DESIGN OF DISCRETE CONTROLLERS FOR CONTINUOUS SYSTEMS USING HYBRID CHI [1]

D. Albert van Beek, **J.E. Rooda**

Eindhoven University of Technology
Department of Mechanical Engineering
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
E-mail: vanbeek@wtb.tue.nl

Abstract. Controllers of continuous systems are usually discrete, because of the sampling required by computer implementations. The controlled physical systems are often nonlinear. It is therefore important that tools and languages for control system design provide adequate support for dealing with these phenomena. The χ language, presented in this paper, provides such support. It is suited to specification and simulation of control systems of a continuous-time, discrete-event or discrete-time nature. Such control systems may range from stand-alone controllers to interacting plant-wide control systems. The language integrates a small number of orthogonal continuous-time and discrete-event concepts. The continuous-time part of χ is based on differential algebraic equations; the discrete-event part is based on a CSP-like concurrent programming language. A case study is presented of a tank level control system. Several control strategies are modelled, including a PI controller with anti-windup.

Keywords. control, simulation, hybrid, discrete, continuous.

1. INTRODUCTION

There is a large number of languages and tools supporting linear continuous control system design. In practice, however, systems are usually nonlinear and control systems are usually discrete. An important cause of nonlinearities is actuator saturation, which may occur in all physical actuators. Physical controllers are usually discrete because of the sampling required by computer implementations. It is therefore important that languages and tools for control system design provide adequate support for dealing with these phenomena.

The χ language, treated in this article, provides such support. It is suited to specification and simulation of industrial systems, including real-time control systems, in the time domain. Such control systems can be of a continuous-time, discrete-event or discrete-time nature. In continuous-time systems, an infinite number of state changes are possible in any given finite time interval. In discrete-event systems there can only be a finite number of changes in a time-interval. The state changes at discrete points of time only. In discrete-time systems, the state changes at equidistant points of time. The χ language is equally well suited to continuous-time, discrete-event and discrete-time modelling. We consider discrete-time systems as a subclass of discrete-event systems.

A well known package for continuous control system design is Matlab. Although it is well suited to linear continuous control system design, it lacks a solver for nonlinear differential algebraic equations. Such solvers can be found in general purpose modelling languages such as ACSL (Mitchell and Gauthier, 1976) and Dymola (Elmqvist, 1994). The main difference between these languages and χ is that the discrete-event part of χ is based on concurrent programming, which is essential for modelling industrial systems that contain interacting controllers operating in parallel.

[1] In *Proceedings of IFAC 7th Symposium on Computer Aided Control Systems Design (CACSD'97), Gent*, April 1997, pp. 9-14.

2. THE χ LANGUAGE

The χ language is based on a small number of orthogonal language constructs which makes it easy to use and to learn. Where possible, the continuous-time and discrete-event parts of the language are based on similar concepts. The language is based on mathematical concepts with well defined semantics. Unlike many other simulation languages, the χ language uses a symbolic notation for specifications and ASCII equivalents for simulations. The symbolic notation makes specifications easier to read and to develop.

The pronunciation of the χ symbols used in this paper is shown in Table 1. In this paper only a (small) subset of the language is treated. We do not treat the language elements for modelling parallel processes that interact by means of message passing and synchronization, nor do we treat the use of systems for hierarchical modelling. For these and other aspects of the χ language we refer to (Arends, 1996; Beek *et al.*, 1996*b*; Beek *et al.*, 1996*a*). The syntax and operational semantics of the language constructs are explained in an informal way.

A process may consist of a continuous-time part only (DAEs), a discrete-event part only, or a combination of both.

proc *name*(*parameter declarations*) =
⟦ *variable declarations*
; *initialization*
| *DAEs*
| *discrete-event statements*
⟧

2.1 *Data types and variables*

All data types and variables are declared as either continuous or discrete. This is an important distinction with other hybrid modelling languages in which the type of a variable is often implicitly inferred from its use.

Table 1. Pronunciation of the χ symbols.

Symbol	Pronunciation
proc	process
syst	system
xper	experiment
⟦	begin block
⟧	end block
$*$	repeat
[	begin { selection / selective waiting / guarded equation
[]	or
]	end
$\longrightarrow$	then
Δ	delta
∇	nabla (or wait until)

The value of a discrete variable is determined by assignments. Between two subsequent assignments the variable retains its value. The value of a continuous variable, on the other hand, is determined by equations. An assignment to a continuous variable (initialization) determines its value for the current point of time only.

Some discrete data types are predefined like bool (boolean), int (integer) and real. Since all continuous variables are assumed to be of type real, they are defined by specifying their units only. For example, the declaration c : [m/s] defines a continuous variable c.

2.2 *The continuous-time part of* χ

The continuous-time part of χ is based on differential algebraic equations (DAEs). A time derivative is denoted by a prime character (e.g. x').

DAEs are separated by commas

$$DAE_1, DAE_2, \ldots, DAE_n$$

If the set of DAEs depends on the state of the system, *guarded DAEs* can be used

$$[\, b_1 \longrightarrow DAEs_1 \,[]\, \ldots \,[]\, b_n \longrightarrow DAEs_n \,]$$

which is pronounced as: 'begin guarded equations, if b_1 then $DAEs_1$, or ... or if b_n then $DAEs_n$, end'. The boolean expression b_i ($1 \leq i \leq n$) denotes a *guard*, which is open if b_i evaluates to true and is otherwise closed. At any time at least one of the guards must be open, so that the *DAEs* associated with an open guard can be selected.

2.3 *The discrete-event part of* χ

The discrete-event part of χ is a CSP-like (Hoare, 1985) real-time concurrent programming language, described in (Mortel-Fronczak and Rooda, 1995) and (Mortel-Fronczak *et al.*, 1995).

Time passing is denoted by

$$\Delta\, t$$

where t is a real expression. A process executing this statement is blocked until the time is increased by t time-units.

Selection ([*GB*]) is denoted by

$$[\, b_1 \longrightarrow S_1 \,[]\, b_2 \longrightarrow S_2 \,[]\, \ldots \,[]\, b_n \longrightarrow S_n \,]$$

which is pronounced as: 'begin selection, if b_1 then S_1, or if b_2 then S_2 or ... or if b_n then S_n, end'. The boolean

expression b_i ($1 \leq i \leq n$) denotes a *guard*, which is open if b_i evaluates to true and is otherwise closed. After evaluation of the guards, one of the statements S_i associated with an open guard b_i is executed.

Repetition of the statement $[GB]$ is denoted by

$$*[GB]$$

The repetition terminates when all guards are closed. The repetition $*[\,\text{true} \longrightarrow S\,]$ may be abbreviated to $*[\,S\,]$.

2.4 *Interaction between the continuous and discrete parts of* χ

In the discrete-event part of a process, assignments can be made to discrete variables occurring in DAEs (e.g. $n := 1$) or in the boolean guards of guarded DAEs. In the former case the DAEs will be evaluated with new values, in the latter case different DAEs may be selected. Continuous variables are initialized immediately after the declarations (e.g. $h ::= h_0$), and may be reinitialized in the discrete-event part.

By means of the *state event* or *nabla* statement

$$\nabla r$$

the discrete-event part of a process can synchronize with the continuous part of a process. Execution of ∇r, where r is a relation involving at least one continuous variable, causes the process to be blocked until the relation becomes true.

3. A TANK LEVEL CONTROL SYSTEM

This section illustrates the use of the χ language for the specification and analysis of discrete-event and discrete-time controllers. The controlled system is a tank shown in Figure 1. The example is based on a study of Van Geldrop (Geldrop, 1995) of discrete-time controller specification in χ.

The constants used in the models are declared in the following way

$$\text{const } g = 9.81,\ \rho = 998$$

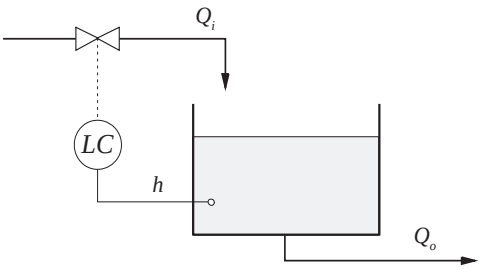

Fig. 1. The tank control system.

3.1 *Discrete-event control*

The first controller is a discrete-event controller that keeps the liquid between two levels, using a hysteresis h_{hys} of 0.1. The χ model of the tank system with controller follows below. The model consists of a single process *TC*. The arguments of the process are listed between brackets. Then the continuous variables h and Q_o are declared. The values of these continuous variables are determined by the equations $Ah' = nQ_{max} - Q_o$, $Q_o = k\sqrt{\rho g h}$. The valve is modelled by the variable n. This variable is declared as a discrete variable, because its value is determined by discrete assignments ($n := 0$ and $n := 1$) only. The height h of the tank is initialized to h_0 ($h ::= h_0$). The operator $::=$ is used, because h is a continuous variable. The controller consists of a repetition ($*[\ \ldots\ ]$). The nabla statement $\nabla h < h_{set} - h_{hys}$ blocks until $h < h_{set} - h_{hys}$ is true. Initially, $h = h_0 = 0.1$ so that the next statement is executed immediately. If the level of the vessel is above $h_{set} - h_{hys}$, the nabla statement blocks until the level sinks below $h_{set} - h_{hys}$. After this, the incoming flow is switched on, which is modelled by the assignment $n := 1$. When the level rises above h_{set} ($\nabla h > h_{set}$), the incoming flow is switched off.

$$
\begin{array}{l}
\text{proc } TC(A, h_0, h_{set}, h_{hys}, k, Q_{max} : \text{real}) = \\
[\![\ h : [\text{m}],\ Q_o : [\text{m}^3/\text{s}] \\
\ ,\ n : \text{int} \\
\ ;\ h ::= h_0\,;\ n := 0 \\
\ |\ Ah' = nQ_{max} - Q_o \\
\ ,\ \ Q_o = k\sqrt{\rho g h} \\
\ |\ *[\ \nabla h < h_{set} - h_{hys}\,;\ n := 1 \\
\qquad ;\ \nabla h > h_{set}\,;\ n := 0 \\
\qquad] \\
]\!]
\end{array}
$$

Below follows the specification of the experiment (xper) that is performed on the model. In this experiment, the process *TC* is instantiated with the values 10, 0.1, 1, . . ., which are assigned to the respective pa-

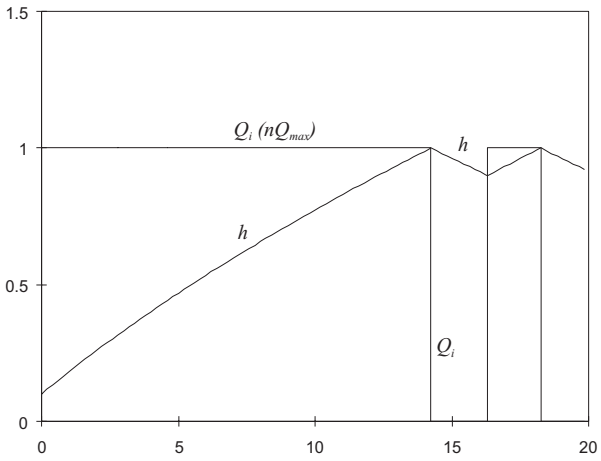

Fig. 2. Discrete-event on/off control.

rameters $A, h_0, h_{set}, \ldots$. The results of the simulation run are shown in Figure 2.

xper $= [\![\ TC(10, 0.1, 1, 0.1, 0.005, 1)\ ]\!]$

3.2 *P control*

The χ model of the tank system with a discrete-time proportional controller follows below. In the repetition of the discrete-time controller, the error e ($e := h_{set} - h$) and the control output u ($u := K_p e$) are calculated first. The valve is modelled by the selection statement

$$
\begin{array}{l}
[\ u < 0 \quad\ \longrightarrow Q_i := 0 \\
[]\ 0 \le u \le 1 \longrightarrow Q_i := u\,Q_{max} \\
[]\ u > 1 \quad\ \longrightarrow Q_i := Q_{max} \\
]
\end{array}
$$

It is assumed that the valve operates in its linear mode for $0 \le u \le 1$ ($0 \le u \le 1 \longrightarrow Q_i := u\,Q_{max}$). If the valve is fully opened ($u > 1$), the input flow equals Q_{max}. If the output u becomes negative ($u < 0$), the flow becomes zero. After setting the input flow, the sampler waits for t_s seconds ($\Delta\, t_s$). In the χ language, assignments and selection statements do not cause the simulation time to advance. Therefore, in this model of the controller, the samples are spaced at exactly t_s seconds.

```
proc TC(A, h0, hset, k, Kp, Qmax, ts : real) =
[[ h : [m], Qo : [m^3/s]
, e, u, Qi : real
; h ::= h0 ; Qi := 0
| Ah' = Qi - Qo
,  Qo = k*sqrt(rho*g*h)
| *[ e := hset - h
   ; u := Kp*e
   ; [ u < 0       --> Qi := 0
     [] 0 <= u <= 1 --> Qi := u*Qmax
     [] u > 1       --> Qi := Qmax
     ]
   ; Delta ts
   ]
]]
```

Below follows the specification of the experiment that is performed on the model. The proportional gain K_p is set to 20, and the sampling time to 0.1 seconds. The results of the simulation run are shown in Figure 3. The final error $h_{set} - h$ approaches 0.024.

xper $= [\![\ TC(10, 0.1, 1, 0.005, 20, 1, 0.1)\ ]\!]$

3.3 *PI control*

In order to eliminate this final error, an integral action is introduced in the process *TC*, as shown below. The integral term is represented by the discrete variable I_e.

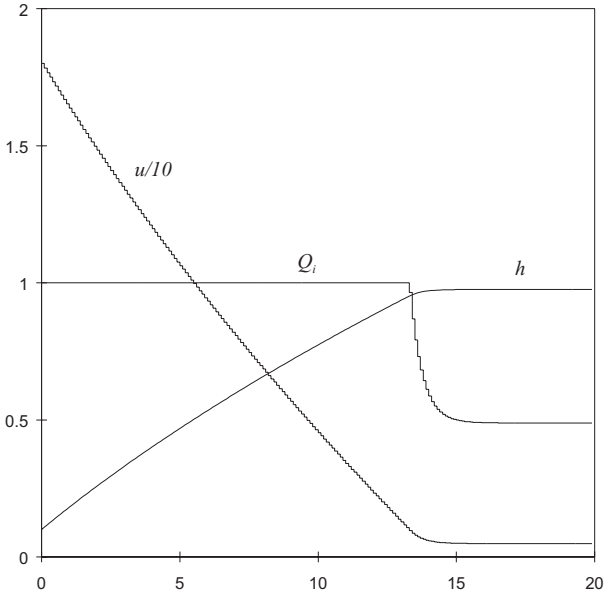

Fig. 3. P control ($K_p = 20$).

The dots $\cdots$ are used to indicate parts of the specification that have not changed. The results of the simulation run are shown in Figure 4. The final error $h_{set} - h$ now approaches zero, but the settling time becomes much longer and there is a considerable overshoot. This integrator windup is caused by the fact that the integrator keeps integrating when the output u saturates ($u > 1$).

```
proc TC(A, h0, hset, k, Ki, Kp, Qmax, ts) =
[[ h : [m], Qo : [m^3/s]
, e, Ie, u, Qi : real
; h ::= h0 ; Qi := 0; Ie := 0
...
```

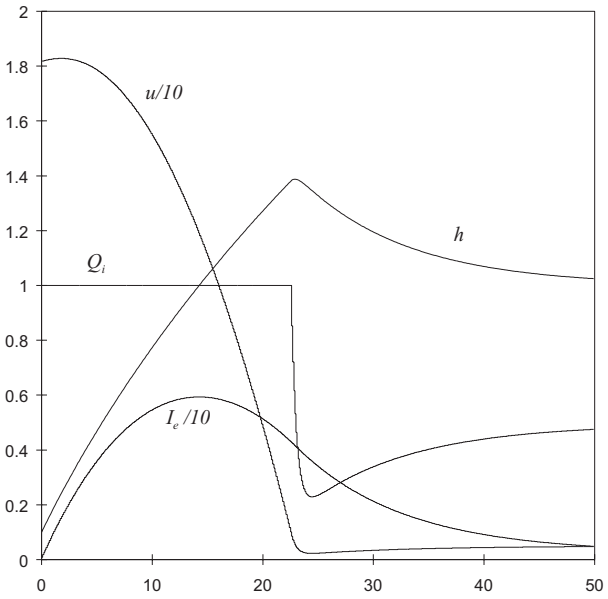

Fig. 4. PI control ($K_p = 20,\ K_i = 2$)

```
| *[ e := h_set − h
   ; I_e := I_e + e t_s
   ; u := K_p e + K_i I_e
   ; [ u < 0        ⟶ Q_i := 0
     [] 0 ≤ u ≤ 1   ⟶ Q_i := u Q_max
     [] u > 1       ⟶ Q_i := Q_max
     ]
   ; Δ t_s
   ]
]|
xper = |[ TC(10, 0.1, 1, 0.005, 2, 20, 1, 0.1) ]|
```

3.4 *PI control with anti-windup*

Several anti-windup strategies are treated in the literature (e.g. (Bohn and Atherton, 1995)). Here we use a strategy for discrete-time controllers. The process *TC* with anti-windup is shown below. The amount of saturation of the output ($(u-1)$ for $u > 1$, and u for $u < 0$) is subtracted from the integral term I_e in order to reduce the saturation effect. This may initially lead to negative values of I_e, as is shown in Figure 5. The values shown in the figure are the values of the variables just before and just after the $\Delta\, t_s$ statement.

```
proc TC(A, h_0, h_set, k, K_i, K_p, Q_max, t_s) =
...
| *[ e := h_set − h
   ; I_e := I_e + K_i e t_s
   ; u := K_p e + I_e
   ; [ u < 0        ⟶ Q_i := 0
                      ; I_e := I_e − u
     [] 0 ≤ u ≤ 1   ⟶ Q_i := u Q_max
     [] u > 1       ⟶ Q_i := Q_max
                      ; I_e := I_e − (u − 1)
     ]
   ; Δ t_s
   ]
]|
xper = |[ TC(10, 0.1, 1, 0.005, 10, 20, 1, 0.1) ]|
```

4. THE χ COMPILER AND SIMULATOR

The χ simulator for the discrete-event part of the language (Naumoski and Alberts, 1995) is being used in a large number of cases. The cases deal with modelling and simulation of complex discrete-event manufacturing systems, such as production facilities for integrated circuits. The first version of the hybrid χ simulator for combined continuous-time / discrete-event systems (Fabian and Janson, 1996) has recently been completed. A χ model is first compiled and linked (using the DASSL (Petzold, 1983) DAE solver), and can then be executed. Although execution of the χ models is already quite fast, further work needs to be done on optimization of the algorithms for equation solving and discontinuity handling. The current experimentation language (xper) is rather limited. Future enhancements will include a more elaborate experimentation language. The experimentation model will then be read during the execution of the compiled χ model. In this way, experimenting with different settings will be possible without recompiling or relinking the model. Currently, experimenting with different settings is done by explicitly reading these settings from a file. In the example, this file input is not shown.

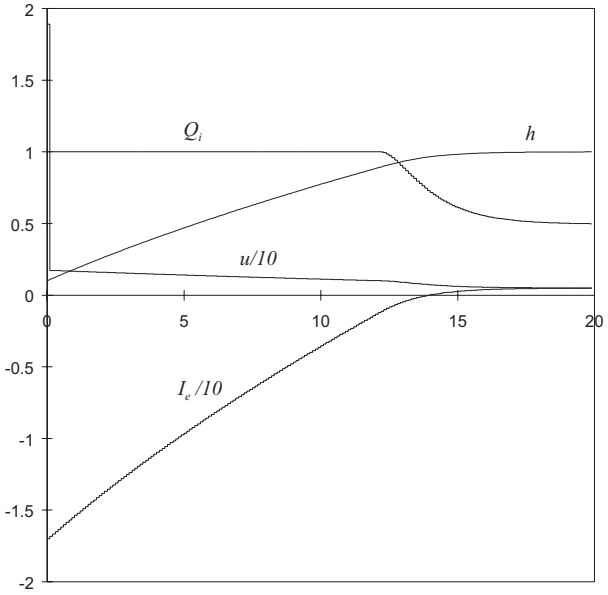

Fig. 5. PI control with anti-windup ($K_p = 20$, $K_i = 10$)

5. CONCLUDING REMARKS

The suitability of the χ language to modelling and simulation of stand-alone discrete-event and discrete-time control systems has been illustrated using a basic tank level control system. The χ language is, however, not limited to stand-alone controllers, but can also be used for modelling complex systems consisting of many interacting controllers. Indeed, plant wide control systems are currently being modelled in χ. Research is also done on error handling and actual real-time control using the χ language. In (Beek and Rooda, 1996), a proposal dealing with an advanced exception handling mechanism for concurrent control system specification is treated.

Despite the wide range of control systems that can be modelled using the χ language, it is relatively easy to use and to learn because of the small number of orthogonal language constructs. In this way, the use of the χ language and simulator can be an important aid for the development of a wide range of robust control systems.

6. REFERENCES

Arends, N.W.A. (1996). A Systems Engineering Specification Formalism. PhD thesis. Eindhoven University of Technology. The Netherlands.

Beek, D. Albert van, S.H.F. Gordijn and J.E. Rooda (1996*a*). Integrating continuous-time and discrete-event concepts in modelling and simulation of manufacturing machines. *Simulation Practice and Theory*. To be published.

Beek, D.A. van and J.E. Rooda (1996). A new mechanism for exception handling in concurrent control systems. *European Journal of Control* (2), 88–100.

Beek, D.A. van, J.E. Rooda and S.H.F. Gordijn (1996*b*). Hybrid modelling in discrete-event control system design. In: *CESA'96 IMACS Multiconference, Symposium on Discrete Events and Manufacturing Systems*. Lille. pp. 596–601.

Bohn, C. and D.P. Atherton (1995). An analysis package comparing PID anti-windup strategies. *IEEE Control Systems* **15**(2), 34–40.

Elmqvist, H. (1994). *Dymola–Dynamic Modeling Language–User's Manual*. Dynasim AB. Lund, Sweden.

Fabian, G. and P. Janson (1996). Simulator for combined continuous-time / discrete-event models. Final report of the Postgraduate Programme Software Technology. Eindhoven University of Technology, Stan Ackermans Institute. The Netherlands.

Geldrop, P.A.B.F. van (1995). The specification and analysis of discrete-time systems. Report WPA 420082. Eindhoven University of Technology, Department of Mechanical Engineering. The Netherlands.

Hoare, C.A.R. (1985). *Communicating Sequential Processes*. Prentice-Hall. Englewood-Cliffs.

Mitchell, E.E.L. and J.S. Gauthier (1976). Advanced continuous simulation language (ACSL). *Simulation* **26**(3), 72–78.

Mortel-Fronczak, J.M. van de and J.E. Rooda (1995). Application of concurrent programming to specification of industrial systems. In: *Proceedings of the 1995 IFAC Symposium on Information Control Problems in Manufacturing*. Bejing. pp. 421–426.

Mortel-Fronczak, J.M. van de, J.E. Rooda and N.J.M van den Nieuwelaar (1995). Specification of a flexible manufacturing system using concurrent programming. *Concurrent Engineering: Research and Applications* **3**(3), 187–194.

Naumoski, G. and W.T.M Alberts (1995). The χ engine: a fast simulator for systems engineering. Final report of the Postgraduate Programme Software Technology. Eindhoven University of Technology, Stan Ackermans Institute. The Netherlands.

Petzold, L.R. (1983). A description of DASSL: A differential/algebraic system solver. *Scientific Computing* pp. 65–68.