Real Time Control using Matlab and Java

Ove Ewerlid Claes Tidestav Mikael Sternad

Signals and Systems Group, Uppsala University

Abstract

In the course "Process Control" offered at Uppsala university, groups of four students use Matlab to design and implement advanced controllers for a laboratory-scale plant.

To implement the graphical user interface, the students use Java. The graphical user interface and the controller communicate over a network in client-server mode.

The Matlab process which controls the plant runs under Linux, a public domain Unix dialect available for e.g. Intel PC:s. The Linux kernel has been extended with real-time capabilities.

The results from the 1997 project course were very encouraging. In particular, non-compiled Matlab 5 code turned out to provide adequate computational speed. By using Matlab for both design and implementation, the development process could be simplified considerably.

1 Background

The course Process Control [1, 2, 3] is taught during the eighth semester of the Engineering Physics Program at Uppsala university. In this project course, groups of four students are required to

- design and implement a controller for a two input-two output system;
- design a user interface for this controller.

Our aim is to provide a bridge between theory and practice: advanced control strategies are implemented, and have to work on a real plant. At the same time, the course provides extensive practice in project work, programming and user interface design. Ideally, the programming skills acquired in this course should also be useful for the students after graduation in a wide area of applications. It is therefore important that we use appropriate and efficient programming tools.

This course was given for the first time in 1982. At that time, the control loop was implemented in FORTRAN on ABC80 computers. The user interface was text based. As experience grew, the controllers improved. Matlab became the tool used for controller design, while the controllers were implemented in Turbo Pascal on 286-computers. However, at the beginning of the 90's, the software tools used in the course were out-of-date. A major revision was necessary.

In 1993, the operating system was changed from DOS to UNIX and a new programming environment was introduced. The controller and the user interface were programmed in C++. The graphical user interface (GUI) was implemented using X Windows primitives and different widget sets, e.g. the Athena Widget Set and Motif. The controller and the graphical interface were separate UNIX processes, communicating using a shared memory area.

The course was now modern. It also provided skills in mastering tools which were widely used in industry. However, both the controller and the interface implementation were prone to errors. Insights in automatic control and user interface design were obscured by the relatively low-level programming. Also, the system was very complex, and the students had no real possibility to grasp the entire setup. It was realized that an implementation based on Matlab could remove these drawbacks.

Compiled languages had so far been required, due to timing constraints: a rather complex controller had to be implemented with a sampling rate of at least 50 Hz. During the spring of 1996, experiments with control loop implementation in Matlab were conducted. Those experiments indicated that with powerful PC:s, it should indeed be possible to run the control loop from within Matlab.

In 1997, this intention was carried out. The controller was implemented in an m-file, which was *interpreted* by Matlab. The graphical interface was designed as a *Java applet*. As a result, both controller implementation and GUI programming were raised to a higher level. The students could focus on controller and GUI design, rather than searching for programming bugs. Instead of sharing memory, the controller and the user interface communicated over the network, making it possible for the two processes to reside on different computers.

The paper is organized as follows. In Section 2, the system which is to be controlled is described. In Section 3, a brief description of the possible control strategies is given. The overall implementation of the control system is described in Section 4. The controller loop, the graphical user interface and the communication between them are presented in Sections 6, 7 and 8 respectively. In Section 10 the the hardware requirements and the software complexity are discussed. Finally in Section 11 some conclusions are drawn.

2 The Coupled Electric Drives plant

The plant to be controlled is a laboratory process built and marketed by TecQuipment Ltd [4]. This process illustrates the problem of controlling tension and speed in material handling and transport and consists of a rubber belt, suspended over three wheels as shown in Fig. 1. The two lower

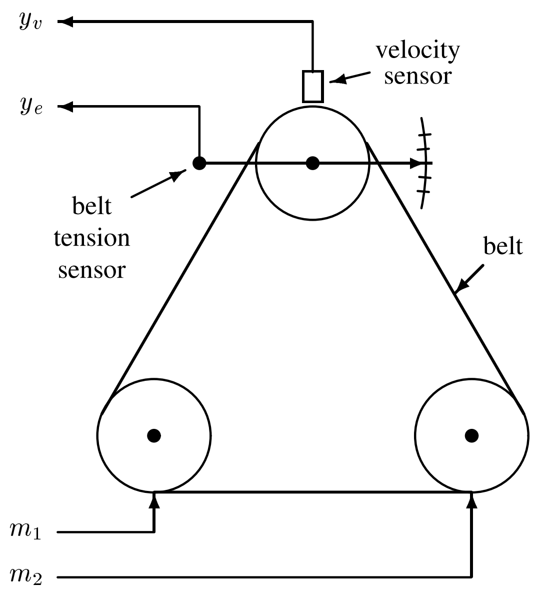

Figure 1: The Coupled Electric Drives laboratory-scale plant

wheels are connected to separate DC motors. The voltages to the two motors are the two input signals used to control the plant.

The upper wheel is mounted on a rotating arm, which is suspended by a spring. The velocity of the upper wheel can be measured, as can the angular position of the arm. This angular position indicates the tension in the upper part of the belt.

Thus, the control problem to be solved is:

> *Use the voltages to the two motors to control the velocity y_v and the tension y_e of the rubber belt, as measured by the velocity at the upper wheel.*

For most project variants, we have chosen to reduce the control problem to two scalar problems. With a simple decoupling link, the system can be transformed into two single input-single output systems. Two observations lead to this decomposition. First, the belt tension will depend mainly on the difference between the two motor voltages. Second, due to symmetry, the belt velocity will depend mainly on the sum of the two motor voltages. If we thus use the transformation

$$\begin{aligned} u_v &= m_1 + m_2 \\ u_e &= m_1 - m_2 \end{aligned} \tag{1}$$

then u_v will mainly affect the velocity of the belt, whereas u_e will mainly affect the tension in the belt. The transformation (1) is invertible. It is therefore possible to transform the two input-two output system approximately into two single input-single output systems, see also Fig. 2.

When designing controllers for the Coupled Electric Drives plant, the students model the two subsystems separately and then design two single input-single output controllers. These single input-single output controllers compute u_v and u_e, while m_1 and m_2 are computed from the inverse of (1).[1]

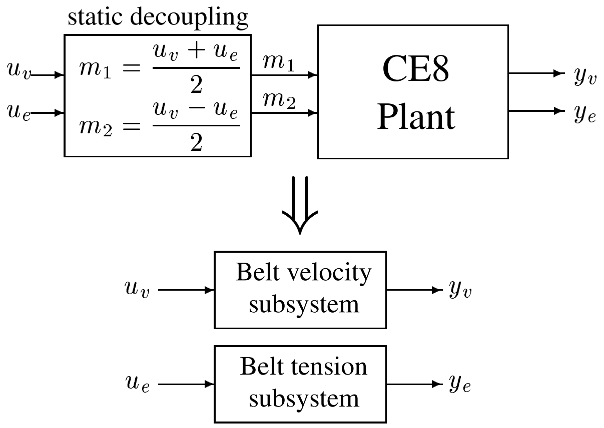

Figure 2: Transformation of the two input-two output system into two single input-single output systems. The velocity subsystem has rather well damped third order dynamics, whereas the tension subsystem is governed by a fourth order system with a lightly damped 7 Hz oscillatory mode.

3 Different control strategies

All the groups implement PID controllers for the velocity and the tension. In addition, each group selects another control strategy to implement advanced single input-single output controllers for both entities. These advanced controllers are based on modeling, process identification or adaption. The model based controllers the students may choose are

- linear input-output controllers based on transfer function models;
- linear state space controllers based on state space models;

In addition, fuzzy logic controllers have also been investigated. The adaptive methods [6] used in the course over the years are

- the generalized minimum-variance controller of Clarke and Gawthrop;
- indirect adaptive controllers based on recursively estimated input-output models;
- auto-tuned PID controllers.

To model the plant, different groups of students use:

- physical modeling;
- frequency domain identification, either frequency analysis or spectral analysis;

[1] During the spring of 1997, one group successfully tried a more complex approach: identify a two input-two output model of the system without the static decoupling. The model parameters were adjusted using subspace identification [5]. Then, a two input-two output controller was designed based on this model and applied to the plant without the static decoupling. Thereby, the cross couplings in the system can be accurately modeled and compensated for.

- time domain parametric identification.

4 Overall implementation

The entire control system is implemented under *Linux*, a UNIX dialect freely available for e.g. Intel PC:s, see e.g. [7]. Small alterations in the Linux kernel were sufficient to enable operation in a real time framework.

The two major parts of the control system are very different. The controller should run exactly once every sampling period, whereas the graphical user interface is asynchronous and event driven. Since the two parts have so different demands, the two tasks should be separated as clearly as possible. The controller and the graphical user interface have therefore been implemented as *separate UNIX processes*.

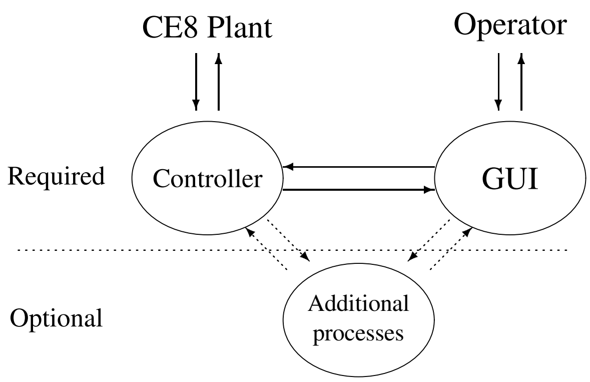

Figure 3: Overall implementation of the control system.

Communication between the controller process and the GUI is accomplished via the computer network. The controller sends information about the state of the process to the user interface, whereas the GUI sends information to the controller about user interactions in client-server mode.

As indicated in Fig. 3, additional processes can also be managed. Such a process could for example be an additional Matlab process which receives information from the controller about the system, computes some interesting quantity and transmits this information to the GUI.

As a starting point, the students are provided with a primitive controller process, containing proportional controllers, functions for inter-process communication and a primitive user interface. The students subsequently modify and extend these programs.

Since only two laboratory setups are available, a simulator has been implemented. This simulator mimics the multivariable dynamics of the true plant, allowing the students to develop their programs without access to one of the Coupled Electric Drives plants. Interfacing with the simulator is identical to interfacing with the real plant; only a single instruction is necessary to switch between the two.

5 Real time and IO extensions

The Linux kernel has been extended with a real time scheduler.[2] The scheduler determines what UNIX processes are run and when they are run. In Matlab, these extensions are made available as a number of C-MEX functions. Linux has also been extended with drivers for the AD/DA card. The implementation of such AD/DA drivers is straightforward, and will not be described any further.

The control algorithms need to be run at a sampling rate of at least 50 Hz. To accommodate these real time requirements, the standard Linux scheduler must be modified. Only the process implementing the controller needs hard real time support so the modifications can be kept simple.

Two parameters are available for the control of the real time scheduler: the number of ticks[3] per sample and the number of ticks before preemption. The real time task is initiated once every sampling period, and *suspended* when preemption occurs. When the next sampling period begins, execution proceeds at the point of preemption. [4] Fig. 4 illustrates how the scheduler operates. To simplify the design the rate of the default scheduler clock was increased from 100 Hz to 1000 Hz. Sampling rates of up to 500 Hz can be handled with this setup.

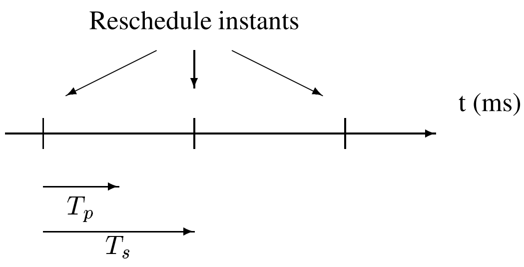

Figure 4: The real time scheduler. The real time task is rescheduled every T_s seconds. After T_p seconds preemption occurs, i.e. the real time task is suspended.

Our real time extensions are confined to the scheduler, leaving some real time aspects open, e.g., disk and network IO. In practice, this is not a problem, since the disk and ethernet support use bus mastering. The real time task is scheduled directly by the timer interrupt, yielding latencies which are typically better then 100 μs. In fact, Matlab itself is a greater source of timing jitter.

Another important aspect to take into account in a real time implementation is *garbage collection*. Garbage collection is the process of deallocation unreferenced memory. Garbage collection in Matlab is to a great extent avoided by pre-allocating matrices.

[2]Today there are a number of real time extensions for Linux that could be used. See e.g. [8]. In 1993 no such implementations were available.

[3]A tick is a period of the scheduler clock.

[4]Obviously, the controller should be designed so that preemption occurs at the very end of the control loop, in the waiting state.

6 The controller process

As stated earlier, the entire controller is implemented in a single m-file. We use Matlab 5.1 for the implementation. The possibility to use the **case** statement in Matlab 5 is vital for the performance: under worst case conditions, execution would be considerably slower if a large number of **if/then/else** clauses were used.

The commands to be executed during each sampling interval are contained in a **while/end** block. During each iteration of this block, the following events occur:

- the output signals from the plant are measured;
- the control signals are computed;
- the control signals are fed to the plant;
- the plant is checked for possible alarm conditions;
- interesting quantities are transmitted to the interface;
- wait!

At the end of the control loop, execution enters a *waiting state*. This state is interrupted at the beginning of the next sampling interval by the real time scheduler. See Section 9 for an control loop example.

7 The graphical user interface

The graphical user interface is implemented using *Java* [9]. Java has many advantages when used for user interface programming:

- Java is relatively *simple and high-level*, making it possible for the students to concentrate on overall design issues rather than on finding programming bugs.
- Despite being an *interpreted* language, Java has relatively *high performance*.
- Java has *generic network support*, making networked communication easy.
- Java is *multi-threaded*, making it for instance possible for a single executable to both listen to a network port and to wait for keyboard and mouse events.
- Java is *portable*. This makes it possible for the students to run their control system interface on different types of computers.
- Java has a rich set of GUI components.
- Java is hot . . .

The *Java Development Kit (JDK)*, version 1.02 was used as programming environment. To make the interface even more portable, it was designed as a Java *applet*, rather than as a stand-alone Java application. The resulting design can then be run from a Java capable WWW browser, such as Netscape Communicator. The interface can also be a part of a Web site.

Some interface components were specific to our application. Those classes were custom designed. Graphs scrolling in real time were examples of such components. Also, some communication primitives had to be designed from scratch.

As a future development, an interface builder could (and should) be used. All custom designed Java classes would then be implemented as *Java beans* and used in the GUI builder. This is the way interface design is going in commercial applications.

8 The inter-process communication

The communication between the processes is performed via the computer network. It is therefore possible for the controller process and the GUI to reside on different computers. The entire capacity of one computer can then be reserved for running the time consuming controller loop whereas the GUI is run on another computer.

Communication takes place using the *User Datagram Protocol (UDP)* [10]. UDP packets are sent by one process to a specific port on a specific computer. A process on the receiving computer listens to that port. Since one of the involved processes is a Matlab process in our case, and Matlab only handles Matlab arrays, special functions must be used to pack, send, receive and unpack packets containing Matlab arrays. Such functions have been implemented for numerical Matlab arrays, both in Matlab and in Java. The UDP primitives are available in Matlab through a number of C-MEX functions. In Matlab, one, two, three or four arrays are packed into one UDP packet. Also, an arbitrary string is included in the packet to provide a mechanism for packet identification. The maximum size of each packet is 64 kB. In practice, these packets are almost always smaller than the MTU[5]. UDP is not lossless like TCP[6], but in our setup, with a dedicated, full duplex, point-to-point 100 Mbits/s ethernet link, the loss is negligible.

In Java, packets are received and unpacked. Each Matlab array is transformed into a two-dimensional array of Java **Double**:s. The information string in the packet is extracted to a Java **String**. The transmission of UDP packets from the Java process to the Matlab process is analogous.

9 Sample code

The sample code below illustrates the basic structure of the controller. In practice, the **switch** statement contains a large number of cases and possibly nesting of other **switches**. This example contains code that collects the absolute sample time in microseconds since from program start and sends it to the GUI.[7] In line five in this example, the sampling interval is set to 20 ms and the preemption time is set to 10 ms. Note that the C-MEX function *udp_receive* returns the tag *nothing to read* if no packets are available in the queue.

[5] Maximum Transmit Unit - typically 1500 bytes for ethernet.

[6] Transmission Control Protocol

[7] The GUI is free to ignore packets tagged *sample-time* but during the design phase this is an interesting piece of information.

```
AD_data = [0 0 0 0 0 0 0 0];
DA_data = [0 0];

proc_set_target (0);       % Use "real" plant
proc_set_realprio(20,10);  % 50 Hz sampling rate

keep_going = 1;

proc_tic_toc;              % Reset usec-timer

while (keep_going)

  proc_click_speaker (1000); % Good debug aid:
                             % one "click" per
                             % sample

  proc_adda (AD_data, 1, DA_data, 0); % Sample
                                      % plant

  %%% ... control algorithm ...

  proc_adda (AD_data, 0, DA_data, 1); % Control
                                      % plant

  % Deal with GUI (or additional Matlab process)
  [tag, str, m1, m2, m3, m4, srcip, srcport]
    = udp_receive (1); % m1-m4 are Matlab arrays

  switch (str)
  case 'nothing to read'
    % Do nothing
  case 'applet_address'
    applet_ip   = srcip;   % Register applet IP
    applet_port = srcport; % Register applet port
    fprintf (stdout, 'Setting applet IP/PORT\n');

  %%% ... more cases here ...

  otherwise
    fprintf (stdout,
             'unknown pkt: tag=%d, str="%s"\n',
             tag, str);
  end

  if (applet_port > 0)   % Send absolute
                         % sample time to GUI
    usec = proc_tic_toc;
    udp_send (applet_ip, applet_port,
              0, 'sample-time', usec, 0, 0, 0);
  end

  proc_yield; % Sleep until next sample

end

proc_set_realprio(0,0); % No real time priority
```

(Due to security issues, the applet needs to register the IP address and port it uses. This cannot be statically allocated.)

10 Hardware requirements and software complexity

In the 1997 version of the course, two computers were used to control the Coupled Electric Drives plant. The controller process executed on a 200 MHz Pentium Pro computer, which was equipped with 32 MB RAM, and a 100 Mbits/s ethernet card. It interfaced with the Coupled Electric Drives by an ISA card, having eight 12 bit A/D-channels, multiplexed at 50 kHz, and two 12 bit D/A-channels.

The graphical user interface resided on a 90 MHz Pentium PC. This computer was also equipped with 32 MB RAM and a 100 Mbits/s ethernet card.

With this hardware configuration, it was for instance possible to run two indirect adaptive controllers, one for the velocity control and one for the tension control. During each sampling period, models of the belt velocity subprocess and the belt tension subprocess were updated with the Extended Least Squares algorithm [6]. Each model had 8 to 12 parameters. The complete controller loop lasted for about 7 milliseconds, leaving a margin of 3 milliseconds of the sampling period at a sampling rate of 100 Hz or 13 milliseconds at a sampling rate of 50 Hz. As often as possible, the desired pole placements were re-calculated using LQG design and new controller parameters were computed. These computations took about 130 milliseconds to perform and were implemented in a separate Matlab process, cf. Fig. 3. At a sampling rate of 50 Hz, the controllers would be updated five times per second, whereas with a sampling rate of 100 Hz, new controllers can be computed twice every second.

The algorithms necessary for the above operations were all interpreted by Matlab. The m-files for a controller of this complexity typically consist of between 1000 and 1200 lines of Matlab code.

11 Conclusions

It is indeed possible to use Matlab without a compiler to run an advanced controller loop at 100 Hz sample rate. The use of Matlab both for design and implementation leads to less programming bugs and makes it possible for the students to focus on control problems.

Java is a good language for design of graphical user interfaces. Again, the students spend more time on design issues than on low-level implementation and debugging.

To summarize, we now have a complete software environment which enhances the theoretical understanding of practical automatic control, rather than being an obstacle. As an added bonus, the students develop a thorough understanding of Matlab and Java, languages that many of them will continue to use after graduation. They also gain valuable insights into client-server relations.

References

[1] “Process control — course description,” http://www.signal.uu.se/Courses/Descr9798/ procreg.html.

[2] “Process control 1997 — home page,” http://www.signal.uu.se/Courses/CourseDirs/Procreg/Proc97.html.

[3] Mikael Sternad, *Modelling and Control — Lecture Notes for the project oriented course in process control*, UPTEC 94025K, Dept. of Technology, Uppsala university, March 1994.

[4] P. E. Wellstead, *CE8 Coupled Electric Drives*, TecQuipment Ltd., Long Eaton, England, 1981.

[5] Mats Viberg, "Subspace methods in system identification," in *Proceedings of IFAC SYSID'94*, Copenhagen, Denmark, July 1994.

[6] Karl Johan Åström and Björn Wittenmark, *Adaptive Control*, Addison-Wesley, Reading, Mass., 2nd edition, 1995.

[7] Jessica Perry Hekman, *Linux in a Nutshell*, O'Reilly & Associates, February 1997.

[8] Michael Barabanov, "A linux-based real-time operating system," M.S. thesis, New Mexico Institue of Mining and Technology, 1997, Also published at http://luz.cs.nmt.edu/ baraban/thesis/index.html.

[9] David Flanagan, *Java in a Nutshell*, O'Reilly & Associates, May 1996.

[10] Andrew S. Tanenbaum, *Computer Networks*, Prentice Hall, 3rd edition, March 1996.