REAL TIME DIGITAL CONTROL OF AN ACTIVE MAGNETIC BEARING USING OPEN SOURCE SOFTWARE

H. E. Alpaugh, F. C. Nelson, D. W. Fermental
Tufts University, Medford, Ma. 02150 frederick.nelson@tufts.edu

***Abstract* - A digital control system using a real time variant of Linux is developed and implemented on a bench-top rotor suspended on active magnetic bearings. The combination of simple hardware and open source software provides a useful educational and research platform for digital control methods that is unencumbered by proprietary constraints as well as one that is more cost effective than available commercial systems. A real time operating system must ensure that specific tasks are executed at a fixed rate regardless of other system level demands. This is achieved with special purpose kernels that allow Linux to run as a low priority task within the real time operating system. Digital filters are constructed and coded in C, the native language of Linux. The controller is implemented on an off-the-shelf Pentium 3 personal computer operating in a multiple channel configuration at 10 KHz.**

***Index Terms* – Active Magnetic Bearing; Digital Control; RTLinux; Real Time Programming; Rotordynamics**

I. BACKGROUND

The use of open source software, in particular Linux, as an environment for dynamic control law development and research has several advantages, especially in an educational setting in which students may not be required to learn various proprietary software tools. This paper presents work at Tufts University, which uses open source Linux to control an active magnetic bearing (AMB) and to access the data stream from the controller. It is an expansion of earlier work published in the Linux Journal [1].

There are several advantages for magnetic bearings that justify their incorporation into an educational setting; they eliminate physical contact between the shaft and the support, minimize friction and eliminate wear inherent with conventional bearings. A recent application under development in Japan is an implantable heart pump [2, 3]. Gas turbines are another major area under consideration [4] .

II. TEST SETUP

A photo of the experimental test rig is shown in Fig. 1. A physical schematic showing the relationship between the various functional parts is shown in Fig. 2. Each AMB consists of four pole pairs of laminated stator assemblies with individual windings on each of the poles. The bearing assembly includes inductive gap sensors at the centerline of each pair of diametrically opposed poles. These bearings support a 0.6 m. long x 9.5 mm. diameter stainless steel shaft driven by a brushless DC motor.

A digital controller uses the signal from the gap sensors to adjust the current from the power amplifiers driving the magnetic coils to keep the rotating shaft centered in the gap.

The digital controller is implemented on an Intel based Pentium3 personal computer (PC) with a multi-channel data acquisition board and a multi-channel analog output board. The RedHat® version 7.2 distribution of Linux was selected as the operating system (OS). The PC is not connected to a network. The digital controller, described in this paper, replaces the original analog controller. In 2002 the PC and boards cost on the order of $5000.

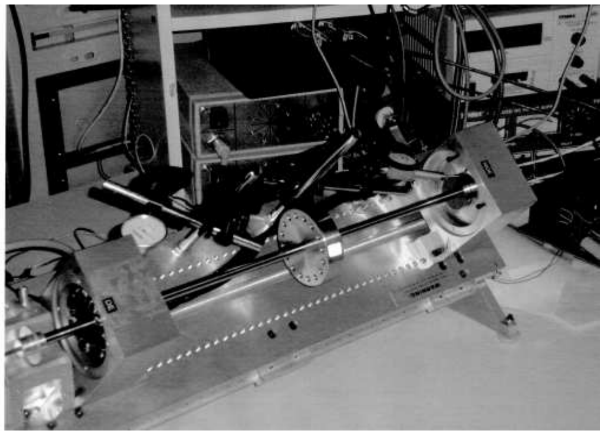

Fig. 1 – Photograph of Rig at Tufts University.

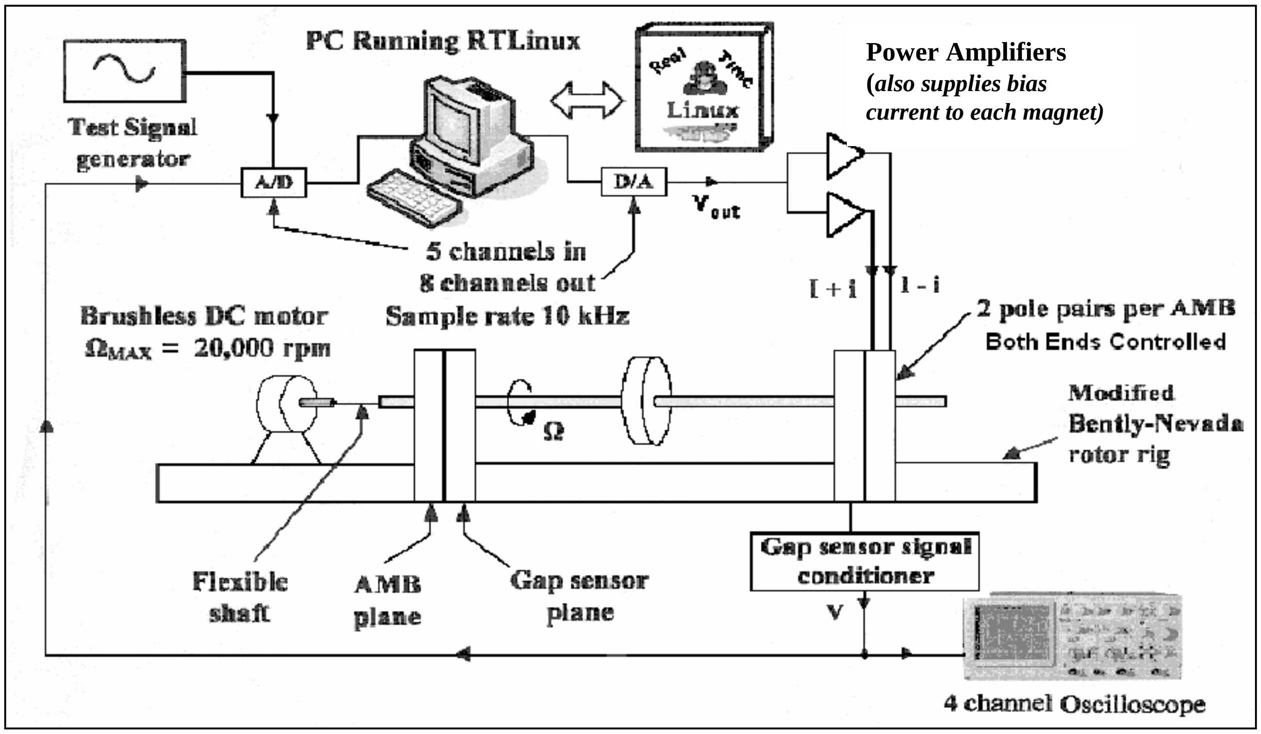

Fig. 2 – Physical Schematic of Rig at Tufts University showing functional connections.

III. Real time architecture

The heart of the Linux system is the kernel, which provides the features and functions common to all contemporary computer operating systems such as multitasking, memory management and allocation, file system supervision, I/O drivers and networking. To accomplish this, the Linux OS controls the CPU to optimize system resource use.

In contrast, a real time OS must ensure that a specific task executes at a fixed rate regardless of the many system level demands that burden the OS. To meet this requirement, two organizations, FSMLabs [5] and RTAI [6], have developed special purpose kernels that run Linux as a low priority task within a real time OS. This substantially reduces the timing from the hundreds of milliseconds of desktop systems to the microsecond range and allows the user to precisely control the timing of critical control processes.

RTLinux®, developed by Michael Barabanov and Victor Yodaiken in 1996, is currently marketed by FSMLabs, a private company located in New Mexico. They provide two versions: RTLinux/Free and RTLinux/Pro. RTLinux/Free is used for this work. FSMLabs holds a software patent on RTLinux, but the patent license allows it to be used in projects under the GNU General Public License (GPL) [7].

Conceptually, RTLinux splits the OS into user-space and a real-time kernel. One may think of this as two separate cities, walled off from each other and able to communicate only by guarded pathways, such as real time first-in-first-out devices (RT-FIFO's). User space is the familiar Linux system with all its friendly utilities such as the vi editor, the gcc compiler and the exit command. The real time kernel is the Spartan-like environment which executes the real time task regardless of the activities in user space.

Real time programs are coded as modules, not as the *main{}* program construct of most C programs. A real time program requires two functions: *init_module* called when starting and *cleanup_module* when turning off the real time module. Listing 1 shows the basic construct for coding the real time module. The *init_module* creates the entry point for the real time module and allocates the RT-FIFO's used to communicate with user space. The real time code starts at *Periodic_function_entry* and uses a *while(1)* code construct to repeatedly run the control loop.

The real time module is started with the i*nsmod* command. Much like the proverbial sorcerer's apprentice, once the real time module starts it can only be stopped by the *rmmod* command or by literally pulling the plug on the PC. It is quite unnerving to a new user of RTLinux to discover that despite issuing the exit command the controller continues to run.

IV. Some control theory

First, the quantity to be controlled is instrumented and measured, here it is the gap between the rotating bearing and the stationary magnetic poles of the bearing. This gap is converted to a voltage with signal conditioners and input to an analog/digital (AI) board. In the test setup four separate gap sensor signals control the rotating shaft. All four gap signal voltages are sampled simultaneously and sequentially downloaded to the PCI bus.

The gap is controlled by the current through the magnets which are driven by eight power amplifiers. The power amplifiers are controlled by the voltage from a separate digital/analog output board. The analog output (AO) board receives a digital input from the PCI bus and converts it to a voltage which is held constant until the next time period.

In the control loop the AO board receives the processed signals from the AI board after numerical processing. In an ideal digital controller both AI and AO operations occur simultaneously at precise constant intervals. Although it is impossible to achieve this ideal, one must ensure that the code within the control algorithm runs efficiently.

The numerical operations within the control program include the history of the input, x, and the output, y, of the controller for several previous steps. These are stored in memory and shifted one increment each time the control loop executes. The history is incorporated in a difference equation:

$$y(n)=A*y(n-1)+B*y(n-2)+\dots+C*x(n) + D*x(n-1) + \dots \tag{1}$$

where y(n) is the output of the controller for the current time step, y(n-1) is the output of the controller in the previous time step, y(n-2) is the output two steps in the past, y(n-3) three steps in the past, and so forth to the depth demanded by the sophistication of the control algorithm. Similarly, x(n) is the input voltage for the current time increment and x(n-1) is the input for the previous increment. The particular control law implementation determines the constant coefficients A, B, C, D, etc. The bilinear transformation is used to approximate the continuous control law (s-domain) in the discrete time domain:

$$\frac{1}{s} = \frac{T}{2}\frac{(z+1)}{(z-1)} \tag{2}$$

where T is the sample time and z is the discrete time operator.

In the magnetic bearing test setup y is the voltage driving the power amplifier and x is the signal from the gap sensor. Three previous values are used in the magnetic bearing difference equation.

V. Digital control implementation

There are three possible approaches for selection of the digital acquisition and control (DAC) boards:

1. Write the required board driver software.
2. Obtain a driver from an open source [8].
3. Use vendor supplied driver software.

The first and second options require a high level of sophistication and expertise with Linux and data acquisition programming. The second option reflects the open source nature of the Linux system, but the selection of vendors is limited and the latest products are often unavailable. The third option, although requiring the least expertise, places the user at the mercy of the board vendor. The vendors supplying and supporting the necessary drivers are limited and quite often use the same sources as the second option. The third option was chosen and two PCI bus multi-channel DAC boards were purchased from United Electronics Inc. [9]. These were supplied with the required real-time Linux software drivers.

Before the digital controller was implemented on the rotor rig tests were run to characterize the digital system behavior. These tests were various program codes to evaluate the conversion and timing interactions of the boards.

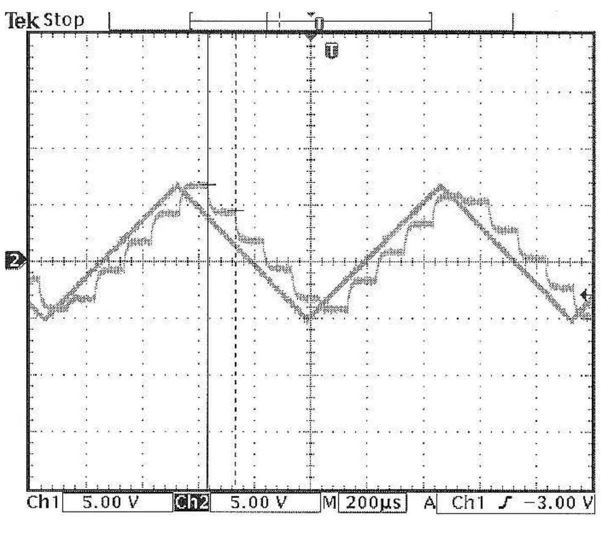

Fig. 3 – Analog to digital to analog conversion test showing typical zero order hold behavior.

LISTING 1 – SKELETON CODE OF REAL TIME MODULE.

```
#define PERIODIC_FREQ_HZ 10000.0
#define FRAME_PERIOD_NS ( … );
pthread_t periodic_thread;
void *Periodic_function_entry(…)
{
  pthread_make_periodic_np( … );
//Initialize various parameters.
while (1)
    {
    // real time code loop here
        pthread_wait_np();
    }
}
int init_module(void)
{
    // create the thread
    pthread_create( … );
    pthread_wakeup_np( … );
}
void cleanup_module(void)
{
   read_delete_np( … );
   exit(0);
}
```

The primary functional test consisted of a C language module designed and coded to read analog data on the analog input board, convert it to floating point variables, convert it back to a digital variable, then output the signal via the analog output board. Fig. 3 shows a typical record of the performance from this test. The main loop in the software is set at 10 kHz in this plot; the analog input is a 1000 Hz sawtooth. The output shows the step waveform characteristic of sample-and-hold operation.

At the heart of the real time control program, shown in abbreviated form in Listing 1, is the RTLinux function, *pthread_wait_np* which suspends execution of the currently running realtime thread until the start of the next period. This thread is marked for execution with *pthread_make_periodic_np*. The thread gives up control until the next time period. The default arithmetic in RTLinux is integer, but this control application uses floating point which is turned on by *pthread-setfp_np* .

The original analog controller is a simple lead compensator implemented with a single order pole-zero pair on each of the four bearings which is duplicated with the difference equation:

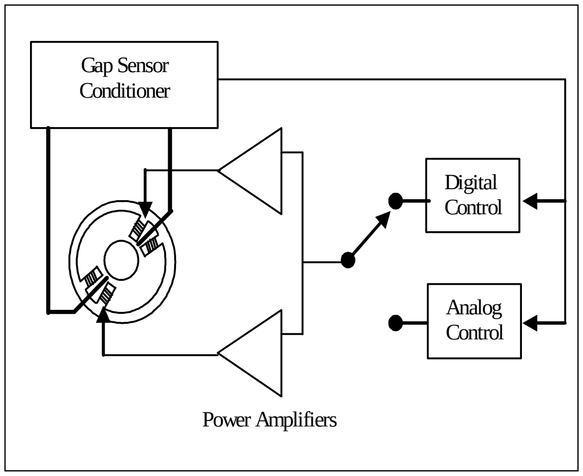

Fig. 4– Single Axis of AMB showing analog and digital controllers.

$$y(n) = 0.7467*y(n-1) + 4.6380*x(n) + 4.5189*x(n-1). \quad (3)$$

The analog and digital controls were connected in parallel as shown in Fig. 4 to compare responses. Fig. 5 shows both the digital and analog controller responses to a mechanical impulse on the shaft; they are virtually identical. The digital loop operates at 10 KHz in a MIMO (multiple input-multiple output) configuration (4 channels input, 8 channels output).

VI. SOME ROTORDYNAMICS

As they say, the proof of the pudding is in the eating. In other words, can the digital controller just described control the rotor shown in Fig. 1 in a predictable way? To answer this question, at least in part, the controller model was linked to a low order model of the AMB and rotor as a classical negative feedback loop as shown in Fig. 2

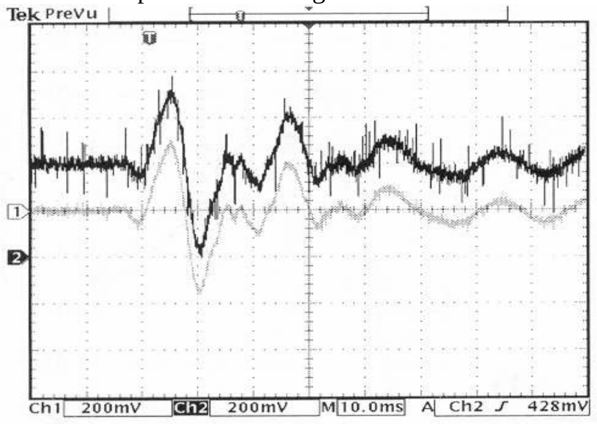

Fig. 5– Oscilloscope traces of a single axis of the AMB comparing analog (lower) and digital (upper) response to a mechanical impulse.

The open loop AMB/rotor system (the plant) has a pole in the right half plane, signifying that the system is unstable. Simulation of the corresponding closed loop system indicates that this pole migrates to the left half plane when the controller gain reaches a value of 27; hence the system becomes stable. Simulation further indicates that the closed loop system will remain stable until this gain reaches 169.

The agreement between the simulation and experiment proved to be within ten percent. This level of agreement extends to rotor spin speeds up to at least 1500 RPM. This suggests that the pudding may well be worth the eating. One may also infer that the fundamental resonance frequency can be varied by changing the controller gain. It has not gone unnoticed that this capability may have useful application in the real world of rotordynamics.

VII. Advanced Experiments

Alternate control laws are easily implemented in C and experimentally verified. Higher order control laws and different laws on the orthogonal axes have been implemented. The rotor has successfully spun up to 11000 RPM, with the AMB under full digital control. This presents an extreme test for the bearings.

Communication between Linux user space and the RTLinux kernel is done with RT-FIFOs (first-in-first-out files). Because RT-FIFOs are uni-directional two separate files must be created for two way communication with the control module. Function *rtf_create(fifo_id_no, fifo_length)* allocates a buffer of the specified size for the specified FIFO (/dev/rtf0, /dev/rtf1,…./dev/rtf64). It must be called from *init_module()*. Function *rtf_destroy* deallocates the FIFO at the completion of execution. It can be called from *init_module( )* or *clean-up_module()*.

The control law may be changed 'on-the-fly' by changing the difference equation coefficients while the real time module is running. The real time function *rtf_get(fifo_id,&variables,sizeof(variables))* within the real time thread reads the coefficients in a non-blocking mode. The user space code for sending the coefficients to the real time module is:

```
ctl = open("/dev/rtf1",O_WRONLY);
write(ctl,&coeffs,sizeof(coeffs));
ctl = close(ctl);
```

This code is embedded in an NCURSES [10] interface which allows the coefficients to be changed with manual entry in a simple interactive interface as the rotor is spinning.

In a similar way the data stream can be accessed in the control program and sent to user space. The appropriate function in the real time module is *rtf_put(framerate_rtfifo_id, volts, offset)*. In user space *cat /dev/rtf0 > file* sends the output to a file. A simple C program converts the data stream from the controller to a readable form and stores it on a file. It is currently implemented to acquire a slice of ten data items at 10000 continuous time intervals as the user requests.

VIII. Summary and Closure

RTLinux is used to control a working rotor test rig at Tufts University. The controller is realized on a conventional Pentium3 personal computer using the RTLinux extension of the Linux operating system. The control algorithm is implemented using the C language compiler. Various control laws can be implemented and tried on an actual experiment.

An additional advantage is the elimination of a target computer, since the real time OS operates on the same processor as the host computer. Most applications developed as digital control systems launch as a startup executable on a proprietary real-time (RT) target computer. The approach presented here differs; it does not target a RT controller based on a proprietary development system. It uses a Linux software environment developed for applications in control and data acquisition requiring hard real time (deterministic) execution.

Acknowledgments

This work was performed at Tufts University of Medford, Massachusetts, USA. It was partially funded by C. S. Draper Lab. of Cambridge, Massachusetts.

References

[1] Alpaugh, H., Real-Time Control of Magnetic Bearings Using RTLinux, Linux Journal, no. 135, www.linuxjournal.com/issue/135, July 2005.

[2] Onuma, H., Masuzawa, T., Matsuda, K., Okada, T., Magnetically Levitated Centrifugal Blood Pump with Radially Suspended Self Bearing Motor, Proceedings of the Eighth International Symposium on Magnetic Bearings, 26-28 August 2002.

[3] Onuma, H., Murakami, M., Masuzawa, H., Novel Maglev Pump with a Combined Magnetic Bearing, ASAIO Journal, vol. 51, no. 1, January/February 2005, pp 50-55.

[4] Clark, D., Jansen, M., Montrague, G., An Overview of Magnetic Bearing Technology for Gas Turbine Engines, NASA/TM-2004-213177, Aug.2004 (http: //gltrs.grc.nasa.gov).

[5] FSMLabs, Inc., Socorro, NM 87801 USA, www.rtlinuxfree.com.

[6] RTAI, Department of Aeronautical Engineering, Politecnico di Milano (DIAPM), Milan, Italy, www.rtai.org.

[7] Free Software Foundation, Cambridge, MA, 02139 USA www.gnu.org.

[8] Tennis, C., Data Acquisition with Comedi, Linux Journal, no. 124,www.linuxjournal.com/issue/124, August 2004.

[9] United Electronics Industries, Canton, MA 02021 USA www.ueidaq.com.

[10] Padala, P., NCURSES Programming HowTo, www.tldp.org/HOWTO/NCURSES-Programming-HOWTO.