The Real Time Controls Laboratory, an Open Source, Hard Real Time, Controls Implementation Platform

Edgar F. Hilton (efhilton@fsmlabs.com), Victor Yodaiken
FSM Labs, Inc., Socorro, NM, USA

Marty A. Humphrey, Paul E. Allaire
University of Virginia, Charlottesville, VA, USA

Abstract

Modern automatic control systems need 1) logically complex and computationally expensive controller algorithms and I/O; 2) real time data storage of plant information such as states, inputs, and outputs; 3) real time plotting capabilities; 4) real time controller parameter updates (both scalar parameters and matrices); and 5) real time access to reference signals; 6) remote monitoring for safety purposes. Control systems based upon embedded Digital Signal Processors (DSP) boards often require specialized programming and development tools, may lack flexibility when computational or timing requirements change, and may not directly address the aforementioned needs. Even worse, newer and faster DSPs may not be fully pin compatible with their predecessors, thus requiring total redesign of the embedded electronics for a given project. A novel controller implementation system – the Real Time Controls Laboratory (RTiC-Lab) – has been developed explicitly to address these problems. It relies on both commodity personal computers and Real Time Linux to guarantee satisfying hard real time constraints in light of maintaining soft real time requirements (such as plant monitoring and parameter updates). RTiC-Lab is Open Source and is thus intended to serve as a communal effort among both controls engineers and the Real Time community. Discussion is presented on the design of RTiC-Lab's software architecture, the controller API, and the IPC-API which allows other software packages to communicate with RTiC-Lab. An example application is presented.

1 Introduction

Modern automatic control applications (ACA) require the following:

1. *temporal and logical correctness*: a control law must operate on time dependent data, and provide time dependent control signals. Failure to produce a logically correct output at the proper time could result in catastrophe [7, 6, 10].

2. *logically complex and computationally expensive controller algorithms and I/O*: as CPU speeds increase, controller algorithms will match the CPU capabilities [9].

3. *real time data storage of plant information such as states, inputs, and outputs*: this data should be easily stored for post analysis and filtering by any mathematical and visualization packages such as Matlab, Scilab, and TecPlot.

4. *real time plotting and filtering capabilities*: the data should be easily sent in soft real time to virtually any software package that can be used for visualization and run-time filtering and which may, via some onboard logic, choose to update any controller parameters or signal reference information – for example an adaptive algorithm may first adapt to the controlled plant's parameters and then it updates the controller gains for performance.

5. *real time controller parameter updates (both scalar parameters and matrices)*: controller algorithms take many shapes, forms, and sizes. And, as a rule of thumb, the controller algorithm will *never* be correct on its first implementation. Consequently, the controller requires that the architecture allow for an easy means to update parameters both during stoppage time and in soft real time.

6. *real time access to reference signals*: for example, an autonomous underwater robot may be asked to carry out the remainder of its mission from a new depth.

7. *remote monitoring*: this is especially true for dangerous applications – such as active magnetic bearing applications with high speed rotors –, or environments in which it is inconvenient, dangerous, or cost ineffective to have a human being present.

This is the case with active magnetic bearing (AMB) control systems such as AMB supported artificial hearts [5, 2, 1] and high speed energy storage flywheels for powering communication satellites [4, 8].

Traditionally, control systems have relied on embedded Digital Signal Processors (DSP). However, controls engineers have often had to learn a new API and programming

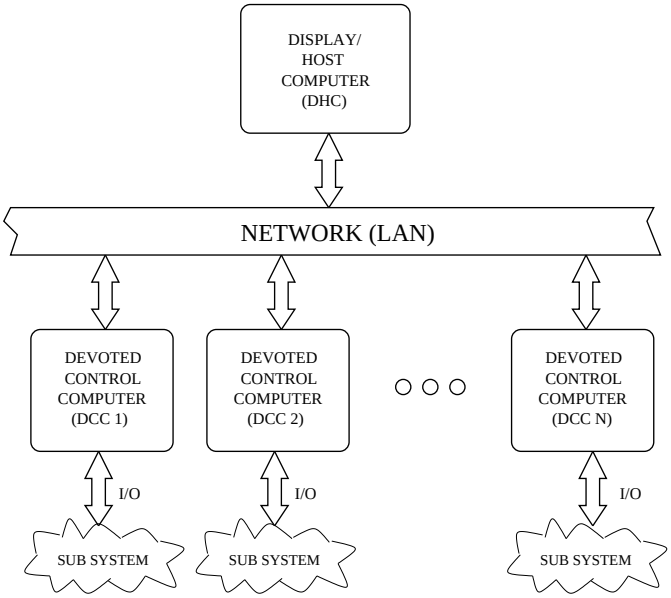


Figure 1: Overview of the Real Time Controls Laboratory network environment

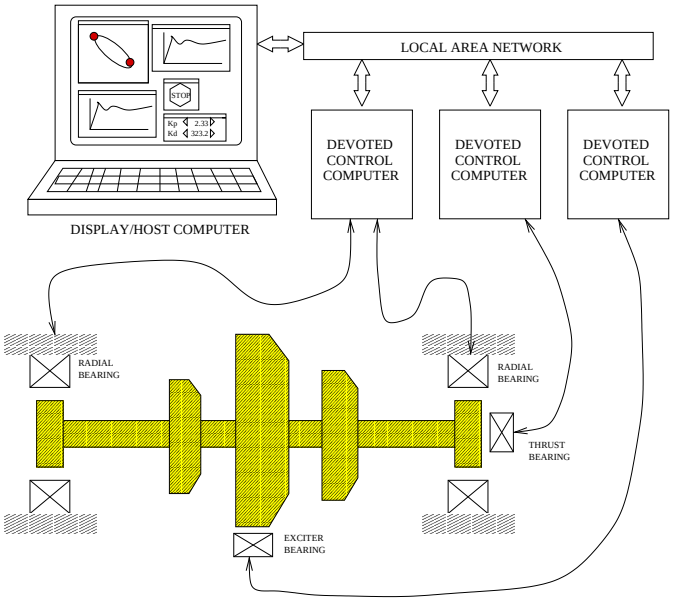


Figure 2: Example of the applicability of RTiC-Lab on an AMB system.

language for each new target DSP. DSPs require specialized programming and development tools, may lack flexibility when computational or timing requirements change, and may not directly address the aforementioned needs. The development time for faster DSP systems is much slower than that of commodity PCs. Even worse, newer and faster DSPs may not be fully pin compatible with their predecessors, thus requiring total redesign of the embedded electronics for a given project.

A novel controller implementation system – the Real Time Controls Laboratory (RTiC-Lab) – has been developed explicitly to address the aforementioned controller requirements. It relies on both commodity personal computers and Finite State Machine Lab's (FSMLabs) Real Time Linux (RTLinux) to guarantee satisfying hard real time constraints in light of maintaining soft real time requirements (such as plant monitoring and parameter updates). RTiC-Lab is and will be – as its underlying Linux and RTLinux platforms – Open Source Software released and protected under the Free Software Foundation's General Public License. That is, users of RTiC-Lab can download the source code, use it, enhance it, and share it with their colleagues.

The following sections are set up as follows. First, RTiC-Lab is presented. Especially, discussion focuses on the design of RTiC-Lab's software architecture, the controller API, and the IPC-API which allows other software packages to communicate with RTiC-Lab. Finally, implementation results are shown which demonstrate the predictability of RTiC-Lab.

2 The Real Time Controls Laboratory, (RTiC-Lab)

Control of ACAs require an exhaustive tuning and characterization process during the early stages of the ACA's life. RTiC-Lab, is explicitly designed to be used not only during these early stages of controller design and plant characterization, but also during subsequent monitoring and control. It provides an environment in which to implement controller algorithms while providing real time access to controller states, plant outputs, controller actions, controller parameters, and other controller information. All this information can be plotted and filtered – via user defined filters – in soft real time. The user can further filter the necessary data either in soft real time or *post mortem*. Last and most importantly, the controller parameters can be updated in real time through a user-defined graphical user interface.

The general scheme used in the design of RTiC-Lab is shown is Figure 1. A devoted display or host computer (DHC) is networked via 10 or 100 Mb/s TCP/IP network to a set of devoted controls computers (DCCs).

The controls engineer sits at the DHC (which may or may not be at the same room or even building as the DCCs) and coordinates, codes, and synchronizes all DCCs from the DHC. Run time parameters, such as sampling rate, startup delay, and networking parameters can be set for each of the DCCs from the DHC.

Each of the DCCs is a stripped down computer system having no keyboard, mouse, video card, or monitor. These only have both the necessary I/O cards which are used to interface to the plant hardware and the necessary Ethernet card to communicate with the DHC.

An AMB example of RTiC-Lab is shown in Figure 2. A single DHC interfaces with three DCCs which in turn interface to the AMB rotor system. The first DCC handles all radial control of the AMB, while a second (and slower) DCC controls the thrust direction of the AMB, and a third DCC is used to add either some excitation or synchronous forces to cancel out rotor imbalances at the midspan. Both controller parameters can be updated through the graphical user interface, and all data is plotted in soft real time

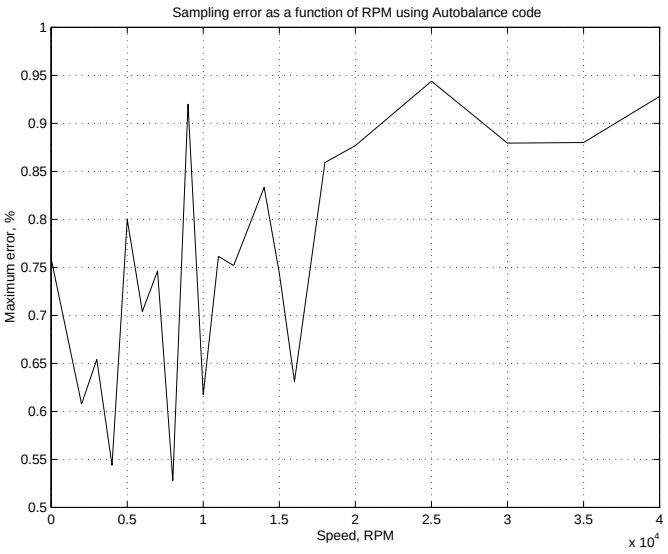


Figure 3: Maximum sample period error for the suspension controller as a function of spin speed

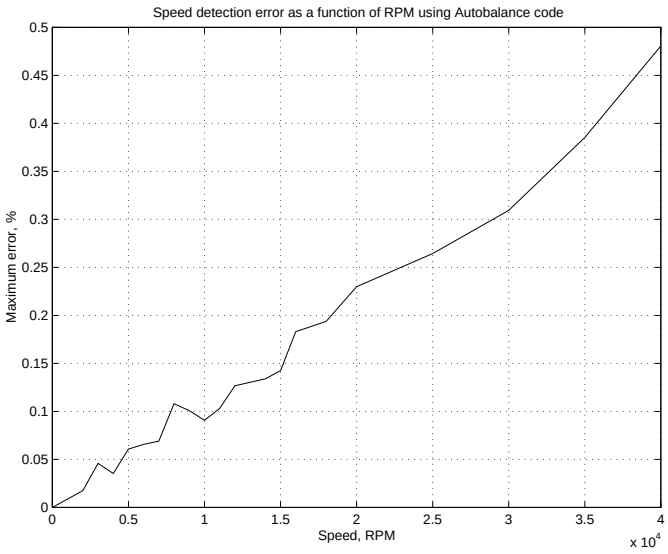


Figure 4: Maximum RPM error of the sampled speed as a function of spin speed

at the DHC.

In the event that the controlled plant is both computationally simple and safe enough to be handled exclusively in a single computer, then RTiC-Lab will collapse into one single computer to control the entire plant. Stated differently, the same computer both implements the controller in hard real time and saves, plots, and updates parameters in soft real time.

As of the time of this writing (version 0.6.4), RTiC-Lab's networking capabilities are handled by exporting windows in Linux. However, true distributed functionality is currently in the works. Networking is thus not mentioned further in this paper.

RTiC-Lab uses RTLinux's fixed priority scheduler, and assigns task priorities according to Liu and Layland's RMA scheduling algorithm. Data is transmitted from the hard real time or "embedded" tasks to the soft real time or "reactive" tasks (henceforth referred to as XRTiC) via real time FIFOs (RTFIFOs). Both scalar and matrix parameters are transferred between XRTiC and the embedded tasks via RTFIFOs and shared memory.

In accordance with the RTLinux paradigm [3], RTiC-Lab separates the ACA's controller into the embedded part and the reactive part. The embedded part of the controller includes all tasks having hard timing constraints. The reactive task, or XRTiC, is a multi-threaded, user-space application running in Linux. Users can use XRTiC's IPC API to create their own stand-alone Linux applications which accesses XRTiC to not only obtain both embedded data and status information, but also to update controller parameters. RTiC-Scope, the RTiC-Lab Oscilloscope emulator, is one such application which uses RTiC-Lab's IPC API.

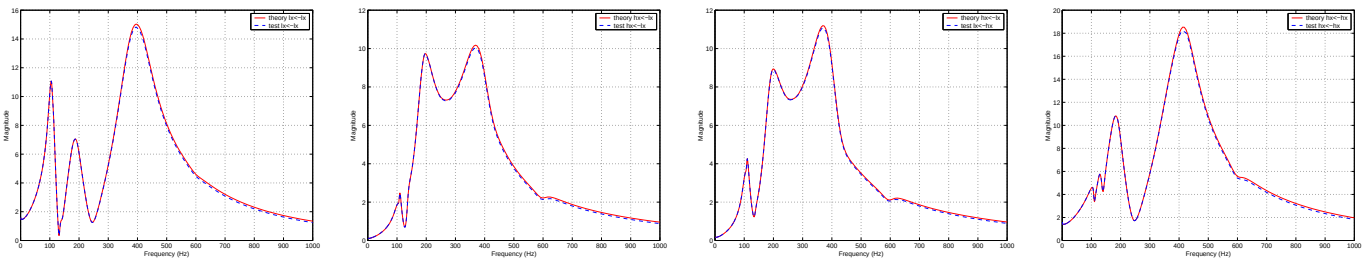
For example, for an AMB application, the embedded part handles: 1) the AMB suspension controller(s) (both periodic and event driven), 2) a software watchdog, and 3) a set of interrupt service routines that are used for communication with XRTiC. XRTiC would perform the following functions: 1) communicate with the embedded tasks via RTFIFOs, 2) display a graphical user interface for the user through which the user can start, stop, and update parameters, 3) perform error checking of the user's controller code, 4) send parameter updates to the embedded tasks as requested by user, 5) either save to data to a file or print it to `stdout`, and 6) send data to, and receive parameters and commands from an external user-supplied

3 Implementation Results

Of most importance to the implementation of an ACA is the predictability of RTiC-Lab. Figure 3 shows the sampling error obtained on a Pentium III 600 MHz computer running the following algorithms: 1) a full μ-synthesis, 44th order, four input, four output, state space, AMB radial magnetic bearing controller (8 kHz), 2) a strictly proper PID axial AMB controller (8 kHz), 3) an AMB auto-balance algorithm (8 kHz), 4) a key phasor monitoring task which monitors the rotor spin speed and from which (combined with time information) we can extract the actual rotational angle of the rotor (once per rotor revolution, interrupt driven, parallel port). Note that items 1, 2, and 3, above, all run in one task, while item 4 runs as a separate task.

First, we measure RTLinux's ability to both schedule and handle our main periodic tasks irrespective of bus load as would be expected in a high speed AMB rotor. As can be seen from the plot, the periodic tasks' sampling period error is bounded below an outstanding 1% even at simulated rotor spin speeds approaching 40,000 RPM (667 Hz).

The next point of interest is determining the maximum error of the sampled rotor spin speed itself. Errors here would measure any latencies in the priority based scheduler in RTLinux. Figure 4 shows the error of the sampled simulated spin speed for the same controller as above. As



(a) Low X to Low X (b) Low X to High X (c) High X to Low X (d) High X to High X

Figure 5: Computer's I/O characteristics. Theoretical and measured amplitudes versus frequency

can be seen, the maximum error is also below and impressive 0.5% at simulated speeds approaching 40,000 RPM.

Last and no less important, both RTiC-Lab and RTLinux, together, must satisfy both the logical and temporal correctness of the aforementioned controller as seen by the controlled plant. Figures 5 show the theoretical and the measured input/output characteristics of the *mu*-synthesis controller of the low x to low x, low x to high x, high x to low x, and high x to high x input to output ports of the controller, respectively. The measured response denotes the results obtained from a sine sweep as measured at the I/O ports of the computer by use of a Stanford Research Systems two channel dynamic signal analyzer (Model SR785). As can be seen from the plots, both the theoretical and measured values match perfectly.

4 Conclusion

The Real Time Controls Laboratory, or RTiC-Lab, was developed to aid in the ACA implementation process. It relies on FSM Lab's RTLinux for meeting its strict time lines. And, consistent with the underlying operating system, RTiC-Lab is Open Source. Most importantly, tests have shown that the predictability of the RTLinux/RTiC-Lab combo are quite remarkable and thus both systems are capable of safely controlling high performance systems.

Controller developers and real time programmers who are interested in using and/or enhancing this software are encouraged to download the software from `http://www.-rtic-lab.sourceforge.net`. Please submit all modifications and enhancements to `efhilton@fsmlabs.com`. Special forums for discussion may be accessed also from this site.

5 Acknowledgements

Work for RTiC-Lab has been made possible due to generous contributions and donations by the National Aeronautics and Space Administration (NASA); American Flywheel Systems, Inc.; the Rotating Machinery and Controls Laboratory at UVA; and the Finite State Machine Labs, Inc. (FSM Labs).

References

[1] P. E. Allaire, H. C. Kim, E. H. Maslen, G. B. Bearnson, and D. B. Olsen. Design of a magnetic bearing supported prototype centrifugal artificial heart pump. In *Tribology Transactions*, volume 39-3, pages 663–669, July 1996.

[2] M. Baloh, P. Allaire, E. Hilton, N. Wei, E. Maslen, D. Baun, and R. Flack. Magnetic bearing system for continous flow ventricular assist device. *J. of the ASAIO*, 45(5), 1999.

[3] M. Barbanov and V. Yodaiken. Introducing Real-Time Linux. *Linux Journal*, 34:19–23, February 1997.

[4] R. Bartlett, J. Coyner, S. Djouadi, P. Allaire, E. Hilton, J. Luo, P. Tsiotras, F. Maher, and R. Strunce. A simulation of spacecraft energy momentum wheels using advanced magntetic bearing controllers. In *1999 Invitational NASA/Air Force Flywheel Workshop*, NASA Glenn Research Center, 1999. Invited Paper.

[5] E. Hilton, P. Allaire, M. Baloh, N. Wei, G. Bearnson, D. Olsen, and P. Khanwilkar. Test controller design, implementation, and performance for a magnetic suspension continous flow ventricular assist device. *Artificial Organs*, 23(8):785–791, 1999.

[6] P.A. Laplante. *Real-Time Systems Design and Analysis: An Engineer's Handbook*. IEEE Press, IEEE Computer Society, New York, second edition, 1997.

[7] N. Nissanke. *Realtime Systems*. Prentice Hall, London, 1997.

[8] U. Schönhoff, J. Luo, G. Li, E. Hilton, R. Nordmann, and P. Allaire. Implementation results of μ-synthesis control for an energy storage flywheel test rig. In *7th International Symposium on Magnetic Bearings*, August 23-25 2000.

[9] J. Stankovic. Misconceptions about real-time computing. *IEEE Computer*, 21(10), October 1988.

[10] J. Stankovic and G. Buttazzo. Implications of classical scheduling results for real-time systems. *IEEE Computer*, 28(6), June 1995.