Regsim: A Software Tool for Real Time Control and Simulation

Thomas Gustafsson
Control Engineering Group

University of Luleå, S-971 87 Luleå, Sweden
thomas@sm.luth.se

Keywords: Real time computer systems, Simulation, Software tools, CACE

Abstract

This paper presents a software tool, Regsim, that bridges the gap between simulation and real world experiments of control systems. The ideas behind Regsim are that it should be interactive, easy to use and that it should behave uniformly irrespective of the use, whether it is simulation or real-time control.

1. Introduction

When a control engineer has designed a new control algorithm, he is usually keen to validate the design in a laboratory experiment. Regsim is a software tool that makes the step from design to experiment short and encourages the engineer to continuously work with the real world.

In the projects described in Gustafsson [4] a lot of the time was spent working in the laboratory with experiments trying to verify theories and control algorithms designed at an earlier stage. We found that the success of an experiment was often dependent of the quality on the software that was used.

This was especially true in the first project dealing with the control of a Cargo Rotating Device. The control software was mainly hand coded in the Z80-assembly language and executed on a CP/M-computer. A few simulations were made with a small BASIC-program on a somewhat larger computer. The main experience from this period was that control experiments should be avoided unless you were extremely interested in debugging assembler programs.

For a control engineer this is not a satisfying situa-

tion since it is our belief that a true control engineer should both master the theory and be able to convert the theory into useful, functional control algorithms. These have to work in the real world that not always pays attention to the crucial assumptions you were forced to make in the process of constructing proofs for your theorems.

In a subsequent project dealing with the control of a rotary crane system the lesson was learned and the excellent simulation program Simnon (Elmqvist [2]) was acquired. The available computers were, however, not of the right brand and the version of Simnon that we had, used a slow interpreter and not a fast compiler as in other implementations. This in conjunction with that the computer was heavily overloaded by diligent students made it in practice impossible to use the program. A simple simulation of a second order non-linear system could take up to one hour. This was of course a disappointment since we had learned to appreciate the way dynamic systems were described in Simnon code, but it was also the injection that later gave birth to Regsim.

As a provisional arrangement we wrote a simple simulation program in Pascal that had a rudimentary interactive interface that made it possible to change parameters and choose what signals to plot or save. The simulation models were hard coded in Pascal procedures, which made it necessary to recompile the program each time a change in the simulation model was made. This was of course time consuming but the gain in simulation speed was considerable and more than enough to motivate the use of the program.

During the project more and more features were added to the simulation program and it was also ported to a PC-compatible computer and made use of its graphic capabilities. In the latter part of the project when the experimental phase was about to begin and a search for a suitable platform to implement the real time controller system had begun, then the memories from the first project were haunting

us. Since the experiments this time would be out of town and we had a limited amount of time to our disposition, there was no room for debugging or other unessential exercises.

It was soon clear that we should use a PC computer and the idea was born that we should make use of the simulation program that at this time had a rather good user interface. The main idea was that the code describing the real time controller should be exactly the same as the code used for the controller in the simulation program. That way all errors due to the implementation would be avoided since it was possible to carefully test the controller in advance with simulations.

Now since we had already implemented a real time kernel that was heavily used in the control laboratory it was a rather simple task to incorporate real time capabilities into the simulation program.

The combination of a simulation program and a real time implementation of a control system proved to be successful at the full scale experiments. Unexpected behaviors of the process¹ could sometimes be reproduced in a simulation. Then an enhanced control algorithm that managed the simulated process usually proved to work even on the real process.

After the encouraging experiences from the full scale experiments the program, now called Regsim, was completely rewritten in Modula-2 and more important a compiler was written to translate dynamic system models described in a Simnon like language to machine code. This made the program a lot easier to handle and the simulation speed was increased considerably. It is possible to go much further in the optimization of the generated code with a specialized compiler that has knowledge of how to simulate a dynamic model, compared to what a Pascal or Modula-2 compiler can achieve.

2. Design Outline

Regsim is composed of several interacting software components where the most important ones are shown in figure 1. The database shown in the middle is not a real component, it only indicates the existence of a common set of data that are accessible by all other components. The access is, however, controlled by the real time kernel, thus avoiding the classical read-write problem in a real time system.

¹The most unexpected behavior was not in the process, but in the computer. The floating point processor showed to have a bug: multiplying zero with a number smaller than 10^{-308} sometimes produced infinity as a result !

Figure 1: Block scheme showing the principle structure of Regsim

The size of the source code to Regsim is about 31,000 lines of original Modula-2 code. The simulation engine and the real time kernel are both small with about 500 lines of code each. The compiler is together with the user interface largest and consists of 7000 lines of code each.

Some of the more important components will be described in detail starting with the compiler.

2.1. Compiler

The compiler is a fairly standard recursive descent compiler that produces optimized code directly for a numerical coprocessor. The code is, however, generated in two steps. Firstly an intermediate machine independent code for a fictive stack based processor is produced. Then a small (500 lines) machine dependent part of the compiler translates the intermediate code into machine code. There is also an alternative where the intermediate code is executed directly by an interpreter.

A non standard but essential feature is that all equations are sorted before the code is generated. The precondition for the sorting algorithm is that all variables except state variables and parameters are undefined and in order to evaluate an equation and calculate a value for an undefined variable, then all variables used in the equation must be defined. This means that recursion or algebraic equations of the type

$$y = y - 1$$

are not allowed since y can not be defined before the evaluation.

The code produced for each system is classified in three categories, which are slightly different depending on whether it is a continuous, time discrete or a discrete event system. The common category is the time independent code that has to be executed only once, typically calculation of variables only depending on parameters. Some care must, however, be taken when parameters are changed in the middle of a simulation or real time control experiment. All variables depending on the changed parameter must also be

changed to ensure a correct result. In a simulation this is not a big problem since the evaluation is sequential. The problem occurs when systems are run in real time and the parameters are changed asynchronously. Then the real time kernel is responsible for ensuring that all dependent variables are changed before they are used in a discrete system.

An option in the compiler is to evaluate time independent subexpressions only once, e.g the PI controller in listing 3.1 has the line

```
ni = i + k*h/Ti*e
```

where k , h and T_i are parameters and the subexpression $k*h/T_i$ is then only evaluated once.

For a continuous system the code used for calculating derivatives of the state variables is minimized and evaluated separately since it is heavily used by the simulation engine to integrate the state variables. The remaining code is only necessary to execute if the variables are plotted or exported to another system.

2.2. Real Time Kernel

The real time kernel is responsible for scheduling the execution of the real time processes. There are mainly five different types of processes each with a different priority. They are listed below with decreasing priority

- Each discrete system constitutes a separate process with the sampling period defined by the parameter `samptime`. The sampling time must be fixed, but it can be changed at any time by the operator.
- If logging is asked for, then the sampling period of the logging process can be given explicitly or it can be automatically determined from the logged variables.
- Plotting can be made in four different windows each with a unique sampling period that can be changed with a simple command.
- Every second a small process is executed that shows the elapsed time in a small information window
- The command interpreter runs as a background process without priority. This can be a problem if the other processes are time consuming. Then there is very little time left for the command interpreter and the result is a sluggish user interface.

The real time kernel itself is executed periodically with a period time that is calculated as the greatest common divisor of all the sampling periods of the running processes. The administrative duties of the kernel are kept to a minimum to ensure a short overhead at process switches. The overhead on a 80486

computer is about 50 μ s and the shortest sampling period is set to 1 ms, although it is possible with sampling periods down to 0.1 ms.

A very important detail is that it must be possible to define an execution order between the discrete systems.² This is essential as it otherwise could lead to unexpected time delays in the signals between two connected systems. The problem can be illustrated with the simple example in listing 2.1.

```
DISCRETE SYSTEM reg
TIME t
TSAMP ts
u = k*( ref - y[proc] )
ts = t + samptime/1000
k : 1
END

DISCRETE SYSTEM proc
STATE x
NEW nx
TIME t
TSAMP ts
nx = x + u[reg]
y = x
ts = t + samptime/1000
END
```

Listing 2.1: A controller – process pair with an inherent conflict, the execution order is indeterminable.

The execution order between the controller `reg` and the process `proc` is indeterminable since a system is not allowed to execute before all inputs to the system are calculated. In this case, then `reg` is depending on the input `y` calculated in `proc`, but `proc` is depending on `u` calculated in `reg` and the recursion goes on forever.

This is, however, normally not a problem in Regsim since the discrete systems are, if necessary, divided into two parts, one minimized part that calculates the outputs and one part where the rest of the variables are calculated.

In the actual case the execution order will be

```
proc: y = x

reg:  ts = t+samptime/1000
      u = k*(ref-y[proc])
```

²One could argue that if the systems have different sampling periods and are sampled at different times then the problem never occurs, but since the sampling period is an integer value then the sampling times eventually will coincide, regardless of the sampling periods.

```

proc: nx = x+u[reg]
      ts = t+samptime/1000

```

Notice that this problem does not occur with continuous systems since all continuous systems are lumped together in one system and there is no need to establish an execution order between the systems.

As an alternative to a fixed sampling period there is an option to control the sampling of a discrete system with external events. This has been used by Hillerström & Sternby [7] to implement a periodic disturbance rejection algorithm to control the angular rate of a peristaltic pump. The sampling was synchronized with the rotation angle of the pump axis.

2.3. User Interface

Regsim contains an interactive command driven user interface, with a repertoire of about 60 commands. Most of the commands can be used even when the real time kernel is active. This is a key feature of Regsim since it is very important for instance to be able to change parameter values or change what variables should be plotted during an experiment without the need to restart the controller.

Almost all commands that take an argument use the compiler to evaluate expressions given as arguments. This is neat if one for instance wishes to increase the value of the parameter k with $\sqrt{15}$ which is easily done with the command

```
par k k+sqrt(15)
```

Or if one wants a plot of the difference between two variables then the command

```
plot x-y
```

does the work. All variables defined in the systems are accessible during an evaluation of a command argument.

The presentation of the simulation or real time control results can be done with different types of graphics, as curves, bar graphs, or simple animated figures. Figure 2 shows a snapshot from a session with Regsim on a Macintosh computer, where both curves and an animated picture is displayed. There is no limit on the number of graphical objects, and it is possible to mix all types of objects in a single window.

Notice in figure 2 the window in the lower left corner. It contains a list of all variables and their values. It is possible to show live values during a simulation. This decreases, however, the simulation speed considerably. The user can also easily change any parameter value and immediately see the effect in the simulation and in the values of all dependent variables. Thus it is possible to use Regsim as a simple spread sheet pro-

Figure 2: Screen snap shot from the Macintosh version. The window named "Grafik 1" displays a 3-D animation of a bifilar pendulum.

gram. A natural extension, to increase the similarity, would then be to both show the equations and allow the user to change them.

A useful feature is the command files that can be used to create new commands. Listing 2.2 shows an example of a command file.

```

MACRO fig8 pole
default pole w
comp pendel reg2
load luta.SAV
maxwin 1
size -x 10 340 -y 100 230
scale -x 0 40 -y -0.02 0.12
Alias go (init;par jtet 0.8;init;simu -w 20;
          par jtet 0;simu -w 40)
axes; init
# position controller
par kj 1; par pteta $(pole)
plot td[kran] 0 -h " "
go
# rate controller
par pteta 0
plot -i 2 td[kran] -h " "
go
dumpeps -f fig8.eps

```

Listing 2.2: An example of a command file.

Command files can be nested and take arguments and there are several control structures as loops and if-then-else constructions that can be used. This makes it possible to automate simulation experiments.

Another way to augment the command repertoire is to use the alias command which is a simpler form of

a command file. The advantage, compared to command files, is that an alias can be used even when the real time kernel is active.³ This makes it possible with a custom made interface e.g. in control lab courses. A command file can be used to compile the system descriptions and define menus and simple alias commands etc. that facilitates the use of Regsim. The student can then concentrate the work on automatic control.

2.4. Simulation Engine

This is a very important part, since the quality of a simulation depends mainly on the algorithms used in the integration method. This is especially true for stiff systems where there exist different time scales. There are today a vast number of integration methods that adapt to the model. In Regsim two integration methods are used, one third order Runge-Kutta method due to Fehlberg [6] with automatic step size selection and one fifth order Runge-Kutta method due to Dormand and Prince [1] but with a step size selection scheme proposed by Gustafsson [3]. The second method is especially good for stiff systems. It has, however, some problems if the model has hard nonlinearities such as relays. Then the step size tends to be very short unless some special care is taken to avoid the problem. The first method is sufficiently good in most cases even with hard nonlinearities but if the model is stiff, then its step size selection scheme can be unstable.

There is one disadvantage with good integration algorithms. Sometimes the step size becomes too long with the effect that a plotted curve can be broken. This is of course only a cosmetic error but there is an algorithm in the simulation engine that estimates the interpolation error made in the plotting and assures that it is below a limit selected by the user.

The simulation can be synchronized to run in real time, or in a scaled real time if a slower or faster simulation is preferred. This is especially useful when the simulation result is presented with animation.

If discrete events are present in the simulation model, special care is taken to find the correct time for the events. An event is usually specified as a boolean expression that includes an inequality, the expression is transformed to an auxiliary real-valued function such that the event will occur when the function goes from a negative to a positive value. The time for the event can then be found by searching for zeros of the auxiliary function.

³The file system can not be used when the real time kernel is active since DOS disables all interrupts during a disc access thus preventing the real time kernel from doing its job.

3. Some Examples

A main feature with Regsim is that it is possible to use the same code for simulation and for real time control. In listing 3.1 an example of a PI controller that can either be used to control a simulated process or a real process connected with A/D and D/A converters. This can easily be accomplished by using the boolean function `simulating()`. For example, one row of the code is

```
e = yr-(IF simulating() THEN y ELSE adin(0))
```

Thus the controller error `e` is `yr-y` when the system is simulated and `yr-adin(0)` when the real time kernel is active. The output from the controller is directed in a similar manner.

```
DISCRETE SYSTEM preg
INPUT y
OUTPUT u
STATE i
NEW ni
TIME t
TSAMP ts
INITIAL
samptime = 100
SORT
utemp = k*e + i
e = yr - ( IF simulating() THEN y ELSE adin(0) )
ni = i + k*h/Ti*e
u = IF simulating() THEN utemp ELSE daut(0,utemp)
ts = t + samptime/1000
k : 1
Ti : 10000
END
```

Listing 3.1: A Regsim PI Controller that works both in simulation and as a real time controller

The description language in Regsim is, as mentioned before, closely related to the description language in Simnon. There are, however, some differences. One is that it is possible to use vectors and matrices in Regsim, and almost all functions take matrices as argument. This makes it for instance easy to implement neural network algorithms and recursive identification algorithms. A complete description of the description language is found in Gustafsson [5].

The full capacity of Modula-2 can be utilized if necessary, since it is possible to include modules written in Modula-2. This is certainly necessary when the communication with the process is done by other means than with A/D and D/A-converters. The normal way to enhance the capabilities of Regsim is to write functions that can be called inside a Regsim system but it is also possible to create new commands and write

discrete systems in Modula-2. All variables defined in a Modula-2 system can if necessary be accessed by Regsim systems and vice versa.

The modules written in Modula-2 are loaded at run-time so there is no need to change Regsim. This is in fact the way different drivers for A/D and D/A-converters are loaded and it makes it very easy to change the hardware. Usually there is no need to rewrite the systems descriptions interfacing the hardware if standard names and arguments are used for the interface functions.

4. Conclusions

Regsim has proved its all-round capabilities in many real time control implementation from the control of a model-sized bridge crane to a full-size 60 m high harbor crane. Other uses have been the control of autonomous vehicles, robots, peristaltic pumps and ore crushers to mention a few.

The most common use is perhaps as an educational tool in the control laboratory where it gives the students an opportunity to experiment with different types of controllers and designs in a very flexible and straightforward manner.

The next step is to integrate Regsim in a CACE environment. To facilitate this work, Regsim has been translated to C++ and an object-oriented programming style has been adopted.

References

- [1] J. R. Dormand and P.J Prince. A family of embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19-26, 1980.
- [2] H. Elmquist. Simnon user's manual. Technical Report 7502, Department of Automatic Control, Lund Institute of Technology, April 1975.
- [3] K. Gustafsson. *Control of Error and Convergence in ODE Solvers*. Doctor of technology thesis, Department of Automatic Control, Lund Institute of Technology, May 1992.
- [4] T. Gustafsson. *Modelling and Control of Rotary Crane Systems*. Doctor of technology thesis, University of Luleå, June 1993.
- [5] T. Gustafsson. Regsim manual. Technical report, Department of Automatic Control, University of Luleå, 1994. (In Swedish).
- [6] E. Hairer, P. Nørsett, and M. Roche. *Solving Ordinary Differential Equations I - Nonstiff Problems*, volume 8 of *Springer Series in Computational Mathematics*. Springer-Verlag, 1987.
- [7] G. Hillerström and J. Sternby. Application of repetitive control to a peristaltic pump. In *Proceed-*

ings of American Control Conference, pages 136-140, San Francisco, June 1993.