Fixed-point digital controller

Olivier Chételat
Swiss center of electronics and microtechnology
CSEM SA, Jaquet-Droz 1
2007 Neuchâtel; Switzerland
olivier.chetelat@csem.ch

Abstract— **After system modeling, controller synthesis and closed-loop simulation, the control engineer has still to implement the controller. In many applications, the controller must be realized with a fixed-point digital processor. The translation of the synthesized controller to its fixed-point realization is intrinsically difficult, because there are an infinite number of solutions and the optimum criterion is not well defined. One would like a fixed-point controller to require as little as possible computing power while at the same time to be the closest from the ideal controller, i.e., the synthesized controller. If it is difficult to define and reach the optimal controller, it is already not obvious to find a sub-optimal but working fixed-point controller. This paper presents the key points that allow a computer to automatically derive a fixed-point realization of any given discrete-time transfer function. The derivation can be controlled by three parameters, one being the number of initial derivative, another the number of initial delay and the last one the number of bits of the processor words. An increase of any of these three parameters enhances the quality of the realization for a relatively small extra cost of computing power.**

I. INTRODUCTION

One may think that in a near future all controllers will be implemented with floating-point arithmetics and that fixed-point controllers are relics of the first age of computer science. The decreasing cost of floating-point microprocessors and the possibility of using the same (floating-point) controller for simulation and real experiment support this opinion. However, for mass production, one can argue that a fixed-point processor is smaller, simpler and consequently cheaper, and it is worth to invest more effort on implementation issues. Since the power consumption of the controller is directly related to the computing power, in the applications the processor is a significant user of battery power, a fixed-point controller generally means a much longer autonomy. Moreover, for micro-controllers, the silicon saved by a smaller fixed-point ALU (arithmetic logic unit) can be used for other functionalities. Last but not least, for a given technology, a fixed-point controller can be significantly faster than its floating-point counterpart.

However, beside the not-obvious extra work required, a fixed-point controller implementation is subject to detrimental effects due to saturation, roundoff of internal signals, and roundoff of parameters. All these effects can lead to controller fragility, instability, longer settling time, permanent error, noise, etc. In addition, the number of possible realizations is infinite and the optimal realization not well defined and application dependent—some applications need more robustness while others require minimum computing power, or minimum noise, etc. Many of these desired specifications are actually contradictory to some extent. However, a good compromise can be achieved without too much effort when the good options are chosen. Below, we will give a set of motivated choices (design drivers) that will allow a computer to automatically generate the C code of a satisfactory fixed-point realization of any discrete-time transfer function, with minimum user interaction. The latter point is important, since many fixed-point development tools (e.g. Matlab) provide simulation support, but no or not much help for the translation process 'transfer function $\rightarrow$ fixed-point control algorithm' (this is all the more surprising that the authors of several papers—see references in [1]—actually claim that they wrote such a tool). Consequently, many fixed-point implementations are still in practice tediously derived by hand. This is costly, time consuming and error prone. Obviously, in many applications, a program automatically generating a decent implementation of a fixed-point controller is a significant advantage. In our company, such a program has been written in Matlab and is used to quickly implement fixed-point controllers. The theory of this program is explained in this paper and illustrated by an example.

Our first design driver concerns the saturations (or overflows) that we do not allow. This choice is first motivated by the existence of an exact and non-conservative theory (see for example [2]) that supports an easy computation of the correct scaling of all internal signals of the controller. With this scaling, no saturation is needed because the maximum absolute value of any signal cannot be exceeded. However, the maximum absolute value of every signal can be reached, which makes scaling optimal. The second motivation for a saturation-free algorithm comes from the detrimental non-linear effects of saturations that can easily upset a great deal of the care brought to the controller design.

Then, as the scaling is optimal, we can hope for minimal effects of rounding of internal signals. Indeed, our second driver is to go on in this direction and chose a controller realization that minimizes the numerical noise. Mathematical rigor is here more difficult to keep, but some statistical approaches have been proposed. A good rule of thumb is to avoid cascading subsystems. Any input of the controller is free of numerical noise (by definition), while the output of any subsystem is already polluted with numerical noise

This project has been partly supported by the Swiss government.

that can just be further amplified. Therefore, the shorter is the 'path' from an input to an output, the better. Another pragmatic rule is to sum up small numbers first.

The third design driver deals with efficiency. As the main advantages of fixed-point over floating-point implementations are related with computing power, it is important to keep the number of operations as low as possible. Many studies (for example [3], [4]) work with a state-space representation. This has the big advantage of a good mathematical framework able to describe many (but not all) possible realizations of a controller, each of them corresponding roughly to a given state matrix. This matrix is square and mostly zero for the most efficient realizations (sparse matrices [5]), but in general, it contains numbers different from zero. For example, a ten by ten matrix would generally describe a realization with 10^2 non-trivial multiplications while in most cases, 10 (and possibly less) would suffice.

For some realization, the roundoff of the parameters may significantly shift the zeros/poles/gain of a transfer function. It is not rare that a robust controller becomes fragile or even unstable for some fixed-point realization. Some researches focus on this point and propose methods supported by solid theoretical backgrounds to find realizations that save as much as possible the robustness properties of the synthesized controller. However, these optimizations cannot simultaneously follow the second and third design driver. Our approach is to relax this criterion. We consider that the two previous design drivers are more important. A realization can always be rigorously checked *a posteriori*. If it is robust enough, no problem. Otherwise, we try and solve the problem with a larger number of bits for the parameters. This will take more computing power, but probably less than for a random state matrix. In addition, the numerical noise will be lower.

However, an optimized accuracy of the poles/gain of the transfer function can be required without conflict with the other design drivers. This is the fourth design driver.

The zeros are the looser. Their accuracy cannot be directly and perfectly optimized. However they can be indirectly improved through an optimization of the accuracy of the impulse or the step response. This is the fifth design driver.

For the sixth design driver, one may wish that the approach be efficiently extensible to MIMO systems (multiple inputs, multiple outputs).

The last design driver is the possibility to easily enhance the quality of the realization when needed. Obviously, one can first increase the number of bits used (for example, to switch from 16-bit to 32-bit implementation). However, before using the 'brute force', other possibilities using less computing power would be welcome.

Section II recalls the standard control architecture. Section III discusses about different realizations of the same transfer function. The parallel form is shown to be the best trade-off. The next section presents some artifices (not published to the best of our knowledge) that can significantly improve a realization for a low additional cost of computing power. Section V develops the mathematical formulation of the parameters. The section that follows deals with scaling and finally, the last section before the conclusion describes our approach to automatically generate the fixed-point algorithm of any discrete-time transfer function.

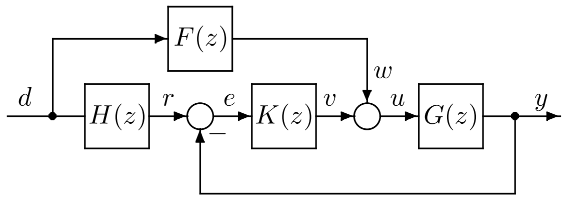

Fig. 1. System architecture

II. Control Architecture

A controller is not limited to the compensator $K(z)$. Fig. 1 shows the diagram of the complete system. A feed-forward signal is generated by the transfer function $F(z)$. The role of the transfer function $H(z)$ is important. In the absence of disturbances and model uncertainties, the signal r (output of $H(z)$) should be exactly the signal y (output of $G(z)$).

The feed-forward transfer function $F(z)$ would be ideally the inverse of the system to control $G(z)$. However, in many cases, the system to control $G(z)$ has some non-minimum phase zeros. Consequently, the inverse is either non-causal or unstable.

To make the feed-forward transfer function $F(z)$ both causal and stable, one can pre-filter the desired signal d (typically with zeros that cancel the unstable poles). In practice, the pre-filter must be distributed both on $F(z)$ and $H(z)$, because the zero/pole cancellation must be analytically done before any implementation. Additionally, one would need to limit the feed-forward signal w, because this signal would be infinite every time the desired signal d is changed. A low-pass pre-filter will generally solve the problem. In this case, one has the choice either to use a dedicated pre-filter directly filtering d, or to have two pre-filters, one embodied in $F(z)$ and the other in $H(z)$. The first option seems more attractive, because only half of the operations are needed for the pre-filtering. However, cascading two fixed-point filters may significantly increase the numerical noise.

III. Structure

At least for the compensator $K(z)$, a good location of poles and zeros is important to prevent the controller to become fragile. A good way to keep a good control on both poles and zeros is to cascade simple cells of one pole and possibly one zero. However, we have just seen above that cascading is a risky approach and need at least to take into account the problem of the numerical noise. Moreover, the order of the cells and the way poles are paired with

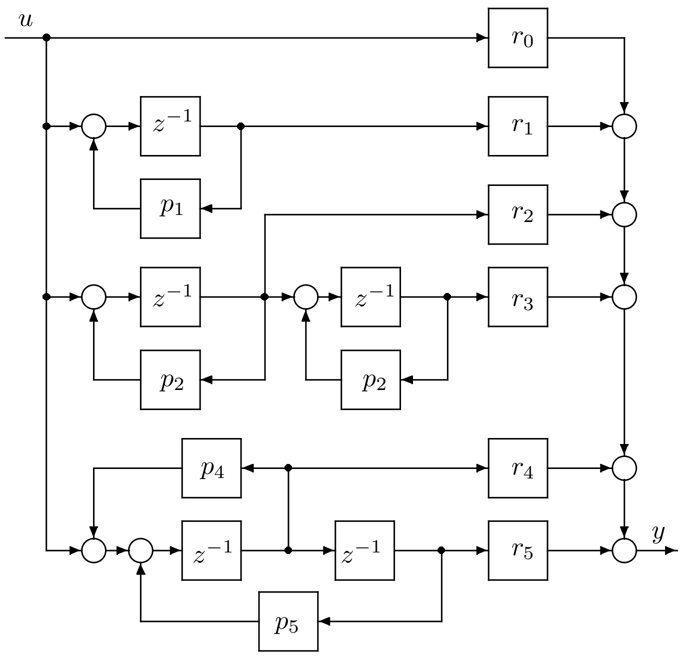

Fig. 2. Parallel form—illustration with a feed-through (r_0), one single pole (p_1, r_1), one pair of poles (p_2, r_2, r_3) and one pair of complex conjugate poles (p_4, p_5, r_4, r_5)

zeros is important and unknown *a priori*. Because of these drawbacks, we dismiss the cascade structure.

The direct form is popular in basic books, because polynomials are usually written with their coefficient, but it is probably the worse approach, since the location of both poles and zeros cannot be controlled. Already for small orders, the output of the realized filter can be totally wrong, and in most case, the stability is lost.

A structure that keeps a good control on both numerical errors and pole location is the parallel form (i.e., partial fraction expansion), see fig. 2. The good behavior regarding numerical errors comes from the short 'paths' linking the input to the output. The input is free from numerical errors. Each pole (except the multiple poles) has the same clean input. The numerical errors come from the realization of the poles and the final sum. The contribution of the latter can be minimized if the smallest signals are summed up first. The 'size' of the signals will be given by the scaling described in section VI.

The direct and parallel forms (but not the cascade structure) are sub-cases of a more general family known as state-space formulation. All state-space realizations have the same (minimum) number n of states. In the state-space formulation, the further values of the states are a linear combination of the actual values of the states. In the general case, the number of operations is of the order $\mathcal{O}(n^2)$, but for the special cases presented above (cascade of simple cells, direct form and parallel form), the number of operations is of the order $\mathcal{O}(n)$. For the compensator $K(z)$ of our example, the number of state is $n = 17$. A general state-space realization will therefore require 17 more operations than, e.g., a parallel form. This is significant, because for the same expense, one can use numbers more than four times ($\sqrt{17}$) larger. In other words, if a general realization is proved best for a 16-bit implementation, a parallel form can still challenge it with a 64-bit implementation.

Other well-known structures (not described by the state-space formalism) include the lattice structure, and the LC structure. They use twice as many multiplications as the parallel form, but in some applications, they have interesting properties. The lattice structure is especially useful for adaptive filtering, because its stability can be guaranteed if its parameters are kept within the range $]-1,1[$. The parallel form has a similar property (the stability can be checked on the poles), but the structure of the parallel form is not well-posed and change if the poles are real or complex, or, single or multiple. The LC structure makes easy to guarantee that the zeros alternate with the poles. This property is important, for example, for controllers that must stay passive.

The cascade form can easily and efficiently be used for MIMO system (the poles are shared by all inputs of the transfer matrix, only the residues are specific to each output).

IV. IMPROVEMENTS

As just discussed, a way to enhance the quality of a realization is to increase the number of bits. For the parallel form, the pole location can be improved if the pole parameters p (see fig. 2) are encoded with more bits. The quality of the zeros follows the precision of the residues r. Finally, the numerical noise is decreased when the words used to contain the signals are widen.

But before increasing the number of bits, there are other less costly improvements to try. A typical problem of some realization shows a permanent error of the step response. Fig. 3 gives a good example of this phenomenon. As a fixing, we propose to pre-filter the input signal with the 'derivative' $1-z^{-1}$. Of course, the transfer function of the filter to be realized in parallel form has to be revised with an added integrator, and becomes $F(z)z/(z-1)$ ($F(z)$ being the overall transfer function to implement). This artifice costs almost nothing (an extra state, two additions and the multiplication of the integrator residue). It works very well (see fig. 4), because the derivative makes the signal to vanish after the transient. All residues r multiply then a null signal, except the integrator $1/(z-1)$. Thus, for final time, the fixed-point approximation is localized to the integrator residue only. The control engineer reflex may make think that the derivative $1-z^{-1}$ adds some noise. In this case, this is wrong. No significant bit is lost by the derivative. An integrator $1/(z-1)$ following a derivative $1-z^{-1}$ perfectly reconstitute the signal with no added noise. Note also that for this same reason, the integrator added to the parallel form can be omitted, since its output is actually the derivative input. The derivative idea can be extended

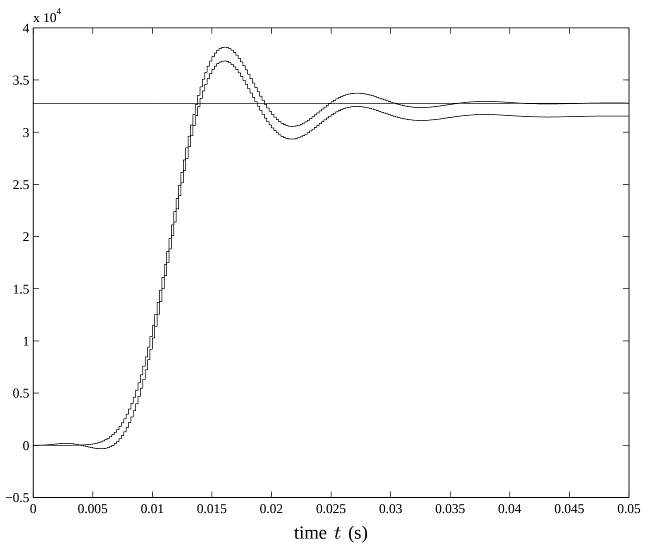

Fig. 3. Fixed-point (lower line) compared with floating-point (upper line) step response of a parallel realization of a cascade of two Butterworth filters of 5^{th} order each (cutoff frequency is 0.02 sampling frequency, implemented with $n = 32$ bits for the signals and $m = 16$ bits for pole parameters and residues)

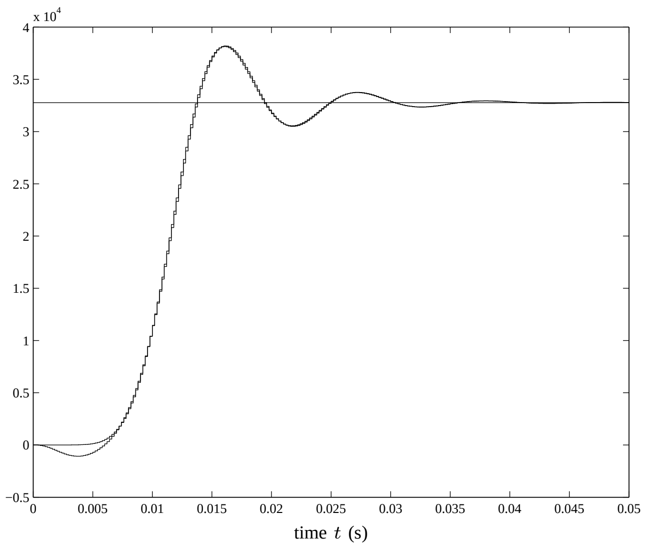

Fig. 4. Step response of the same filter as fig. 3, but implemented in fixed-point with the initial 'derivative' $1 - z^{-1}$

further, and if necessary, more than one derivative can be inserted.

The problem of the permanent error is solved at fig. 4, but still, the beginning of the step response is not correct. The impulse response (which is the derivative of the step response) is also wrong at initial time. Again, the reason of this comes from the approximative zeros. Actually, the signals generated by the poles (modes) are big at initial time and quickly vanish. However, at the beginning, the impulse response should be small. This is unfortunate, because to generate such a response, the residues have to make differences of big and close numbers and lots of

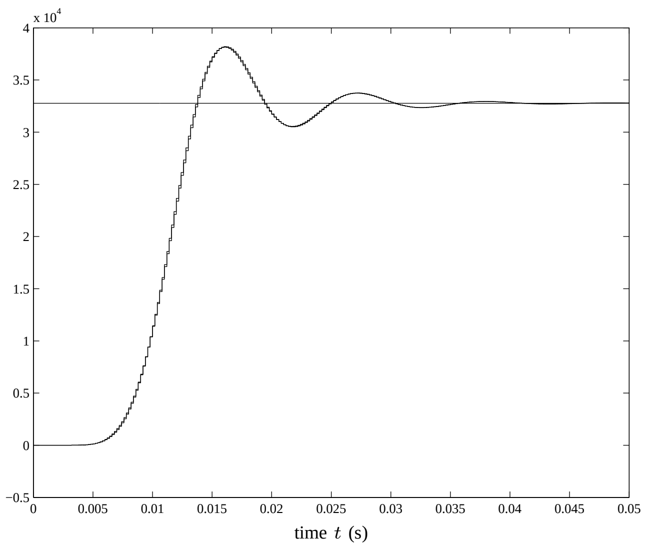

Fig. 5. Step response of the same filter as fig. 4, but implemented in fixed-point with 20 initial delays, most of them having a multiplier too small to really contribute to the output y

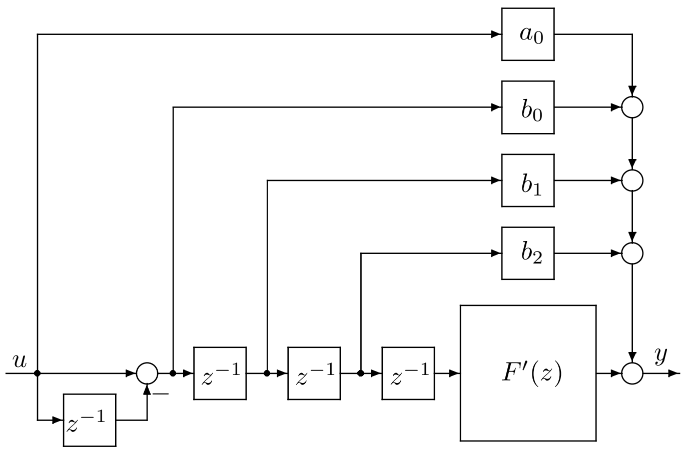

Fig. 6. Parallel realization $F'(z)$ (see fig. 2) improved with initial delays (here 3) and derivative (here 1)

significant bits are lost. To make things worse, at 'medium' time, the magnitude of the modes is much smaller, but the impulse response should be big. Again, the residues have to laboriously make up for the situation. The solution here is to delay the modes until they can be big where the impulse response has to be big. For the initial time no longer covered by the modes, the samples of the impulse response can be independently and accurately controlled by dedicated multipliers. Again, the extra cost is small (one more state, an addition and a multiplication for every inserted delay), but the result is comparatively spectacular (see fig. 5).

Fig. 6 summarizes the derivative and delay artifice described above, and the next section gives more details about the computation of the parameters p, r, a and b of fig. 2 and 6.

As for a given filter, the poles are getting closer to 1 when the sampling rate increases, for transfer function with high sampling rate, it has been proposed [1], [6] to replace the delay operator z^{-1} by the delta operator $\delta = z - 1$ to improve the realization. However, the benefit of the delta operator greatly depends on the structure. For a parallel form, both operators are essentially equivalent.

V. Computation of the Parameters

Assume that the transfer function to realize is $F_0(z)$. To improve the realization, $m \geq 0$ initial derivatives and $n \geq 0$ initial delays are added.

After the first initial derivative, the residual transfer function is

$$F_1(z) = \frac{z-1}{z}\big[F_0(z) - a_0\big]$$

As the step response of $F_1(z)$ vanishes for infinite time, we have $F_1(1) = 0$ and $a_0 = F_0(1)$. For more derivatives, we have the following recursive equations

$$F_i(z) = \frac{z-1}{z}\big[F_{i-1}(z) - a_{i-1}\big] \qquad a_{i-1} = F_{i-1}(1)$$

for $i = 1 \ldots m$.

After the first initial delay, the residual transfer function becomes

$$F_{m+1}(z) = \frac{1}{z}\big[F_m(z) - b_0\big]$$

As the impulse response of $F_{m+1}(z)$ is null for time 0, we have $b_0 = F_m(\infty)$. For more delays, the recursive equations are

$$F_{m+i}(z) = \frac{1}{z}\big[F_{m+i-1}(z) - b_{i-1}\big], \quad b_{i-1} = F_{m+i-1}(\infty)$$

for $i = 1 \ldots n$.

Let us denote the transfer function to implement in parallel form $F'(z)$. This function is equal to $F_{m+n}(z)$. The partial fraction expansion is

$$F'(z) = r_0 + \sum_{i=1}^{N} \frac{r_i}{(z - c_i)^{\mu_i}}$$

where r_0 is the coefficient of *feed-through*, r_i the residues, and c_i the poles with multiplicity μ_i. The partial fraction expansion is known to be ill-posed. When two poles are close, the computation of the residues can lead to numerical problems. To avoid this, we recommend in these cases to extend the partial fraction expansion by increasing the multiplicity of the close poles. The sum (V) will contain more terms with residues that should be theoretically null. Then we compute the numerical values of the residues by a least mean square regression on the impulse response of $F'(z)$ with respect to the linear combination of the modes (a mode is the impulse response of $1/(z-c_i)^{\mu_i}$). This may cost a little more computing power (two multiplications by added poles), but it is numerically well-posed.

Poles are real or complex conjugate. If the pole c_i is real, then we have

$$p_i = c_i$$

If c_i is complex conjugate with c_{i+1}, we have

$$p_i = c_i + c_{i+1} \qquad\qquad p_{i+1} = -c_i c_{i+1}$$

VI. Scaling

Assume that the fixed-point numbers for the parameters p, r, a and b are m-bit signed integers (i.e., covering the range $[-2^{m-1}, 2^{m-1} - 1]$). Theoretically, scaling consist in splitting a number into two factors, one *a priori* and the other such that the product gives the number. However, in practice, in order to minimize the needs of computing power, the *a priori* factor is chosen a power of 2, because in such case, the multiplication can be replaced by the (much faster) logical shift. The binary logarithm of the *a priori* factor gives the position of the period in the number in binary radix. It is called the exponent E, and is optimally defined such as the other factor, called the mantissa M is maximized.

$$E = \begin{cases} \lfloor \log_2 \big(f/(2^{m-1}-1)\big) \rfloor & \text{for } f \geq 0 \\ \lfloor \log_2 \big(f/(-2^{m-1})\big) \rfloor & \text{for } f < 0 \end{cases}$$

where f is the floating-point number (here the parameters p, r, a or b), $\log_2$ the binary logarithm, and $\lfloor x \rfloor$ the integer part of x. The mantissa M is given by

$$M = \lfloor f/2^E + 0.5 \rfloor$$

The mantissa M, which is the fixed-point parameter, is in the range $M \in [-2^{m-1}, 2^{m-1} - 1]$. Moreover, the corresponding floating-point parameter $\tilde{f}$ (which is approximatively the original floating-point parameter f) can be computed by

$$\tilde{f} = M \cdot 2^E$$

The effects of the quantization of the parameters f becoming $\tilde{f}$ can therefore be studied by simulation and by analysis.

Assume that the computer words are n-bit signed integers (i.e., covering the range $[-2^{n-1}, 2^{n-1} - 1]$). All signals (including internal signals) must be scaled in order to never exceed the computer word boundaries, while being as large as possible to minimize the effects of roundoff of internal signals. For LTI (linear time invariant) systems as considered here, there is a theory (see for example [2]) that allows the exact computation of the maximum and minimum reached by any internal signal. Let us focus on one particular internal signal s of the system of fig. 1. The signal s can be any internal signal in the implementation of any of the transfer function $F(z)$, $G(z)$ or $H(z)$. Let us call $T(z)$ the transfer function linking the input signal d to the signal s. The signal d is assumed to be within the same range as the output y. In practice, this makes sens, because the digitizer of the output y codes the quantity to control, let us say, in n_0-bit signed integers. It is also assumed that a white noise is a possible signal for the input d. This means that a given sample can be any thing in the range $[-2^{n_0-1}, 2^{n_0-1} - 1]$ independently of what it was before. Note that when this assumption is not true, the input is actually r and not d, and an appropriate filter H has to be inserted (see fig. 1) such that d can be a white noise.

The signal s can be computed for any input d by a convolution

$$s(k) = \sum_{i=0}^{\infty} h(i)d(k-i)$$

where $h(k)$ is the impulse response of $T(z)$. For $k \to \infty$, the value of $s(k)$ can be maximized if the magnitude of $d(k-i)$ is maximized and its sign the same as the sign of $h(i)$. Similarly, the minimum of $s(k)$ is reached when the magnitude of $d(k-i)$ is maximized and its sign opposite to the sign of $h(i)$. Thus, the bounds A and B of the signal $s \in [A, B]$ are

$$A = -2^{n_0-1} \sum h_+(k) + (2^{n_0-1} - 1) \sum h_-(k)$$
$$B = -2^{n_0-1} \sum h_-(k) + (2^{n_0-1} - 1) \sum h_+(k)$$

where $h_+(k)$ is $h(k)$ if $h(k) \geq 0$ and 0 otherwise, and $h_-(k) = h(k) - h_+(k)$. Note that when the bounds of s have the same magnitude, the expression is simpler, because $\sum |h(k)| = \|h\|_1$. As in our case, the bounds differ only by one, we can redefine A and B in a slightly conservative way

$$A = -B \qquad\qquad B = (2^{n_0-1} - 1)\|h\|_1$$

In a similar way to the scaling of the parameters, the scaling of the signal s is given by

$$X = \lfloor \log_2 \left(B/(2^{n-1} - 1) \right) \rfloor \qquad\qquad \tilde{s} = N \cdot 2^X$$

where N is the fixed-point representation of s with a bit shift of X. The corresponding floating-point signal is $\tilde{s}$.

VII. Sequence of Computation

In practice, the automatic generation of fixed-point code from a transfer function $F(x)$ needs three passes. At the first pass, Matlab-Simulink s-functions are generated. For a control scheme like in fig. 1, an s-function is generated for $F(z)$, $H(z)$ and $K(z)$. These s-functions simply perform the filter computation in floating-point (with rounded parameters), following the computation diagrams of fig. 2 and 6. Only one simple operation (e.g., one addition or one multiplication) is performed by statement. The s-functions output all internal signals s, and of course the filter output.

The diagram of fig. 1 is run in Simulink with an impulse signal as input d until the output y vanishes. The magnitude of this impulse signal is $2^{n_0-1} - 1$. The internal signals s computed by the s-functions are actually the impulse response of all internal signals. The 1-norm is used to get the bound B (and consequently A) of every internal signals according to the theory described above. The bounds B of the signals s at the output of the residues r (see fig. 2) and at the output of the coefficients a and b (see fig. 6) are used to rearrange the order of the computation of the final sum. The new sum starts with the signals with smaller B, and adds progressively signals with growing bound B.

Then, the s-functions are corrected with regard to the order of the final sum, and a second pass is run. With the newly computed bounds B, all signals can be properly scaled, and the final pass generates the fixed-point algorithm in C.

Note that there is no disturbances in the diagram of fig. 1. Actually, the computation of the bounds B during passes 1 and 2 should also take into account any potential disturbance. This can be done by computing the bounds B for each system input (d and disturbances) in turn (the others being set to zero), and eventually summing them up.

VIII. Conclusion

There are an infinity of fixed-point realizations of a given transfer function. The goal for the engineer is to find one that works well for a low computing power. As the task is not easy and costly to perform by hand, a way to automatically generate the C-code algorithm is proposed. The choice of the 'best' realization structure is motivated, and two artifices that allow one (when needed) to significantly improve the quality of the realization for a low cost of computing power are proposed.

References

[1] H. Hanselmann, "Implementation of digital controllers: A survey," *Automatica*, vol. 23, pp. 7–32, 1987.

[2] M. Steinbuch, G. Schootstra, and H.-T. Goh, "Closed-loop scaling in fixed-point digital control," *IEEE transactions on control systems technology*, vol. 2, no. 4, pp. 312–317, December 1994.

[3] J. Whidborne, D.-W. Gu, J. Wu, and S. Chen, "Optimal controller and filter realisations using finite-precision floating point arithmetic," *submitted to Int. J. Systems Science*, 2003.

[4] J. Wu, S. Chen, J. Whidborne, and J. Chu, "A unified closed-loop stability measure for finite-precision digital controller realizations implemented in different representation schemes," *IEEE trans. automatic control*, vol. 48, no. 5, pp. 816–822, May 2003.

[5] M. Gevers and G. Li, *Parametrizations in control, estimation and filtering problems: accuracy aspects*. London: Springer Verlag, 1993.

[6] G. Orlandi and G. Martinelli, "Low sensitivity recusive digital filters obtained via the delay replacement," *IEEE trans. circuits and systems*, vol. 31, no. 1, pp. 453–460, 1984.