations (see the example in section 1.1).

Both scalability and computational efforts are key points of the any time paradigm. As it is evident from the analysis of the FES approach, there is a trade–off among scalability, computational time, simplicity of the controllers design and their complexity with respect to performance satisfaction and control cheapness. For generic systems, the FES approach is extremely suitable for the any time paradigm once an *Optimal Design* can be undertaken. For *Standard Design* and, particularly, for *Worst Design* cases more aspects should be investigated to get rid from this leap in the dark.

Mutual Exclusion Scalability (MES). Consider an any time controller where each component controller poles obstruct the design of the subsequent component controllers. The controllers $K_i(z)$ design is then difficult, assuming that a performance scalability is taken into account. Furthermore, the complexity of each controller grows together with the control effort. Instead of compensating the negative effects of the previously added poles, consider an independent design for each controller $K_i(z)$, leaded by a performance scalability. The design of each controller is then stand-alone and fits the performance set. The series connection of the controllers is then obtained noting that $K_i(z) = C_i(z)\ C_{i-1}(z) \ldots C_1(z)$, that easily carries to component controller $C_i(z) = K_i(z)\ C_1(z)^{-1}\ C_{i-1}(z)^{-1}$. Even in the case that the causality of each controller would be satisfied, this approach leads to unnecessary computational burden, since the computation of each component controller firstly eliminates the effects of previously computed controllers and then adds its control job. Rather than a series connection, a switching architecture (see figure 1.5) seems to be more suitable, in this case a set of mutually exclusive controllers (multi-version controller) is defined.

The switching connection is the simplest any time architecture, that could be regarded as an end point of a variety of choices, where each controller is simply substituted to the previously computed controllers in a single closed loop connection. Therefore, the scalability concept is simply drawn to its extreme point, discarding all the previous computations, but it also represents the design maximum freedom for each independent controller. Although this *Mutual Exclusion Scalability* (MES) approach could lead to large state spaces, this is not always the maximum computational wasted architecture.

More precisely, MES is achieved if:

1. $K_N(z) = K(z)$ where $K(z)$ is the target controller;
2. $N_c = 0$;
3. For each $i = 2, \dots, N$, each $K_i(z)$ is a stand-alone controller, without any component controller.

Let us take a look to the code implementation of MES with mutually exclusive single closed loop. For each controller $K_i(z)$, $u = y_{k_i}$ and $u_{k_i} = r - y$

$$\begin{cases} x_{k_i}^+ &= A_{k_i}\, x_{k_i} + B_{k_i}\, (r - C_s\, x_s) \\ y_{k_i} &= C_{k_i}\, x_{k_i} + D_{k_i}\, (r - C_s\, x_s). \end{cases}$$

The closed loop state space realization under the i–th controller is

$$\begin{cases} \begin{bmatrix} x_s \\ x_{k_i} \end{bmatrix}^+ &= \begin{bmatrix} A_s - B_s\, D_{k_i}\, C_s & B_s\, C_{k_i} \\ -B_{k_i}\, C_s & A_{k_i} \end{bmatrix} \begin{bmatrix} x_s \\ x_{k_i} \end{bmatrix} + \begin{bmatrix} B_s\, D_{k_i} \\ B_{k_i} \end{bmatrix} r \\ y &= \begin{bmatrix} C_s & 0 \end{bmatrix} \begin{bmatrix} x_s \\ x_{k_i} \end{bmatrix}. \end{cases}$$

Although the MES is the most simple structure, it has a state space of dimension

$$n_s + \sum_{i=1}^{N} n_{k_i}$$

where n_{k_i} is the state space dimension of a minimal realization of the controller $K_i(z)$.

The generic closed loop state space dynamics will be

$$\begin{cases} \begin{bmatrix} x_s \\ x_{k_1} \\ \vdots \\ x_{k_i} \\ \vdots \\ x_{k_N} \end{bmatrix}^+ &= \begin{bmatrix} A_s - B_s\, D_{k_i}\, C_s & 0 & \dots & B_s\, C_{k_i} & \dots & 0 \\ 0 & 0 & \dots & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -B_{k_i}\, C_s & 0 & \dots & A_{k_i} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} x_s \\ x_{k_1} \\ \vdots \\ x_{k_i} \\ \vdots \\ x_{k_N} \end{bmatrix} + \begin{bmatrix} B_s\, D_{k_i} \\ 0 \\ \vdots \\ B_{k_i} \\ \vdots \\ 0 \end{bmatrix} r \\ y &= \begin{bmatrix} C_s & 0 & \dots & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} x_s \\ x_{k_1} \\ \vdots \\ x_{k_i} \\ \vdots \\ x_{k_N} \end{bmatrix} \end{cases}$$

representing the whole state space. The states of the controllers $K_1(z)$ through $K_{i-1}(z)$ that are computed and discarded become part of the *sleeping states.*

Due to unpredictable events but satisfying the feasibility of the mandatory part, the scheduler will execute one of the possible controller $K_i(z)$ at any task period iteration. An example of the code skeleton for the MES approach would be:

```
read inputs(r, y);
yk[1] := Ck1*xk[1] + Dk1*(r - y);   } Mandatory part
xk[1] := Ak1*xk[1] + Bk1*(r - y);
write outputs(yk[1]);

yk[2] := Ck2*xk[2] + Dk2*(r - y);
xk[2] := Ak2*xk[2] + Bk2*(r - y);   } Optional part 1
write outputs(yk[2]);
restore xk[1];
    ⋮
yk[i] := Cki*xk[i] + Dki*(r - y);
xk[i] := Aki*xk[i] + Bki*(r - y);   } Optional part i
write outputs(yk[i]);
restore xk[i-1];
    ⋮
yk[N] := CkN*xk[N] + DkN*(r - y);
xk[N] := AkN*xk[N] + BkN*(r - y);   } Optional part N
write outputs(yk[N]);
restore xk[N-1];
```

Analyzing the controller code implementation, it is worth noting that the *sleeping states* are restored (`restore xc[i-1];`), discarding the previously computed values. Keeping instead of discarding the computed evolution may be of interest too, but, at this point there is no evidence on what is better to choose.

Albeit the scalability is not so evident, it is trivial that all the any time constraints defined in section 1.2 are satisfied by this implementation. Nevertheless, this approach pays its extreme simplicity in possible memory wasting, unacceptable overheads and excessive computational efforts to complete. Indeed, the last four code lines, i.e. the code implementation of the target controller $K(z) = K_N(z)$ equal to (1.3), are executed after the computational overheads of the smaller controllers $K_1(z)$ to $K_{N-1}(z)$.

It is worthwhile to note that the assumptions on atomicity should be extended to the code lines

```
write outputs(yc[i]);  } atomic.
restore xc[i-1];
```

Considering SISO systems, the main advantages of the MES technique rely on:

1. Maximal flexibility on the controller design. $K_i(z)$ is completely independent from $K_{i+1}(z)$.

2. The controller state space realizations are also unconstrained. Therefore, unifying the degrees of freedom of design and realization, more complex characteristics can be taken into account, e.g. switching quadratic stability or minimum norm controllers.

Drawbacks are related to:

1. Absence of incremental computation scalability.

2. In general it is a worsening in the state space dimension and in computational time if an *Optimal design* or even a *Standard design* can be carried out with an FES approach.

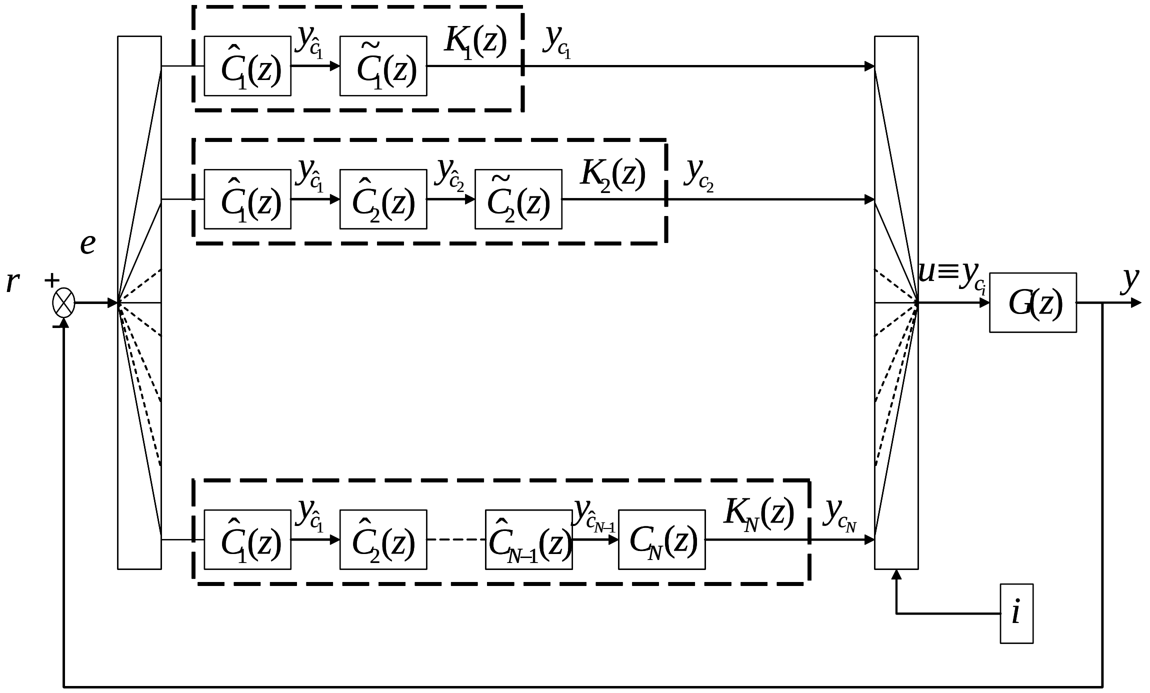


Figure 1.11: Controllers (series) connection for the *Partial Exploitation Scalability* approach.

The MES has its main attractiveness in the independent design and state space representation. Due to the controllers design simplicity and to the scalability features involved (the "extreme" concept of scalability), the MES approach can be regarded as a comparison benchmark for all the possible any time architectures, easily implementable in any possible situation.

Partial Exploitation Scalability (PES). It is straightforward that *Optimal Design* implies the choice of FES to achieve an optimal any time paradigm satisfaction. On the other hand, in the *Worst Design* case MES is the right choice. Between this two extreme points (both from a scalability and computational complexities variety of choices) there are a lot of different situations where the solution is not so easily pointed out.

"In medio tutissimus ibis" ("In the middle of things you will go most safe", Ovidio): a more reliable approach is just to aim between FES and MES. The *Partial Exploitation Scalability* (PES) approach preserves some singularities in switching among the controllers and discards the others. A logical scheme is represented in figure 1.11, showing an intuitive visualization of the connections for the series case. The switching index is controlled as in the other connections.

Let p_i and z_i be the sets of poles and zeros respectively of the component controller $C_i(z)$. Each singularity set is split into two parts

$$p_i = (\hat{p}_i, \tilde{p}_i) \quad z_i = (\hat{z}_i, \tilde{z}_i) \ ,$$

where $(\tilde{p}_i, \tilde{z}_i)$ are the obstructing singularities. For each $i = 1, \ldots, N-1$, the component controller singularities impose a logical subdivision $C_i(z) = \tilde{C}_i(z)\ \hat{C}_i(z)$. In order to preserve already performed calculations and improve time computational savings, but discarding the obstructing poles, PES is achieved if:

1. $K_1(z) = C_1(z) = \tilde{C}_1(z)\ \hat{C}_1(z)$;

2. $N = N_c$;

3. For each $i = 2, \ldots, N$, if a general connection of component controllers $\hat{C}_1(z), \hat{C}_2(z), \ldots, \hat{C}_{i-2}(z)$ and $C_{i-1}(z)$ constitutes the controller $K_{i-1}(z)$, than the controller $K_i(z)$ is constituted by the

same connection among the component controllers $\hat{C}_1(z), \hat{C}_2(z), \ldots, \hat{C}_{i-1}(z)$ (i.e. the non–obstructing parts) and $C_i(z)$.

Consider the open loop dynamics of the PES approach with series connection (see again figure 1.11), dividing the updating equations (i.e. the state space variables x_{c_i}) following the obstructing property (i.e. $x_{c_i} = [\hat{x}_{c_i}, \tilde{x}_{c_i}]^T$). For the first, simplest controller $K_1(z)$

$$\left\{ \begin{array}{rcl} \hat{x}_{c_1}^+ & = & \hat{A}_{c_1}\, \hat{x}_{c_1} + \hat{B}_{c_1}\, (r - C_s\, x_s) \\ \hat{y}_{c_1} & = & \hat{C}_{c_1}\, \hat{x}_{c_1} + \hat{D}_{c_1}\, (r - C_s\, x_s) \\ \tilde{x}_{c_1}^+ & = & \tilde{A}_{c_1}\, \tilde{x}_{c_1} + \tilde{B}_{c_1}\, \hat{y}_{c_1} \\ \tilde{y}_{c_1} & = & \tilde{C}_{c_1}\, \tilde{x}_{c_1} + \tilde{D}_{c_1}\, \hat{y}_{c_1} \end{array} \right. \begin{array}{l} \left.\vphantom{\begin{array}{c} a \\ a \end{array}}\right\} \hat{C}_1(z) \\ \left.\vphantom{\begin{array}{c} a \\ a \end{array}}\right\} \tilde{C}_1(z) \end{array} \tag{1.6}$$

where the controlled system input is $u = y_{k_1} = \tilde{y}_{c_1}$ and the controller input is $u_{k_1} = \hat{u}_{c_1} = r - y$ and where the series connection between the obstructing/non–obstructing parts is emphasized. For each controller $K_i(z)$, with $i = 2, \ldots, N-1$, the open loop dynamics is equal to (1.6), where the controlled system input is $u = y_{k_i} = \tilde{y}_{c_i}$ and the controller input $\hat{u}_{c_i} = \hat{y}_{c_{i-1}}$, i.e. the output of the non–obstructing part. The last controller $K_N(z)$ which is not necessarily split into two parts, furnishes the control $u = y_{k_N} = y_{c_N}$ to the system $G(z)$ and its input is $u_{c_N} = \hat{y}_{c_{N-1}}$. It is worthwhile to note that, as long as the number of obstructing poles decreases, the PES approach converges to the FES. On the other hand, if the number of obstructing poles increases, the PES approaches the MES. Therefore, the closed loop dynamics is straightforward and can be easily obtained from the closed loop scheme of the FES series connection for the non–obstructing poles (i.e. the $\hat{x}_{c_i}$ state variables) and from the closed loop scheme of the MES for the obstructing singularities (i.e. the $\tilde{x}_{c_i}$ state variables). Further extending this "middle approach concept", the PES has *sleeping states* which belong both to a controller non computed part and to discarded states.

The state space dimension of the closed loop PES is

$$n_s + \sum_{i=1}^{N-1} (\tilde{n}_{c_i} + \hat{n}_{c_i}) + n_{c_N} = n_s + \sum_{i=1}^{N} n_{k_i} - \sum_{i=1}^{N} (N-i)\hat{n}_{c_i} \ ,$$

that again highlights how it is possible to reduce to FES (when $\tilde{n}_{c_i} = 0$ in the left part of the relation) and to MES (when $\hat{n}_{c_i} = 0$ in right part of the relation) state space dimensions.

The PES compatible code skeleton is:

```
read inputs(r, y);                                                  }
hat_yc[1] := hat_Cc1*hat_xc[1] + hat_Dc1*(r - y);           } Ĉ1(z) }
hat_xc[1] := hat_Ac1*hat_xc[1] + hat_Bc1*(r - y);           }       } Mandatory part
tilde_yc[1] := tilde_Cc1*tilde_xc[1] + tilde_Dc1*hat_yc[1]; } C̃1(z) }
tilde_xc[1] := tilde_Ac1*tilde_xc[1] + tilde_Bc1*hat_yc[1]; }       }
write outputs(tilde_yc[1]);                                         }

hat_yc[2] := hat_Cc2*hat_xc[2] + hat_Dc2*hat_yc[1];         } Ĉ2(z) }
hat_xc[2] := hat_Ac2*hat_xc[2] + hat_Bc2*hat_yc[1];         }       }
tilde_yc[2] := tilde_Cc2*tilde_xc[2] + tilde_Dc2*hat_yc[2]; } C̃2(z) } Optional part 1
tilde_xc[2] := tilde_Ac2*tilde_xc[2] + tilde_Bc2*hat_yc[2]; }       }
write outputs(tilde_yc[2]);                                         }
restore tilde_xc[1];                                                }
   ⋮
hat_yc[i] := hat_Cci*hat_xc[i] + hat_Dci*hat_yc[i-1];       } Ĉi(z) }
hat_xc[i] := hat_Aci*hat_xc[i] + hat_Bci*hat_yc[i-1];       }       }
tilde_yc[i] := tilde_Cci*tilde_xc[i] + tilde_Dci*hat_yc[i]; } C̃i(z) } Optional part i
tilde_xc[i] := tilde_Aci*tilde_xc[i] + tilde_Bci*hat_yc[i]; }       }
write outputs(tilde_yc[i]);                                         }
restore tilde_xc[i-1];                                              }
   ⋮
yc[N] := CcN*xc[N] + DcN*hat_yc[N-1];                               }
xc[N] := AcN*xc[N] + BcN*hat_yc[N-1];                               } Optional part N
write outputs(yc[N]);                                               }
restore tilde_xc[N-1];                                              }
```

It is worth noting that the *sleeping states* are restored (`restore tilde_xc[i-1];`), discarding the previously computed values. Once again, there are no evidence in performing rather than discarding the "restore" operation. Nevertheless, atomicity of the `write outputs(tilde_yc[i]);` and the restoring operation must be guaranteed.

¿From a code optimization point of view, the computation and the subsequent rejection of *sleeping states* could be better implemented simply delaying the possibly rejectable part computation in the next mandatory part. However, the computational overhead introduced in the mandatory part depends on the last computed controller and could be too expensive.

The PES technique presented as the final, more flexible approach to the any time constraints' satisfaction keeps positive features also for the complexity and computational error sensitivity, due to its degrees of freedom in the state space realization choice.

Considering SISO systems, the main advantages of the PES technique are:

1. The any time paradigm is naturally and completely fulfilled. Within the PES approach, all the scalability possibilities are preserved and taken into account.

2. The controller state space realizations are unconstrained. Therefore, complexity of the implementation can be taken into account.

Remark 2 *Consider a scalable any time controller $K_1(z), \ldots, K_N(z)$. Each component controller $C_i(z)$ is split according to the PES approach and realized in a generic state space representation. If there are common poles between the component controller $C_{i-1}(z)$ and $C_{i+1}(z)$ but they are obstructing poles for the component controller $C_i(z)$, they have to be computed two times. This restrictive property* *can be better explained noting that $\tilde{K}_i(z) = \tilde{C}_i(z)$ and $\hat{K}_i(z) = \hat{C}_i(z)\ \hat{C}_{i-1}(z)\dots\hat{C}_1(z)$, in order to obtain $K_i(z) = \tilde{K}_i(z)\ \hat{K}_i(z)$. Therefore*

$$\hat{C}_j(z) \in \hat{K}_i(z)\ \forall i = 1,\dots,N\ ,\forall j = 1,\dots,i$$

This constraint persists also if a realization in Jordan canonical form is chosen.

It has to be noted that a code optimization on the specific any time realization should be pursued to overcome this problem.

Drawbacks are related to:

1. A controller design that is able to fulfill the separation between obstructing and non–obstructing parts is very difficult. Moreover, systematic techniques are not well suited for the design of a controller satisfying the additional splitting constraint.

2. The causality of each split controller $\hat{C}_i(z)$ and $\tilde{C}_i(z)$ must be preserved. Therefore the separation among obstructing parts cannot be arbitrarily performed.

Hence, the PES represents the more flexible architecture for the any time paradigm that naturally guides the realization of each controller toward the FES, rather than MES, depending on the controllers design.

1.5.1 Complexity comparison among FES, MES and PES

Regardless of the adopted design technique, a comparison among the different approaches will be carried out in this section. As emerged in section 1.3, the best choice for a controller realization seems to be the Jordan canonical form, easily implementable in each of the scalability approaches presented (FES, MES, PES).

For simplicity's sake, the comparison among the different approaches is guided by the subsequent assumptions:

1. Each controller $K_i(z)$ and its relative component controllers $\hat{C}_i$ and $\tilde{C}_i(z)$ are realized in Jordan form. With this choice, complexity comparison is performed assuming the computational time amount equal to the state space dimension of the controller;

2. Each realization is minimal;

3. The input vector B is filled with 1's and 0's, while the output vector C is completely filled with rational numbers (only SISO systems are considered);

Remark 3 *It is worthwhile to note that FES implies $\tilde{C}_i(z) = I\ \forall i$ (I is the identity matrix of suitable dimensions), MES implies $\hat{C}_i(z) = I\ \forall i$, PES has no constraint.*

Remark 4 *For SISO systems, the input vector b or the output vector c could have almost any desired values as long as the realization is chosen to be reachable and observable. For MIMO systems, this assumption is still valid provided that the realization is no longer minimal (see [14, 13] for more details).*

Consider a set of stand-alone controllers $K_i(z)$, with $i = 1,\dots,N$. The controller $K_1(z)$ ensures the stability of the system $G(z)$ (in case $G(z)$ is already stable, it is $K_1(z) = I$), while the controller $K_i(z)$ ensure the performance set $(\hat{P}_1, \hat{P}_2, \dots, \hat{P}_{i-1})$, with $i = 2,\dots,N$.

The less computational efficient any time technique is to compute the i–th controller after that all the controllers from 1 to $i-1$ are computed and then discarded. The underlying assumption of this method is that there are no common singularities among the controllers (i.e. $\hat{K}_i(z) = I \; \forall i$), consequently the computation furnishes the last computed control $K_i(z)$ and discards all the previously, uselessly computed controls. The maximum cost for the i–th controller is then

$$\bar{C}_i = \sum_{j=1}^{i} \underline{n}_{k_i} \tag{1.7}$$

where $\underline{n}_{k_i}$ is the state space dimension of the controller $K_i(z)$. The maximum cost is then equal to the cost of the MES approach ${}^{mes}C_i = \bar{C}_i$. Any other technique which admits common singularities should have a computational cost less then or at least equal to the maximum allowable cost ${}^{mes}C_i$. So, the MES approach defines an upper bound on the computational cost limit: any other approach that has a computational cost greater than this easily computable limit is less efficient and should be avoided.

Let us now consider a PES approach and assume that n_{k_i} is the dimension of the controller $K_i(z)$. Due to the minimality of the controllers in the MES approach (recall that each controller is designed to be stand-alone), $n_{k_i} \geq \underline{n}_{k_i}$, but this property is not in contrast with the fundamental requisite ${}^{pes}C_i \leq {}^{mes}C_i$ that permits to PES to be more attractive than MES. For, consider the dimension of each controller (for instance, in a series connection)

$$K_i(z) = \tilde{K}_i(z)\hat{K}_i(z) \Rightarrow n_{k_i} = \tilde{n}_{k_i} + \hat{n}_{k_i} \geq \underline{n}_{k_i} \; \forall i$$

where $\tilde{n}_{k_i}$ singularities are computed uselessly at each step. For example, consider the component controllers $C_i(z) = \tilde{C}_i(z)\hat{C}_i(z)$ and a series connection among them

$$\begin{aligned} \tilde{K}_i(z) &= \tilde{C}_i(z) \\ \hat{K}_i(z) &= \hat{C}_i(z)\,\hat{C}_{i-1}(z)\ldots\hat{C}_1(z) \end{aligned}$$

For the computation of the cost index ${}^{pes}C_i$ it is necessary to take into account the partial superimposition among the controllers

$$\begin{array}{lcccl} n_{k_1} & = & \tilde{n}_{k_1} + \hat{n}_{k_1} & & \\ n_{k_2} & = & \tilde{n}_{k_2} + \hat{n}_{k_2} & = & {}^{a}n_{k_2} + \hat{n}_{k_1} \geq \underline{n}_{k_2} \\ n_{k_3} & = & \tilde{n}_{k_3} + \hat{n}_{k_3} & = & {}^{a}n_{k_3} + \hat{n}_{k_2} \geq \underline{n}_{k_3} \\ \vdots & & & & \\ n_{k_i} & = & \tilde{n}_{k_i} + \hat{n}_{k_i} & = & {}^{a}n_{k_i} + \hat{n}_{k_{i-1}} \geq \underline{n}_{k_i} \\ \vdots & & & & \end{array}$$

where ${}^{a}n_{k_i}$ is the dimension of the added singularities to the existing common singularities $\hat{n}_{k_{i-1}}$ in order to achieve the total number of singularities of $K_i(z)$. Due to the scalability property, the common parts are already computed and the global computational cost of the i–th controller is then

$${}^{pes}C_i = n_{k_1} + \sum_{j=2}^{i} {}^{a}n_{k_j}$$

Noting that $n_{k_1} = \underline{n}_{k_1}$, since the smaller controller ensures only the stability, it is possible to assert that ${}^{pes}C_i \leq {}^{mes}C_i$ if and only if the relation

$$\sum_{j=2}^{i} {}^{a}n_{k_j} \leq \sum_{j=2}^{i} \underline{n}_{k_j}$$

is verified $\forall i = 2, \ldots, N$, which implies

$${}^{a}n_{k_i} \leq \underline{n}_{k_i} \; \forall i = 2, \ldots, N \tag{1.8}$$

It is straightforward that FES is the end point of the presented variety of choices. Indeed, it simply uses all the calculations performed, regardless of their usefulness. Each controller $K_i(z)$ designed with this approach will be minimal, but the presence of the obstructing poles in $K_{i-1}(z)$ will bring to a state space whose dimension is $n^\star_{k_i} \geq n_{k_i} \geq \underline{n}_{k_i}$, in general larger than any other representation. As it was highlighted for the PES, also in this approach the controllers are logically split into two parts based on the obstructing/non–obstructing property of the singularities involved. Although the PES approach discarded all the obstructing poles, in the FES all the singularities are reused: the partition is highlighted only for comparison clarity. The dimension of each controller is

$$n^\star_{k_i} = \tilde{n}^\star_{k_i} + \hat{n}^\star_{k_i} \geq \underline{n}_{k_i} \ \forall i$$

and the computation of the cost index ${}^{fes}C_i$ is carried out in the same way as PES

$$\begin{array}{rcccl}
n^\star_{k_1} & = & \tilde{n}_{k_1} + \hat{n}_{k_1} & = & n_{k_1} = \underline{n}_{k_1} \\
n^\star_{k_2} & = & \tilde{n}^\star_{k_2} + \hat{n}^\star_{k_2} & = & {}^{a}n^\star_{k_2} + \hat{n}_{k_1} + \tilde{n}_{k_1} \geq n_{k_2} \geq \underline{n}_{k_2} \\
n^\star_{k_3} & = & \tilde{n}^\star_{k_3} + \hat{n}^\star_{k_3} & = & {}^{a}n^\star_{k_3} + \hat{n}^\star_{k_2} + \tilde{n}^\star_{k_2} \geq n_{k_3} \geq \underline{n}_{k_3} \\
\vdots & & & & \\
n^\star_{k_i} & = & \tilde{n}^\star_{k_i} + \hat{n}^\star_{k_i} & = & {}^{a}n^\star_{k_i} + \hat{n}^\star_{k_{i-1}} + \tilde{n}^\star_{k_{i-1}} \geq n_{k_i} \geq \underline{n}_{k_i} \\
\vdots & & & &
\end{array}$$

where ${}^{a}n^\star_{k_i}$ is again the dimension of the added dynamics to the existing singularities $n^\star_{k_{i-1}}$ in order to achieve the desired behavior for $K_i(z)$. Due to the presence of the obstructing poles, the added singularities are split into two parts: the singularities added to increase the performance and the singularities added to compensate the presence of obstructing poles at the step $i-1$

$${}^{a}n^\star_{k_i} = {}^{a}\hat{n}^\star_{k_i} + {}^{a}\tilde{n}^\star_{k_i}$$

Recalling that $n^\star_{k_1} = \underline{n}_{k_1}$, the cost index is

$${}^{fes}C_i = \underline{n}_{k_1} + \sum_{j=2}^{i} ({}^{a}\hat{n}^\star_{k_j} + {}^{a}\tilde{n}^\star_{k_j})$$

and the condition ${}^{fes}C_i \leq {}^{mes}C_i$ is verified if and only if

$$\sum_{j=2}^{i} ({}^{a}\hat{n}^\star_{k_j} + {}^{a}\tilde{n}^\star_{k_j}) \leq \sum_{j=2}^{i} \underline{n}_{k_j} \tag{1.9}$$

The comparison between FES and PES is not so easily identifiable, since the set of obstructing poles at step $i-1$ for the controller $K_i(z)$ may be non–obstructing for the controller $K_{i+1}(z)$ and then keeping them may be an improvement in efficiency. Considering only computational complexity FES can be regarded as a special case which PES naturally tends to, nevertheless a trade–off will appear once design effort and obstructing singularities identification are also taken into account. The overall comparison dramatically depends on the particularity of the application and it is no longer investigate in this work. However, the guide lines for the choice of an approach in the proposed set are given analyzing the scenarios presented in this section and assuming that the obstructing property is a persistent characteristic, in order to get

$${}^{fes}C_i = {}^{pes}C_i + \sum_{j=2}^{i} {}^{a}\tilde{n}^\star_{k_j}$$

Fixing the upper complexity bound with ${}^{mes}C_i$ in (1.7), three main situations are analyzed for comparison purposes:

1. *Optimal Design*: noting that

$$ {}^{a}n^{\star}_{k_i} = {}^{a}\hat{n}^{\star}_{k_i} = {}^{a}n_{k_i} $$

the PES is equal to the FES, but the latter is preferable since no analysis is carried out in the identification of the obstructing poles. Furthermore

$$ \underline{n}_{k_i} = n_{k_i} = n^{\star}_{k_i} $$

for the minimality of MES and the minimality of FES w.r.t. Optimal Design.

The cost index for the FES is

$$ {}^{fes}C_i = {}^{pes}C_i = \underline{n}_{k_1} + \sum_{j=2}^{i} {}^{a}n_{k_j} = \underline{n}_{k_i} $$

and then it is easy to assert that ${}^{fes}C_i < {}^{mes}C_i$ since

$$ \underline{n}_{k_i} < \sum_{j=1}^{i} \underline{n}_{k_j} $$

Equivalently, expressing the upper bound index with respect to the non–obstructing parts

$$ {}^{mes}C_i = \sum_{j=1}^{i} \underline{n}_{k_j} = \underline{n}_{k_1} + \sum_{j=2}^{i} ({}^{a}n_{k_j} + \hat{n}_{k_{j-1}}) = \underline{n}_{k_1} + \sum_{j=2}^{i} ({}^{a}n_{k_j} + n_{k_{j-1}}) $$

the condition (1.9) is verified since

$$ ({}^{a}\hat{n}^{\star}_{k_j} + {}^{a}\tilde{n}^{\star}_{k_j}) = {}^{a}n_{k_j} < {}^{a}n_{k_j} + n_{k_{j-1}}, \ \forall j = 2, \ldots, N $$

Therefore, the Optimal Design with FES is the best choice in the best case. ${}^{fes}C_i$ can be regarded as the lower limit of the any time complexity

$$ {}^{fes}C_i = \underline{n}_{k_1} + \sum_{j=2}^{i} {}^{a}n_{k_j} $$

2. *Worst Design*: in this situation

$$ {}^{a}n^{\star}_{k_i} = {}^{a}\hat{n}^{\star}_{k_i} + {}^{a}\tilde{n}^{\star}_{k_i} $$

which yields to

$$ n^{\star}_{k_i} = {}^{a}\hat{n}^{\star}_{k_i} + {}^{a}\tilde{n}^{\star}_{k_i} + \tilde{n}^{\star}_{k_i} > \underline{n}_{k_i} $$

The comparison index (1.9)is then violated $\forall i$ and then ${}^{fes}C_i > {}^{mes}C_i$. Furthermore the cost index for the FES is

$$ {}^{fes}C_i = \underline{n}_{k_1} + \sum_{j=2}^{i} {}^{a}n^{\star}_{k_j} = {}^{pes}C_i + \sum_{j=2}^{i} {}^{a}\tilde{n}^{\star}_{k_i} $$

that highlights that ${}^{fes}C_i > {}^{pes}C_i$. In order to compare the PES with MES, consider the condition (1.8): $\hat{n}_{k_{i-1}} = 0$, $\forall i$ since all the singularities are obstructing, hence ${}^{a}n_{k_j} \geq \underline{n}_{k_i}$ for the minimality of the MES approach, and then ${}^{pes}C_i \geq {}^{mes}C_i$.

Therefore, for Worst Design the best choice is the MES, i.e. discarding all the already performed computations. It is a matter of fact that ${}^{mes}C_i$ is regarded as the upper limit of the any time complexity. Applying the FES in this situation may lead to an excessive computational burden.

3. *Standard Design*: within this most general scenario

$$^{a}n^{\star}_{k_i} = {}^{a}\hat{n}^{\star}_{k_i} + {}^{a}\tilde{n}^{\star}_{k_i}$$

and the cost index for the FES is again

$$^{fes}C_i = \underline{n}_{k_1} + \sum_{j=2}^{i} {}^{a}n^{\star}_{k_j} = {}^{pes}C_i + \sum_{j=2}^{i} {}^{a}\tilde{n}^{\star}_{k_i}$$

that highlights that $^{fes}C_i > {}^{pes}C_i$. It is worthwhile to remember that this condition is valid as long as the non–obstructing property is permanent and when the design problems are not taken into account.

The condition (1.8) should be analyzed carefully depending on the particular situation.

It is worthwhile to note that discarding design problems have leaded to the obvious conclusion of always using the more flexible approach presented: PES. Indeed, it simply converges to one of the two limit choices (MES or FES) depending on the superimposition properties among the controllers. Problems related to controller design are analyzed in the next section.

1.6 Stability issue

The open loop system endowed with an anytime controller can be regarded as a LTI autonomous switched system

$$x^{+}_{cl} = A_{cl_i} x_{cl} \tag{1.10}$$

where x_{cl} is the vector stacking the state vectors of the open loop system and of the controllers, and A_{cl_i} are the closed loop dynamic matrices related to the system and to the controller K_i. This notation is allowed by the introduction of *sleeping states*, that is states related to idle controllers which are frozen or decreased with a power law.

Ensuring the asymptotic stability of a system with an anytime controller is no longer a matter of single controllers, but of all controllers and of the switching law specifying which controller is active in each instant. In other words it is a problem of stability of switched systems. There exists a wide literature on this subject (see [17], [18] and [19] and references therein), but, unfortunately, few results can be used for the anytime problem. First of all, the numerical problems affecting the computations of the controllers impose specific state space realizations, seriously limiting a stability oriented design of the controllers. That is, we cannot choose state space realizations guaranteeing asymptotic stability for arbitrary switching (see [16]). The realizations driven by computational constraints, may result in a stable system for any switching law, but it cannot be systematically ensured. This suggests to face the problem from a stabilizability point of view, that is to design specific switching laws ensuring the stability.

There exist many results also for the stabilizability problem, but they usually require the computation of complex functions to ascertain which subsystem can be activated next time. Since the reduction of computation at each step is a main concern of the anytime paradigm, we cannot make use of a switching logic potentially requiring more computations than the controller itself. Therefore, we present two switching laws based on the norm of closed loop matrices instead of on the online measure of the norm of the state vector. We make the conservative assumption that the scheduler can make preemption, thus changing the available time for computation, at each step. This means that a switching logic, allowing at each step the computation of the maximal controller for the available time, produces a system subject to an arbitrary switching law, hence, potentially unstable. It is worth noting that any switching logic can only further limits the available time granted by the scheduler, deciding to stop the computation for stability reasons before the scheduler has made preemption. The preemption event is mandatory and cannot be delayed.

1.6.1 (N, n) switching logic

The basic idea behind this logic is to exploit the repeated application of the minimal controller (whose complete computation is always guaranteed by the *feasible mandatory constraint*) to tame the expansive behavior of an arbitrary, but time-limited, sequence of controllers. More in depth, this logic opens a time window of n steps, during which the controller order 'follows' the available time, that is computations are interrupted only on the occurrence of a preemption event and the provided controller is the greatest possible for the available time. After that, a sequence of N successive applications of the minimal controller follows, resulting in an overall contractive behavior.

Define with $I = \{1, \dots, N_k\}$ the set of controller indices, with $\mathcal{A}_{cl} = \{A_{cl_i}\}_{i \in I}$ the set of closed loop dynamic matrices, with $\sigma_n = (i_1, \dots, i_n)$, $i_j \in I$, a sequence (string) of n indices, with Σ_n the set of all the strings σ_n, with $A_{cl_{\sigma_n}} = \prod_{\substack{j=1 \\ i_j = \sigma_n(j)}}^{n} A_{cl_{i_j}}$ the dynamic matrix after n steps corresponding to σ_n and with $\mathcal{A}_{cl_{\sigma_n}} = \left\{A_{cl_{\sigma_n}}\right\}_{\sigma_n \in \Sigma_n}$ the set of all previous matrices.

Suppose the present state is x, after an evolution of n time steps following the sequence of indices σ_n, the state will be $x^{n+} = A_{cl_{\sigma_n}} x$. If we do not measure the actual state, we can make a conservative estimation of the norm of the state after n steps considering the initial state x uniformly distributed on a unit sphere. Moreover, if we do not record the specific sequence of indices, we can estimate the effect of the arbitrary sequence on the norm of the state vector by taking the maximum over all sequences

$$B = \max_{\substack{\|x\|_2 = 1 \\ A_{cl_{\sigma_n}} \in \mathcal{A}_{cl_{\sigma_n}}}} \left\|A_{cl_{\sigma_n}} x\right\|_2 = \max_{A_{cl_{\sigma_n}} \in \mathcal{A}_{cl_{\sigma_n}}} \bar{\sigma}\left(A_{cl_{\sigma_n}}\right)$$

where $\bar{\sigma}(\cdot)$ is the maximum singular value. B is an estimation of the maximum expansion of the system after n steps, that is any evolution of the system starting from an initial condition on the unit sphere yields, after n steps, a final point included in the sphere of radius B. If our goal, after $n + N$ steps of evolution, is to produce an overall contraction at the minimum rate of $1 - \varepsilon$ with $0 < \varepsilon < 1$, we must find a number of steps N such that $B \cdot b \leq 1 - \varepsilon$ with

$$b = \bar{\sigma}\left(A_{cl_1}\right)$$

and A_{cl_1} closed loop matrix related to the minimal (mandatory) controller. With this switching law the switched system is guaranteed to be asymptotically stable (the system sampled any $n + N$ steps is power law stable with minimum rate $1 - \varepsilon$).

This logic has the merit to be very simple and to require no online computations (n and N are fixed) or storing data, but has the drawback to be too stiff. At the expanse of some online recordings and little memory consumption, a more flexible switching logic can be defined.

1.6.2 Power sequence switching logic

Suppose to build all the power sequences $A_{cl_i}, A_{cl_i}^2, \dots, A_{cl_i}^{n_i}$ for every $A_{cl_i} \in \mathcal{A}_{cl}$, with n_i chosen such that $\bar{\sigma}\left(A_{cl_i}^{n_i}\right) \leq 1 - \varepsilon$ $(0 < \varepsilon < 1)$. Moreover, for every power sequence consider the sequence of maximum singular values $\bar{\sigma}\left(A_{cl_i}\right), \bar{\sigma}\left(A_{cl_i}^2\right), \dots, \bar{\sigma}\left(A_{cl_i}^{n_i}\right)$ and define the following quantities

$$B_{ij} = \bar{\sigma}\left(A_{cl_i}^{j}\right) \max_{l \in I} \bar{\sigma}\left(A_{cl_l}\right), \qquad j < n_i, \ i \in I.$$

The B_{ij}'s represent an estimation of the expansion induced on the norm of the state by each sequence of $j < n_i$ consecutive applications of the i-th controller followed by a step of an arbitrary controller $l \in I$. As for the previous logic we use the minimal controller to reduce the norm of the state vector. Indeed, we find positive integers N_{ij} such that $B_{ij} \cdot b_{ij} \leq 1 - \varepsilon$ with

$$b_{ij} = \bar{\sigma}\left(A_{cl_1}^{N_{ij}}\right).$$

N_{ij} is the number of consecutive applications of the minimal controller needed to reduce the expansion B_{ij} and to produce a minimal overall contraction of $1-\varepsilon$ after $j+1+N_{ij}$ steps. With these definitions it is now possible to explain the power sequence switching logic as follows:

1. Try to compute the greatest possible controller for the available time (say k the index of this controller);
2. For the following $n_k - 1$ steps try to compute the controller k (the switching logic force the controller k even if the available time would allow the computation of more complex controllers);
3. If a preemption event occurs after r steps (with $r < n_k$) from the beginning of this power sequence, that is if the available time is reduced below the value needed for the computation of the controller k, a step of the greatest possible controller for the reduced available time is computed. For the following N_{ij} steps the minimal controller (index 1) is computed;
4. If the step 2) (n_k time steps) or the step 3) ($r+1+N_{ij}$ time steps) is completed, than go to step 1).

This logic introduces more flexibility as it allows the execution of sequences of variable length and is based on the implicit assumption that the consecutive application of the same controller produces better results than an arbitrary sequence of controllers even if of greater complexity. For this reason the switching logic come unhooked by the scheduler and limits further the computation time by freezing the controller order.

Respective to the simpler (N, n) switching logic, a “lookup table” is needed (computed off-line) and a counter to compare with n_k is each time updated. This little memory consumption, carried out with neglected computational time, allows more flexible and efficient switching laws.

1.6.3 Power sequence + norm switching logic

If a measure of the state vector is available, a more flexible switching logic can be set up at the expense of further computations. In order to limit these computations, the 1-norm can be chosen to evaluate the expansion or contraction of the state vector. Indeed, it requires only the sum of the absolute values of the vector components. The use of the actual norm can reduce the conservatism of the previous switching logic based on the worst case expansion, thus allowing the use of shorter power sequences. This is because we replace the contraction condition on the matrix norms with a condition derived from the state norms. Consider an initial state x_I and suppose that after a certain number of steps the actual state is given by x_A. The task of the switching logic is to find a number m of successive application of the i-th controller capable of producing a contraction of the state norm at least of a factor $1-\varepsilon$, namely $\left\|x_A^{m+}\right\|_1 = \left\|A_{cl_i}^m x_A\right\|_1 \leq \left\|A_{cl_i}^m\right\|_1 \|x_A\|_1 \leq (1-\varepsilon)\|x_I\|_1$. From the previous relation we derive the following conservative condition: find m such that $\left\|A_{cl_i}^m\right\|_1 \leq (1-\varepsilon)\frac{\|x_I\|_1}{\|x_A\|_1}$.

As shown before we use the matrix 1-norm for the power sequences of dynamic matrices. Therefore, let us define the positive integer n_i as the number of steps such that $\left\|A_{cl_i}^{n_i}\right\|_1 \leq 1-\varepsilon$ and all the intermediate expansions as $B_{ij} = \left\|A_{cl_i}^{j}\right\|_1$, $j < n_i$, $i \in I$. The power sequence + norm switching logic is as follows:

1. Compute and store the (new) initial state vector norm ($\gamma^I = \|x_I\|_1$);
2. Try to compute the greatest possible controller for the available time (say k the index of this controller);
3. For the following $n_k - 1$ steps try to compute the controller k;

4. If a preemption event occurs after r steps (with $r < n_k$) from the beginning of this power sequence:

    4.1) Compute and store the actual state vector norm ($\gamma^A = \|x_A\|_1$);

    4.2) If i ($< k$) is the index of the greatest possible controller for the (reduced) available time, for the following m steps try to compute the i-th controller, where m is such that $\left\|A_{cl_i}^m\right\|_1 \leq (1-\varepsilon)\frac{\gamma^I}{\gamma^A}$;

    4.3) If the step 4.2) is completed then go to step 1) else go to step 4.1);

5. If the step 2) is completed then go to step 1).

It is worth noting that this logic relaxes also the constraint of directly using the minimum controller after a preemption event, by allowing the computation of the greatest possible controller for the new available time. However, the asymptotic stability of the system is ensured by the fact that, even in the worst case of a burst of preemptions expanding the norm, eventually the minimum controller is activated and then the norm is reduced at least of the factor $1 - \varepsilon$.

1.6.4 Always measured norm switching logic

If the computation of the state norm is included in the mandatory part, it can be performed at each step. This way a new logic can be defined:

1. Compute and store the (new) initial state vector norm ($\gamma^I = \|x_I\|_1$);
2. Try to compute the greatest possible controller for the available time (say k the index of this controller);
3. Compute the actual state vector norm ($\gamma^A = \|x_A\|_1$), if $\gamma^A \leq (1-\varepsilon)\gamma^I$ then go to step 1) else try to compute another step of controller k and go to step 3);
4. If a preemption event occurs, compute the greatest possible controller for the new available time (say $i < k$ the index of this controller) and go to step 3) with k replaced by i.

Also this logic ensures the asymptotic stability as, in the worst case, the minimum controller is eventually reached and computed for a sufficient number of steps to contract the state norm.

1.7 Simulations results

The mechanical system chosen for the simulations results is reported in figure 1.12. The nonlinear dynamic equations of the presented system are

$$ml^2\ddot{\alpha} - \frac{1}{2}ml^2 \sin 2\alpha\, \omega^2 + mgl \sin\alpha = 0$$
$$(I + ml^2 \sin^2\alpha)\dot{\omega} + ml^2 \sin 2\alpha\, \dot{\alpha} = \tau$$

where $m = 1Kg$ is the hanging mass, $l = 2L = 1m$ is total length of the rigid vertical bar and of the rigid bar that hangs the mass, $I = 10^{-3}Kg/m^2$ is the inertia of the rotating vertical bar and $g = 9.8m/s^2$ is the gravity acceleration. Let α be the orientation angle between the rigid vertical bar and the hanging bar while ω is the angular velocity of the vertical bar. The mechanical system is actuated by the torque control τ applied at the structure basement. Let $\mathbf{x} = [x_1, x_2, x_3]^T = [\alpha, \omega, \dot{\alpha}]^T$ be the state space vector. Consider the linearized system with respect to the equilibrium point

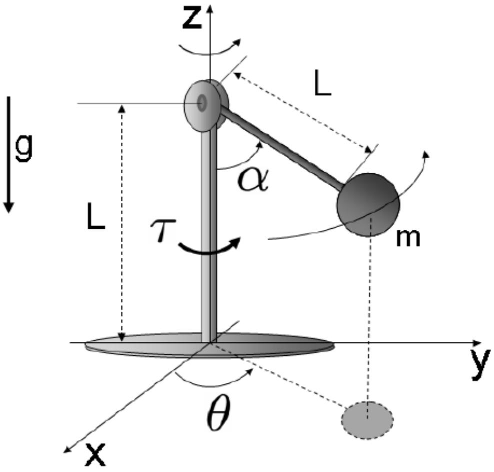


Figure 1.12: Mechanical system adopted for the Any Time Controller simulations.

$\bar{\mathbf{x}} = \left[\pi/4, \sqrt{\frac{g\sqrt{2}}{l}}, 0\right]^T$, with $\tau = 0$. Let now consider the discretized state space with sample time $T = 0.1s$, which yields to the subsequent, open loop unstable, transfer function

$$G(z) = \frac{\alpha(z)}{\tau(z)} = \frac{0.0012178(z + 3.657)(z + 0.2734)}{(z - 1)(z^2 - 1.664z + 1)}$$

To prove the validity of the analysis of the Any Time paradigm, together with the presented stabilizable control laws, a parallel connection has been adopted (see figure 1.7). The controllers are designed based on nested loops, i.e. considering at the level i–th the closed loop transfer function obtained by the system closed with controllers $C_j(z)$ with $j = 1, \ldots, i-1$: $G_{cl_i}(z) = \frac{G(z)}{1+\sum_{j=1}^{i} C_j(z)G(z)}$. Three controllers are obtained

$$C_1(z) = \frac{3.4294(z^2 - 0.6558z + 0.7057)}{(z - 0.091)(z - 0.868)}$$
$$C_2(z) = \frac{900.6711(z^2 - 1.376z + 0.6716)}{(z + 0.2734)(z^2 + 2.307z + 2.695)}$$
$$C_3(z) = \frac{-15.1733(z - 0.5619)(z^2 + 0.569z + 0.09385)}{(z + 0.5874)(z - 0.091)(z^2 + 0.655z + 0.493)}$$

whose closed loop responses to a step command (raised after one second of simulation) are reported in figure 1.13.

Five different policies of the Any Time paradigm are simulated: *No Switching Logic* (the system will execute the controller computation as there is enough time to compute), (N, n) *Switching Logic*, *Power Sequence Switching Logic*, *Power Sequence + Norm Switching Logic* and *Always Measured Norm Switching Logic.* Four different simulations set-up are reported, supposing the system and the controllers at the equilibrium in the initial instant. All the simulations run for 50 seconds. In the first set-up, the system is excited with the reference signal in figure 1.14, on the top left. The closed loop responses reported in figure 1.14 show that the system is asymptotically stable for each stability policy chosen. In figure 1.15, the scheduled time (dotted) and the effectively executed controller (solid) for each policy is reported, assuming that index i means the execution of the parallel controller $\sum_{j=1}^{i} C_j(z)$. To simulate the scheduled time, it has been assumed a statistical distribution of the workload on a general multitasking PC. More precisely, assumptions have been made on the *mean time length* of the activation period of each controller (1/2, 5 and 10 seconds respectively for

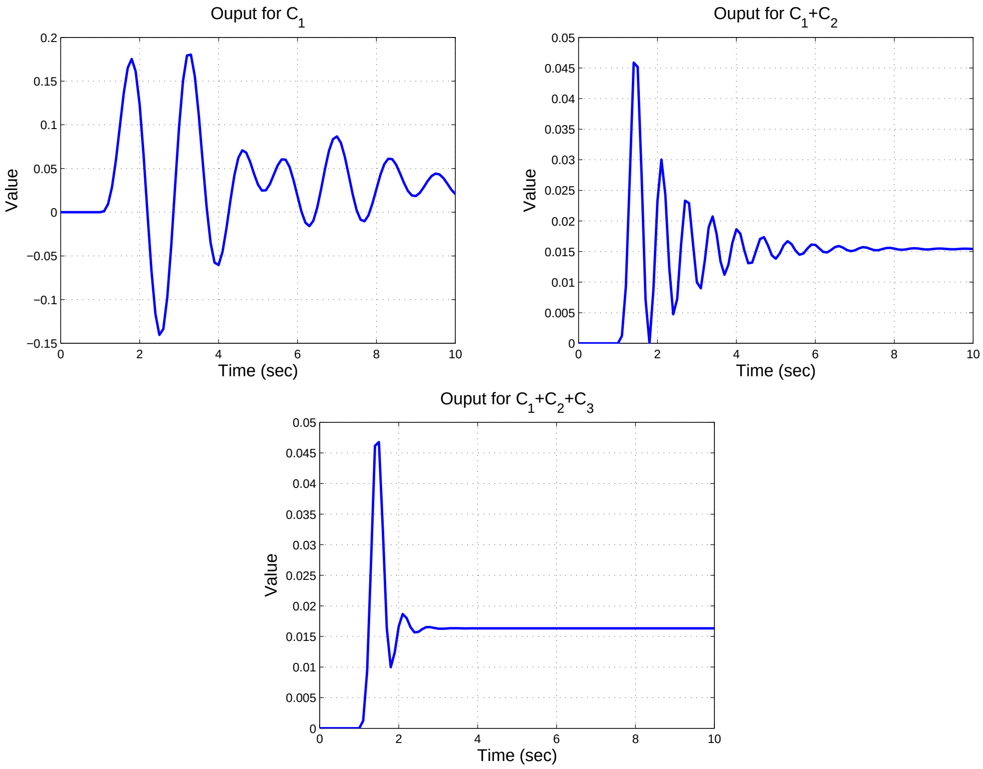


Figure 1.13: Closed loop output in the case of controller $C_1(z)$ (top, left), $C_1(z) + C_2(z)$ (top, right) and $C_1(z) + C_2(z) + C_3(z)$ (bottom).

the controllers $C_1(z)$, $C_1(z) + C_2(z)$ and $C_1(z) + C_2(z) + C_3(z)$) and on the probability of activation of each controller (equal for each of the three connections).

In the second set-up, the system is excited with the same reference signal of example in figure 1.14, on the top left. The mean time length of the activation period of each controller has been set to 1 second for each controller, while the probability of activation is again equal for each of the three connections. In figure 1.16, the scheduled time (dotted) and the effectively executed controller (solid) for each policy is reported, showing how this assumption degrades all the available computational time for each controller. The closed loop responses are reported in figure 1.17. As it will be expected, without any switching policy, the closed system may be unstable.

In the third example, excited with the same reference signal of the previous simulations, the mean time length of the activation period of each controller has been set to 1/2, 1/2 and 10 seconds for each controller, assuming that the system has enough time to execute the larger controller $C_1(z) + C_2(z) + C_3(z)$, although some sporadic task may be executed. To enforce this scenario, the probability of activation of the first controller has been rather set to 1/10 of the probability of the other two tasks. Results for the executed controllers are reported in 1.18, while the outputs are reported in 1.19. In the presented examples, the closed loop behavior is quite good, since available computational time is sufficient. Future works will focus the attention on the general closed loop performances of the whole system.

In the final example, the reference signal is depicted in figure 1.20, on the top left. The mean time length of the activation period of each controller has been set to 1/2, 1/2 and 2 seconds for each controller, assuming an high workload for the multitasking CPU. Again, the probability of activation of the first controller has been rather set to 1/10 of the probability of the other two tasks. Results for the executed controllers are reported in 1.21, while the outputs are reported in 1.20.

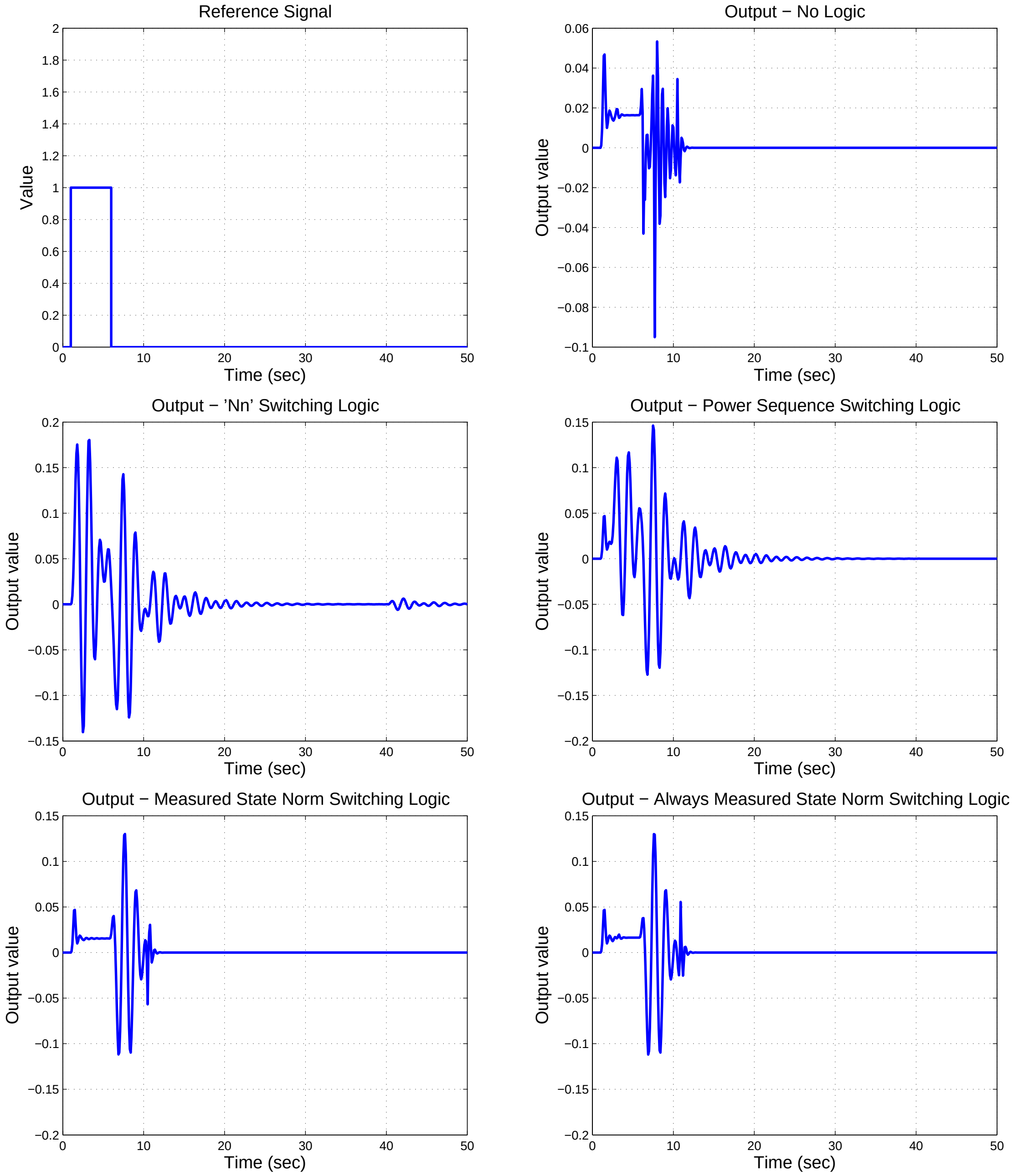


Figure 1.14: Reference signal (top, left) and closed loop outputs for the Any Time Controller implementation for No Switching Logic (top, right), (N, n) Switching Logic (center, left), Power Sequence Switching Logic (center, right), Power Sequence + Norm Switching Logic (bottom, left) and Always Measured Norm Switching Logic (bottom, right).

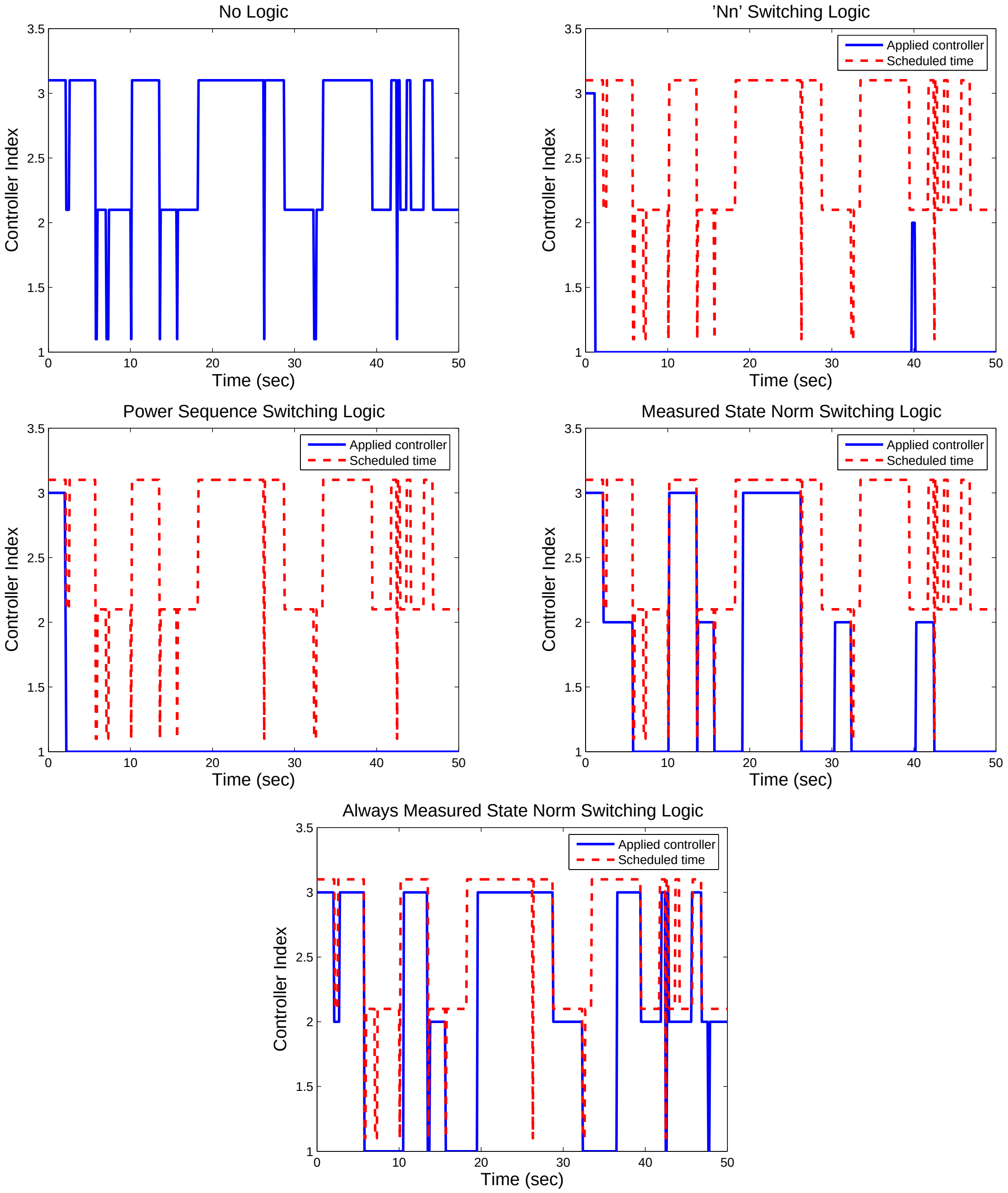


Figure 1.15: Scheduled time (dotted) and executed controller (solid) (the index i means that the parallel controller $\sum_{j=1}^{i} C_j(z)$ is executed) for No Switching Logic (top, left), (N, n) Switching Logic (top, right), Power Sequence Switching Logic (center, left), Power Sequence + Norm Switching Logic (center, right) and Always Measured Norm Switching Logic (bottom).

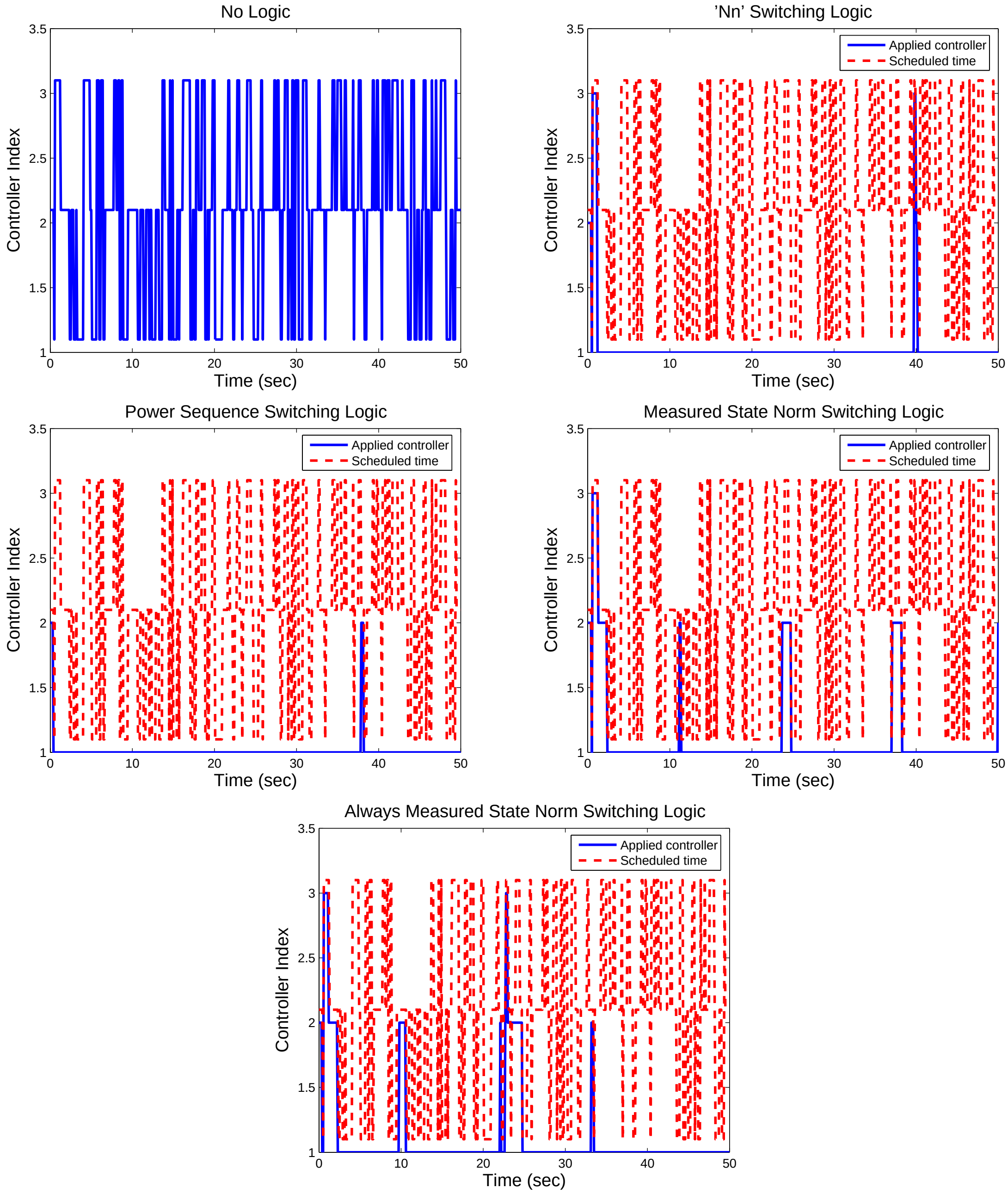
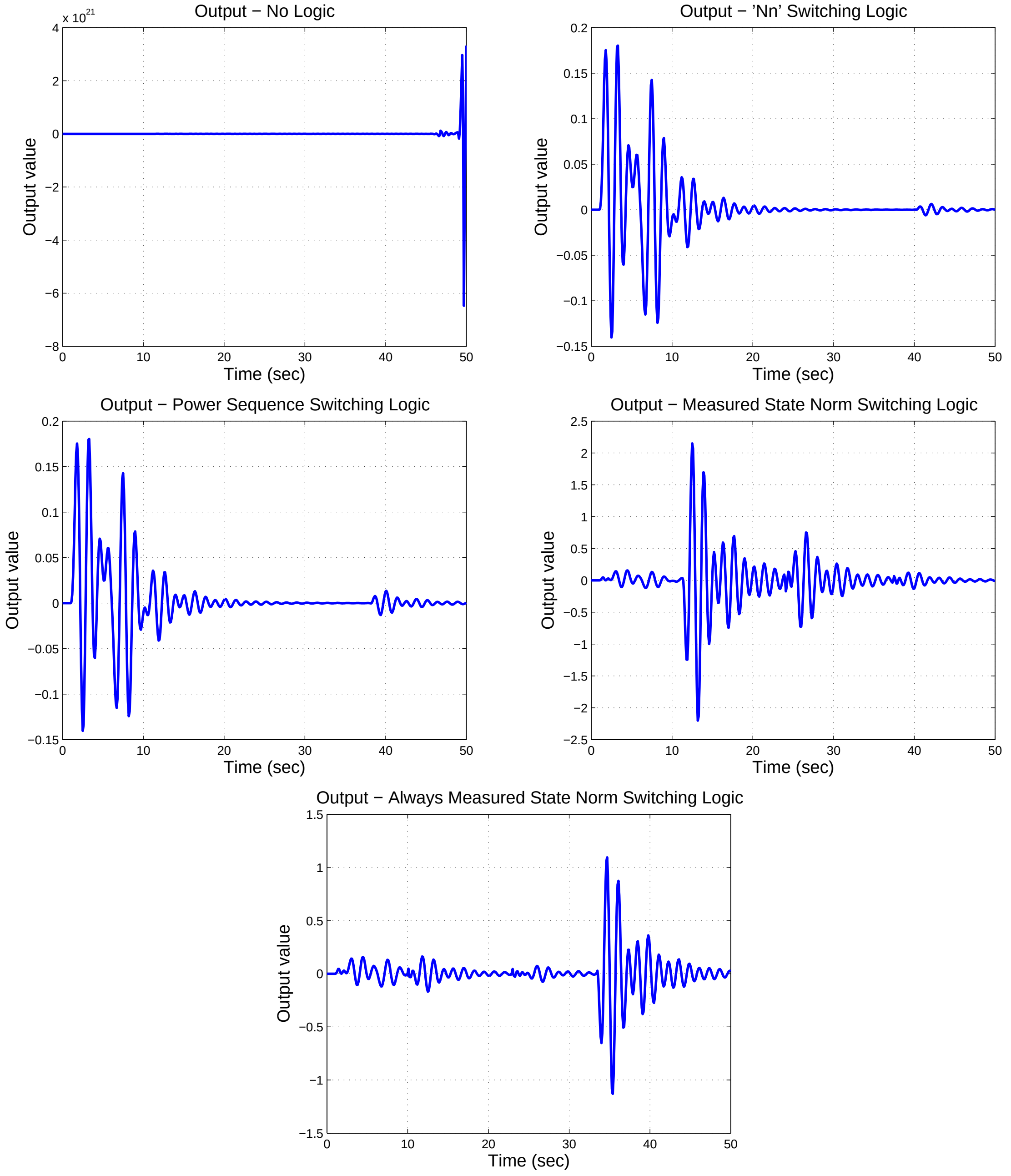


Figure 1.16: Scheduled time (dotted) and executed controller (solid) for No Switching Logic (top, left), (N, n) Switching Logic (top, right), Power Sequence Switching Logic (center, left), Power Sequence + Norm Switching Logic (center, right) and Always Measured Norm Switching Logic (bottom).



Output − No Logic

Output − 'Nn' Switching Logic

Output − Power Sequence Switching Logic

Output − Measured State Norm Switching Logic

Output − Always Measured State Norm Switching Logic

Figure 1.17: Closed loop outputs for the Any Time Controller implementation for No Switching Logic (top, left), (N, n) Switching Logic (top, right), Power Sequence Switching Logic (center, left), Power Sequence + Norm Switching Logic (center, right) and Always Measured Norm Switching Logic (bottom).

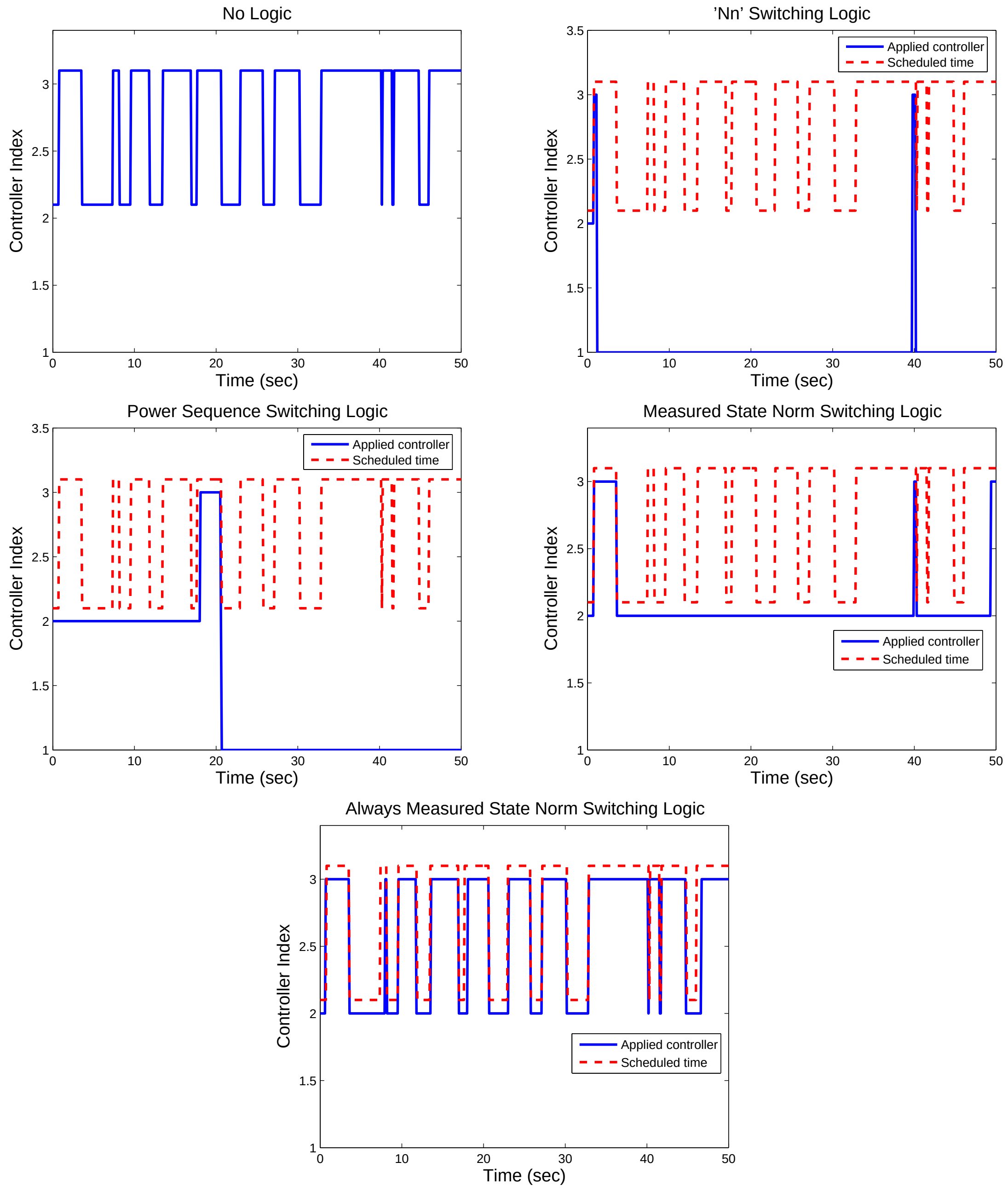


Figure 1.18: Scheduled time (dotted) and executed controller (solid) for No Switching Logic (top, left), (N, n) Switching Logic (top, right), Power Sequence Switching Logic (center, left), Power Sequence + Norm Switching Logic (center, right) and Always Measured Norm Switching Logic (bottom).

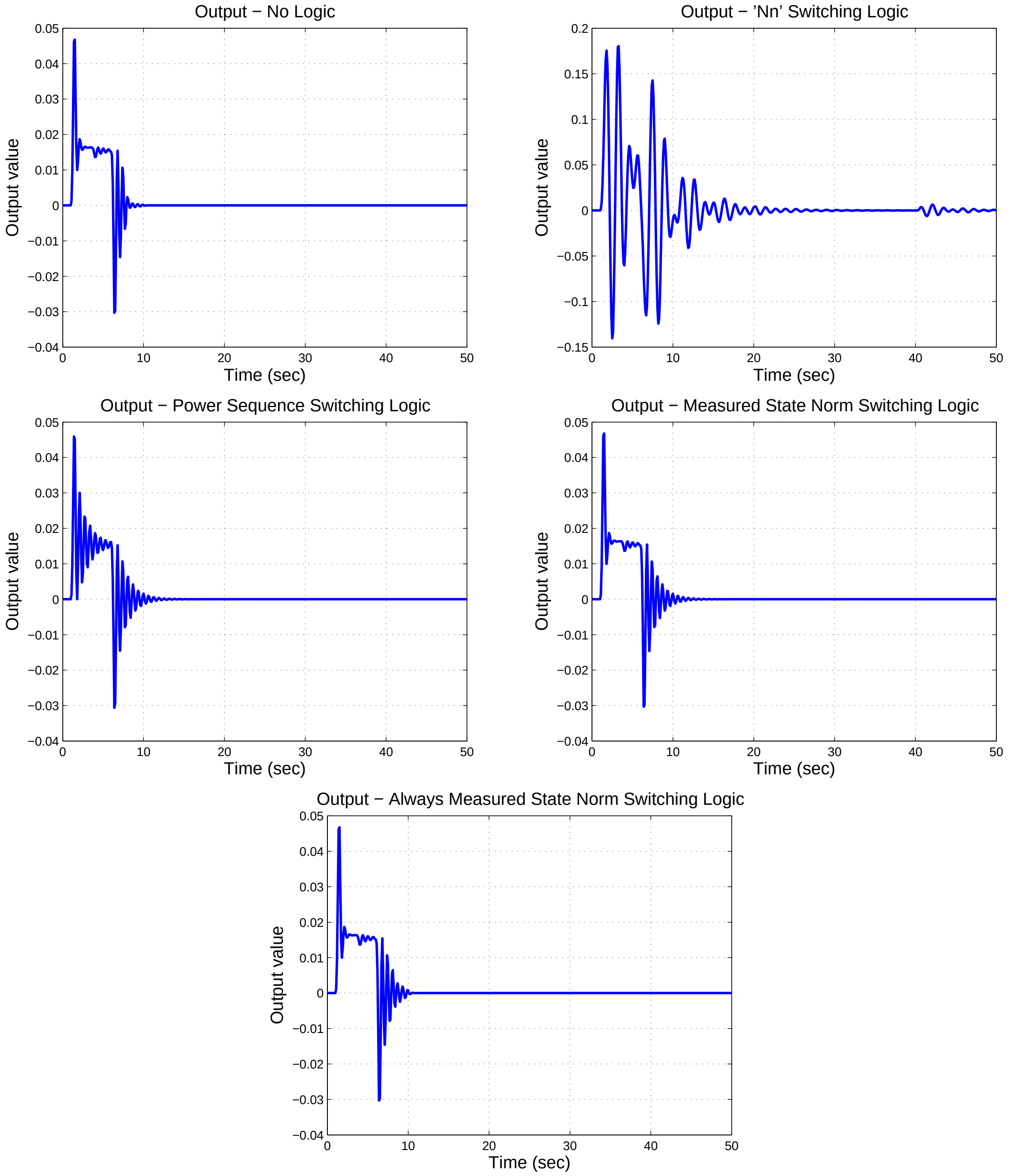


Figure 1.19: Closed loop outputs for the Any Time Controller implementation for No Switching Logic (top, left), (N, n) Switching Logic (top, right), Power Sequence Switching Logic (center, left), Power Sequence + Norm Switching Logic (center, right) and Always Measured Norm Switching Logic (bottom).

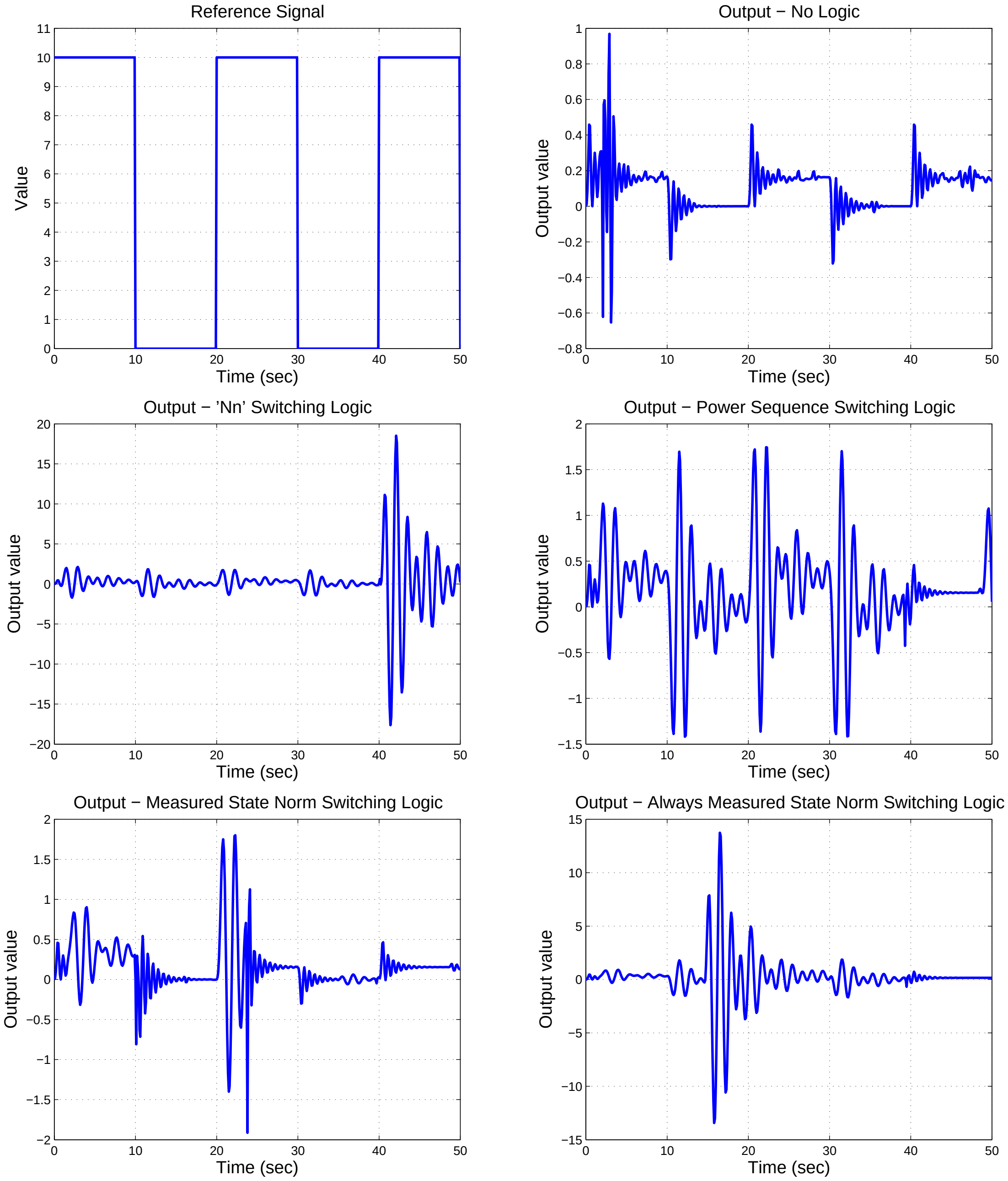


Figure 1.20: Reference signal (top, left) and closed loop outputs for the Any Time Controller implementation for No Switching Logic (top, right), (N, n) Switching Logic (center, left), Power Sequence Switching Logic (center, right), Power Sequence + Norm Switching Logic (bottom, left) and Always Measured Norm Switching Logic (bottom, right).

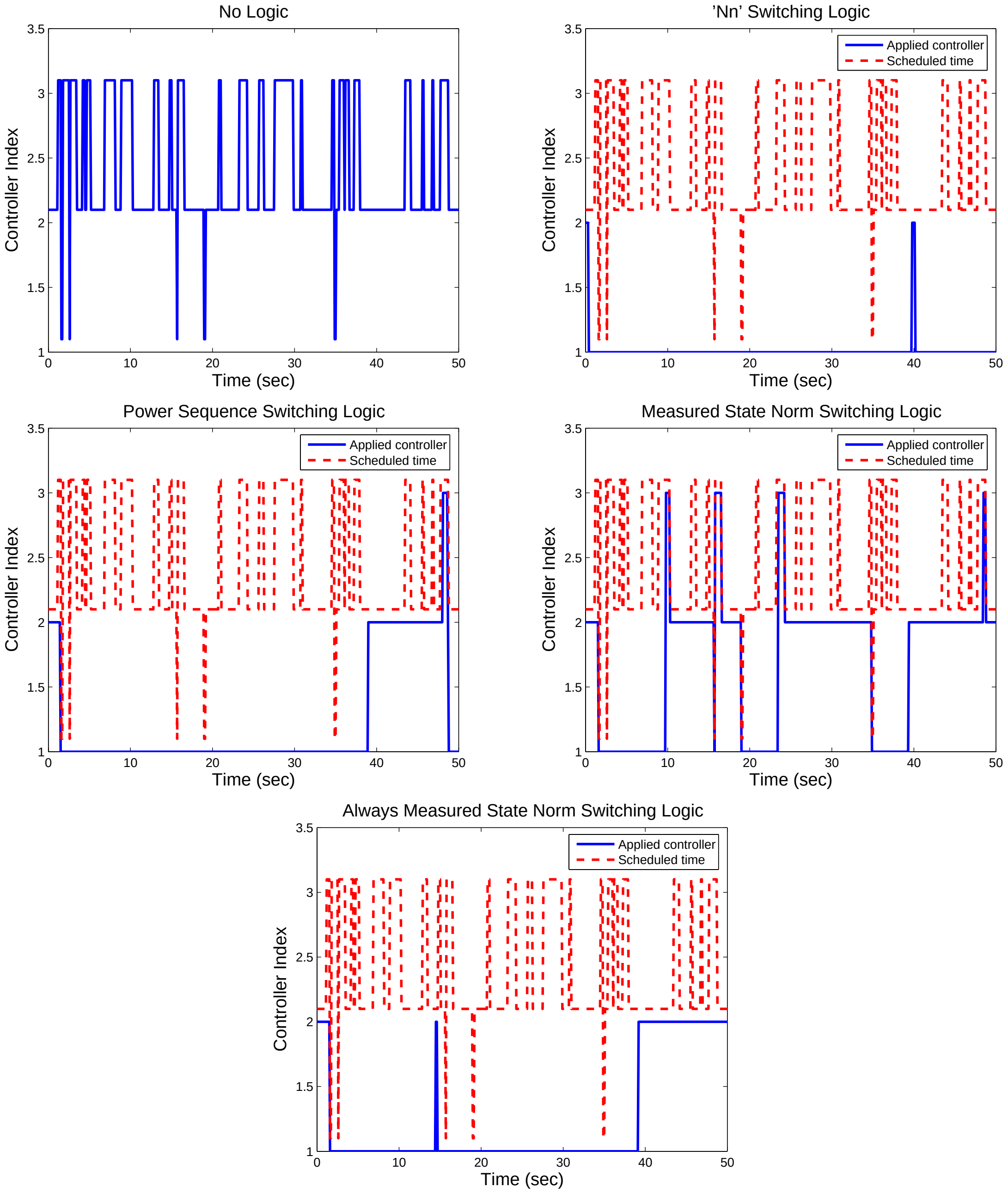


Figure 1.21: Scheduled time (dotted) and executed controller (solid) for No Switching Logic (top, left), (N, n) Switching Logic (top, right), Power Sequence Switching Logic (center, left), Power Sequence + Norm Switching Logic (center, right) and Always Measured Norm Switching Logic (bottom).

Bibliography

[1] K. Årzén, A. Cervin, J. Eker, and L. Sha, "An introduction to control and scheduling co-design," in *Proc. IEEE Intern. Conf. on Decision and Control*, Sydney, Australia, December 2000, pp. 4865–4870.

[2] A. Cervin and J. Eker, "The Control Server: A computational model for real-time control tasks," in *Proc. IEEE Euromicro Conference on Real-Time Systems*, Porto, Portugal, July 2003, pp. 113–120, best paper award.

[3] A. Cervin and B. Lincoln, "Jitterbug: A tool for analysis of real-time control performance," in *Proc. IEEE Intern. Conf. on Decision and Control*, Las Vegas, NV, December 2002, pp. 1319–1324.

[4] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K. Årzén, "How does control timing affect performance?" *IEEE Control Systems Magazine*, vol. 23, no. 3, pp. 16–30, June 2003.

[5] J. W. S. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung, "Imprecise computation," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 83–93, January 1994.

[6] N. Perrin and B. H. Ferri, "Digital filters with adaptive length for real–time applications," in *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium*, Le Royal Meridien, King Edward, Toronto, Canada, May 2004.

[7] J. W. S. Liu, *Real–Time Systems*, T. Holm, Ed. Upper Saddle River, NJ: Prentice Hall Inc., 2000.

[8] K. J. Åström and B. Wittenmark, *Computer Controlled Systems*, T. Kailath, Ed. Prentice Hall Inc., November 1996.

[9] S. Jones, R. Goodall, and M. Gooch, "Targeted processor architectures for high–performance controller implementation," *Control Engineering Practice*, no. 6, pp. 867–878, 1998.

[10] S. P. Panda and C. T. Chen, "Irreducible Jordan form realization of a rational matrix," *IEEE Transactions on Automatic Control*, vol. 14, no. 1, pp. 66–69, February 1969.

[11] Y. L. Kuo, "On the irreducibile Jordan form realization and the degree of a rational matrix," *IEEE Transaction on Circuit Theory*, vol. CT–17, no. 3, pp. 322–332, August 1970.

[12] F. W. Fairman, "Jordan form realization via singular value decomposition," *IEEE Transaction on Circuits and Systems*, vol. 35, no. 11, pp. 1431–1434, November 1988.

[13] T. Kailath, *Linear systems*, T. Kailath, Ed. Prentice–Hall Inc., 1980.

[14] C. T. Chen, *Linear system theory and design*, M. E. V. Valkenburg, Ed. Holt, Rinehart and Winston, 1984.

[15] D. G. Luenberger, "Canonical forms for linear multivariable systems," *IEEE Transactions on Automatic Control*, vol. 12, no. 3, pp. 290–293, June 1967.

[16] J. P. Hespanha and A. S. Morse, “Switching between stabilizing controllers,” *Automatica*, no. 38, pp. 1905–1917, June 2002.

[17] R. A. Decarlo, M. S. Branicky, S. Pettersson, and B. Lennartson, “Perspectives and results on the stability and stabilizability of hybrid systems,” *Proceedings of the IEEE*, vol. 88, no. 7, pp. 1069–1082, July 2000.

[18] D. Liberzon and A. S. Morse, “Basic problems in stability and design of switched systems,” *IEEE Control Systems Magazine*, vol. 19, no. 5, pp. 59–70, October 1999.

[19] Z. Sun and S. S. Ge, “Analysis and synthesis os switched linear control systems,” *Automatica*, no. 41, pp. 181–195, 2004.