Real-Time Control System Software for Intelligent System Development: Experiments and an Educational Program*

Veysel Gazi, Mathew Moore, Kevin M. Passino
Department of Electrical Engineering
The Ohio State University
2015 Neil Avenue
Columbus, Ohio 43210

Abstract

The National Institute of Standards and Technology (NIST) has been developing the "Real-Time Control Systems (RCS)" software for more than two decades and using it for the design and implementation of complex intelligent control systems. Present applications of RCS include mining, manufacturing, robotics, and autonomous vehicles. In this paper we summarize the efforts at Ohio State University to develop an educational program for the NIST RCS that includes classroom instruction, a strong laboratory component (complete with small-scale RCS implementations), and a software handbook.

Keywords: Software, real-time control, intelligent control, education.

1. Introduction

With the evolution of control systems and with development of the theories of intelligence, intelligent autonomous control, and hierarchical functional architectures [1, 2], a need for more complicated controllers performing many different tasks in real time has arisen. Systems operating with such controllers have been called Real-Time Control Systems (RCS). The RCS library, developed by the National Institute of Standards and Technology (NIST), is generic software developed for the design and implementation of RCS applications. (In this paper, RCS will refer to the RCS software library.)

We begin this paper with a brief description of the need for such a library. Then we show how the RCS software can be used to implement complex and distributed control systems. We provide an overview of a software manual that is being written for RCS. Next, we describe two experiments that we have developed at OSU to use for teaching RCS applications. These experiments currently operate under RCS and are used in class room instruction. Moreover, we describe an RCS design problem and its solution for an automated highway system; this problem was used as a final project in the class. We provide a syllabus of a course that is taught at OSU on RCS and close the paper with a few brief remarks.

2. RCS Software for Complex Control Systems Development

Several practical issues surround the development of complex real-time control applications. These range from the development of communications between the operator and the plant to more complicated issues of sharing information between multiple, separately operated processes that are interdependent. Furthermore, the number of actuators and sensory processing modules of large complex applications makes it difficult to develop a practical control system. The RCS Library is a generalized controller software and development tool that helps to alleviate the difficulties in setting up both simple and complex control routines. It provides a general architecture for control systems development and the ability to produce hierarchical controllers and a communications base to link independent control systems together.

RCS contains an inherent structure that allows for the decomposition of control systems into several layers, each of which may contain its own sensory processing and actuation components. These layers need not be limited to one computer system. RCS provides a communication management system (CMS) that allows different routines to "talk" to each other. This allows for the modeling and control of an arbitrary number of actuators by linking several applications across multiple backplanes. Since RCS provides a generalized structure for control systems development, it can be applied to essentially any control application (e.g., the reference model architecture described by Albus in [1, 2] and the intelligent autonomous controller architecture described in [2]).

Consider the case of a process control experiment in which chemicals are combined and mixed in a set pat-

*This work was supported by the Intelligent Systems Division of the National Institute of Standards and Technology (NIST) and the Center for Intelligent Transportation Systems at OSU. Please address all correspondence to K. Passino ((614)292-5716, k.passino@osu.edu).

tern. In the simplest case, we control the mixing of two chemicals in which both level and temperature values are crucial. The easiest way to implement this structure is to develop two controllers that are implemented serially (that is, one at a time) in one program. However, this quickly becomes a vast waste of resources as the complexity of the plant that is controlled grows. Say we were to add to the process the mixing of several chemicals, each of which must be heated (or refrigerated) to its own specific temperature, and must be mixed in a particular order. In this case, we must control the level in the storage tank of each chemical, including several mixing tanks and the level inside them, as well as numerous temperature controllers. Furthermore, we control the level via actuating a number of pumps. As one can see, the number of actuators and sensors can grow quite large in even a fairly simple process control experiment such as this. It does not make sense to create one large program to run all control for the entire system in a centralized fashion; time constraints and required sampling rates may not allow this, or physical constraints may dictate the need for distributed control. If, however, the control system is distributed, the running of separate controllers on several computer systems often results in the need for communication between different controllers and higher level coordination of their activities.

With RCS, hierarchical levels of control can be implemented, and this allows the user to spread the controllers across several computer systems, increasing processing power while allowing communication between controllers. Processes can act independently, allowing the user the freedom of choosing separate sampling rates, while at the same time be linked together. Also, RCS allows each layer or process to pass certain information to the other processes acting within the control system, and also allows the user to determine which and how much data is passed between control modules. In most cases, controllers need not share *all* information, only that which is crucial to the operation of the other control modules. As an example, in the process control problem discussed above it is best to spread out the control over several subsystems (and even several computers) and RCS can conveniently provide a method to do this.

The development of a software base to synchronize and link multiple control applications in real-time is expensive and time-consuming. This is compounded by the fact that each control application usually must be developed separately, and previously developed controllers often cannot be reused in different applications. The RCS software base reduces the effort in the development of real-time control systems by providing a portable and reusable software base. The RCS system does not specify implementation details. This means that the RCS structure can be applied to many different types of applications, from simple single-input single-output control systems to a hierarchical structure that controls many complex processes. It can even provide a basis for development of autonomous and intelligent control systems. One can see that RCS offers a wide variety of benefits. By providing only the structure of real-time control systems, a standard interface is available for many different platforms. The user establishes the implementation details. RCS does not limit the possible control applications since it only provides the means for communicating between computers, sensory processing, task decomposition, and developing an operator interface. The user determines the layout and use of these systems such that productivity is maximized. Since RCS is easily ported to different platforms, algorithms written for one application can be reused and incorporated by different controllers.

The keys to the portability and standardized architecture of RCS is the use of CMS and the Neutral Manufacturing Language (NML). CMS is itself a software library that contains several system dependent operating system (OS) calls crucial to establishing communications between separate computers along a network. Several OS are already supported by the RCS software base, including MS-Windows, UNIX, Linux, DOS, and others. RCS provides the ability to pass information between computers running different OS by encoding the information in the neutral manufacturing language format before passing it to another computer. NML provides the software base classes that allow this to happen. RCS controllers communicate with each other via shared memory buffers of user-specified size. That is, sharing of information is obtained by having one process write to the buffer and another process read the information from it. RCS provides the flexibility to establish these buffers anywhere along the network. An NML-server runs in conjunction with the buffer that decodes the NML-formatted information into the native format (e.g., a format that can be used by the other processes). The NML base classes set up what information in a process is written to the buffer on a write cycle, and also provide the means of determining the type of message (e.g., what process the data came from) obtained from a buffer read. The RCS user will program mainly using the NML classes and will not deal with the CMS, though it is important to remember that CMS provides the building blocks for setting up the RCS communications and NML is simply a higher level interface to CMS.

The software manual described in the next section explains how to use the RCS software library to develop both simple control applications and design more complex hierarchical control structures. Actually, the development of a hierarchical system stems directly from the development of several simple control processes linked together using NML buffers. One of the greatest strengths of the RCS control system development approach is that it provides a standardized structure that is simple and portable to different applications. Because of this, the uses of RCS are wide spread.

3. Overview of the Software Handbook

The software handbook is a document that serves as a user's manual for the RCS application programmer. It describes the main components of the RCS library and provides real physical examples of its implementation. For users to get the full understanding of the RCS Library, some knowledge of the C++ programming language as well as a basic understanding of network communications is desirable. For C programmers who have not had much C++ experience, there is an appendix which contains a basic introduction of the C++ language. Furthermore, there is an appendix which provides an introduction to some operating system concepts and network communication protocols. Next, we briefly overview the main contents of this manual.

The second chapter discusses two alternative hierarchical intelligent system structures which can be implemented using RCS. These structures are discussed in [1, 2]. The first architecture is called Reference Model Architecture (RMA) and was developed by NIST and the second architecture is included simply to emphasize that the NIST RCS software has very broad applicability, even to hierarchical intelligent autonomous control systems. The RCS software could be used equally effectively for other control architectures.

The third chapter provides a basic introduction to the terminology used in the RCS library. The main components of the RCS library such as CMS and NML are briefly described. Moreover, a process control experiment, to be used throughout the manual, is stated as a design problem for an NML application. Finally, some guidelines for design of an NML application are presented.

To build an NML application one needs to know basic NML message classes, NML read and write functions and how to run and stop NML servers. These are detailed in the fourth chapter together with some examples. Moreover, this chapter describes some error handling and some command line utilities for NML.

Chapter five provides information on basic process and buffer types that can be defined in an NML configuration file. It describes how a configuration file is written. The user should be familiar with basic operating system concepts so that he or she can choose between the available protocols and define the required type of mutual exclusion for concurrent processes.

Chapter six presents some lower level utilities of the RCS library. These are generic classes or functions which are used throughout the development of the RCS library; however, they can be useful for the application programmer on their own. Such utilities include a timer, semaphore, linked list, and some print and windows functions.

In RCS applications we often need to check the controller status and supervise it remotely. The RCS diagnostics tools can provide this function. The seventh chapter describes a "diagnostic tools" written by the NIST team as a Java applet and can be viewed using any Java compatible Internet browser.

In the final chapter we provide three illustrative examples for programming using RCS library. The first two examples are physical laboratory experiments; the third is a computer simulation process. The laboratory experiments include the level and temperature control of a liquid in surge tank, introduced as a design problem in the second chapter, and the balancing of an inverted pendulum. The computer simulation is an RCS-centered automated highway system (AHS) platoon consisting of (at least) three vehicles. The chapter describes the definition of message classes, writing configuration and architecture files, coding format functions, creating and running NML servers and programming module components of these experiments.

If you would like to get more information on RCS see [3].

4. The RCS Laboratory Experiments

At this time we have two experiments operating under RCS and have included the full details of the implementation in the RCS software handbook. Moreover, there is a RCS design problem for an automated highway system application that we will simulate using computers.

4.1. Rotational Inverted Pendulum Experiment

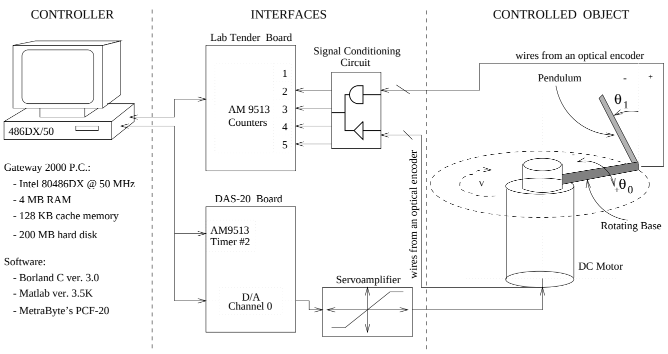

Figure 1: Experimental set up of the pendulum.

The swing-up and balancing of an inverted pendulum is a typical non-linear academic problem that has been used for testing a multitude of controllers. It is an ideal experiment for an application example for the use of RCS since the control problem is well understood and hence we can just focus on RCS component. The plant is shown in Figure 1 and consists of a pendulum, a rotational base, sensors and an actuator. The actuator is simply a motor that drives the rotational base. The pendulum is connected to the optical encoder tied to the base in such a way to allow the pendulum to swing freely. By rotating the base, the pendulum swings in response. The controller must rotate the base in such a way that the pendulum swings up and balances vertically at its unstable equilibrium point.

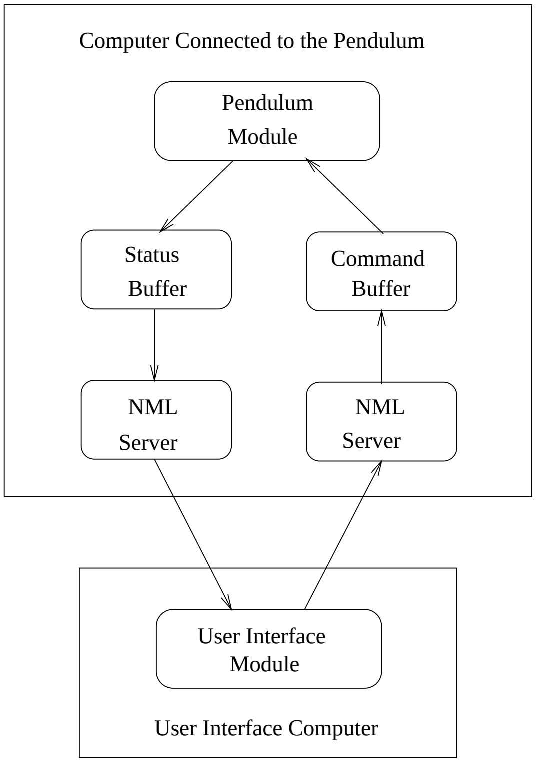

Figure 2: Shared buffers and NML servers for the pendulum.

This system is simpler than the process control experiment treated below as far as the use of RCS and that is why we treat it first. While we use only one running module this application is still important since it serves to introduce basic RCS ideas and it shows that RCS can be used for plants with relatively small sampling intervals. Furthermore, we show the versatility of several of the lower library functions and how they are used.

This experiment can be performed both under DOS and Linux environments on the same PC, and a Windows NT workstation is used for diagnostics purposes. We need two NML shared memory buffers for reading the status of the control module and for sending commands to it via the diagnostics tool. The NML shared memory buffers are located on the NT station while running under DOS and can be located both on NT station and Linux station while running under Linux OS. The simple reason for this is that one needs to run NML servers, which read from and write to the shared memory buffer on behalf of the remote processes, for every buffers which will be accessed remotely. Since DOS does not allow multitasking one cannot run the servers on the DOS PC because there is already a controller module running on it. Figure 2 shows a possible implementation of the buffers and modules while running the experiment under Linux. Note that, although the computer we use are in the same lab, the host called "User Interface Computer" can be located on any host on the internet. We will not discuss in detail the control techniques used, as emphasis is on using RCS for the experiment. All data acquisition is obtained using a Keithley DAS20 Data Acquisition Card. The functions accessing the DAS20 are contained within the DOS executable file.

4.2. Process Control Experiment

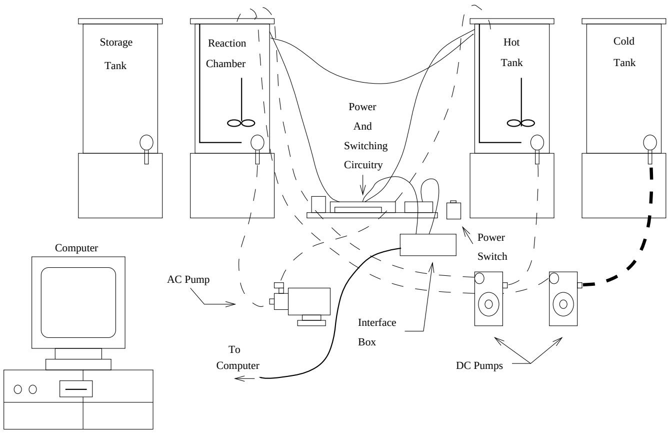

Figure 3: Setup of the process control experiment.

The experimental setup of the process control experiment is shown in Figure 3. Similar to the pendulum experiment this one can also be run both under DOS and Linux OS. On another PC running under Windows NT we run the Diagnostics tools so that we can view the status of the process and send commands to the modules in controller. In other words the DOS (Linux) PC is the computer on which the control modules run and the Windows NT PC is the user interface computer. As in the pendulum we locate the shared memory buffers on the NT station for DOS and on the Linux station for Linux application. They can be located on the NT station also but we recommend to put them on the computer the controllers are running on since this makes writing to them easier and therefore increases the speed of the process. In this application there is a need for temperature and level control so there will be a module for each and one supervising module. Figure 4 illustrates how one can locate the status and command buffers for this experiment. Note that the supervisor module can access the buffers of the level and temperature modules. This allows it to check the status of the modules and to send commands to them. As in the pendulum experiment, the user can access all the buffers from a remote host. Low level controls are simple PID loops. The RCS code illustrates how multiple modules can be defined to interact on different machines.

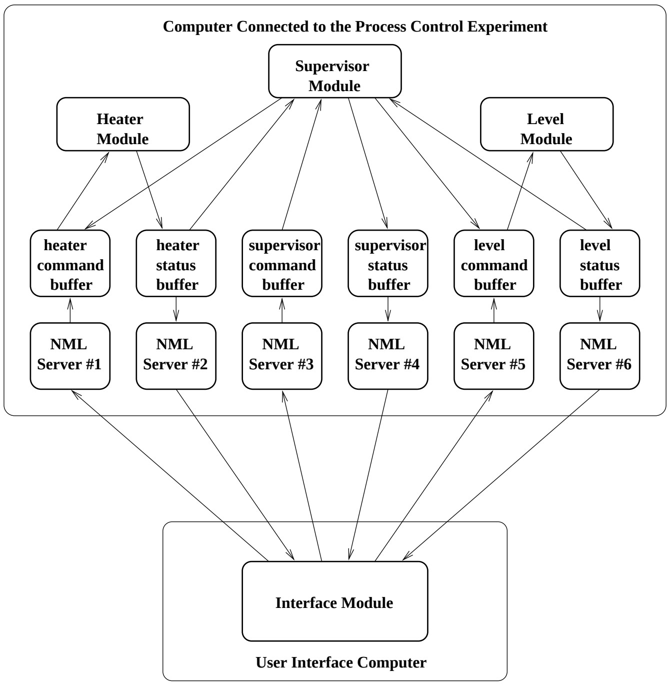

Figure 4: Shared buffers and NML servers for the tanks.

4.3. The Automated Highway Example

The AHS problem is the most involved RCS design project of the laboratory course, and because of this, it serves as the students' final project. This ensures the students have had plenty of exposure to RCS coding from the previous experiments before attempting to solve the AHS problem.

The AHS setup demands that the students design controllers for longitudinal and lateral control for a vehicle with simple dynamics. Furthermore, the student must develop the higher level decision-making process based on state table analysis of the situation a platoon of three vehicles may be introduced to (e.g., when a car should change lanes, speed up, etc.). The basic requirements insist that the students simulate each of the three vehicles on three separate computers (simulating the vehicle independence that occurs in a real AHS). The overall AHS system will be operating on a two lane (one-way) road with specified lane width. An additional requirement makes it possible for each vehicle to set its own desired speed. Other than using RCS modules for the control algorithms and simulating vehicle dynamics and RCS messages for any inter-vehicular communication, the actual details of the design are left to the students. The student determines what information is communicated and how.

A simple solution to this problem is briefly presented to the students to act as a starting point. In the solution, two RCS modules are created—one to act as the supervisor, and the other is a basic template for the vehicle. Dynamic simulation and the low-level controller code is included in the vehicle module. The higher level decision-making is done by the supervisor, which acts as a parent to each of the vehicles. The supervisor reads the current speeds and positions of each of the subordinate vehicles via RCS messages, determines the current situation to handle and sends commands back to the vehicles with the goal of providing efficient and safe highway travel. Possible commands could be for the vehicle to slow down/speed up or to switch lanes. The advantages to this design include the modularity of the code, as each vehicle code will be the same except for a few parameter changes, and the hierarchical design which is promoted by the RCS architecture.

5. The RCS Course

In this section we provide an overview of the syllabus that is used for a course on the NIST RCS in the Department of Electrical Engineering at The Ohio State University in the Spring Quarter, 1998.

The course (EE 758) is one of two laboratory courses we offer at the graduate level in the controls area (the other, EE 757, is on digital control and focuses on low level implementation details and controller development for a variety of conventional control strategies). At other times EE 758 has been run as a lab course in conventional control and more recently it has been associated with our course in intelligent control (where fuzzy/neural/genetic adaptive estimation and control are taught). EE 757 is not a prerequisite for EE 758, but students who have already had EE 757 tend to have a deeper understanding of the low level operation of the controllers. Prerequisites for the current EE 758 include a knowledge of how to program in C (or C++) and an understanding of a few control system design methodologies (e.g., PID and state feedback control design), although a student who has had many graduate level courses in control may have a slight advantage. The course has lecture sessions as well as lab sessions. In the lecture sessions the theory and use of RCS is detailed while in the lab sessions students develop controllers and implement them using RCS.

The stated objectives of the laboratory are: To develop and implement distributed real-time control systems using the NIST RCS and to develop control modules using a variety of conventional and intelligent control methods. OSU is on the quarter system which has ten weeks. The class and laboratory topics for these ten weeks are:

1. Class: RCS Introduction (CMS/NML Overview); Lab: Laboratory software (C, C++ programming, OS, network)

2. Class: Programming in the neutral manufacturing language (NML); Lab: Laboratory hardware (data acquisition card)

3. Class: Writing NML configuration files, RCS diagnostics; Lab: Pendulum experiment (controller development)

4. Class: Other classes and functions in the RCS library; Lab: Pendulum experiment (RCS implementation)

5. Class: RCS applications (overview and simulation); Lab: Tank experiment (level, temperature controller development)

6. Class: RCS development project overview; Lab: Tank experiment (RCS implementation)

7. Class: Distributed control problem (background, real-world details); Lab: Project: low level control design (distributed control experiment)

8. Class: RCS Design review; Lab: RCS design/implementation

9. Class: RCS design review; Lab: RCS implementation and testing

10. Class: RCS design report; Lab: RCS demonstration

6. Concluding Remarks

In this paper we have overviewed some experiments we developed that use the NIST RCS software package and have described an educational program we implemented at OSU for teaching RCS. This involves class room instruction and laboratory experimentation. At the present time we are revising the educational materials based on our experience in teaching the material and the materials will be made available to the public by Sept. 1998.

Acknowledgements: The authors would like to thank Fred Proctor, Will Shackleford, and James Albus of NIST for their assistance in every aspect of this project. It must emphasized that the RCS software was developed over many years at NIST by these and other members of NIST; here, we simply reported on the development of an educational program for RCS at OSU.

References

[1] J. S. Albus, *Outline for a Theory of Intelligence*, IEEE Trans. on Systems, Man, and Cybernetics, Vol. 21, No. 3 May/June 1991

[2] P. J. Antsaklis and K. M. Passino (eds.), *An Introduction to Intelligent and Autonomous Control*, Kluwer Academic Press, MA 1993

[3] Will Shackleford, *Real-Time Control Systems Library: Software and Documentation*, Internet Location, http://isd.cme.nist.gov/proj/rcs_lib/