REAL TIME CONTROL IN LINUX

UNIVERSITY
of
GLASGOW

Xiaoyu Duan

A Thesis Submitted to the Faculty of Engineering of the University of Glasgow for the Degree of Master of Science

Acknowledgements

The author wishes to express appreciation to Dr. Donald Ballance for his guidance and support throughout the whole project work. Thanks to Mr. Chunming Xia and Miss. Xiaoyun Zang for the advice regarding control theory. Also thanks to Mr. Kenneth Stevenson for all the help in terms of computer and experimental equipment. Finally, I wish to give special thank to Dr. Yubin Shi for his constructive suggestion for the graphical interface part and I wish to acknowledge all the support of staff in the Department of Mechanical Engineering throughout my study in University of Glasgow.

Abstract

In this thesis, the approaches to achieving real time control under Linux operating platform are presented and four different real time control applications are discussed. The driver of the AD512 data acquisition card is programmed to enhance hardware supported by COMEDI through which the connection between computer and DAQ boards are built up. A simple project combining RTAI (Real Time Application Interface) with COMEDI is introduced together with the discussion of one SISO (Single-Input Single-Output) control project and two SIMO (Single-Inputs Multi-Output) control projects based on different controllers, and RTLab is selected to provide us with real time functionalities as it combines COMEDI with RTAI or RTLinx very well in Linux. Further more, to enhance the observability and maneuverability of RTLab, additional custom plugin graphic windows have also been made for every application in the project.

Table of Contents

List of Figures

Chapter 1 Introduction

1.1 Background

Along with the development of modern industry, the demands on control systems are getting higher and higher. Accuracy is no more the only requirement of the control system. Real time operating system (RTOS) shows its great potential and becomes an overwhelming trend of modern science technology. The requirement of stability, reliability, real time capability and maneuverability makes Linux the first choice for real time operating system irresistibly.

1.2 The Overall Objective

This thesis advances several Linux operating system based real time controlling approaches, using different controllers applied to different systems. Several virtual applications are also introduced to enhance the discussion.

The demands of real time require the system to respond and process quickly and accurately. This means that the result of controlling not only relies on the control accuracy but also on the time of responding. A real time scheduler arranges the response priority in real time Linux system. FIFO and other Inter Process

Communication (IPC) methods have been used to achieve data transfer between real time and non-real time parts. Following is the connection and relationship between the major components.

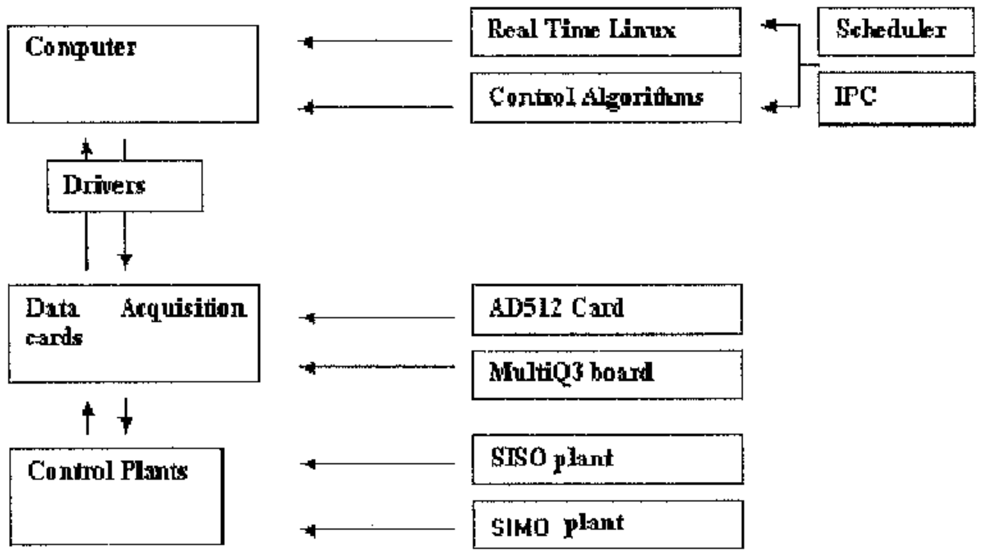


Figure 1.1: The relationship between major components in the project

1.3 Outline of The Thesis

Chapter 2 contains some fundamental background to Linux and real time Linux which is the basis of the thesis. As the operating platform, Linux has a number of characteristics that make it different from other OSs. To equip Linux with real time functionalities, real time Linux has to be involved in the project. The reason why

Linux was selected, and why real time Linux is needed will be discussed later in this chapter.

The principles, equipment and software of data acquisition are described in Chapter 3. The two data acquisition (DAQ) cards, and COMEDI, the software for data acquisition in Linux are also introduced in this chapter. A driver for an AD512 card, which is one of the DAQ cards used in the project, was written to enhance the support hardware of COMEDI. A brief description about writing drivers for COMEDI is given and physical hardware used in the applications in this project is also introduced.

Chapter 4 introduces RTLab, which was one of the major software packages required for the project. It combines COMEDI with real time Linux to achieve real time control using the Linux operating system. Currently the graphical interface in RTLab is only available for inputting information, the user is able to input parameters to system but unable to get any feedback from the system. A feedback graph was therefore added into the original parameter window to display the current status of control plant in the form of graphics.

By applying alternative scheduling policies, different results may be obtained. For instance, the response time in hard real may be quite different from that in soft real time. A discussion about the approaches achieving control in real time with different schedulers is given in Chapter 5.

Chapter 6 will detail four real time control applications: one simple program which combines COMEDI with RTAI LXRT but without graphical interface, and three applications in the form of RTLab plugins in which different controllers are used according to the complexity of the applications, and meanwhile the plugin windows have also been altered to meet the requirement discussed in Chapter 4.

The discussion and conclusions will be made in Chapter 7 and some future directions will be suggested in the last chapter, Chapter 8.

1.4 Work in This Project

The following work has been done in this project based on the previous work:

- Driver of AD512 data acquisition card. The driver of MultiQ3 data acquisition board is provided with Comedi distribution package (with bugs, fixed in this thesis) however that of AD512 is not included. This was programmed in order to use AD512 card in this project.

- A Real Time control program by combining Comedi with RTAI LXRT. There are a number of methods to combining Comedi with RTAI to achieve real time performance under Linux. All these methods have been discussed in chapter 4 and with a real time control program discussed. A motor servo is controlled to change its rotating velocity as a sine wave via DAQ board in the program and RTAI LXRT was used together with Comedi. User can change the controlling setpoint by simply changing the counterpart code in the program.

- Control programs for three different control applications in RTLab. Three different control algorithms were adopted with separate controller for different control projects in this project:
    - Control object—Motor servo. Controller—PID controller.
    - Control object—Rotary Flexible Link with Straingage. Controller—PID controller.
    - Control object—Rotary Flexible Link with Straingage. Controller—LQR controller.
- The custom graphic window in RTLab plugin. Only control parameters were displayed in the original plugin window, which means, the original graphical interface for RTLab is purely for inputting information, and by which users can only change control parameters online. In this project, a custom graph has been added into the original graphical interface in which a graph of the motor servo is displayed. The current angular position will appear in the graph and it

changes along with the virtual plant. Users may also designate its refreshing frequency. The graphical interface can be either for inputting or for outputting.

Chapter 2 Background to Linux and Real Time Linux

This chapter contains some fundamental background information on Linux and real time Linux that forms the basis of this thesis. The history, characteristics and the functionalities of Linux will be introduced. This chapter also addresses the concept, categories and usage of real time Linux that provides us with real time abilities. Finally, two of the most commonly used real time Linux branches, RTAI and RTLinux, and the Inter-process Communication (IPC) methods available in real time Linux will be discussed.

2.1 History of Operating Systems

Computer technology has progressed rapidly over the past several decades, from the first Mini computer to Microcomputer, from Apple to Pentium. Operating systems have also progressed as well as the computer itself. DOS (Disk operating system) is probably the earliest popular operating system. It is based on the basic command line input mode, and requires users to remember quite a number of commands with predefined formats. Windows series are the milestone in the history of operating system. These systems allow users to execute commands, run programs, or perform certain operations simply by clicking on the icons in the screen, and this makes it a suitable operating system for all-level users.

Linux came forth in the late 1980s. It was pioneered by a Finnish university student, Linus Torvalds, who was studying an operating system named Minix which was developed by computer scientist Andrew S. Tanenbaum at that time. In the early days, Linux was regarded as an operating system used by hackers and was put onto an FTP server free for download. It has now become a POSIX compatible operating system with all UNIX characters.

2.2 Linux Operating System

2.2.1 General Description of Linux

When Linus was studying Minix, he noticed that the functionalities of Minix were not complete and he therefore programmed another operating system running in protection mode, and that was the prototype of Linux.

Linus announced his first version of Linux, version 0.02 on Oct 5, 1991. At that time, nothing could be done in Linux except running bash (the GNU Bourne Again Shell) and gcc (the GNU C compiler), and the development was focused on the kernel part as it was just regarded as a hacker's system. Linus made the source code of Linux public since its first appearance, and also put it onto some FTP servers where it is free for download. The Administrator figured it as the Minix of Linus hence he named it Linux. Linux has its own name since ([14]).

In the next few years, Linux developed at an amazing speed and far from what Linus expected. On March 14 1994 its first official version, version 1.0 was announced; Linux forum has become one of the most popular forums in USENET. Meanwhile, the kernel version developed rapidly as well, the latest kernel version is 2.6 and is still developing rapidly.

2.2.2 Linux and GNU

It is hard to describe Linux without the introduction of GNU. GNU is the abbreviation of GNU's Not Unix. It was initially established by Richard M. Stallman, the chairman of Free Software Foundation, in 1984. Stallman worked in the Artificial Intelligence laboratory of Massachusetts Institute of Technology at that time and he is regarded as one of the top class programmers in the world. He was convinced that even if UNIX is not the best operating system in the world, it was not too bad, and it had the potential to be something more than it showed. The major goal of the development on this system is making it free to every user, that is to say everyone can acquire, copy, modify and redistribute the source code with no extra cost.

GNU also has the copyright announcement of its own, General Public License (GPL), saying famous copyleft. It states 'one can redistribute this library and/or modify it under the terms of the GPL as published by the Free Software Foundation; either version 2 of the License, or any later version, and it is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose ([14]). At a word, GPL ensures GNU is always free and public.

2.2.3 *General Distribution Versions of Linux*

(1). Debian GNU/Linux

Distributed by Free Software Foundation (FSF), suitable for high-level users.

Website: http://www.debian.org. ftp://ftp.debian.org/debian.

(2). Redhat Linux

Created Redhat Package Manager (RPM) to manage software, the best choice for beginners.

Website: http://www.redhat.com. ftp://ftp.redhat.com

(3). Slackware

Website: http://www.slackware.com.

(4). Mandrake

Website: http://www.mandrakelinux.com.

(5). SuSe

Now owned by Novell. In addition to systems and application software for private users, SUSE Linux provides services to the deployment in the enterprise.

Website: http://www.suse.com/us/.

2.2.4 Linux Commands

Different from other operating systems, LINUX is not directly intuitive. Many commands have seemingly queer names or formats, and may have different effect from that of their MS-DOS counterparts even though they may appear to be similar. Benefiting from years of experience with standard UNIX utilities and advances in computer science, programmers on the GNU project have managed to create versions of standard tools that have more features, run faster and more efficiently, and lack the bugs or inconsistencies that persist in the original standard versions ([9]).

The `Shell` is not only used to accept and execute commands as a command interpreter in Linux, but also works as an interface between operating system and users ([2]). Different shells may provide dissimilar commands. The familiar shells under Linux are:

- Bourne shell, /bin/sh;
- Cshell, /bin/csh;
- Kornshell, /bin/ksh;
- Bourne again shell, /bin/bash.

The shell currently being used can be determined by the command `echo $shell`. The most commonly used shell is Bourne again shell, which is provided by almost all Linux systems.

User can use "-help" argument to acquire online help for every command, or "man" to browse more detailed information.

2.2.5 The Linux Kernel

The Linux kernel is the central part of Linux. It is the code that controls the interface between user programs and hardware devices, schedules multitask processes, and manages many of the other functionalities of the system ([14]). Similar to other Unix kernels, Linux kernel needs to accomplish:

- Managing file system and I/O operations;
- Managing processes, allocating resource for program, and communicating between programs;
- Managing and allocating memory and virtual memory;
- Managing network, connections.

The kernel allocates hardware resources to tasks running simultaneously, and has them running individually and safely. Linux kernel is known as a monolithic kernel because all device drivers are components of the kernel properties. Some operating systems make use of microkernel architecture, in which device drivers or other code are loaded and executed on demand, and with no necessity to reside in the memory all the time.

Linux kernel is composed of 5 subsystems ([14]):

- Process scheduling (SCHED). This is the heart of an operating system and the objective of it is managing the access to CPU;
- Memory Management (MM). MM controls the access to system memory;

- Virtual File System (VFS). It provides a uniform interface for all hardware equipment;
- Network Interface (NET). It gives Linux the access to network;
- InterProcess Communication (IPC). In case of multiprocess, communication between processes is necessary.

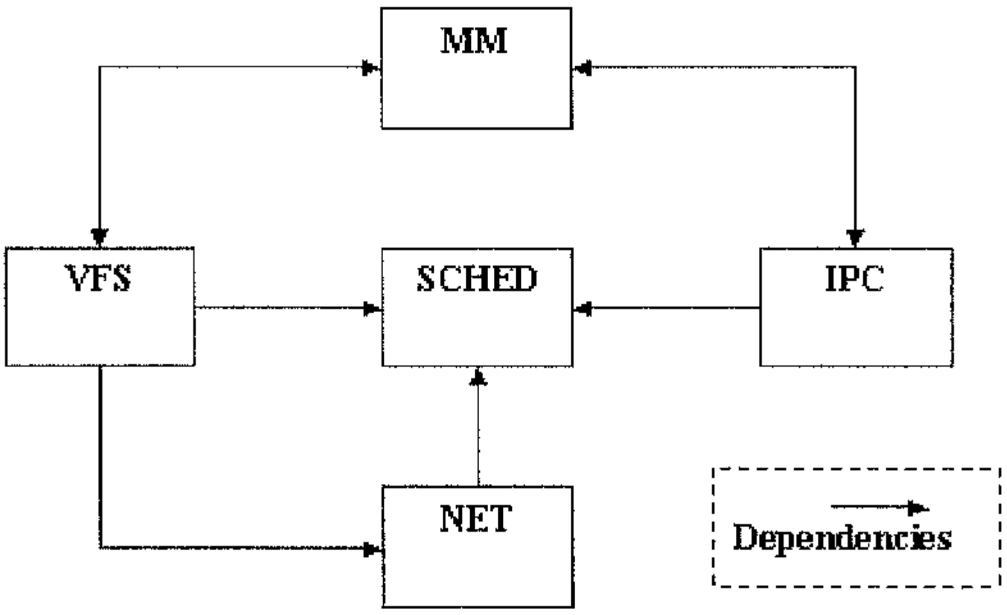


Figure 2.1: Relationship between subsystems

2.2.6 Linux Loader (LILO)

LILO is a general-purpose boot manager which can be used to boot almost every operating system in current use including Linux ([2]).

There are several ways of configuring LILO. Two most common methods are:

- Installing LILO on the master boot record (MBR) of the hard drive;
- Installing LILO as a secondary boot loader for Linux only.

The most common way to boot Linux from the hard drive is doing it by LILO. The kernel itself is stored on the hard drive so that no boot floppy is needed. Once the kernel is loaded into the memory, control will be transferred to the kernel instantly. If LILO is installed in the master boot record of the drive, it will be executed first when the hard drive is booted. The user can then select the operating system they prefer such as Windows or Linux at boot time. However it must be noted, if the user desires to have both Windows or OS/2 and Linux residing in the same machine simultaneously, it is recommended to install Windows prior to installing Linux. This is because both OS/2 and Windows have their own boot managers which occupy the MBR. If the user is using one of these systems, he may have to install LILO as the "secondary" boot loader for Linux only in order to boot Linux from the hard drive. In this case LILO is installed in the boot record for the Linux root partition only, and the boot manager software will run LILO from there when the user wishes to boot Linux. If however Windows is installed after Linux, it will occupy the master boot record despotically and destroy what Linux has set before. This will result in the machine not being able to boot from Linux unless the user uses a boot floppy or re-installs it.

2.2.7 Programming in Linux

(1) C and Linux

C is probably the most common used programming language nowadays. It is not only suitable in terms of application program producing but also in system software programming. In the early days, Programmers were using assembly languages which relied on the hardware to a great extent to program system software such as the operating system (including UNIX operation system), and this caused bad readability and portability. Therefore high-level languages began to be used. The C language came into being in the early 1970s, almost concurrently with the early development of the Unix operating system. In 1978, the first description work of the C language appeared—*The C Programming Language,* often called the `white book' or `K&R'. Soon after, it was officially standardized by the ANSI X3J11 committee who made the further changes in the mid 1980s. Today it has become one of the most prevalent languages in the computer industry ([13]).

C's popularity and tremendous vitality come from its distinctive characteristics (13):

- C only has 32 keywords, 9 kinds of scripts. This makes it simple and flexible;
- It has abundant operation symbols and data structures, can achieve operations which other programming languages are unable to accomplish;

- C allows user to access physical addresses directly and achieve many functions which were only available to assembly language before;
- Good portability

As the clone of UNIX, Linux has a very close relationship with the C language. Actually, Linux itself is actually written in C. All the Linux systems support C/C++ well in spite of some of the characteristics that may differ from system to system.

(2) GCC Compiler

GNU C compiler (GCC) is a full-functional ANSI C compatible compiler. Enter command gcc -v after shell prompt and the version will be shown on the screen. GCC is based on command line input mode, and is often used together with options and filenames as parameters.

Following is an example of general gcc structure:

```
Gcc [options] [filenames]
```

GCC provides over 100 compiling arguments for instance:

User can specify a filename which will be created after compiling by using **–o** argument:

```
gcc  -o test test.c
```

Or use **–c** argument to omit compiling and linking stages

It must be noted, that the symbol "-" must be used for only one argument rather than a set of arguments each time.

(3) Using GDB

GDB is the abbreviation of GNU project DeBugger, it allows users to inspect what happens "inside" a program when the program is running, or what is causing the program to crash.

The functions GDB can achieve are ([14]):

- Observing the variable value when the program is running;
- Stopping the program at any specific step by setting setpoints;
- Examining what is happening when program stops.

The program debugged can be written in C, C++, Pascal and many of other languages. They can be executed either on the same machine in which GDB (local) locates or another machine (remote), and GDB can run in most of UNIX and Microsoft Windows operating systems.

(4) Using makefile

In Linux, creation and maintenance of the object program is achieved by the command make. make is a general-purpose program that builds target files from object files. The target file could be an executable program, a postscript document, etc. The object file can be C code, a text file, and so on.

The Example Makefile ([31]):

```
project.exe : main.obj io.obj
        tlink c0s main.obj io.obj, project.exe,, cs /LF:\bc\lib
main.obj : main.c
        bcc -ms -c main.c
io.obj : io.c
        bcc -ms -c io.c
```

make reads its instructions from text files. An **initialisation file** is read first, it holds the instructions for make and is used to customize the operation of make. make automatically reads the default initialisation file (normally named Makefile) whenever it starts up, and user can also specify it to other filenames by the command with the format as:

```
make -f myfile
```

The basic goal of make is to let the user built a file in small steps. If the final executable file is made up of many source files, make can give a user the flexibility of changing one of them and rebuilding the executable file without having to recompile everything.

This makefile file has three main rules, one each for making project.exe, main.obj, and io.obj. These rules are called **explicit rules** since they are supplied explicitly in the makefile. make also has **inference rules** that generalize the make process. The lines within the colon ":" are called **dependency lines**, the filename in the left hand side of colon ":" is the target of the dependency, and the filename in right side is the source needed to make the target. For example, project.exe: main.obj io.obj means "project.exe depends on main.obj and io.obj". At run time make compares the time that project.exe was last changed to that of main.obj and io.obj. If either source is newer than project.exe, make rebuilds project.exe ([31]).

The lines that follow each dependency line are called **shell lines**. Shell lines tell make how to build the target.

When each shell line has been executed, make checks the shell line **exit status**. By convention, programs return an exit status of zero if they finish without error and non-zero if an error occurs ([31]).

The user can also employ macro definitions in makefile. This makes it very convenient and flexible to change compiling requirement without re-typing long compiling commands each time. A macro definition line is a makefile line with a macro *name*, an equals sign "=", and a macro *value*. In the makefile, the form $(*name*) or ${*name*} is replaced with *value*. If the macro name is a single letter, the parentheses or braces are optional (*i.e.* $X, S(X) and ${X} all mean "the value of macro X").

makefile can also be used to save to list of filenames, executable filenames, compiler command arguments and so on, it is an important component of the program and programming could be much more efficient if makefile is made good use of.

2.3 Real Time Linux

2.3.1 What is a Real Time Operating System?

A Real time operating system is a system that must respond to inputs or events with predefined time limits. The system must operate within a specific time constraints and be capable of predicting and controlling plants when different computation algorithms are applied ([14]). It is the vital component of the technological infrastructure of an industrial nation and is widely used in modern telecommunication systems, automated factories, defence systems, power plants, aircraft, airports, spacecraft, medical instrumentation, and SCADA systems.

The biggest difference between a Real Time Operating System (RTOS) and a normal operating system is that the Real Time Operating System must satisfy the relationship between processing and time ([14]). In real time computation, the accuracy of the system not only relies on the correct result of the computation, but also on the time in which results are generated. Real time system must respond to urgent events quickly and predictably, have high-level schedulability, and stability under transient overload.

The most important requirement for a RTOS is that it must have the capability of responding to and processing internal or external events in a pre-defined time ([14]). An RTOS it must have effective capability of:

- Processing interrupt;
- High efficiency I/O ability;
- Processing asynchronism;
- Receiving data and sending application within strict time limitation.

The detailed requirements are:

- System should be capable of distinguishing and processing discrete events in a pre-defined time;

- System can process and store a huge amount of data that the control system needs.

The most important component in RTOS is the Real Time Multi-task Kernel, which is used to accomplish the functions of:

- Tasks management;
- Timer management;
- Memory management;
- Resource management;
- Events management
- System management;
- Message management;
- Queue management;
- Semaphore management;

2.3.2 Some Popular RTOSs

- **QNX**

QNX is an embedded, expandable real time operating system. It abides by POSIX.1 (programming interface), POSIX.2 (shell and tools), and POSIX.1b (real time expansion) partially. QNX was pioneered in 1980, and has developed rapidly in recent years [14].

QNX is a microkernel real time operating system. The QNX kernel provides 4 kinds of services: process scheduling, inter-process communication, network communication and interrupt processing. All the OS services are regarded as cooperative user processes therefore QNX kernel is very small (about 12 KB for QNX4.x) and rapid [14].

- **LynxOS**

Similar to QNX, LynxOS is also an embedded, expandable real time operating system. It abides by POSIX.1a, POSIX.1b, and POSIX.1c standard, pioneered in 1988. The microkernel (28 KB) of LynxOS provides the services as: kernel startup and termination, memory management, error processing, etc [14].

LynxOS supports threads concept ([4]), uses hard real time priority scheduling and preemptable RTOS kernel.

- **VxWorks**

VxWorks is a real time operating system which was developed by a company named Wind River Systems in United States. It is widely used in the area with high real time requirements such as communication and aviation [14].

- **RT-Linux**

RT-Linux is an embedded hard real time system, support POSIX.1b standard partially [14].

A small, simple, real-time kernel is inserted beneath the normal Linux kernel in RT-Linux, having Linux as a task which only runs when there is no real-time task running at the same time. Different from the microkernel and normal kernel, RT-Linux belongs to real time EXE structure. A more detailed description will be given later.

- **RTAI**

RTAI is another real time extension which is developed by Dipartimento di Ingegneria Aerospaziale - Politecnico di Milano (Department of Aerospace Engineering-Polytechnic of Milan). It is selected to provide real time functions in this project.

- **KURT-Linux**

KURT-Linux is a "strict" real time system. The KURT-Linux kernel includes two major parts: Kernel and Real time modules. The kernel is responsible for real time tasks scheduling, provides specific real time services for user processes. KURT-Linux can run in two different modes: normal and real time. All processes can run in normal mode, but some of the kernel services might cause unexpected interrupt. In real time mode, the only process allowed to run is real time process [14].

KURT-Linux Supports the scheduling method of: FIFO (First In First Out) scheduling, recursive scheduling, UNIX time-shared scheduling and SCHED-KURT [14].

2.3.3 RTOS Designing

There are a number of ways to design RTOS according to applications. For instance, a RTOS can be a periodic (time-sharing) or an aperiodic (event-driven) system,. Periodic system means the system uses a sensor to probe external changes periodically, and then responds to it. Aperiodic means external events take place recursively but not regularly ([14]). Time-sharing means the operating system will change at time intervals, and event-driven means the system will only change in response to events or interrupts, and it is commonly associated with cooperative operating system in which system waits for a process to surrender control.

There are two main ways to react to an event: polling method and interrupt driven method. In polling mode, the application program continually polls the different system peripherals to check if they need service. When a peripheral is ready for servicing, it must wait until the software polls this peripheral. Therefore, polling-mode peripherals experience longer response time from the processor as more peripherals are added to the system. Therefore polling-mode systems can become unstable in this case since the response time of each peripheral is affected. Generally it is only appropriate when the system is small enough. In interrupt-driven mode, each peripheral usually has one interrupt indirectly feeding into the processor's interrupt port via an interrupt controller. The interrupts coming from peripherals can be prioritized. The processor always services the interrupt with the highest priority first. Consequently, the response time of an interrupt-driven system is much faster. However, in interrupt-driven systems there is the possibility that lower-priority peripherals are never serviced. Thereby interrupt-driven systems must be carefully designed according to the real time requirements of various peripherals. There is generally more stability with interrupt-driven systems since the response time for each interrupt can be estimated with more accuracy and peripherals can be added to the system without affecting the response time of existing peripherals. Usually real time operating systems use interrupt-driven methodology.

A RTOS can also be a hard real time or soft real time system. Hard real time system means the system must respond to the affair in time to avoid major damage. For example, the pilot system of an airplane has to respond to the control signal quickly to ensure safe flying. In soft real time system, system is allowed to respond to the event a bit later than the responding deadline in case of overloading, and this may not result in big loss or disaster. For example, in communication system, one call it allows to omit one call among 105 calls.

The biggest difference between hard real time system and soft real time system is the scheduling policy. User can use static periodic scheduling or FIFO (First In First Out) scheduling to schedule tasks. However both static periodic scheduling and FIFO scheduling are monopolized algorithms. In other words, neither of them allows preempting. This will be introduced in detail in later chapters.

2.3.4 How Does Real Time Linux Work?

It is possible to take control of a camera, robot, or other scientific equipments by a personal PC via Linux. However, Linux itself cannot control devices with hard real time requirement reliably. For example, connecting a speaker to a pin of parallel interface, and then run a program to play music. If this program is the only one running, speaker will emit stable music. If refreshes of the window occur every 2 seconds, user can notice that the music may change slightly. The sound will become irregular if two or more windows are being opened at the same time. Further more, if running Netscape in one of the windows, the music will become discrete and distorted.

Like some of other operating systems, Linux optimizes every function and tries to allocate time to each process equally. This is essential to normal operating system, but in real time operation, counting and forecasting functions are much more important than others. For example, a camera is required to fill the buffer every one microsecond, and it may cause data loss if the process taking charge of filling the buffer is delayed for just for even a small time.

Linux with real time function can achieve lots of tasks and operations as the user desires. In real time Linux, Linux must be cleared from the CPU whenever real time task needs it. Generally, there is no need for Linux to know how the RTOS runs, how it sends interrupts or controls hardware devices, but real time tasks can run in a high accuracy level. In a 120-MHz Intel Pentium (P120) testing system, a series of tasks can run orderly with an error of just 20 microseconds.

The following graph shows the relationship between Real Time Kernel and Linux Kernel.

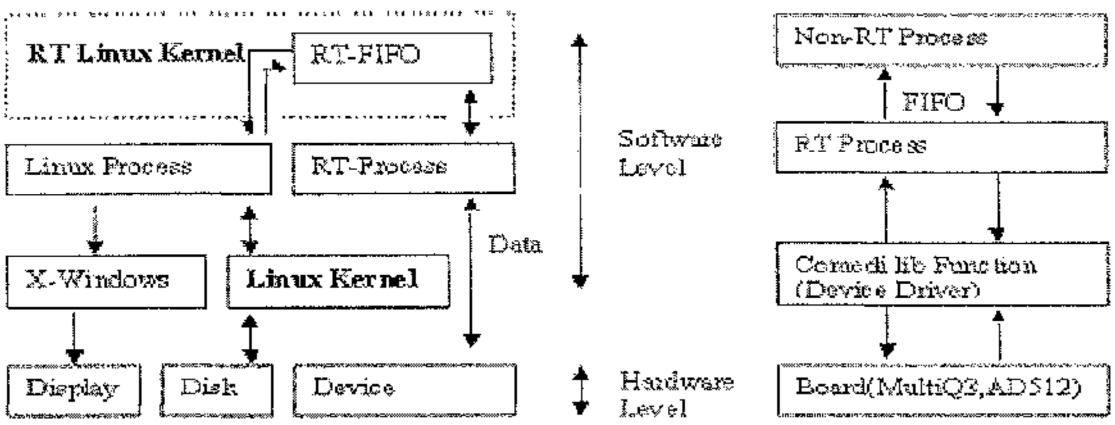


Figure 2.2: The relationship between real time kernel and normal Linux kernel

There are two primary variants of hard real time Linux available: RTAI and RTLinux. The real time kernel in RTAI is called Real Time Hardware Abstraction Layer (RTHAL), it intercepts all hardware interrupts and routs them either standard Linux or to real time tasks depending on the requirements of the RTAI schedulers ([6]). Compared to RTHAL, the real time kernel in RTLinux is known as RTCore which allows users to get hardware-limit real time performance with all of the flexibility provided by Linux. Details about RTAI and RTLinux will be introduced later.

2.3.5 Linux Loadable Kernel Module (LKM)

The simplest method of adding code into a Linux kernel is to add some source files to the kernel source tree and recompile the kernel. The file which should be included for compiling is set in the kernel configuration.

It is also possible however to add code into the Linux kernel directly while it is running. A chunk of code added in this way is called a loadable kernel module (LKM), and this module can be a device driver, a file system driver or even a system call.

All the drivers and plugins are integrated in the form of LKMs in this project.

Loadable kernel modules have a lot of advantages ([17], [5]):

- Not necessary to rebuild kernel time after time. This prevents the user from wasting too much time on rebuilding and reinstalling the base kernel;
- LKMs help user to diagnose system problems. A bug in a device driver which is bound into the kernel can stop the system from booting, and it is very hard to know where the problem resides in such a case. If the device driver is inserted as LKM, the base kernel could run before the device driver is loaded, and if the system crashed after a certain module was loaded, it is easy to track and fix the problem;
- LKMs save memory. Kernel modules will be loaded only when actually needed;

- It is very fast to maintain and debug LKMs;
- Running LKMs is not slower than running base kernel modules.

LKMs can be used for [17]:

- `Device drivers`. The kernel uses a device driver which is designed for a specific piece of hardware to communicate with that piece of hardware without having to know any detail about how that hardware works;
- `Filesystem drivers`. Filesystem is the content of a disk drive generally. A filesystem driver interprets the content of a filesystem as files and directories. Files and directories can be stored on disk drives, network servers or other places, but for each case, a file system driver is needed;
- `System calls`. User space programs use system calls to get services from the kernel. Although most of the system calls are integrated into the system and are very standard, users can make system calls of their own;
- `Network drivers`. A network driver interprets a network protocol;
- `Executable interpreters`. An executable interpreter is used to load and run an executable file or task;
- `TTY line disciplines`.

After creating the desired LKMs, user can operate on them with the following utilities:

- `insmod`. Insert an LKM into the kernel;
- `rmmod`. Remove an LKM from the kernel;
- `depmod`. Determine interdependencies between LKMs;
- `kerneld`. Kernel daemon program, it allows kernel modules to be loaded automatically;
- `ksyms`. Display symbols that are exported by the kernel for use by new LKMs;
- `lsmod`. List currently loaded LKMs;
- `modinfo`. Display content of modinfo section in an LKM object file;
- `modprobe`. Insert or remove an LKM or set of LKMs intelligently;

A user can use command `cat /proc/ksyms` to list every symbol that is exported by the kernel and command and `cat /proc/modules` to see the presently loaded LKMs.

Generally, module files can be found in the directory */lib/modules*, divided into subdirectories.

The kernel initialises an LKM when the kernel is loaded, and it initialises a bound-in module at boot time.

It is necessary to introduce the concept of kernel space and user space here. User space is a term for combined address of all user-level applications. The kernel itself has its own address space called kernel space. Generally speaking, kernel space is where the kernel code resides, and user space is where the user programs live. A kernel is all about access to resources which might be a sound card, a video card, a hard drive or memory. Programs often compete for the same resource and the kernel needs to keep everything in order. A CPU can run in different modes, and each mode gives a different freedom level. A user space is an environment where low-priority tasks run. Basically, library functions are used in user mode. The library function calls one or more system calls, and these system calls execute on the library function's behalf, but they do this in supervisor mode because they are part of the kernel. Once the system call completes its task, it returns and execution is transferred back to user mode.

Working in user space provides a better system stability, easier access to library functions such as math-library and compared to working in kernel space, it is also easier to debug in user space when problems occur. However, it provides a better access to resources when working in kernel space. Clearly, one of the main advantages of RTAI over RTLinux (which are discussed later) is that RTAI provides a means of developing in user space (via LXRT).

The main advantages of RTHAL are [6]:

- The changes needed to the standard Linux kernel are minimal. This improves the code maintainability and makes easier to keep the real time modifications up-to-date with the latest release of the Linux kernel.
- The real time extensions can be easily removed by replacing the interrupt function with the original Linux routines. This is especially useful in certain debugging situations when it is necessary to remove the extensions and when verifying the performance of standard Linux with or without the real time extensions.

2.4 Real Time Linux—RTAI

2.4.1 General Description of RTAI

RTAI (Real Time Application Interface), was initially developed by The Dipartimento di Ingeneria Aerospaziale Politecnico di Milan (DIAPM- Department of Aerospace Engineering-Polytechnic of Milan) as a variant of RTLinux developed by the New Mexico Institute of Technology (NMT), at that time neither floating point support nor periodic mode scheduling was provided by RTLinux ([6]).

RTAI is not an intrusive modification of the kernel, it uses HAL (Hardware Abstraction Layer) to provide fundamental functions and get information from Linux, HAL does not depend greatly on the Linux kernel and this provides RTAI with a very good portability.

2.4.2 *RTAI Modules*

There are a numbers of modules provided by RTAI, and the user can load the modules to accomplish every required RTAI functionality ([6]).

1) `rtai`. `rtai` module is the basic RTAI framework. It initialises all of its control variables and structures, makes copies of the `idt_table` and the Linux irq handlers' entry addresses, and initialises the interrupt chips management functions.

2) `rtai_sched` this is a real time, pre-emptive, priority-based scheduler module. `rtai_sched` is in charge of distributing the CPU resource to different tasks in the system. The scheduling occurs when tasks perform certain system calls and timer handler activates. Tasks with different priorities will be arranged at different time. RTAI regards the priority 0 as the highest priority and 0x3fffFfff the lowest. Linux is given priority 0x7fffFfff.

   RTAI supports both periodic and one-shot modes for the real time scheduler.

Three schedulers are available in RTAI:

- ***UP***, only for uniprocessors;
- ***SMP***, for multiprocessors;
- ***MUP***, only for multiprocessors.

The scheduler services are:

- Task functions;
- Timing functions;
- Semaphore functions;
- Mailbox functions;
- Intertask communication functions;

3) `rtai_fifos`. This is the module that implements the FIFOs and semaphores services for RTAI. It is used to achieve communication between the real time system and Linux side, such as managing the data logging and displaying. The real time interface includes creation, destruction, reading and writing functions which are performed by `rtai_fifos` module. User processes consider real time fifos as ordinary character devices.

4) `rtai_shm`. This is a RTAI specific module that allows sharing memory

among different real time tasks and Linux processes. The first allocation does a real allocation, Any subsequent call to allocate with the same name from Linux processes just maps the area to the user space or return the related pointer to the space already allocated in kernel space. A user can also use the 'mbuff' module for access to shared memory.

5) LXRT. The LX (LinuX) RT (Real Time) module, which implements services to make any of the RTAI schedulers functions available to Linux processes. Users can share memory, send messages, use semaphores and timings between Linux and Linux, Linux and RTAI, or RTAI and RTAI.

6) `rtai_pqueue`. Posix RTAI modules. `rtai_pthread.o` provides hard real-time threads, where each thread is a RTAI task. All threads are executed in the same address space and work simultaneously on shared data. `rtai_pqueue.o` offers kernel-safe message queues.

7) `rt_mem_mgr`. Dynamic memory management for real time.

2.4.3 LXRT

LXRT provides the same set of RTAI API calls available for RTAI applications in user space ([6]). It enhances its 'soft' real-time performance by requiring the programmer to change the Linux scheduler's policy from SCHED_OTHER to SCHED_FIFO.

SCHED_OTHER is the standard Linux default scheduling policy used by most processes. SCHED_FIFO and SCHED_RR are used for special, time-critical applications which have high requirement on control precision. Processes scheduled with SCHED_OTHER have a static priority of 0. The scheduler selects which process to run from a waiting list by the level of these processes. Processes scheduled with SCHED_FIFO are assigned static priorities in the range 1 to 99. When a process begins running it will pre-empt a running SCHED_OTHER processes or a SCHED_FIFO process of lower priority. A FIFO (first in first out) policy is applied to processes of the same priority. And SCHED_RR is a simple enhancement to SCHED_FIFO, each process is only allowed to run for a maximum time period before being re-scheduled, this type of scheduling policy is seldom used in LXRT.

LXRT has the important features as following:

- The tasks can execute under the Linux memory protection scheme;
- LXRT allows a system to be easily divided into hard real-time and soft real-time parts in that the LXRT modules will execute at a higher priority than normal Linux processes;
- The tasks can be debugged using standard Linux user-space debug

tools;

- User can move a task into kernel space right after debugging it;
- The tasks can use the standard RTAI API, which makes it very easy to move tasks between hard real time and soft real time parts;
- Once a root user has installed the required modules, these modules can be called by normal users;
- Real time tasks no longer carry kernel dependencies because they are no longer implemented as kernel modules.

Under LXRT, the real time task is implemented as a user space task, but actually, it is scheduled by the real time scheduler after being moved into hard real time. User can achieve this simply by inserting the LXRT module and using LXRT API within user space task.

It must be noted that there is a fixed sequence of inserting RTAI modules, rtai.o must be inserted first, and rtai_sched.o, lxrt.o last, otherwise "unresolved symbol" errors will occur due to module dependencies and insertion will fail.

2.4.4 The Official Website of RTAI

Website: http://www.rtai.org/.

2.5 Real Time Linux—RTLinux

2.5.1 General Description of RTLinux

RTLinux was initially developed by Victor Yodaiken and Michael Brananov who worked in the Computer Science Department of University of New Mexico in United States at that time. It is another real time branch of Real Time Linux. RTLinux has the same principle as the RTAI but they have different API functions and modules.

From the point of view of compatibility with the data acquisition software used in this project—COMEDI, RTAI is selected as it is supported better for the COMEDI nowadays. The detail of COMEDI will be introduced later.

2.5.2 RTLinux Standard Modules

RTLinux provides the following modules ([14]):

- rtl_sched provides scheduling methods based on priority, supports POSIX interface and version 1.0 RTLinux API functions;
- rtl_time provides real time timers;
- rtl_posixio provides the read, write and call operations of driver in POSIX means;

- rtl_fifo provides the communication interface between real time tasks and Linux processes;
- Semaphore is the module which gives information value to real time tasks;
- Mbuff is the shared memory driver for the communication between Linux user process and kernel process.

In these modules, following API functions are provided:

- Interrupt controlling API functions;
- Clock controlling and acquiring;
- Thread creating and deleting, priority and schedule controlling API functions;
- POSIX interface;
- FIFO driver;
- Series port driver API functions;
- mbuff driver API functions;
- Floating point number support API functions

2.5.3 The Official Websites of RTLinux

Website:

- http://www.rtlinux.com/,

- http://www.rtlinux.org/,
- http://www.fsmlabs.com/

All the three websites above direct to the same webpage.

2.6 Inter-process Process Communication (IPC)

There are a number of approaches to achieving Inter-Process Communication (IPC) between realtime tasks and non-real time tasks within real time Linux operating system. Inter-Process Communication (IPC) means passing messages between active processes or between tasks. FIFO is probably currently the most commonly used IPC approach in real time process. Semaphores, mailboxes, shared memory remote procedure call (RPC) functions and POSIX APIs are also available for the same purpose:

- Real time FIFO: A FIFO (First In First Out) is a read/write buffer used to asynchronously transfer data between real time Linux tasks and processes. Similar to a pipe, one end opened for writing, and another end opened for reading operation ([10]). RTAI supports two RT_FIFO implementations, "oldfifos" and "newfifos". Oldfifos are based on the original NMT-RTL FIFOs, while newfifos are based on completely new code but maintain fully compatibility with the basic services provided by its original NMT-RTL counterpart while adding some additional features ([10]).

- Semaphores: They are used to achieve synchronization between tasks either with regard to access to shared resources or as a simple binary, message-passing system.
- Mailboxes: They provide the capability to transfer data of user-defined sizes between Linux and RTAI.
- Shared memory: They provide a means of transferring data between real time and user space tasks, in which a portion of physical memory is set aside for sharing between them.

- RPCs: RPCs are similar in operation to QNX-style messages available to real time tasks. They can either transfer an unsigned integer or a pointer to the destination task.

Shared memory is selected in this project and this will be detailed in later chapters.

Chapter 3 Data Acquisition and Physical Equipment

Both hardware and software are crucial for data acquisition in a control system. MultiQ3 and AD512 are two of the data acquisition (DAQ) boards used in this project by which the user is able to build up data transfer between the physical environment and the computer. As the interface between DAQ boards and the computer, COMEDI links the 2 sides by adding drivers for the DAQ boards which the user wishes to use into the COMEDI driver library. The introduction of MultiQ3 board, AD512 card, and control plants will be given in this chapter.

3.1 Hardware for Data Acquisition

3.1.1 Data Acquisition (DAQ) Boards

3.1.1.1 Quanser Consulting MultiQ3 Board

The MultiQ3 is a general purpose data acquisition and control board which has 8 single ended analogue inputs, 8 analogue outputs, 16 bits of digital input, 16 bits of digital output, 3 programmable timers and up to 8 encoder inputs decoded in quadrature. Interrupts can be generated by any of the three clocks, one digital input line and the end of conversion from the A/D ([27]).

The system is accessed through the ISA slot and is addressable via 16 consecutive memory mapped locations which are selected through a DIP switch located on the board.

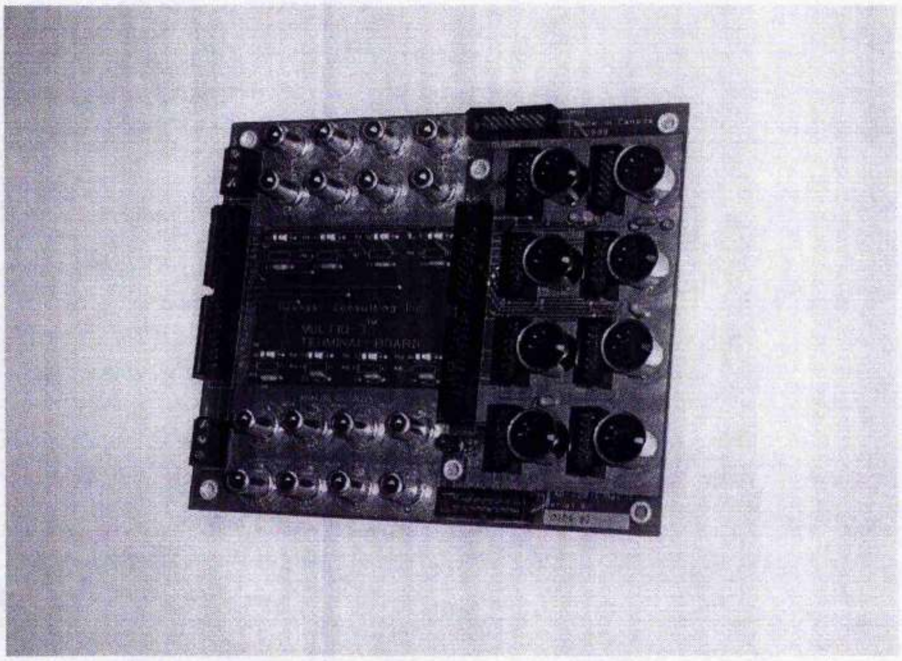


Figure 3.1: MultiQ3 data acquisition board

An online manual is available at:

http://mechanical.poly.edu/faculty/vkapila/ME325%5CMultiQ%5Cmq3_manual.pdf

3.1.1.2 Humusoft AD512 Data Acquisition Card

The AD512 data acquisition card is another general data acquisition and control board. It contains a 100 kHz throughput 12 bit A/D converter with sample/hold circuit, four software selectable input ranges and 8 channel input multiplexer, 2 independent double buffered 12 bit D/A converters, 8 bit digital input port and 8 bit digital output port ([26]).

The AD512 card is designed for standard data acquisition and control applications and optimized for use with Real Time Toolbox for MATLAB ([7]). The AD512 can be used not only in desktop computers but also in portable computers due to its small size and low power consumption ([26]). To be aware, when working with notebooks, it can only be used in ones which have device such as docking station with ISA slot. It is recommended to use PCMCIA boards in this case.

An online manual is available at:

http://www2.humusoft.cz/www/datacq/manuals/ad512um.pdf

3.1.2 Control Plants and Other Physical Equipment

Two sets of plants will be controlled in this project: a rotary position motor servo (single –input single-output) and a rotary flexible straingage with a motor servo (single-input multi-output). Both of them are manufactured by Quanser and driven by an UPM-15-03 power module.

3.1.2.1 Rotary Position Motor Servo SRV02

The rotary position servo consists of a DC servomotor and a built-in gearbox whose ratio is 14 to 1. The output of the gearbox drives a potentiometer and an independent output shaft to which a load can be attached. SRV02 is equipped with only one potentiometer and has a tachometer attached to the back of the motor. The position of motor shaft is measured by a sensor attached to the shaft.

The control objective for this plant is to implement a controller to control the position of the output shaft

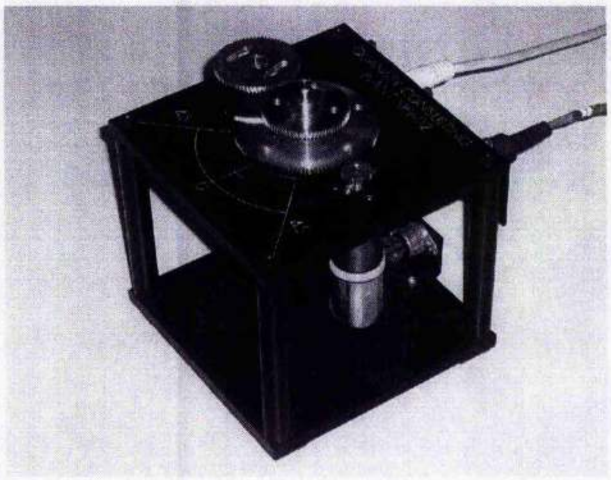
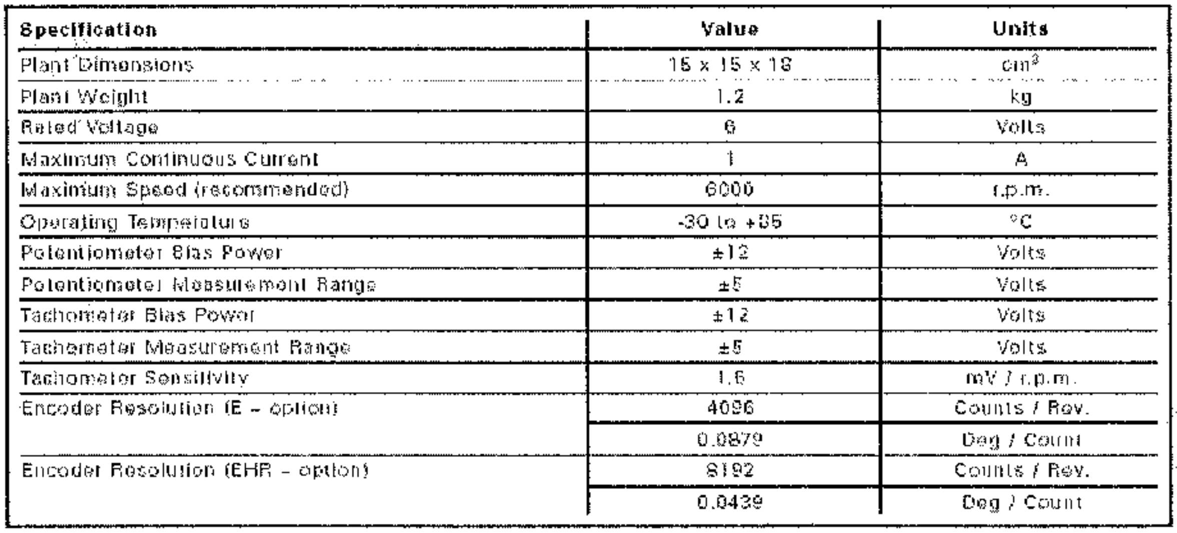


Figure 3.2 Rotary position motor servo SRV02

System parameters of servo SRV02:



Specification	Value	Units
Plant Dimensions	15 x 15 x 18	cm^3
Plant Weight	1.2	kg
Rated Voltage	6	Volts
Maximum Continuous Current	1	A
Maximum Speed (recommended)	6000	r.p.m.
Operating Temperature	-30 to +85	°C
Potentiometer Bias Power	±12	Volts
Potentiometer Measurement Range	±5	Volts
Tachometer Bias Power	±12	Volts
Tachometer Measurement Range	±5	Volts
Tachometer Sensitivity	1.5	mV / r.p.m.
Encoder Resolution (E – option)	4096	Counts / Rev.
	0.0879	Deg / Count
Encoder Resolution (EHR – option)	8192	Counts / Rev.
	0.0439	Deg / Count

Figure 3.3 System parameters of servo SRV02

3.1.2.2 Rotary Flexible Link with Motor Servo SRV02

A straingage is mounted at the clamped end of a flexible link. The output is an analog signal which is proportional to the deflection of the link. This system is mounted on a motor servo plant (SRV2 in this project) to perform flexible link control experiments. The straingage is calibrated to give 1 volt per inch of the deflection at the tip.

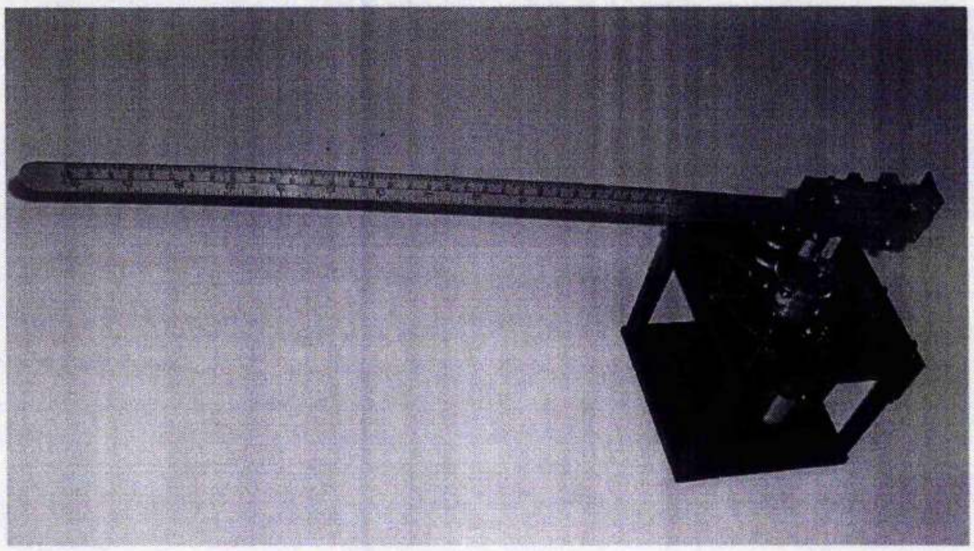
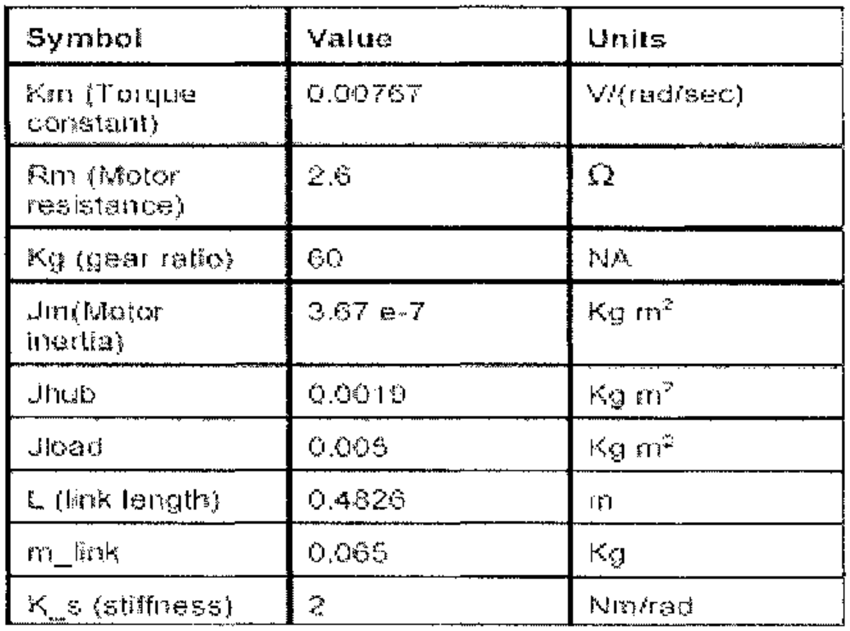


Figure 3.4 Rotary flexible link with motor servo SRV02

This control project involves positioning the flexible link to a set point using a feedback controller to damp out the vibration at the tip of the link as quickly as possible with minimal vibrations ([20]). The objectives of this project are:

- To obtain a linear state-space model for the Flexible Link module.
- To design a state feedback controller that damps out the vibrations at the tip of the beam.

System parameters of the flexgage module:



Symbol	Value	Units
Km (Torque constant)	0.00767	V/(rad/sec)
Rm (Motor resistance)	2.6	Ω
Kg (gear ratio)	60	NA
Jm(Motor inertia)	3.67 e-7	Kg m^2
Jhub	0.0019	Kg m^2
Jload	0.005	Kg m^2
L (link length)	0.4826	m
m_link	0.065	Kg
K_s (stiffness)	2	Nm/rad

Figure 3.5 System parameters of the flexgage module

3.1.2.3 Power Amplifier and Supply

A Quanser UPM1503 is the power module used in this project. The module is equipped with a 1-ampere +/- 12-volt regulated DC power supply for signal conditioning of external analog sensors.

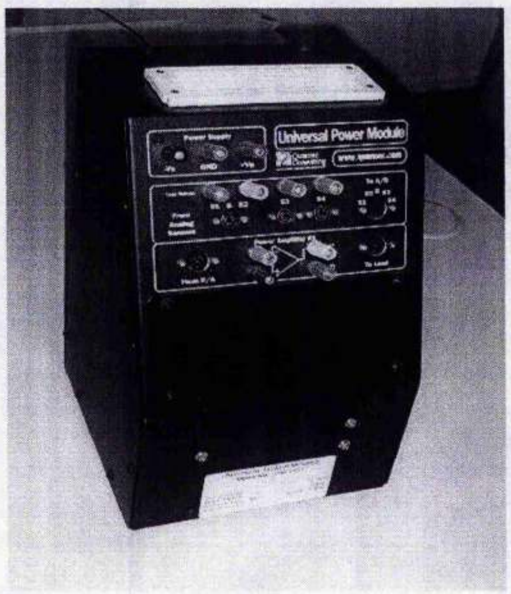


Figure 3.6 Power amplifier UPM1503

3.2 Software for Data Acquisition

3.2.1 Data Acquisition

Data acquisition is the most elementary work in this project. In control systems, the first and one of the most important tasks is acquiring data from the control plant accurately and promptly. The control signal will be sent back after comparing with desired setpoints with successful data acquisition. Data is transferred from the output interface of the plant to computer environment through analogue input channels and some kind of interface between computer environment and data acquisition card in the format which is recognizable to the computer.

3.2.2 COMEDI

Comedi is a Linux control and measurement device interface, its project develops open-source drivers, tools, and libraries for data acquisition ([11]). It includes:

- **Comedi**. A collection of drivers for a variety of common data acquisition plug-in boards. The drivers are implemented as a core Linux kernel module providing common functionality and individual low-level driver modules;
- **Comedilib. A** user-space library that provides a developer-friendly interface to Comedi devices. Included in the Comedilib distribution is documentation, configuration and calibration utilities, and demonstration programs;
- **Kcomedilib.** A Linux kernel module (distributed with Comedi) that provides the same interface as Comedilib in kernel space, suitable for real-time tasks. It is effectively a "kernel library" for using Comedi from real-time tasks.

Comedi has the following features:

- Integrated real-time support for most hardware;
- High-level library (comedilib);
- Application-level device independence;

- Works with Linux 2.0, 2.2, and 2.4 kernels.

The latest version (20/06/05) of Comedi and Comedilib are:

- comedi-0.7.70;
- comedilib-0.7.22.

Comedi designates a separate subdevice number to every subdevice in the board like analogue input, encoder, and a separate channel number for every channel residing in the same subdevice as well. The user can talk to whichever subdevice or channel by giving the proper number as parameter in Comedi function calls. For example, sending a voltage value out through analogue output (subdevice 1) channel 0 by `./outp -s 1 -c 0 xx (xx is the voltage value)`. The user can observe the information such as subdevice and channel numbers of the board by command `cat /proc/comedi`.

It should be noted that 2 different Comedi libraries are provided in the same release package: Comedilib and Kcomedilib. Both of them provide almost the same functionalities however they should be used in different cases. Comedilib is a user-space library and Kcomedilib is a Linux kernel module that provides the same interface as Comedilib in kernel space, whereas it should be used for real-time process only. Misusing of the library will cause errors in compilation.

3.2.3 Drivers for Data Acquisition Cards

3.2.3.1 Quanser Consulting MultiQ-3 Board Driver

The Comedi driver for MultiQ-3 board is provided with Comedi distribution, usually in file /$ComediDIR$/comedi/drivers/multiq3.c.

There is no necessity to put the whole driver program here however, it must be pointed out that there are 2 bugs in the original MultiQ3 driver provided with Comedi 0.7.66: One is in `multiq3_ai_insn_read()` function:

```
for(n=0;n<insn->n;n++){

    hi = inb(dev->iobase + MULTIQ3_AD_CS);
    lo = inb(dev->iobase + MULTIQ3_AD_CS);
    data[n] = ((hi << 8) | lo) & 0xfff;
```

This code results in the cutting off of minus input part and only 0 and positive voltage input can be read in. During the project it was modified to the following:

```
for(n=0;n<insn->n;n++){

    hi = inb(dev->iobase + MULTIQ3_AD_CS) &0xff;
    lo = inb(dev->iobase + MULTIQ3_AD_CS) &0xff;
    data[n] = (((hi << 8) | lo + 0x1000) & 0x1fff;
```

This ensures the whole range of the input is kept during reading-in.

Another bug resided in `multiq3_ai_insn_read()` function. The last line of the function `return i;` causes error messages as following in kernel space:

```
Oct 2 16:16:16 Ctrl6-PC kernel: BUG: result of insn!=insn.n
Oct 2 16:16:16 Ctrl6-PC last message repeated 1689 times
Oct 2 16:16:16 Ctrl6-PC kernel: BUG: result of insn!=insn.n
Oct 2 16:16:16 Ctrl6-PC last message repeated 122 times
Oct 2 16:16:16 Ctrl6-PC kernel: BUG: result in insn!=insn.n
Oct 2 16:16:16 Ctrl6-PC last message repeated in insn!=insn.n
Oct 2 16:16:16 Ctrl6-PC last message repeated 123 times
Oct 2 16:16:16 Ctrl6-PC kernel: BUG: result in insn!=insn.n
.........
```

Even thought it is just a simple consistency check and does not affect the acquisition, it can be easily fixed by changing `return i` into `return n`.

3.2.3.2 Humusoft AD512 Data Acquisition Card

The AD512 card is not in the supported hardware list of the Comedi distribution. A driver for the board was therefore written (see Appendix B).

3.2.4 Writing a New Driver in Comedi

To write a new driver in Comedi, several steps need to be followed:

1. Put the driver into `/$ComediDIR$/comedi/drivers/mydriver.c`
2. Edit `/comedi/config.in` and add a new "`dep_tristate`" line. Invent a meaningful name for the driver's variable.
3. Add a line with the name of new driver to `/comedi/drivers/Makefile`.

Each driver has to register two functions which are called when configuring and deconfiguring the DAQ board: mydriver_attach() and mydriver_detach. In mydriver_attach() function all properties of the device and subdevice and defined, mydriver_detach destroys all the settings and definition.

Instructions (insns) are low-level functions for accessing all kinds of channels. Drivers for digital inputs and outputs must have the following two functions ([32]):

insn_bits ()—Drivers set this if reading and writing multiple bits in a digital I/O subdevice at the same time is supported.

insn_config ()—Implements INSN_CONFIG instructions, used for configuring the direction of digital I/O lines.

Similarly, drivers for analogue inputs and outputs must implement the following two functions:

insn_read ()—Required for analogue inputs.

insn_write ()—Required for analogue outputs.

Several of tasks need to be done in the initialisation function of the driver ([32]):

- Announce that the hardware driver has begun initialisation by a printk("comedi %d: driver: ", minor);
- Check and request the I/O port region, IRQ, DMA, and other hardware resources. It is convenient here if user verifies the existence of the hardware and the correctness of the other information given;
- Fill in the comedi_device structure,
- Allocate user private data structure and subdevices;
- Set up each subdevice;
- Return 0, indicating success. If there were any errors along the way, the appropriate error number should be returned. In this case, the _detach function is called. The _detach function should check any resources that may have been allocated and release them as necessary. dev->subdevices and dev->private do not need to be freed in _detach as the comedi core does that.

Chapter 4 Graphical User Interface (GUI) and RTLab

RTLAB is one of the major software package required in this project which combines COMEDI with real time Linux to achieve real time control using the Linux operating system. Qt is adopted as the graphical interface which makes it possible to have additional custom windows within RTLab plugins. The introduction about RTLab and Qt, and a simple example that shows how Qt works in conjunction with RTLab is given in this chapter.

4.1 Graphical User Interface (GUI)

4.1.1 General Description of Qt

User interface plays a very important role in software development. There are different GUIs for different programming languages as well as different operating systems.

A comprehensive introduction to GUIs is given in http://www.geocities.com/SiliconValley/Vista/7184/guitool.html.

In this project, Qt is adopted as the graphical interface toolbox. Qt is a multiplatform, C++ application frame work that lets developers write one application which can run in different platforms such as Windows, Linux/Unix, Mac OS X, and so on ([33]).

Qt includes a rich set of Widgets (visual elements that are combined to create user interfaces) that provide standard GUI functionality. Signals and Slots are used to achieve inter-object communication. Qt also offers a conventional Event model to handle mouse clicks, key presses, and so on. The relationship between each element in Qt is in Figure 5.1:

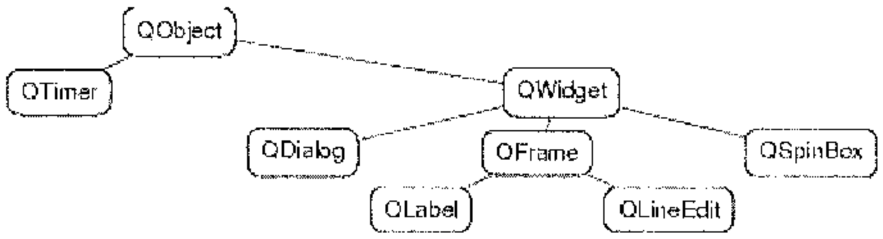
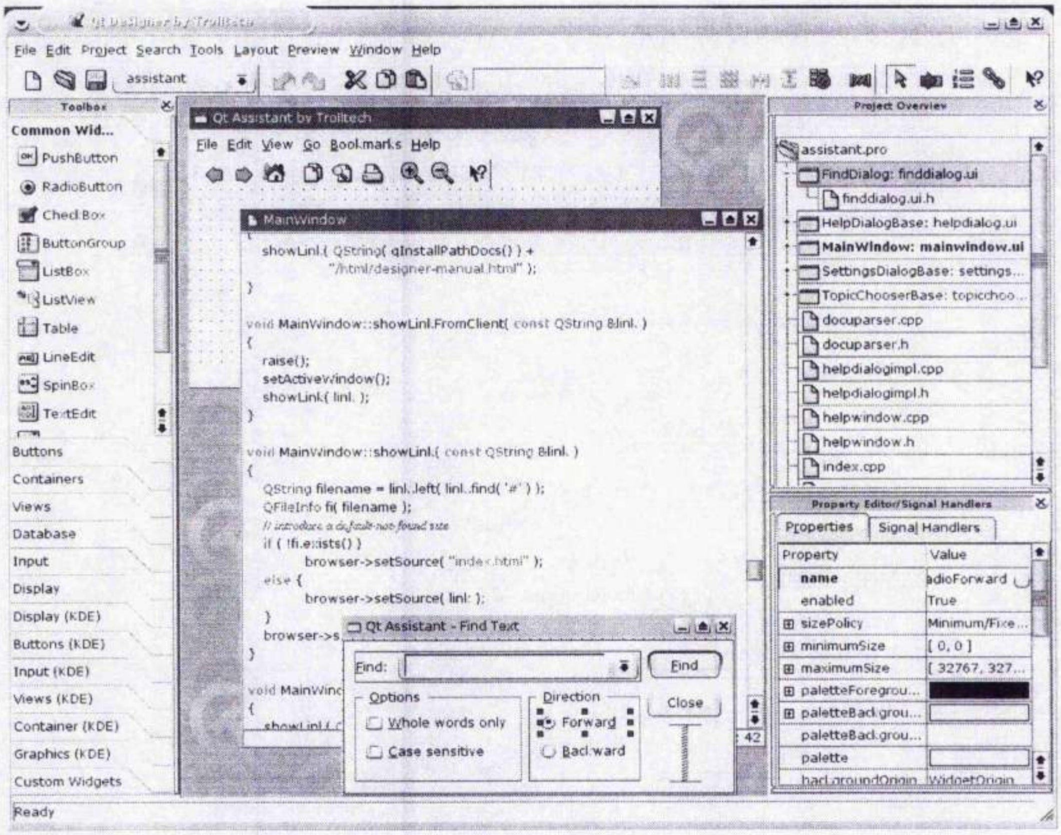


Figure 4.1: Elements in Qt

The QWidget class is the base class of all user interface objects. It receives events like mouse, keyboard from the window system, and paints on the screen. QObject, QDialog, QLabel, QFrame, QLineEdit, and QSpinBox are all the classes which can be used for a QWidget and each class has its own function. For instance, QLabel provides a text or image display, QLineEdit is a one-line text editor, and the QSpinBox class provides a pop-up menu.

To make programming simpler and more convenient, Qt provides Qt Designer, a graphical designing tool for user interfaces. It allows users to build interfaces with layout tools that move and scale widgets automatically at runtime, and generates code with its built-in code editor.



```
    showLink( QString( qInstallPathDocs() ) +
            "/html/designer-manual.html" );
}

void MainWindow::showLinkFromClient( const QString &link )
{
    raise();
    setActiveWindow();
    showLink( link );
}

void MainWindow::showLink( const QString &link )
{
    QString filename = link.left( link.find( '#' ) );
    QFileInfo fi( filename );
    // introduce a default-not-found site
    if ( !fi.exists() )
            browser->setSource( "index.html" );
    else {
            browser->setSource( link );
    }
    browser->s
}

void MainWinc
{
    showLink( c
```

Property	Value
name	adioForward
enabled	True
sizePolicy	Minimum/Fixe...
minimumSize	[0, 0]
maximumSize	[32767, 327...
paletteForegrou...	■
paletteBackgrou...	□
paletteBackgrou...	
palette	□
backgroundOrigin	WidgetOrigin

Figure 4.2 A screenshot of Qt Designer

Qt is a very powerful and convenient tool in GUI designing. In this project, Qt works together with RTAI and COMEDI in RTLab. More details will be given later in the thesis.

4.1.2 A Simple Qt Example

Here is a simple "hello world" example. It shows the basic rule by which a Qt application is working.

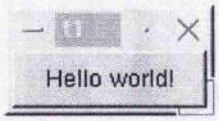


Figure 4.3 A snapshot of "hello world" example

```
/****************************************************************
**
** Hello World Example
**
****************************************************************/

#include <qapplication.h>
#include <qpushbutton.h>


int main( int argc, char **argv )
{
    QApplication a( argc, argv );

    QPushButton hello( "Hello world!", 0 );
    hello.resize( 80, 25 );

    a.setMainWidget( &hello );
    hello.show();
14      return a.exec();
15  }
```

The first 2 lines include the QApplication and QPushButton class definitions. There has to be exactly one QApplication object in every application. QPushButton is a standard GUI push button that the user can press and release. There are a series of classes available in Qt such as QSlider, QLabel, QLineEdit, and QComboBox besides QPushButton.

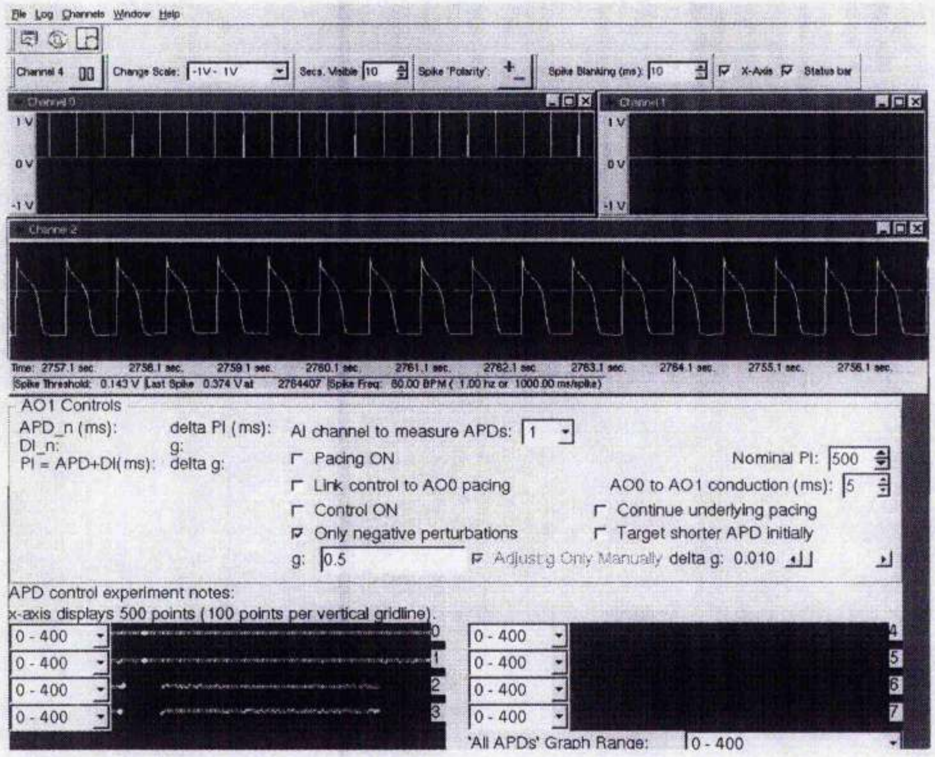
Line 9 and line 10 define the content and size of the QPushButton which will appear in the graph of this QApplication.

Lines 12 to 14 set up and show the main widget. A widget is a user interface object that can process user input and draw graphs. In this example, A QPushButton is all that in the main widget. One thing to be noted, line 13 is essential because a widget is never visible until show() is called.

4.2 RTLab

4.2.1 General Description of RTLab

RTLab is an ongoing project to develop a general-purpose, open-source, hard real-time experiment interface software system ([19]). All applications in RTLab exist in the form of RTLab plugin modules. The following graphical interface will appear after invoked RTLab by entering "./daq_system" in the RTLab source directory and the plugin module loaded:



```
Channel 4   Change Scale: -1V - 1V   Secs. Visible 10   Spike 'Polarity': +_   Spike Blanking (ms): 10   ☑ X-Axis ☑ Status bar
Time: 2757.1 sec.  2758.1 sec.  2759.1 sec.  2760.1 sec.  2761.1 sec.  2762.1 sec.  2763.1 sec.  2764.1 sec.  2755.1 sec.  2756.1 sec.
Spike Threshold: 0.143 V  Last Spike 0.374 V at 2764407  Spike Freq: 60.00 BPM ( 1.00 hz or 1000.00 ms/spike)
AO1 Controls
APD_n (ms):          delta PI (ms):   AI channel to measure APDs: 1
DI_n:                g:
PI = APD+DI(ms):     delta g:         ☐ Pacing ON                          Nominal PI: 500
                                      ☐ Link control to AO0 pacing         AO0 to AO1 conduction (ms): 5
                                      ☐ Control ON                         ☐ Continue underlying pacing
                                      ☑ Only negative perturbations        ☐ Target shorter APD initially
                                      g: 0.5     ☑ Adjust g Only Manually  delta g: 0.010
APD control experiment notes:
x-axis displays 500 points (100 points per vertical gridline).
0 - 400  0      0 - 400  4
0 - 400  1      0 - 400  5
0 - 400  2      0 - 400  6
0 - 400  3      0 - 400  7
'All APDs' Graph Range: 0 - 400
```

Figure 4.4: RTLab graphical interface

The top half of the graph shows the signals echoed back from analogue input channel 0, 1 and 2, generally this part is the same to every plugin. The bottom half is the plugin window in which the control parameters can be set, and all the contents in this window are decided by the plugin program.

There are several files in RTLab source directory such as: plugin.h, plugin_score.h, plugin_scanner.h, plugin_scanner.cpp, and so on. These files work together with certain pre-defined plugin mechanism and some variables in daq_system files, to ensure that the user plugins can be recognized automatically so long as these plugins were compiled correctly and placed in /$RTLABDIR$/plugins.

Generally, an RTLab plugin is composed of 4 files:

- myplugin.c—Module initialisation and cleanup functions. Shared memory initialisation function, process read function, and control algorithm (usually do_control function) are given in this file. It is the major control implementation part of the plugin.
- myplugin.h—Callback frequency, macros of maximum and minimum values of control signal (in volts), and shared memory structure is defined and declared in this file.
- myplugin.cpp—the main entry of user plugin graphical interface, user plugin constructor, destructor, widgets, signals and slots are declared and called in this file.
- myplugin_private.h – mainly used for user plugin widget declaration.

Each plugin must contain the following symbols ([12]):

- "ds_plugin_ver" int value, indicating the version of the plugin engine this plugin was written for. Must equal DS_PLUGIN_VER pre-processor symbol defined in this file;
- "entry". The entry function of type plugin_enrty_fn_t, should return a valid Plug-in * reference;
- "name". The const char * friendly name of the plugin.

Any plugin missing the above symbols will fail to load.

Optional symbols:

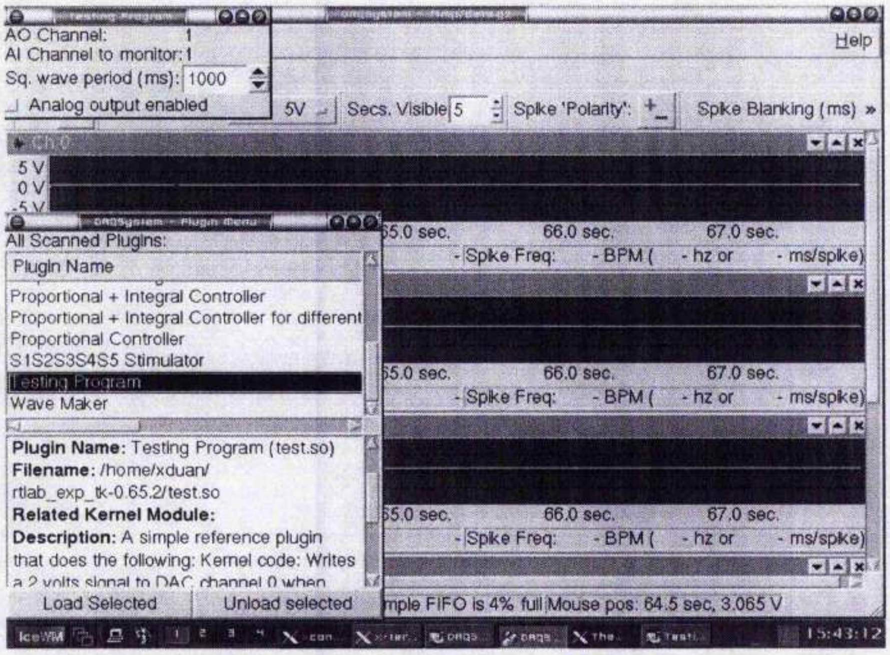
- "flags" int value, indicating what flags this plugin has set;
- "description" const char *, a brief description of the plugin's functionality;
- "author" const char *, authors of this plugin;
- "requires" const char *, a descriptive string explaining what is required to properly load this module.

Shared memory is used to achieve inter-process communication in RTLab.

In this project, the plugin window needed to be altered to show the plant's working status, and this will be detailed in Chapter 6.

4.2.2 How Does Qt Work in Conjunction with RTLab

Here is a simple example that shows how Qt works in conjunction with RTLab:


AO Channel: 1
AI Channel to monitor: 1
Sq. wave period (ms): 1000
Analog output enabled

All Scanned Plugins:

Plugin Name
Proportional + Integral Controller
Proportional + Integral Controller for different
Proportional Controller
S1S2S3S4S5 Stimulator
Testing Program
Wave Maker

Plugin Name: Testing Program (test.so)
Filename: /home/xduan/rtlab_exp_tk-0.65.2/test.so
Related Kernel Module:
Description: A simple reference plugin that does the following: Kernel code: Writes a 2 volts signal to DAC channel 0 when

mple FIFO is 4% full Mouse pos: 64.5 sec, 3.065 V

Figure 4.5 A snapshot of test plugin windows in RTLab

test.c (see Appendix C for source code):

```
/* This file is a part of an example showing how Qt works in
conjunction with RTLab
 * Copyright (C) 2004 Xiaoyu Duan
 */
/**
 * Example RTLab plugin -- Kernel side.
 * This plugin does the following:
 * Kernel code:
 *   Writes a 2 volts signal to DAC channel 0 when receiving a signal
```

```
of any value but 0 from ADC Channel 0.
 *
 * GUI (test.cpp):
 *     Simple GUI to change AI online.
 */
#include ..
#define ..

  Set module author
  Set module description

int ..;

module_init(init);
module_exit(cleanup);

/*----------------------------------------------------------------------
  Some private 'global' variables...
--------*/
static ..;
static const int ..;
static struct ..;
/*----------------------------------------------------------------------
--------*/

int init (void)
{
  Consistency check

  Register callback

   shared memory

  Set callback rate

   the rtlab_comedi_context convenience struct

  Turn callback on
}

void cleanup (void)
```

```
51  {
52    Deactive and unregister do_control function
53    Detach shared memory
54  }
55
56  static int init_shared_mem(void)
57  {
58    Set shared memory structure
59  }
60
61  static int proc_read (char *page, char **start, off_t off, int count,
62                        int *eof,  void *data)
63  {
64    Set the information of each element in shared memory
65  }
66  /**
67   * This function does the following:
68   *   Kernel code:
69   *   Writes a 2 volts signal to DAC channel 1 when receiving a signal
70  of any value but 0 from ADC Channel 1.
71   *
72   *  This function is called by rtlab's core... see
73  rtp_register_function()
74   */
75  static void do_control (MultiSampleStruct * m)
76  {
77    double ..;
78    SampleStruct ..;
79
80    Get value from DAQ device
81
82    Calculate output voltage value
83
84    Return value to DAQ device
85  }
```

Lines 1 to 33 are the variable and constant declarations and the explanation for the program.

Lines 35 to 48 are the initialisation part for the module. Each kernel module must have an "initialisation" part and a "cleaning up" part in which the working environment for the module may be d and cleared when the module is inserted into and removed from the kernel. The scheduler is set up inside the "initialisation" part in the program.

Lines 56 to 59 the shared memory structure which accomplishes the communication between real time processes and non-real time processes.

Lines 61 to 65 set the information of each element in shared memory structure.

Lines 75 to 85 do the control task–output a 2 volts signal to the control plant.

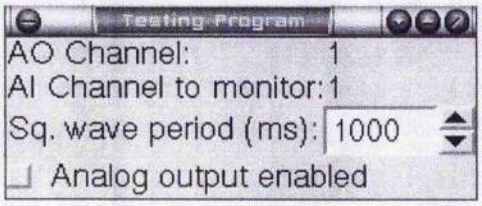
test.cpp (see Appendix C for source code):

```
1   /*
2    * This file is a part of an example showing how Qt works in
3   conjunction with RTLab
4    *
5    * Copyright (C) 2004 Xiaoyu Duan
6    */
7   #include ..
8   #define ..
9
10  extern "C" {
11
12    Set some information needed by plugin
13
```

```
  Plugin * entry(QObject *o)
  {
   and show the plugin widget
  }

};

Test::Test(DAQSystem *d)
  : QWidget(d, PLUGIN_NAME, Qt::WType_TopLevel), ds(d)
{
  Attach to Shared memory

  Create plugin widget variable

  Build up plugin widget buy calling element "buildGUI"

  Connect signals with slots

  Set caption
}

Test::~Test()
{
  Detach shared memory

  Delete widget
}

void Test::buildGUI()
{
  Create layout variable

  Set the number of AI channels

  Set subwidgets and add them to the layout
}

void Test::connectSignals()
{
  Connect signals with slots
}
```

```
55
56      Set parameters in the shared memory region by slots
```

This cpp file is the program actually written using Qt to create the GUI within RTLab. It adds the following window into the RTLab graphical interface.



```
Testing Program
AO Channel:               1
AI Channel to monitor: 1
Sq. wave period (ms): 1000
[ ] Analog output enabled
```

Figure 4.6 Plugin window of "testing program" example

After some definitions and declarations, lines 15 to 18 define the “entry” party of the plugin in which the main Qwidget is created and displayed.

Lines 21 to 33 are the “constructor” of the plugin widget. A constructor is a standard Qt widget constructor that builds up everything needed for the widget. The test Qwidget is set to top-level widget here.

Lines 35 to 40 are the “destructor” which does the tidying up work when the widget is no longer needed.

Lines 42 to 49 set up the graphical window as Figure 4.6 shows. There are 7 elements in this plugin window: QLabel “AO Channel:”, “1”, “AI Channel to monitor”, “1”, “Sq. wave period (ms):”, QSpinBox “1000”, and QCheckBox “Analog output enabled".

Lines 51 to 54 connect the widgets with correspondent signals.

Lines 56 set the parameters in the shared memory region by slots.

Here are the 2 header files needed for this plugin:

test.h

```
/* Xiaoyu Example plugin - The kernel side defs.. */
/* This file is a part of an example showing how Qt works in
conjunction with RTLab
 *
 * Copyright (C) 2004 Xiaoyu Duan
 */

#ifndef TEST_H
#  define TEST_H

#include "rtlab.h"

#ifdef __cplusplus
extern "C" {
#endif

/** Callback frequency -- basically the granularity of our
monitoring.. */
#define TEST_CALLBACK_FREQUENCY_HZ 1000

/*   Max and Min values of control signal (V) */
#define MAX_OUT 5.0
#define MIN_OUT -5.0

/** The shared memory */
26    struct TestShm {
27      int     magic;          /*< Should always equal TEST_SHM_MAGIC    */
28
29      volatile                /** R/W value                             */
30      int     period_milliseconds; /* The period                      */
31
32      volatile                      /** R/W value                       */
33      char    wave_on;        /* if nonzero, do the actual output       */
34
35      int     reserved[4];    /* just so i can look like i know what i am
36    doing... */
37    };
38
39    #ifndef __cplusplus
40      typedef struct TestShm TestShm;
41    #endif
42
43    #define TEST_SHM_NAME "Test Shm"
44    #define TEST_SHM_MAGIC (0xf0015555) /*< Magic no. for shm...
45    'foolzzzz'    */
46
47    #ifdef __cplusplus
48    }
49    #endif
50
51    #endif
```

Lines 8 to 23 are some constant definitions are consistency checks. Lines 26 to 37 defined a structure of the shared memory variables which will be used in test.c and test.cpp.

test_private.h

```
1     /*
2      * This file is a part of an example showing how Qt works in
3     conjunction with RTLab
4      * Copyright (C) 2004 Xiaoyu Duan
5      */
```

```
#ifndef _TEST_PRIVATE_H
#define _TEST_PRIVATE_H

#include <qwidget.h>
#include "plugin.h"

class DAQSystem;
struct TestShm;
struct TestWidgets; /* struct to store all the widgets this class has
                              -- forces header files into .cpp file
*/

class Test: public QWidget, public Plugin
{
  Q_OBJECT

 public:
  Test(DAQSystem *d);
  ~Test();

  const char *name() const; /* overrides Plugin parent class  */
  const char *description() const; /* overrides Plugin parent class
*/


 private: /* methods */
  void buildGUI();
  void connectSignals();

 private slots: /* A Qt-ism -- these methods set variables in the SHM
*/
  void setPeriod(int);
  void setAO(bool);

 private: /* data */
  DAQSystem *ds;
  TestShm *shm;
  TestWidgets *widgets;
};

#endif
```

Lines 18 to 44 defined the Qwidget that will be used in graphical interface. Users are able to talk to the control program by this Qwidget.

Chapter 5 Discussion of Real Time Solutions

There are two different approaches which provide Linux with real time performance, giving the Linux kernel pre-emption ability and adding a new software layer beneath the Linux kernel with full control of interrupts.

For the first approach, TimeSys (website: http://www.timesys.com/) and Linux kernel pre-emption project (website: http://sourceforge.net/projects/kpreempt/) is the software available, and for the second one, the most commonly used approach is RTLinux or RTAI, which has been introduction in previous chapters.

Both RTAI and RTLinux have similar scheduling, inter-task communication methods, and API functions. Although from an academic point of view, RTAI and RTLinux are the same, user can use and modify the code at will, there are still some differences between the two ([28], [29]):

The RTAI team tries to allow proprietary development for zero price (by using LGPL license) while RTLinux does not.

RTAI is based on the GPL mode and it is impossible to change the license without the permission of all the code copyright holders. In other words, work derived from GPL code cannot be released under a different license.

Real Time Linux was developed to provide Linux with Real Time functionalities in its early days. As a variant of RTLinux, RTAI appeared later however is supported better today, and it works better with COMEDI, thus why RTAI is selected for this project.

There are several schemes with which COMEDI can be integrated with RTAI:

- rt_com_lxrt.
- rtai_comedi_lxrt
- RTAI-Lab
- RTLab

5.1 rt_com_lxrt

This is a somewhat out-of-date solution which is integrated in RTAI-2.4.0-0.26. rt_com_lxrt strengthens the RTAI approach of symmetric usage of all services in kernel-user space for soft-hard real time. The files for rt_com_lxrt could be found in the directory /$RTAIDIR$/rt_com_lxrt/. rt_com also has a version number of its own. The rt_com package can be easily linked to user space applications by including rt_com_lxrt in the files using it. To use rt_com_lxrt, several modules like rtai, rtai_sched, rt_com, lxrt have to be inserted into the kernel, and it is required to "make" rt_com_lxrt before using.

Here is an example extracted from rt_com_lxrt directory:

```
/* rt_com-LXRT test
 * ================
 *
 * RT-Linux kernel module for communication across serial lines.
 *
 * Adaptation of rt_com test modules to provide the same examples in
 * RTAI environment using LXRT.

 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>

#define KEEP_STATIC_INLINE
#include <rtai_lxrt_user.h>
#include <rtai_lxrt.h>

#include "rt_com_lxrt.h"
#include "rt_com.h"

int main(int argc, char **argv)
{
      unsigned long testcomtsk_name = nam2num("TESTCOM");
      RT_TASK *testcomtsk;
      char hello[] = "Hello World\n\r";
      int retval = 0;

32          mlockall(MCL_CURRENT | MCL_FUTURE);
33
34          if (!(testcomtsk = rt_task_init(testcomtsk_name, 1, 0, 0))) {
35                printf("CANNOT INIT MASTER TASK\n");
36                exit(1);
37          }
38          rt_set_oneshot_mode();
39          start_rt_timer(0);
40          //rt_make_hard_real_time();
41          // This example use the rt_com_setup() on port 0 so it needs
42          // that you have compiled rt_com.c with rt_com_table[0].used=1
43          // otherwise the rt_com_setup() fails
44
45          if( rt_com_setup( 0, 9600, RT_COM_NO_HAND_SHAKE,
46    RT_COM_PARITY_NONE, 1, 8, -1 ) < 0 ) {
47                printf("hello_world_lxrt: error in rt_com_setup()\n");
48              retval = 1;
49          } else {
50              rt_com_write( 0, hello, sizeof( hello ) );
51              rt_sleep(nano2count(500000000));
52              printf("rt_com_lxrt test: >>%s<< sent.\n", hello );
53              rt_com_setup(0, -1, 0, 0, 0, 0, 0);  // release port
54              printf("rt_com_lxrt test: finished\n");
55          }
56          //rt_make_soft_real_time();
57          stop_rt_timer();
58          rt_task_delete(testcomtsk);
59          exit(retval);
60    }
```

There is only one function—main() in this example. Lines 27 to 39 some variables and the real time task. Lines 45 to 55 do the task which prints a word "hello" out on the screen on success. User can have the task running either in soft real time all the way or in hard real time first and then back in soft real time by adding corresponding script sentences as in line 40 and 56.

5.2 rtai_comedi_lxrt

This is a porting of COMEDI to LXRT/NEWLXRT which is done by using the standard extension feature of LXRT/NEWLXRT. rtai_comedi_lxrt makes COMEDI symmetrically usable in kernel and user space within RTAI, in soft and hard real time. As with LXRT/NEWLXRT and its extensions it is possible to use rtai_kcomedi_lxrt both using static inlines (by adding "define KEEP_STATIC_INLINES" before including rtai comedi_lxrt.h) and using libkcomedi.a found in directory comedi_lxrt/lib. Two simple examples called testa.c and tests.c are available in /$RTAIDIR$/comedi_lxrt/ and an application using this approach will be detailed in Chapter 6.1.

5.3 RTAI-Lab

RTAI-Lab is a tool that allows the use of any set of real time controllers/simulators automatically generated by Matlab/Simulink/RTW. Release 2.24.11 of RTAI allows the integration of COMEDI drivers in Simulink ([8]) schemas and to generate code for (x)RTAI-Lab. If there is no Matlab/Simulink/RTW available, the same thing could also be done under Scilab/Scicos. In this case however an unofficial release of RTAI (at least 2.24.12pre1) is needed, and two files rt_scilab.tgz and xrtailab.pp are also required.

The basic concept of RTAI-Lab is to allow any couple of separated systems, the host and the target, to communicate with each other ([24]). In a distributed implementation, the host is the machine where the generated hard real time codes run, it sends/receives messages, requests the target to accept parameters. The host and the target could be the same machine.

5.4 RTLab

As introduced in Chapter 4, RTLab is an ongoing project to develop a general-purpose, open-source, hard real-time experiment interface software system ([19]). Comedi, RTLinux or RTAI and Qt are integrated in this rapidly developing project and user applications reside in the form of plugins for RTLab. The detailed applications will be given in Chapter 6.

5.5 Real Time Control Solutions

There are several approaches to setting the scheduler up, either in kernel space or user space, either in hard real time or soft real time. Scheduling in kernel space means setting up the scheduler in the initialisation part of the module therefore the scheduling is set right after the module is inserted into the kernel, the tasks then run in the user space repeatedly according to the tick period. Scheduling in user space means setting up the scheduler in the main part of the program rather than in the initialisation part. The system will wait for a fixed interval when the task finishes. That is to say, the whole tick period may be different each time. Hard-real time means giving a Linux process, or a pthread (POSIX thread) hard real time execution capabilities allowing full kernel pre-emption ([25]). Running in soft-real time means giving processes standard Linux behaviours. The kernel allows other processes to run at the same time, and this is forbidden in hard real time.

Figure 5.1 and Figure 5.2 show the difference between the scheduling in kernel space and scheduling in user space.

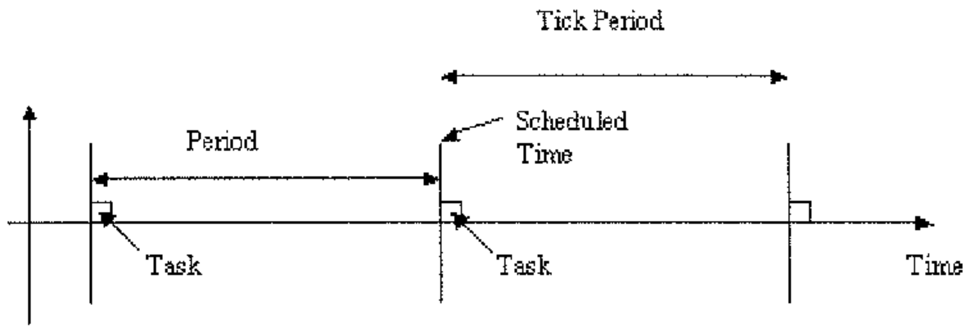


Figure 5.1: Scheduling in kernel space

In Figure 5.1, the task executes within the fixed tick period and it will repeat no matter whether the task has finished or not when period ends. Generally however, the scheduled period is long enough for the task to run.

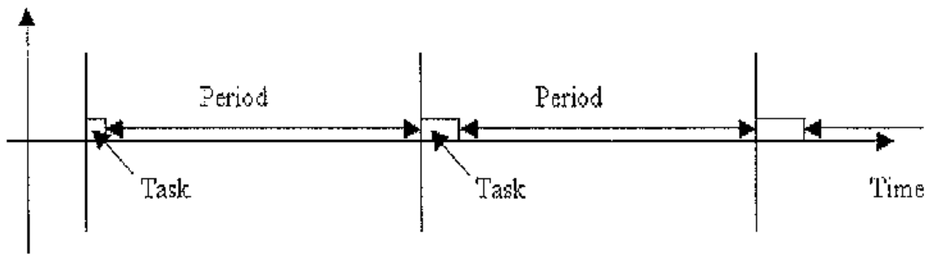
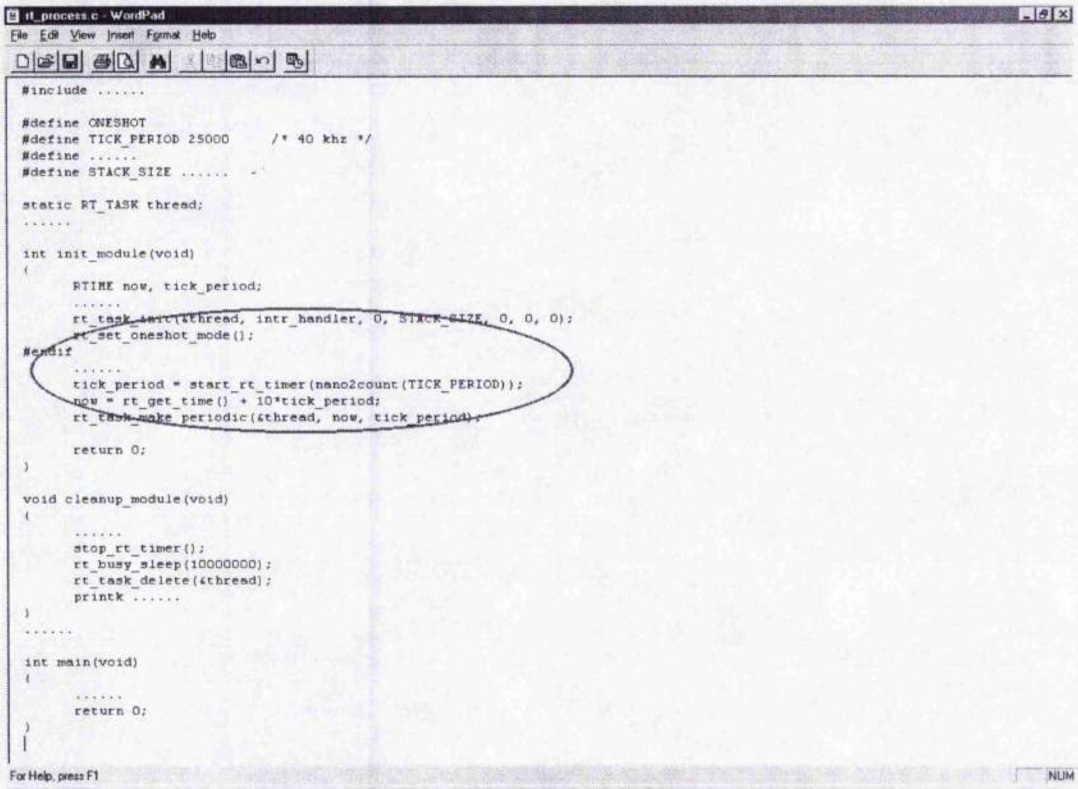


Figure 5.2: Scheduling in user space

In Figure 5.2, the system waits for a period after the task finishes. The period is fixed but the time for the task to run may be unpredictable.

In COMEDI + RTAI schemes, the scheduler may be set up in the following 4 ways:

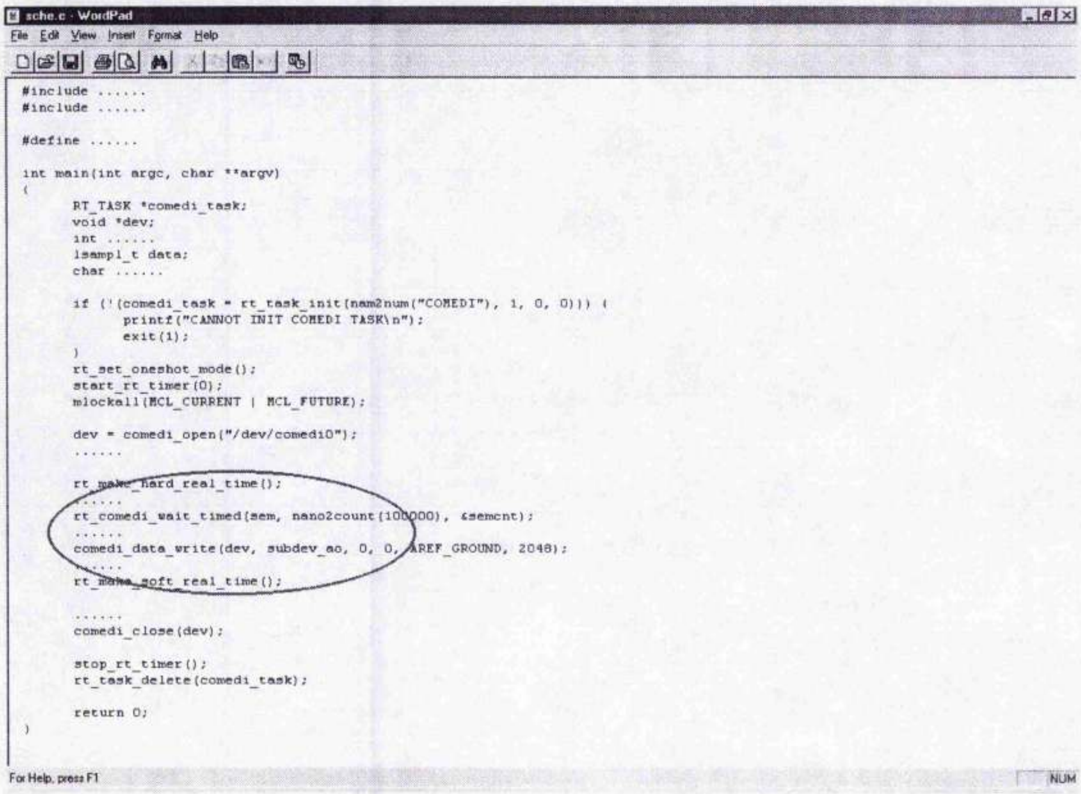
1) Setting up scheduler in "module initial" part for RT scheduling, do not use LXRT extensions, which means, scheduling in kernel space (Figure 5.3).



```
#include ......

#define ONESHOT
#define TICK_PERIOD 25000     /* 40 khz */
#define ......
#define STACK_SIZE ......

static RT_TASK thread;
......

int init_module(void)
{
      RTIME now, tick_period;
      ......
      rt_task_init(&thread, intr_handler, 0, STACK_SIZE, 0, 0, 0);
      rt_set_oneshot_mode();
#endif
      ......
      tick_period = start_rt_timer(nano2count(TICK_PERIOD));
      now = rt_get_time() + 10*tick_period;
      rt_task_make_periodic(&thread, now, tick_period);

      return 0;
}

void cleanup_module(void)
{
      ......
      stop_rt_timer();
      rt_busy_sleep(10000000);
      rt_task_delete(&thread);
      printk ......
}
......

int main(void)
{
      ......
      return 0;
}
```

Figure 5.3 Setting up scheduler in "module initial" part

2) Use LXRT extension with hard-real time process (Figure 5.4).



```
#include ......
#include ......

#define ......

int main(int argc, char **argv)
{
      RT_TASK *comedi_task;
      void *dev;
      int ......
      lsampl_t data;
      char ......

      if (!(comedi_task = rt_task_init(nam2num("COMEDI"), 1, 0, 0))) {
            printf("CANNOT INIT COMEDI TASK\n");
            exit(1);
      }
      rt_set_oneshot_mode();
      start_rt_timer(0);
      mlockall(MCL_CURRENT | MCL_FUTURE);

      dev = comedi_open("/dev/comedi0");
      ......

      rt_make_hard_real_time();
      ......
      rt_comedi_wait_timed(sem, nano2count(100000), &semcnt);
      ......
      comedi_data_write(dev, subdev_ao, 0, 0, AREF_GROUND, 2048);
      ......
      rt_make_soft_real_time();

      ......
      comedi_close(dev);

      stop_rt_timer();
      rt_task_delete(comedi_task);

      return 0;
}
```

Figure 5.4 Setting up scheduler with LXRT extension (hard real time)

3) Use LXRT extension with soft-real time process.

4) Non-real time Loop—without timing.

In approach (1), a scheduler is set up in "module initial" part and the whole process runs in kernel space at a fixed time interval, as the approach shown in Figure 5.1. A "module initial" is a function in the module program by which the environment variables are d when the module is inserted into the kernel. The system ensures that the process run at an exact time period and allocates the rest of time for other tasks.

As in Figure 5.3, the scheduler is set up by "`rt_task_make_periodic()`" statement in function "`init_module()`", and the tick period is set to 25 milliseconds.

In approach (2), it adopts the real time solution shown in Figure 5.2, the process runs in an unknown time period which is relatively very short, then waits for an exact tick period. This may cause a small time error which might be ignored however. As shown in Figure 5.4, the process is set to run in hard real time by calling "`rt_make_hard_real_time()`" and can be set back to run in soft real time at any time by calling "`rt_make_soft_real_time()`".

Approach (3) is almost the same as approach (2) except that the process runs in soft-real time rather than hard-real time.

Approach (4) is the simplest method however it provides the worst performance. A user can neither assign the process an exact time interval nor keep the kernel waiting after the loop is finished. The program execution will fall into an infinite loop, and the system will crash when out of resource. In real time control the control accuracy and promptness are highly required which means this approach could never be used in this project.

Chapter 6 Applications

In this chapter, four real time control applications will be discussed. One simple program combining COMEDI with RTAI LXRT without graphical interface, and three applications in the form of RTLab plugins in which different controllers are used according to the control complexity. Meanwhile the plugin windows have also been altered to meet the requirement discussed in Chapter 4.

6.1 A Simple COMEDI + RTAI LXRT Program

Following is a simple control program that combines COMEDI with RTAI LXRT by rtai_comedi_lxrt which was introduced in Chapter 5. The application is without GUI and all the inputs are via the command line mode. A motor servo is controlled to reach the angular position designated by the user via DAQ board (either MultiQ3 board or AD512 card).

6.1.1 Description

As introduced in Chapter 4, the rotary position servo consists of a DC servomotor and a built-in gearbox whose ratio is 14 to 1. The output of the gearbox drives a potentiometer and an independent output shaft to which a load can be attached. A controller will be implemented to control the position of the output shaft, as shown in Figure 3.2.

6.1.2 Mathematic Model

The model is derived from the basic equations of a DC motor ([18]):

V_{in}	Input voltage
I_m	Motor current
R_m	Motor resistance
K_m	Torque constant
ω_m	Angular velocity of motor shaft
θ	Angular position of output shaft
θ_d	Desire angular position of output shaft
ω_l	Angular velocity of output shaft
K_g	Gear ratio

Electrically:

$$V_{in} = I_m R_m + K_m \omega_m \tag{6.1}$$

$$= I_m R_m + K_m K_g \omega_m \tag{6.2}$$

Mechanically:

T_m	Torque generated by motor
T_o	Torque at the output after the gearbox
J_m	Motor inertia
J_l	Load inertia

$$T_0 = K_g T_m = K_g (J_m \dot{\omega}_m + J_l \frac{\dot{\omega}_l}{K_g}) \tag{6.3}$$

$$= J_m K_g^2 \dot{\omega}_l + J_l \dot{\omega}_l \tag{6.4}$$

$$= \dot{\omega}_l (J_m K_g^2 + J_l) \tag{6.5}$$

But $T_m = K_m I_m$, then

$$I_m = \frac{T_m}{K_m} = \frac{T_o}{K_m K_g} = \dot{\omega}_l \frac{K_g^2 J_m + J_l}{K_m K_g} = \dot{\omega} \frac{J_{eq}}{K_m K_g} \tag{6.6}$$

$J_{eq} = K_g^2 J_m + J_l$ is the equivalent inertia seen at the output of the gearbox.

Then Laplace transform is applied (s is the Laplace operator):

$$\frac{\theta(s)}{V_{in}(s)} = \frac{1}{s(s\frac{R_m J_{eq}}{K_m K_g} + K_m K_g)} \tag{6.7}$$

with the given parameters:

$$\frac{\theta(s)}{V_{in}(s)}=\frac{1}{s(0.0026s+0.1081)} \tag{6.8}$$

6.1.3 Control System Design

According to the transfer function, it is obvious that the plant is unstable due to the pole at the origin. A proportional plus derivate controller is selected in this case:

$$V_{in}=K_p(\theta_d-\theta)-K_d\dot{\theta} \tag{6.9}$$

The feedback gain K_p and K_d are used to design the response of the system. Substituting the Laplace Transform of Equation 6.9 into Equation 6.8 yields:

$$\frac{\theta}{\theta_d}=\frac{K_p}{0.0026s^2+(0.108+K_d)s+K_p} \tag{6.10}$$

If the required rise time is 100 milliseconds and let the damping ratio equal 0.6. The desired characteristic polynomial is:

$$s^2+2\zeta\omega_0 s+\omega_0^2 \tag{6.11}$$

And $t_p=\dfrac{\pi}{\omega_0\sqrt{(1-\zeta^2)}}=0.1$, $\zeta=0.6$

Solving for K_p and K_d results in:

$$K_p = 0.0026\omega_0^2 = 4\,,\ K_d = 0.0052\zeta\omega_0 - 0.108 = 0.015$$

6.1.4 Real Time Control Program in Linux

There are two source programs in this application: mainpro.c and mailbox.c. mainpro.c (see Appendix C) is for output, it does filtering, calculation and sends control signal to the plant. mailbox.c ((see Appendix C)) is for input, it asks the user for the desired angular position value, then sends the value to mailbox. The mailbox is in charge of the communication between these two programs in real time.

`Mainpro.c` (see Appendix C for source code)

```
#include ..
#define ..

double ..;
int ..;
char *subdevice_types[] = {
            ..
            ..
};

double control_algrithm(double volts)
{
double ..;

Convert binary value to actual voltage value

Save previous filtered output

Save old sample
```

```

 filtering

 Compute output voltage value

 Convert it back to binary value

 Return voltage value;

  }

 int main(int argc, char **argv)
 {
 RT_TASK ..;
 int ..;
 double ..;
 lsampl_t ..;
 SEM ..;

 Set mailbox name  /* use mail box to achieve Interprocess
communication here */

 Set scheduling priority

 Recognize mailbox

 Get DAQ device information

 Start control

 Receive value from mailbox

 data = control_algrithm(data); /* Computation*/

 Return value to DAQ device

 Cleanup
}
```

This program consists of three main blocks. These are definitions and declarations (lines 1-9), control algorithm (lines 11- 29), and the main program (lines 31 to 57). It is fairly straightforward—initialise constants, sample the sensor voltage from A/D, convert voltage to desired unit (degree), filter and numerically differentiate data, then compare the angle value acquired with the desired setpoint, calculate the required voltage and output to D/A interface.

Following is the companion program of mainpro.c (mailbox part).

Mailboxpro.c (see Appendix C for source code):

```
#include ..
#define ..

int main(int argc, char **argv[])
{
  Set task name
  Set mailbox name

  double ..;
  int ..;
  RT_TASK ..;
  MBX ..;

  Set scheduling priority

   real time task

  Set timer

  Create mailbox

22    count=5; /* It is possible to enter desired value for 5 times. */
23
24    while(count){
25
26    Send value to mailbox
27
28    Delay for a short period
29
30      count--;
31    }
32
33    Cleanup;
34    }
```

Lines 6 to 20 d the real time task, mail box, and set up the scheduler. Once the mainpro starts to run, it checks the devices information and searches the value in mailbox. Meanwhile when the mailboxpro runs, it prompts the user for the desired angle value, and then sends it to the real time process. The user is allowed to enter desired a value 5 times. The system will prompt for the next input after 2 seconds, this is accomplished in lines 22 to 31.

The following commands are required to the environment for running the application.

```
run:
sync
/sbin/insmod  ../modules/rtai.o
/sbin/insmod  ../modules/rtai_sched.o
/sbin/insmod  ../modules/rtai_lxrt.o
/sbin/insmod  ../modules/rtai_shm.o
```

```
6    /sbin/insmod  /lib/modules/'uname -r'/COMEDIDIR/comedi/comedi.o
7    /sbin/insmod  /lib/modules/'uname -
8    r'/COMEDIDIR/kcomedi/kcomedilib.o
9    /sbin/insmod  /lib/modules/'uname -r'/COMEDIDIR/drivers/multiq3.o
10   /sbin/insmod  /lib/modules/'uname -r'/COMEDIDIR/drivers/ad512.o
11   /usr/sbin/comedi_config  /dev/comedi0  multiq3 0x320, 5
12   /usr/sbin/comedi_config  /dev/comedi1  ad512 0x300
13   /sbin/insmod ./rtai_comedi_lxrt.o
```

"COMEDIDIR" is the directory name in which comedi is installed. Lines 2 to 5 are required to insert RTAI module, lines 6-9 insert the COMEDI library and drivers into the kernel, lines 10 and 11 the two data acquisition boards, and line 12 links the RTAI and COMEDI installation.

rem:

```
1    sync
2    /sbin/rmmod rtai_comedi_lxrt
3    /sbin/rmmod ad512
4    /sbin/rmmod multiq3
5    /sbin/rmmod kcomedilib
6    /sbin/rmmod comedi
7    /sbin/rmmod rtai_shm
8    /sbin/rmmod rtai_lxrt
9    /sbin/rmmod rtai_sched
10   /sbin/rmmod rtai
11   sync
```

There commands are used to do the cleaning up job when the application finished, remove all the modules from the kernel.

To run the process, do the following in comedi_lxrt directory in sequence after compilation:

```
./run
./mainpro
./mailboxpro (run this in another window in order to run at the same time as mainpro)
./rem  (tidy up when programs finished)
```

6.1.5 *Result*

The system failed to perform well with the values of K_p and K_d obtained in Chapter 6.1.3. Thereby the system was adjusted manually and it gave a decent performance, the final values are $K_p = 0.025$, $K_d = -0.007$.

6.2 A SISO PID Control Application with GUI

6.2.1 *Description*

The control plant in this application is completely same as that was used in Chapter 6.1. The only difference is in this section control is achieved through RTLab and a graphic user interface is provided, and this gives a better observability and maneuverability.

6.2.2 *Mathematical Model*

As discussed in Chapter 6.1.2, the mathematical model for this application is:

$$\frac{\theta(s)}{V_{in}(s)}=\frac{1}{s(0.0026s+0.1081)} \tag{6.12}$$

6.2.3 Control System Design

Considering the effects of the clearance characteristic between gears, dead zone dry friction characteristic, saturation characteristic of the amplifier, and nonlinear error in the conversion between Analogue signal and digital signal in practical use, it is relatively difficult to acquire the controller parameters simply by computation. In real control experiment a traditional PID controller ([1], [15], [16]) is used to get an acceptable degree of error reduction simultaneously with acceptable stability and damping:

$$V_{in}=K_p(\theta_d-\theta)+\frac{1}{K_i}\int(\theta-\theta_d)dt+K_d\frac{d\theta}{dt} \tag{6.13}$$

Substitute u for V_{in}, y for θ:

$$u=(K_p+1/(K_i\cdot s)+K_d\cdot s)\cdot e \tag{6.14}$$

e	Error. Equals u-w.
u	Control variable (output voltage)
w	Desired voltage value
y	Read-in value from analogue input

Substitute K for K_p, T_d for K_d, T_i for K_i, and introduce T_f for filtering, then:

$$u = K(w - y) + u/(1 + sT_i) - sT_d/(1 + sT_f)\, y \tag{6.15}$$

The system response is affected by the combination of four coefficients—K, T_i, T_d, and T_f. By increasing K faster responses can be achieved but the response may become more oscillatory and lead to instability. The introduction of Td and Tf brings a stabilizing effect to the system. Theoretically Tf is equal to 0 however in practical use it is needed to balance the physical effect. By adjusting Ti the overshoot and decay ratio can by diminished.

Following parameters are chosen after being adjusted in virtual plant:

$K = 15$

$T_i = 0.12$

$T_d = 0.03$

$T_f = 0.1$

This is tested in the virtual system through SIMULINK. Figure 6.1 is the simulation system in SIMULINK; SRV-3 is the block which is connected to the virtual motor servo equipment. The input to this block will be sent to the motor servo equipment and the output of the motor servo will be feedback to the Matlab workspace via "theta" and "theta dot" outputs in the block. The transfer function of the system is shown on the bottom of the diagram.

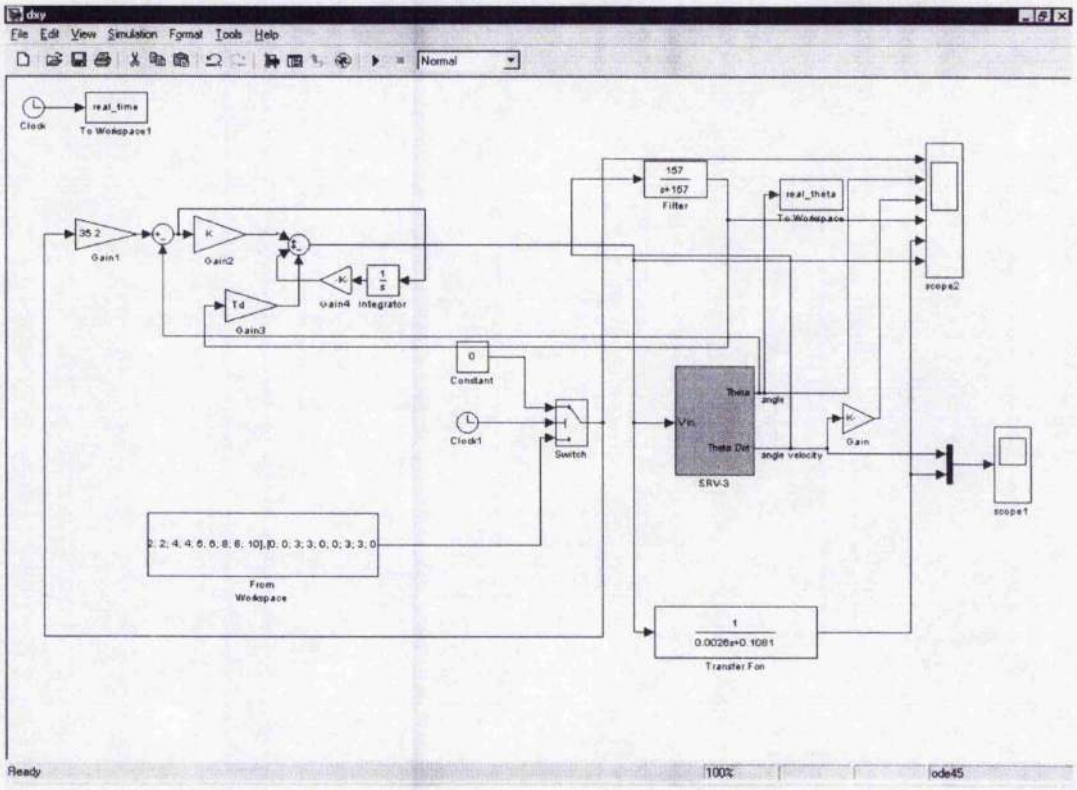


Figure 6.1: Testing module in SIMULINK

The outputs of the system can be observed in the scope block. The output after filtering, angle velocity output of the real equipment, angle output of the real equipment, output after simulation, and the original input are compared with each other in the scope block.

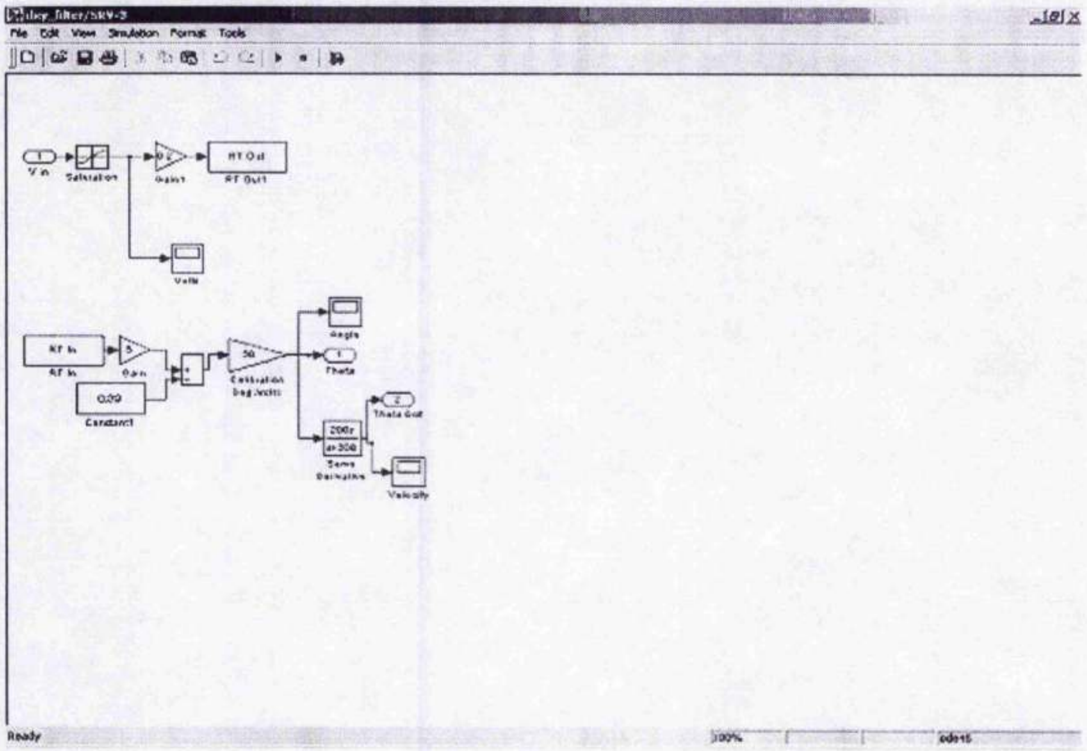


Figure 6.2: Servo03 module in SIMULINK

SRV-3 block is provided with rotary servo plant SRV-03 which is used in this project. AD512 data acquisition card is used as adapter in the system.

When a square wave signal is used as input, the counterpart outputs in "scope2" block in Figure 6.1 will be:

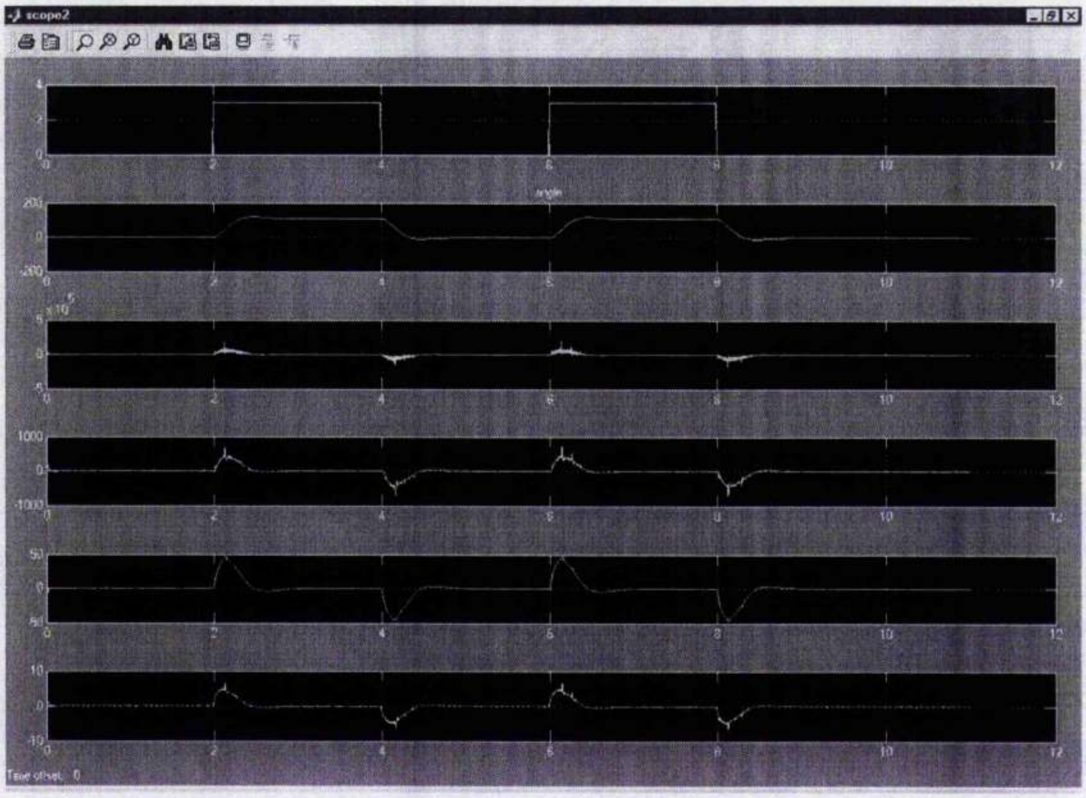


Figure 6.3: Outputs of testing module in SIMULINK

6.2.4 Real Time Control Program in Linux

As mentioned above, a real time control program in Linux resides in the form of RTLab plugin, and it can be detected automatically when all of the 4 files—PluginName.c (where the major do_control part resides), PluginName.cpp (graphical interface program), PluginName.h (shared memory declaration), and PluginName _private.h (plugin widget declaration) are placed in plugin directory properly.

For this application, the corresponding files are piddxy.c, piddxy.cpp, piddxy.h, and piddxy_private.h. Following are the block diagrams for each file, the major part as control algorithm and graphical interface layout setting will be introduced in details separately later:

piddxy.h:

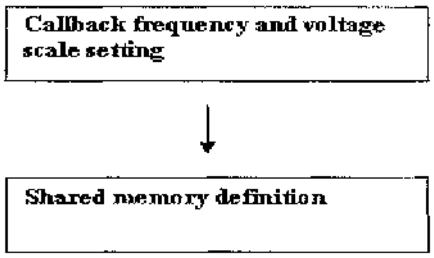


Figure 6.4: Block diagram of piddxy.h

piddxy_private.h:

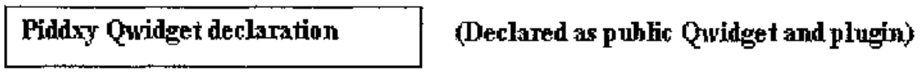

Piddxy Qwidget declaration
(Declared as public Qwidget and plugin)

Figure 6.5: Block diagram of piddxy_private.h

piddxy.c:

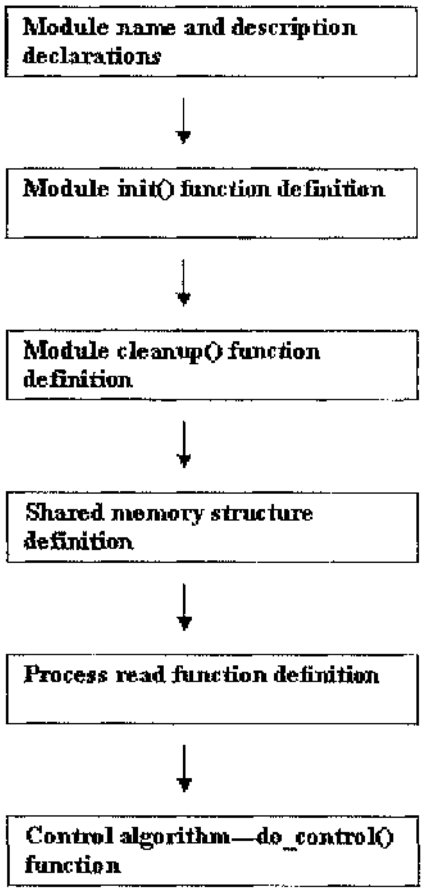

Module name and description declarations
Module init() function definition
Module cleanup() function definition
Shared memory structure definition
Process read function definition
Control algorithm—do_control() function

Figure 6.6: Block diagram of piddxy.c

piddxy.cpp:

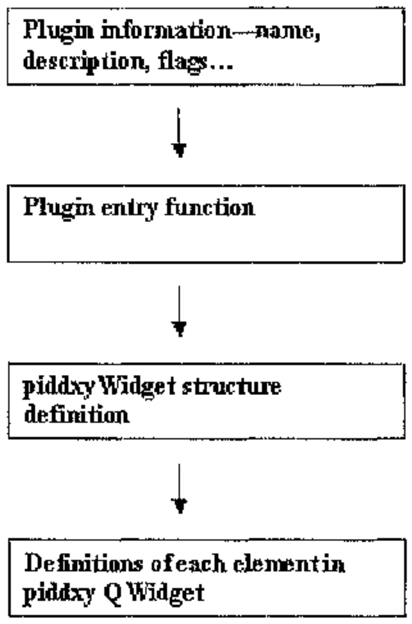

Plugin information—name, description, flags...
Plugin entry function
piddxy Widget structure definition
Definitions of each element in piddxy Q Widget

Figure 6.7: Block diagram of piddxy.cpp

Pseudo code of do_control() function(in piddxy.c):

```
/**
 * This plugin does the following:
 *
 * A Proportional + integral + derivative controller
 * Kernel code:
 *     1. Generates different waves with the counterpart periods and
amplitudes.
 *     2. Reads ADC channel shm->ai_chan and calls the voltage 'y'.
 *     3. Computes u = k*(w-y) + u/(1+sTi) - sTd/(1+sTf) y.
 *     4. Writes u to DAC channel shm->ao_chan.
 *
 * GUI (piddxy.cpp):
 *     Simple GUI to change parameters on-line
 */

 constants
Define module initialisation function
Define module cleanup function
Define shared memory initialisation function
static void do_control (MultiSampleStruct * m)
{
   constants
  Declare phase length, sampling rate, and amplitude of Square wave.

  Check if the plugin is ready to run

  Check the operation wish to perform, input wave type selection:
  0-Fixed position, designated through graphical interface.
  1-Square wave, amplitude and frequency are designated through
  graphical interface.
  2-waves from signal generator, modify wave parameter via adjusting
  signal generator


  If (the input is a square wave
  Get the phase length and frequency of the wave
  Compute the desired voltage value
     }

  Else if (the input is signal from signal generator){
  Sample from signal generator

  Check if the input voltage is in the valid range
  }

  else{
  Check if the input voltage is in the valid range
  Input signal is a fixed value in relation to the angular position
  of the motor servo
  }

  Read input voltage value from analog input channel

  Check if AI monitoring is off or the channel they want to monitor
  is not found

  Get error value through dividing desired voltage value by input
  voltage value

  Compute control. Get angular position and angular velocity of motor
  servo

  Filtering

  Clip

  Optionally echo desired voltage value back to an AI channel for the
  UI

  Echo angle velocity back to channel 2 so that it can be observed
  in Graphical interface.
  Output control signal

  Save output
}
```

The `do_control` function is the main part of .c file. In this function, data is read from the DAQ board, and calculated as certain algorithm, then sent back to the DAQ board. There are three options for setpoints available in this program, 0—fixed position, 1—square wave, and 2—waves from signal generator. For the first and the second option, the detailed values for setpoints may be given via graphical interface, and for the third option, these values are set through the signal generator. The user is able to echo any value back to whichever channel for observation by calling `rtlab_get_sample_by_chan(ChannelNumber, Value)` inside the `do_control()`

function in .c file.

Graphical interface layout setting (in **piddxy.cpp**):

```
  Plugin name, description and entry function declaration
  extern "C" {

  Declare some stuff needed by plugin engine, these symbols are read
  by libdl/dlsym()*/

  ds_plugin_ver is DS_PLUGIN_VER

  Flag is Plugin::RequiresRTLab

  Plugin name is PLUGIN_NAME

  Plugin description is
  "A Proportional + integral + derivative controller\n"
  "Kernel code:\n"
  "    1. Generates different waves with the counterpart periods and
amplitudes.\n"
  "    2. Reads ADC channel shm->ai_chan and calls the voltage 'y'.\n"
  "    3. Computes u = k*(w-y) + u/(1+sTi) - sTd/(1+sTf) y.\n"
  "    4. Writes u to DAC channel shm->ao_chan.\n"
  "\n\n"
  "GUI (piddxy.cpp):"
  "    Simple GUI to change square wave and PID parameters on-line",
  * author = "Xiaoyu Duan",
  * requires =
     "piddxy.o be loaded into the kernel. P+I+D control";

  Plugin * entry(QObject *o)
  {

/*  This is a top-level widget, and the parent is root */

  Issue warning message if plugin loading failed.

  Show the widget
  }

};

  Store some widgets that we need pointers to for
(dis)connectSignals()   and updateStats()
  struct PiddxyWidgets
{
  Declare all the elements needed in this plugin
};

/*  Plugin constructor */
Piddxy::Piddxy(DAQSystem *d)
  : QWidget(d, PLUGIN_NAME, Qt::WType_TopLevel), ds(d)
{
  Attach to shared memory

  Declare new plugin widget

  Build GUI

  Connect Signals

  Set caption name
}

/*  Plugin destructor */
Piddxy::~Piddxy()
{
  Set wave_on to 0

  Set wave_type to -1

  Set channel number for signal generator to -1

  Detach shared memory

  Delete plugin widget; just delete the struct, not the actual widget
}

  Return plugin name
  Return plugin description

/*  Setting up the main layout */
void Piddxy::buildGUI()
{
  Create new layout

  Create shared memory controller constant

  Get number of analogue input channels

  Add AO Channel QLabel to the main layout

  Add AI Channel QLabel and QComboBox to the main layout

  Add wave type QLabel and QComboBox to the main layout

  Add wave period QLabel and QSpinBox to the main layout

  Add wave amplitude QLabel and QLineEdit to the main layout

  Add K value QLabel and QLineEdit to the main layout

  Add Td value QLabel and QLineEdit to the main layout

  Add Tf value QLabel and QLineEdit to the main layout

  Add maximum voltage QLabel and QLineEdit to the main layout

  Add minimum voltage QLabel and QLineEdit to the main layout

  Add Echo output wave to DAQSystem AI chan QLabel to the main layout

  Add wave on QcheckBox to the main layout

  Add input channel number for signal generator QLabel and QcomboBox
to the main layout

  Set text in each element
}
```

```
/* Connect signals with slots */
void Piddxy::connectSignals()
{
  Connect each signal to the counterpart slot
}

/* -----------------------------------------------------------------
-------
   Slots below set parameters in the shared memory region for
notifying the
   real-time process
-----------------------------------------------------------------
------*/

SetAIChannel slot declaration

SetWaveType slot declaration

SetGene slot declaration

SetAngle slot declaration

SetPeriod slot declaration

SetAmplitude slot declaration

SetK slot declaration

SetTi slot declaration

SetTd slot declaration

SetTf slot declaration

SetU_max slot declaration

SetU_min slot declaration

SetAIEcho slot declaration

SetAO slot declaration
```

This program is broken down into several parts: `entry(QObject *o)` is the entry to the plugin, the widget is displayed by calling `show()` in this function. `Piddxy::Piddxy(DAQSystem *d)` and `Piddxy::~Piddxy()` are the constructor and the destructor of the widget which constructs and destroys the widget. `Piddxy::buildGUI()`, `Piddxy::connectSignals()`, and the other `Piddxy::set*(int c)` functions are all the elements of the constructor. The main layout of the graphical interface are built up in `Piddxy::buildGUI()` and the parameters in the shared memory are set by `Piddxy::set*` slots. `Piddxy::connectSignals()` is in charge of connecting the slots with the signals generated by the operations on the graphical interface.

6.2.5 Custom Graph in Plugin Graphical Interface

In order to make the screen output more straightforward for observation when controlling the angular velocity, a custom graph was added into the window for plugin settings (Figure 6.10) in which both the graph of the angular position of motor servo and a numerical text are displayed (in real time). The graph will show the servo angular position as it changes and the current angular position of the motor servo will be displayed in the graph (Figure 6.8). The graph refreshes every 50 millisecond in this project and the user can set it to any kind of desired time interval by changing the corresponding code in the program.

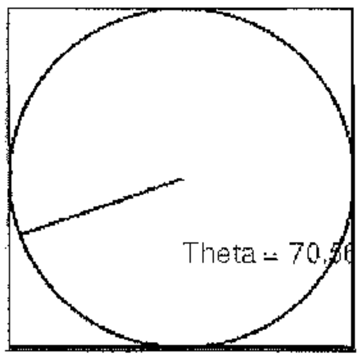


Figure 6.8 Custom window in RTLab plugin

To do this, the following code needs to be added:

In **_private.h** file:

In Pcon widget class declaration, add "void setAng(void);" in "private slots", and "Qtimer * AngTimer;" in "private".

Then declare widget class for motor servo:

```
Class ServoField : public Qwidget
{
  Q_OBJECT
public:
  ServoField( Qwidget *parent=0, const char *name=0 );

  /*QsizePolicy sizePlolicy() const; */

public slots:

private slots:
  void refresh(); /* Use to refresh  widget */

signals:

protected:
  void paintEvent( QpaintEvent * );

private:
  QRect servoRect() const;  /* return motor servo rectangle */
  QTimer * RefreshTimer;
};
```

refresh() is used to refresh the graph, it is connected to a timer—RefreshTimer which makes sure that the time refreshes the window each time when it times out. In pcon widget, AngChange() and AngTimer are used to renew the value of global variable ang.

In **.cpp** file:

```
Piddxy::buildGUI()
{
    ... ...
    int n_ai_chan......(ComediSubDevice::AnalogInput), i;

    AngTimer = new Qtimer ( this, "ang changing handler" );
    AngTimer->start(50);
    ... ...
}

void Piddxy::connectSignals()
{
    ... ...
    connect(Widgets->wave_on, ... ...SLOT(setAo(bool)));

    connect (AngTimer, SIGNAL(timeout()), this, SLOT(setAng()) );
}
```

```

ServoField::ServoField( QWidget *parent, const char *name )
        : QWidget( parent, name )
{
    setMinimumSize( 200, 200 );
    setMaximumSize( 200, 200 );

    RefreshTimer = new QTimer( this, "Graphic refreshing handler" );
    connect( RefreshTimer, SIGNAL(timeout()),
             this, SLOT(refresh()) ); /* connect refresh slot to time
out signal */
    RefreshTimer->start(50); /* unit in millisecond */

/*   setPalette( QPalette( QColour( 250, 250, 200) ) );*/ /* set
background colour here, or ignore it to use default colour  */
}

void Piddxy::setAng(void)
{
     ang = shm->angle ;
}

void ServoField::refresh()
{
    QRegion r(servoRect());
    QRect ServoR = servoRect();
    r = r.unite(QRegion(ServoR));  /* "unit" returns the bounding
rectangle  */
    repaint(r);
}

void ServoField::paintEvent( QPaintEvent * )
{
    QRect cr = servoRect();
    QString s = "Theta = " + QString::number( ang );

    QPixmap pix( cr.size() );   /* create pixmap */
    pix.fill( this, cr.topLeft() );

    QPainter p( &pix );

    p.setBrush( white );     /* set brush  */

    QPen pn=p.pen();         /* set pen */
    pn.setWidth(2);
    pn.setColor(black);
    p.setPen( pn );

    p.translate( 100, pix.height()/2 );     /* move coordinate */
    p.drawRect( QRect(-100,-100, 200, 200));  /* draw rectangle */
    p.drawArc( QRect(-100, -100, 200, 200), 0, 360*16);  /* draw arc
*/
    p.rotate( ang);     /* rotate coordinate  */
    p.drawLine(0, 0, 0, 100);     /* draw line  */
    p.end();

    p.begin( this );     /*  draw pixmap  */
    p.drawPixmap( cr.topLeft(), pix );

    p.drawText( 100,150, s );    /* draw text */
}
QRect ServoField::servoRect() const
{
    QRect r( 0, 0, 200, 200 );
    r.moveBottomLeft( rect().bottomLeft() );
    return r;
}
```

At first we declare and start a Qtimer called AngTimer, which restarts timer after every 50 milliseconds. Connect this timer with a setAng slot in which the value of shm->angle is passed to a global variable ang. The content of setAng will be executed each time when the time times out.

The next is the constructor of the custom widget called ServoField. The maximum and the minimum size are set here, and they are of the same value here which means the size of this widget is not changeable.

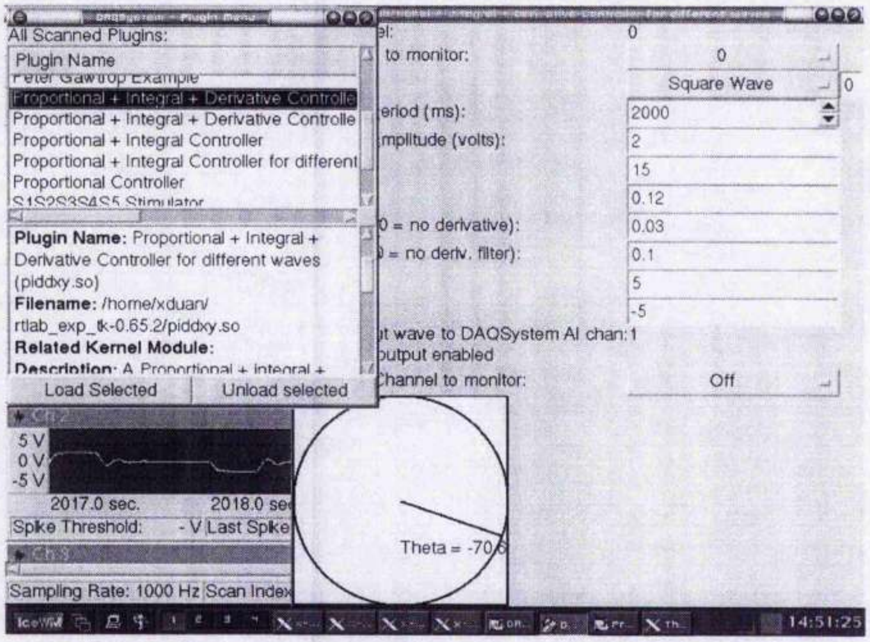
In refresh() slot it repaints the rectangular region of ServoField. In paintEvent(), a numerical text is declared first, it shows the angular position of the motor servo in the form of number, then a pixmap is created where the profile of motor servo is drawn. We can also draw it directly here rather than via a pixmap however a pixmap is useful for reducing flickering, this makes the graph looks smoother.

Lines 55 to 84 are fairly straightforward: fill the pixmap with widget background, set brush colour, set pen, place the coordinate origin in the centre of motor servo rectangle, and draw this rectangle, draw arc, draw line, then display the pixmap. In the last section the motor servo rectangle is defined.

One thing to be noted is that the Qtime used to refresh the pixmap must have a relatively longer period than that of the timer used to renew the value of ang, this ensures the pixmap is drawn each time the value of ang is refreshed.

6.2.6 Results

The module is inserted into kernel by loading it in the plugin menu window within RTLab (Figure 6.9) and the parameters are set in the plugin window (Figure 6.10):


All Scanned Plugins:

Plugin Name
Peter Gawthrop Example
Proportional + Integral + Derivative Controlle
Proportional + Integral + Derivative Controlle
Proportional + Integral Controller
Proportional + Integral Controller for different
Proportional Controller
S1S2S3S4S5 Stimulator

Plugin Name: Proportional + Integral + Derivative Controller for different waves (piddxy.so)
Filename: /home/xduan/rtlab_exp_tk-0.65.2/piddxy.so
Related Kernel Module:
Description: A Proportional + Integral +

Load Selected | Unload selected

Figure 6.9 Load plugin module in RTLab

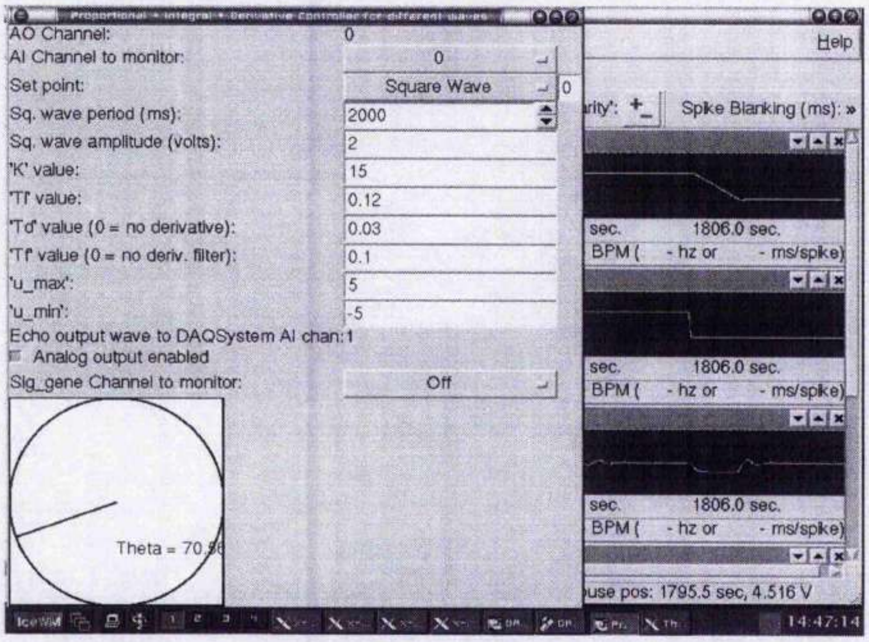

AO Channel: 0
AI Channel to monitor: 0
Set point: Square Wave 0
Sq. wave period (ms): 2000
Sq. wave amplitude (volts): 2
'K' value: 15
'Ti' value: 0.12
'Td' value (0 = no derivative): 0.03
'Tf' value (0 = no deriv. filter): 0.1
'u_max': 5
'u_min': -5
Echo output wave to DAQSystem AI chan: 1
Analog output enabled
Sig_gene Channel to monitor: Off

Figure 6.10 Piddxy plugin window

The graph at bottom left corner is the custom graph window introduced in Chapter 6.2.5. It shows the current angular position of the motor servo SRV02 both in the form of graph and numeral (theta). The period, amplitude, voltage range, channels, and the controller parameters are set in the plugin window.

There are three selections for setpoints available in this application: fixed position, which can be set by entering value in the box at right hand side of the setpoint type box, square wave, and any wave type generated by the signal generator.

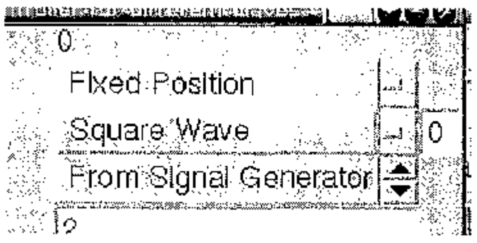
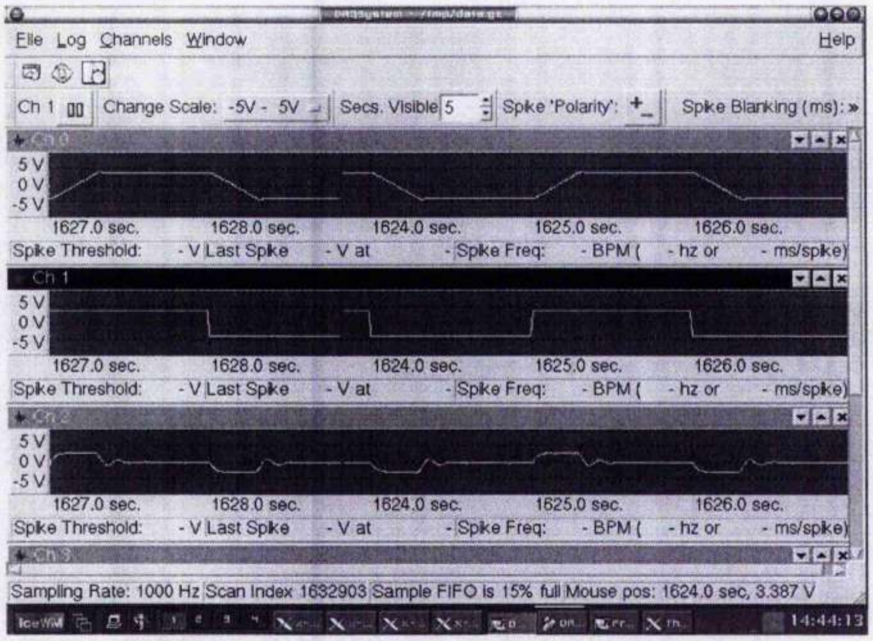


Figure 6.11 Setpoints available in piddxy plugin

The following graphs show the control results of this SISO application when a square wave is selected as the set point:



Ch 1 | Change Scale: -5V - 5V | Secs. Visible 5 | Spike 'Polarity': + | Spike Blanking (ms):

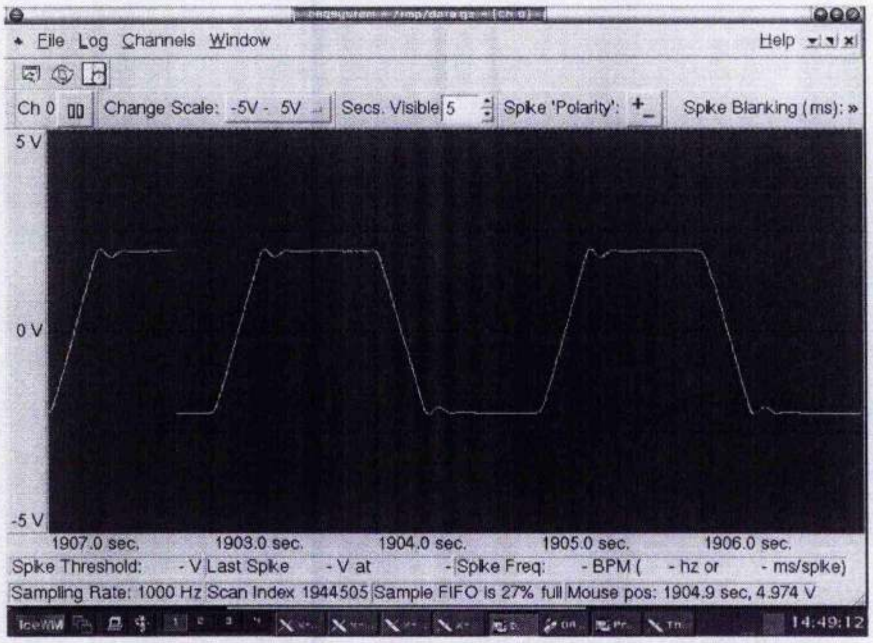
Spike Threshold: - V Last Spike - V at - Spike Freq: - BPM (- hz or - ms/spike)

Sampling Rate: 1000 Hz Scan Index 1632903 Sample FIFO is 15% full Mouse pos: 1624.0 sec, 3.387 V

Figure 6.12 Control results of a square wave

Channel 0 is the AI channel to monitor and it can be set to whichever channel by selecting in the plugin window (Figure 6.10). The graph in this channel shows the angular position of the motor servo. As (-5, 5) is used for the scale of all channels in RTLab, the scale for angular position is also converted into this range. The original scale is (-176 deg, 176 deg). Channel 1 is the wave of desired voltage value, saying setpoint. Channel 2 is the angular velocity of the motor servo, it can be acquired either by differentiating the angular position or by reading from encoder subdevice instead of analog output device directly, the original range is (-880 deg/s, 880 deg/s).

Following are the enlarged graphs for each channel:



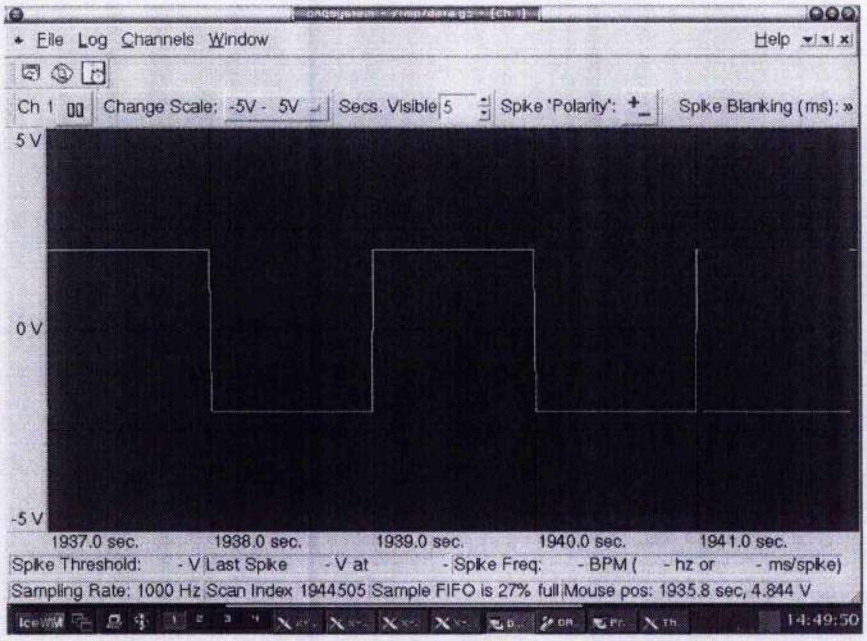
Ch 0 | Change Scale: -5V - 5V | Secs. Visible 5 | Spike 'Polarity': + | Spike Blanking (ms): »

Spike Threshold: - V Last Spike - V at - Spike Freq: - BPM (- hz or - ms/spike)

Sampling Rate: 1000 Hz Scan Index 1944505 Sample FIFO is 27% full Mouse pos: 1904.9 sec, 4.974 V

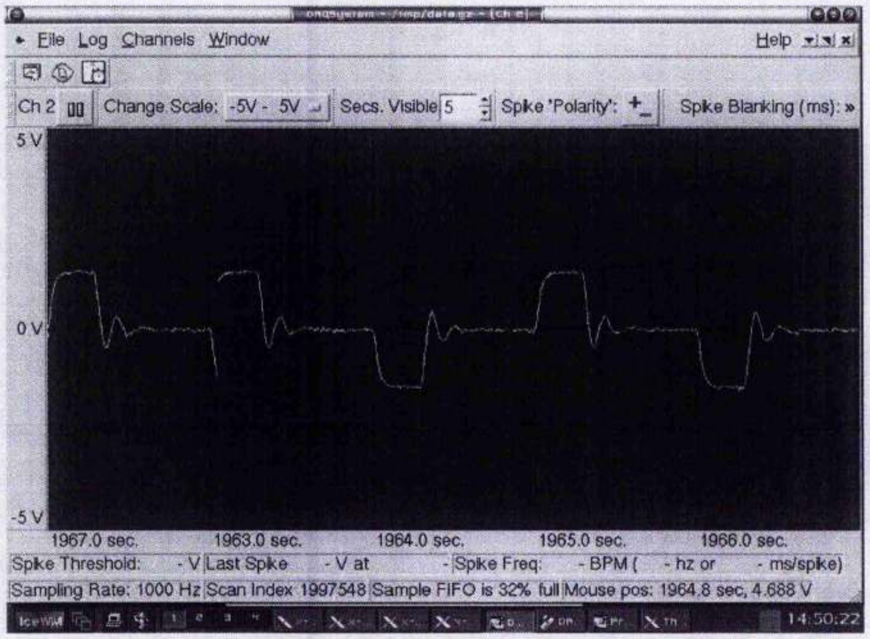
Figure 6.13 Angular position of motor servo SRV02 (theta)

The gains are implemented in the controller running at 2000Hz. The response of the system is shown in Figure 6.13. Compared with the corresponding graph in Figure 6.3 (second top), this result provides a shorter settling time and a smaller overshoot apparently, and the system response matches the design closely.



Spike Threshold: - V Last Spike - V at - Spike Freq: - BPM (- hz or - ms/spike)
Sampling Rate: 1000 Hz Scan Index 1944505 Sample FIFO is 27% full Mouse pos: 1935.8 sec, 4.844 V

Figure 6.14 Setpoint (square wave)



Spike Threshold: - V Last Spike - V at - Spike Freq: - BPM (- hz or - ms/spike)
Sampling Rate: 1000 Hz Scan Index 1997548 Sample FIFO is 32% full Mouse pos: 1964.8 sec, 4.688 V

Figure 6.15 Angular velocity of motor servo SRV02

According to the graphs it is observed that this controller gives a decent loop performance and settling time.

6.3 A SIMO PID Control Application with GUI

6.3.1 Description

As introduced in Chapter 4, this control application involves positioning the flexible link to a set point using a output feedback controller to damp out the vibration at the tip of the link as quickly as possible with minimal vibrations ([34]). The objectives of this project are:

- To obtain a transfer function model for the Flexible Link module.
- To design a PID controller that damps out the vibrations at the tip of the beam.

The Flexgage module consists of a stainless steel link instrumented with a straingage, the straingage is calibrated to give 1 volt per inch of the deflection at the tip.

6.3.2 Mathematical Model

6.3.2.1 Servomotor Model

The servomotor in this control project is the same as in motor servo control project, and all of the parameters are identical.

6.3.2.2 Flexible Link Model

The parameters of the flexible module are defined as follows ([20]):

θ	Servo gear angular displacement
ω	Servo gear angular velocity
α	Link angular deflection
v	Link angular velocity
β	Total deflection ($\beta = \alpha + \theta$)
L	Flexible link length (L =19 inches =0.4826 m)
D	End point arc length deflection ($D = \alpha L$)
m	Mass of flexible link (m =65gm)
J_{hub}	Link's moment of inertia ($J_{hub} = \frac{1}{3} mL^2 = 3.67e^{-7}\ kg \cdot m^2$)
ω_{FL}	Link's damped natural frequency
K_{stiff}	Link's stiffness ($K_{stiff} = \omega_{FL}^2 J_{hub} = 2 \cdot Nm/rad$)
K_{Gage}	Straingage calibration factor (1 Volt/inch)
T_L	Out put torque
B_{eq}	Equivalent viscous friction referred to the secondary gear

R_a Armature inertia

J_{eq} Equivalent inertia seen at the output of the gearbox

η_{mr} Motor efficiency due to rotational loss $\eta_{mr} \approx 0.87$

η_{gb} Gearbox efficiency $\eta_{gb} \approx 0.85$

$\eta = \eta_{mr} \cdot \eta_{gb} = 0.7395$

V_i Input voltage

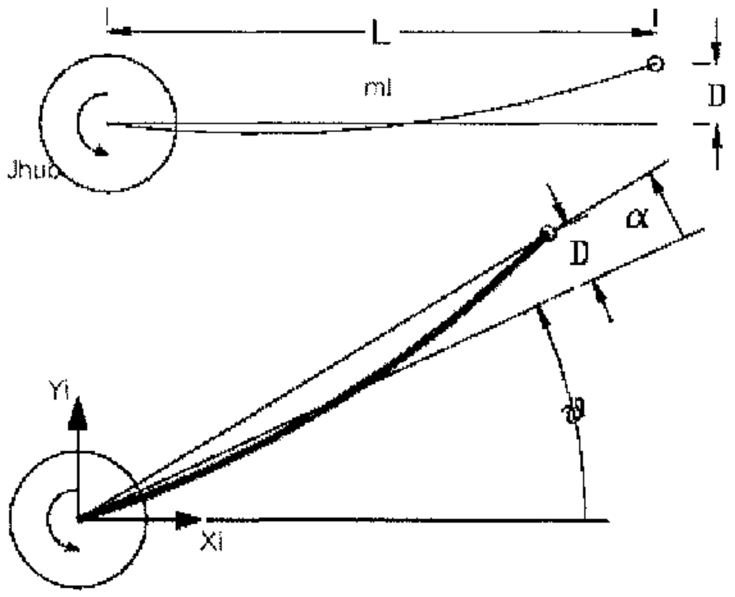


Figure 6.16: A schematic picture of the flexible link

If J_{hub} is the link's moment of inertia, the torque due to the link acceleration is:

$$T_{J_{hub}} = J_{hub}\frac{d^2\beta}{dt^2} = J_{hub}(\ddot{\theta}+\ddot{\alpha}) = J_{hub}(\dot{\omega}+\dot{v}) \qquad (6.16)$$

The link torque due to torsional spring stiffness K_{stiff} is assumed to be proportional to the link's deflection α.

The servomotor output torque in addition to overcoming the inertia torque due to J_{eq} and frictional torque, it is assumed to overcome the torque due to link's acceleration, then,

$$J_{eq}\dot{\omega} + B_{eq}\omega + J_{hub}(\dot{\omega} + \dot{v}) = T_L \tag{6.17}$$

Then,

$$\dot{\omega} = \frac{K_{stiff}}{J_{eq}}\alpha - \frac{\eta K_m^2 K_g^2 + B_{eq}R_a}{J_{eq}R_a}\omega + \frac{\eta K_m K_g}{J_{eq}R_a}v_i \tag{6.18}$$

And

$$\dot{v} = \frac{K_{stiff}(J_{eq} + J_{hub})}{J_{eq}J_{hub}}\alpha - \frac{\eta K_m^2 K_g^2 + B_{eq}R_a}{J_{eq}R_a}\omega - \frac{\eta K_m K_g}{J_{eq}R_a}v_i \tag{6.19}$$

6.3.3 Control System Design—PID Controller

The block diagram for the entire system:

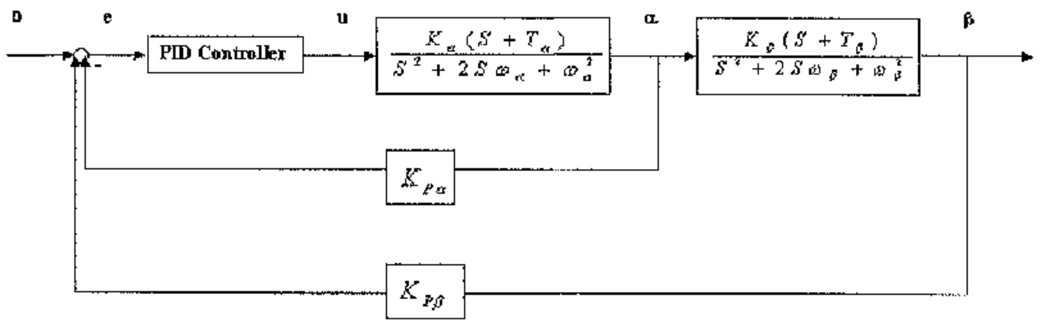

D
e
PID Controller
u
$\frac{K_\alpha (S + T_\alpha)}{S^2 + 2S\omega_\alpha + \omega_\alpha^2}$
α
$\frac{K_\beta (S + T_\beta)}{S^2 + 2S\omega_\beta + \omega_\beta^2}$
β
$K_{P\alpha}$
$K_{P\beta}$

Figure 6.17: Block diagram for the entire system

Different from the SISO application discussed in Chapter 6.2, we have two outputs in Flexible link model project—the angular position output of motor servo (θ) and the displacement output of the straingage tip (α). Taking into account the relationship between these two outputs, Sum of these two outputs (β) is used as the control variable, and almost all the other parts are kept the same.

The controller is the same:

$$u = K(w - y) + u/(1 + sT_i) - sT_d/(1 + sT_f)\,y \qquad (6.20)$$

u Control variable (output voltage).

w Desired value (The desired position of tip)

y Actual position of the tip (alpha + theta)

The system response is affected by the combination of four coefficients—K, T_i, T_d, and T_f, then the following value are obtained after adjusting:

$K = 2$

$T_i = 0.2$

$T_d = 0.05$

$T_f = 0$

6.3.4 Real Time Control Program in Linux

Same as the SISO application discussed in Chapter 6.2, the control programs for this plugin include 4 files: pidstrain_gage.c, pidstrain_gage.cpp, pidstrain_gage.h, and pidstrain_gage_private.h. The programs for these two applications are similar and the main difference is the control algorithm. In this application, the control variable is the sum of both of the two feedback variables so that the corresponding part in `do_control()` function in pidstrain_gage.c will be:

```
static void do_control (MultiSampleStruct * m)
{
  ......
  if (!sample) y = 0; /* AI monitoring is off
                         OR the channel they want to monitor
                         is not found, so ignore vAI term.. */
  else  y = sample->data; /* sample is not NULL, use vAI term..  */

  theta=y/s_ser; /* convert voltage to degree. */
  shm->feedback_angle = theta; /* this value will be showed in the
11   custom window */
12
13     betaDesired = vDesired/s_ser;
14
15     sample = rtlab_get_sample_by_chan(shm->Displacement_inp_channel,
16   m);
17     if (!sample) alpha = 0;
18     else  alpha = sample->data/(s_gag*L);
19     beta = theta + alpha;
20
21     e = betaDesired-beta;
22     /** Compute control */
23     u = shm->k*e + u_f;
24     ......
25
26     /** Now, optionally echo it BACK to an AI channel for the UI */
27     sample = rtlab_get_sample_by_chan(shm->echo_to_ai, m);
28     if (sample) sample->data = vDesired;
29
30     sample = rtlab_get_sample_by_chan(2, m);
31     if (sample) sample->data = vel;
32
33     sample = rtlab_get_sample_by_chan(3, m);
34     if (sample) sample->data = beta*5/176;
35
36     rtlab_data_write(&ctx, u);
37
38     y_old = y;                    /* Save output */
39   }
```

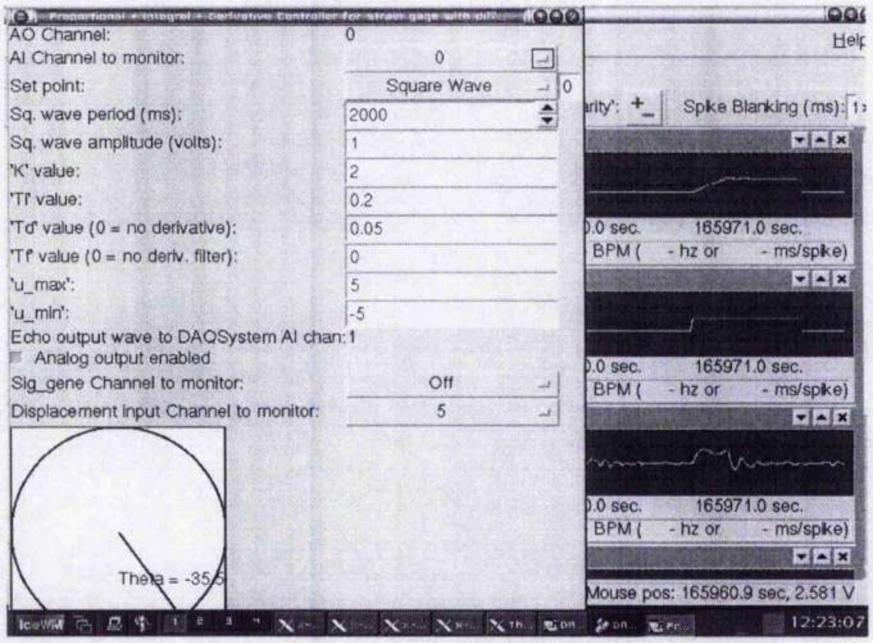
Lines 4 to 18 get the voltage values from two of the analog input channels and convert the unit into degrees. Lines 21 to 23 compute the control. Lines 27 to 34 echo the values to different channels for observation.

In terms of graphical interface, one more element has been added into the main widget so that the user is able to select the analog input channel to get the feedback of the displacement at the tip of the straingage.

```
void Pidstrain_gage::buildGUI()
{
  ......
  layout->addWidget(new QLabel("Displacement input Channel to
monitor:", this), 14, 0);

  widgets->Displacement_inp_channel = new QComboBox(this,
"DISPLACEMENT_INP Chan CBox");
  for (i = 0; i < n_ai_chans; i++)
    widgets->Displacement_inp_channel>insertItem(QString::number(i));
  widgets->Displacement_inp_channel->insertItem("Off");
  layout->addWidget(widgets->Displacement_inp_channel, 14, 1);
  ......
}
```

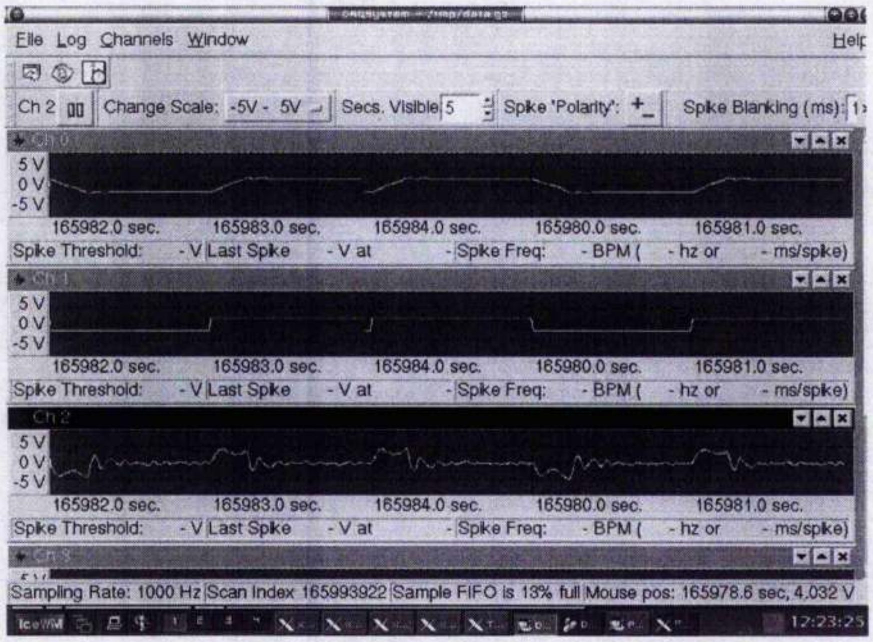
6.3.5 Results

Following is the plugin window of this application:



AO Channel: 0
AI Channel to monitor: 0
Set point: Square Wave 0
Sq. wave period (ms): 2000
Sq. wave amplitude (volts): 1
'K' value: 2
'Ti' value: 0.2
'Td' value (0 = no derivative): 0.05
'Tf' value (0 = no deriv. filter): 0
'u_max': 5
'u_min': -5
Echo output wave to DAQSystem AI chan: 1
Analog output enabled
Sig_gene Channel to monitor: Off
Displacement input Channel to monitor: 5
Theta = -35.5
Mouse pos: 165960.9 sec, 2.581 V

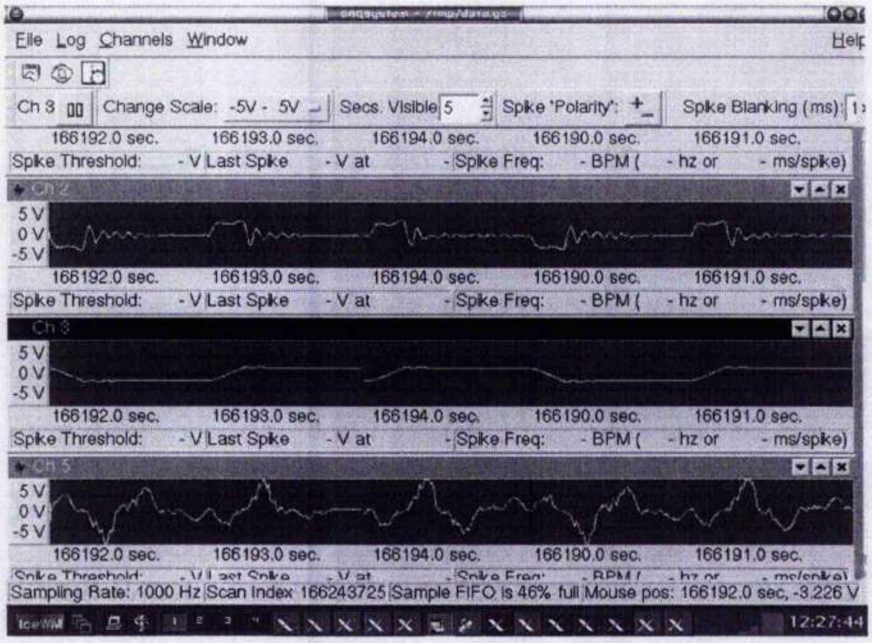
Figure 6.18 Plugin window of pidstrain_gage



Ch 2 | Change Scale: -5V - 5V | Secs. Visible 5 | Spike 'Polarity': + | Spike Blanking (ms):
Sampling Rate: 1000 Hz | Scan Index 165993922 | Sample FIFO is 13% full | Mouse pos: 165978.6 sec, 4.032 V

Figure 6.19 Control results of a square wave (1)

Same as the channels in Figure 6.12, when the voltage changes as a square wave, channel 0 is the AI channel to monitor and it can be set to whichever channel by selecting in the plugin window (Figure 6.18). The graph in this channel shows the angular position of the motor servo. Its original range is (-176 deg, 176 deg). Channel 1 is the setpoint, and Channel 2 is the angular velocity of the motor servo. Its original range is (-880 deg/s, 880 deg/s).

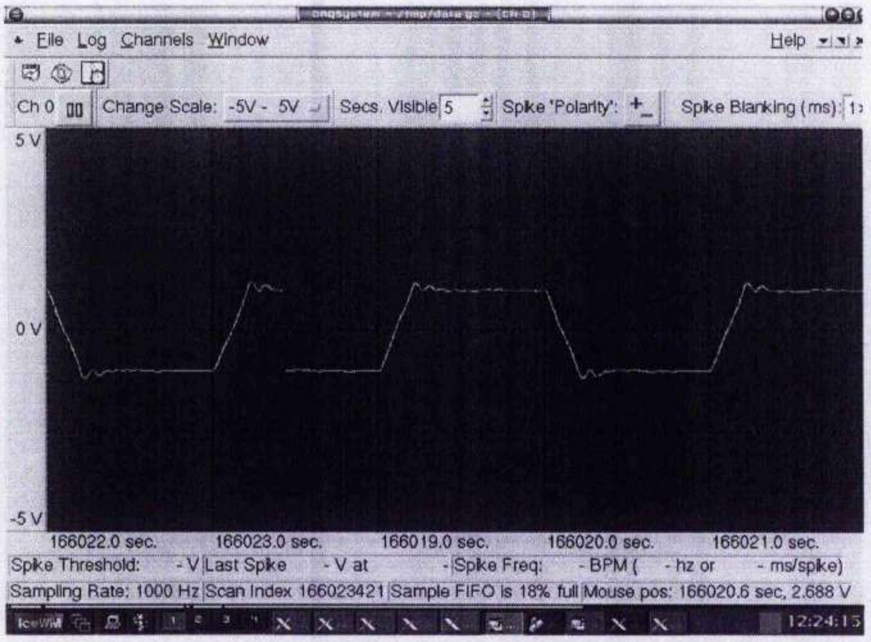

File Log Channels Window
Help
Ch 2
Ch 3
Ch 5
5 V
0 V
-5 V
166192.0 sec. 166193.0 sec. 166194.0 sec. 166190.0 sec. 166191.0 sec.
IceWM
12:27:44

```
Ch 3 | Change Scale: -5V - 5V | Secs. Visible 5 | Spike 'Polarity': + | Spike Blanking (ms): 1
Spike Threshold: - V Last Spike - V at - Spike Freq: - BPM ( - hz or - ms/spike)
Sampling Rate: 1000 Hz Scan Index 166243725 Sample FIFO is 46% full Mouse pos: 166192.0 sec, -3.226 V
```

Figure 6.20 Control results of a square wave (2)

Channel 3 is the relative angular tip position of the straingage to the whole plant system. Its original range is (-176 deg, 176 deg). Channel 5 is the displacement of the tip, original range (-5 inch, 5 inch), or (-12.7cm, 12.7cm).

Following are the enlarged graphs for each channel:

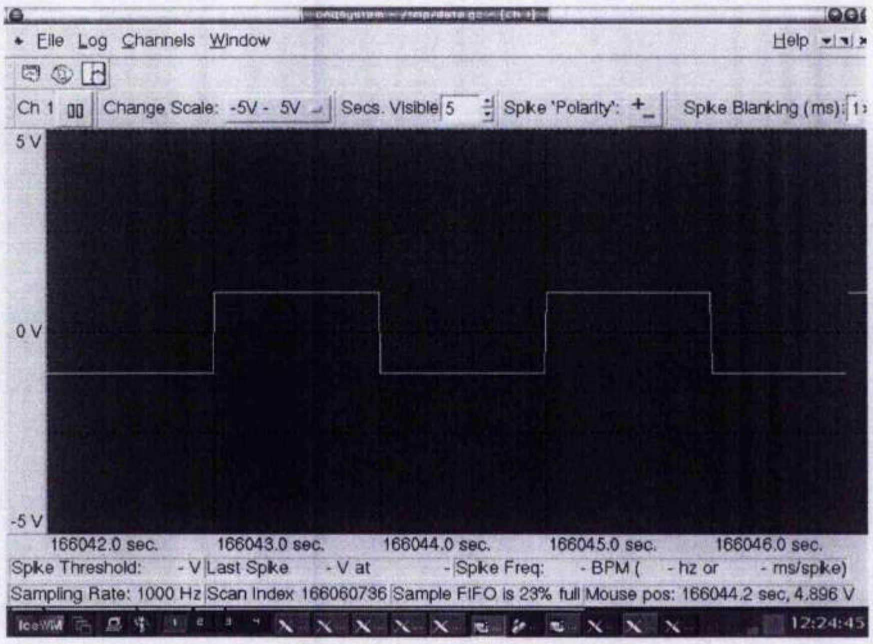


Ch 0 | Change Scale: -5V - 5V | Secs. Visible 5 | Spike 'Polarity': + | Spike Blanking (ms): 1

Spike Threshold: - V | Last Spike - V at - | Spike Freq: - BPM (- hz or - ms/spike)

Sampling Rate: 1000 Hz | Scan Index 166023421 | Sample FIFO is 18% full | Mouse pos: 166020.6 sec, 2.688 V

Figure 6.21 Angular position of motor servo SRV02 (theta)

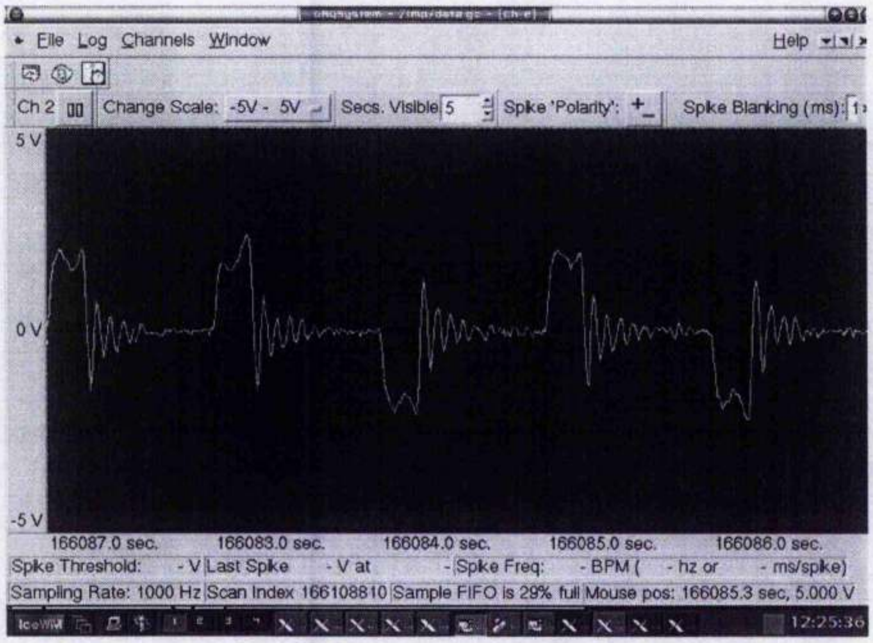


Ch 1 Change Scale: -5V - 5V Secs. Visible 5 Spike 'Polarity': +_ Spike Blanking (ms): 1

Spike Threshold: - V Last Spike - V at - Spike Freq: - BPM (- hz or - ms/spike)

Sampling Rate: 1000 Hz Scan Index 166060736 Sample FIFO is 23% full Mouse pos: 166044.2 sec, 4.896 V

Figure 6.22 Setpoint (square wave)

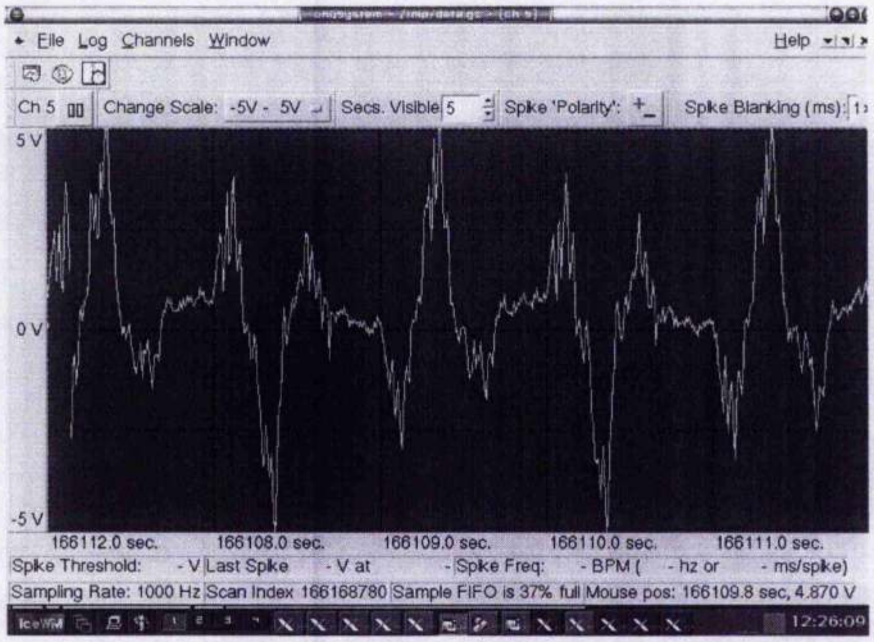


Ch 2 Change Scale: -5V - 5V Secs. Visible 5 Spike 'Polarity': +_ Spike Blanking (ms): 1

Spike Threshold: - V Last Spike - V at - Spike Freq: - BPM (- hz or - ms/spike)

Sampling Rate: 1000 Hz Scan Index 166108810 Sample FIFO is 29% full Mouse pos: 166085.3 sec, 5.000 V

Figure 6.23 Angular velocity of motor servo SRV02

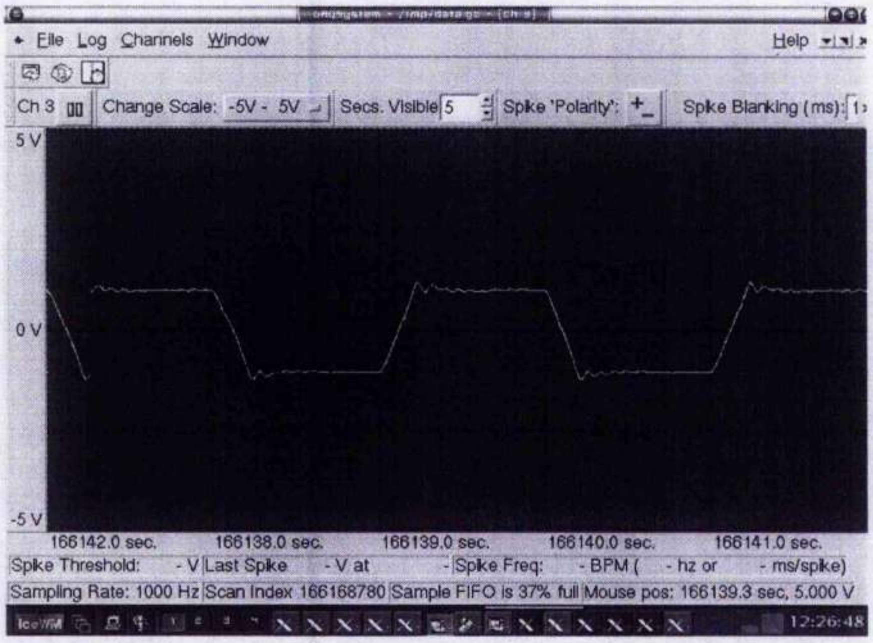


Ch 5 Change Scale: -5V - 5V Secs. Visible 5 Spike 'Polarity': +_ Spike Blanking (ms): 1

Spike Threshold: - V Last Spike - V at - Spike Freq: - BPM (- hz or - ms/spike)

Sampling Rate: 1000 Hz Scan Index 166168780 Sample FIFO is 37% full Mouse pos: 166109.8 sec, 4.870 V

Figure 6.24 Displacement of the tip of the straingage (alpha)



Ch 3 Change Scale: -5V - 5V Secs. Visible 5 Spike 'Polarity': +_ Spike Blanking (ms): 1

Spike Threshold: - V Last Spike - V at - Spike Freq: - BPM (- hz or - ms/spike)

Sampling Rate: 1000 Hz Scan Index 166168780 Sample FIFO is 37% full Mouse pos: 166139.3 sec, 5.000 V

Figure 6.25 Relative angular tip position to the whole system (theta+alpha)

As the straingage is always vibrating when controlling therefore it is very difficult to have a completely steady response for this system and as shown in above graphs, the proposed controller gives an acceptable performance.

6.4 A SIMO LQR Control Application with GUI

6.4.1 Description

The control object is completely the same as that in Chapter 6.3, the only difference is that the controller is designed by an optimal regulator—Linear Quadratic Regulator (LQR) instead of traditional PID method. This introduces a better control result.

6.4.2 Mathematical Mode

This application addresses the problem of controlling a flexible link with a state feedback controller ([20]). A state-space model:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{6.21}$$

$$y(t) = Cx(t) \tag{6.22}$$

$$x(t) = \begin{bmatrix}\theta & \alpha & \omega & v\end{bmatrix}^T \tag{6.23}$$

$$\begin{bmatrix} \dot{\theta} \\ \dot{\alpha} \\ \dot{\omega} \\ \dot{\nu} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \dfrac{K_{stiff}}{J_{eq}} & -\dfrac{\eta K_m^2 K_g^2 + B_{eq} R_a}{J_{eq} R_a} & 0 \\ 0 & \dfrac{K_{stiff}(J_{eq}+J_{hub})}{J_{eq} J_{hub}} & \dfrac{\eta K_m^2 K_g^2 + B_{eq} R_a}{J_{eq} R_a} & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \alpha \\ \omega \\ \nu \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dfrac{\eta K_m K_g}{J_{eq} R_a} \\ -\dfrac{\eta K_m K_g}{J_{eq} R_a} \end{bmatrix} v_i \quad (6.24)$$

After substituting system parameters then obtain the following linear model:

$$A = \begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 1059.1 & -57.6 & 0.0 \\ 0.0 & -1463.4 & 57.6 & 0.0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.0 & 0.0 & 107.2 & -107.2 \end{bmatrix}^T$$

$$X = \begin{bmatrix} \theta & \alpha & \dot{\theta} & \dot{\alpha} \end{bmatrix}$$

6.4.3 *Control System Design—LQR Controller*

Lineal Quadratic Regulator (LQR) is a kind of controller which is designed by the approach of optimal controller designing ([3]). In LQR control, the object is to determine the optimal controller $u(t) = -Kx(t)$ such that a given performance index $J = \int (x^T Q x + u^T R u) dt$ is minimized. This performance index is selected to give the best performance. The balance which consists of the effects of each state variables and individual control inputs will change along with the change of elements Q and R.

For example, using an identity matrix Q for weights all the states equally, a diagonal matrix $Q = diag([260 \quad 3600 \quad 2 \quad 1])$ and $R = 1$ are used here to calculate the optimal feedback matrix K. These values were obtained after optimisation. In Matlab, this can be achieved by using the function $[k, \quad S] = lqr2(A, \quad B, \quad Q, \quad R)$ and it minimizes the cost function J subject to the constraint defined by the state equation.

The design is performed using state feedback controller in the form of $V = -kX$, a LQR controller is used and the resulting feedback gain is:

$$K = [20.0 \quad -130.0 \quad 36 \quad 1.5]$$

The deflection in the tip of gage could be obtained by:

$\alpha = \dfrac{D}{L} = \dfrac{0.0254(m/volt)Vs(volt)}{L_{link}}$ in the unit of radians, where Vs is the input voltage from the sensor in the joint end of straingage and L_link is the length of the link in metres.

Following is the test model in SIMULINK:

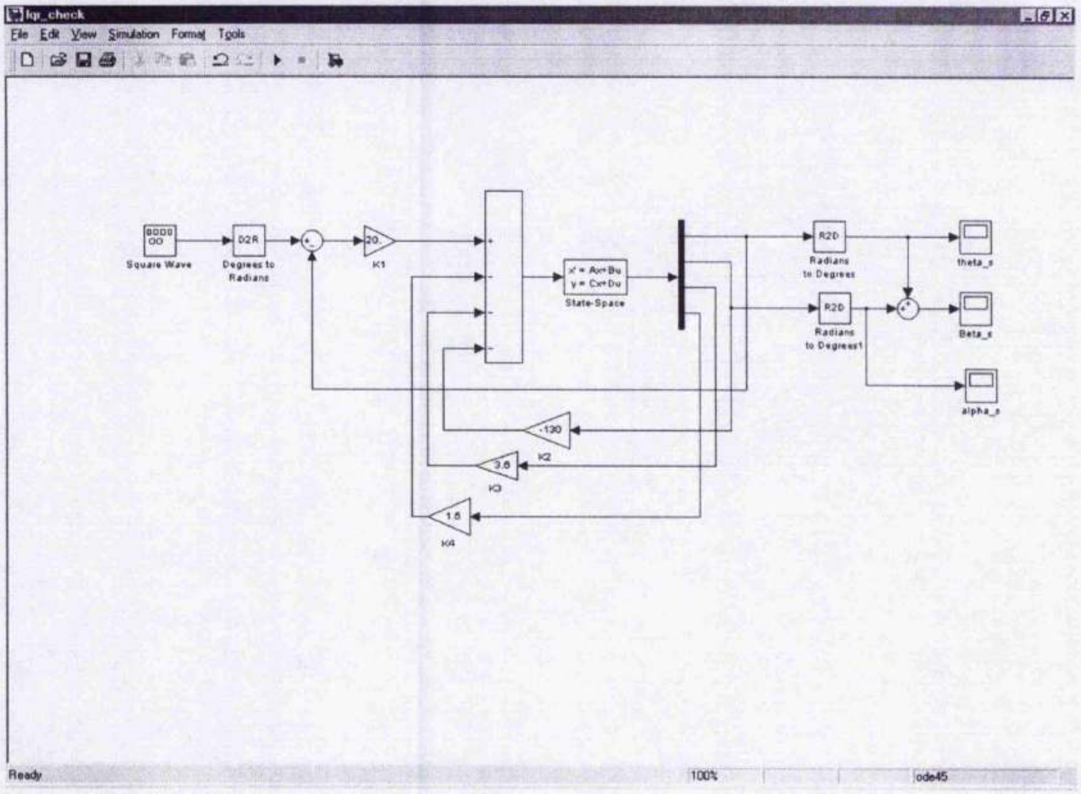


Figure 6.26: LQR control simulation model in SIMULINK

And the outputs:

Theta—The position of motor servo (the upper scope in Figure 6.26):

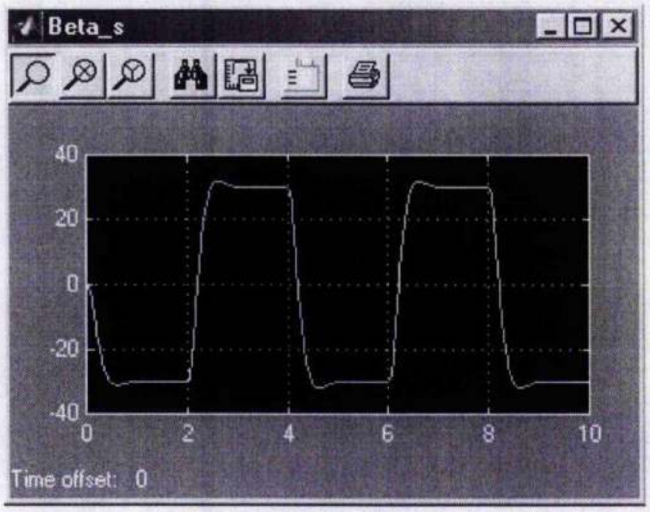


Figure 6.27: Output of theta in LQR control

Beta—The position of tip relative to whole plant (the central scope in Figure 6.26):

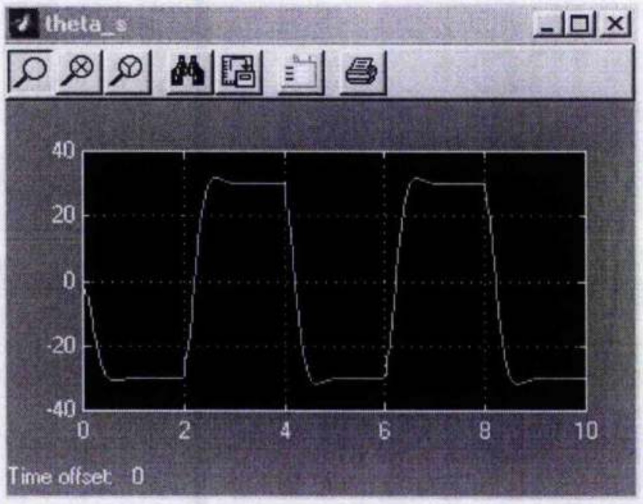


Figure 6.28: Output of beta in LQR control

Alpha—The position of tip relative to joint (The lower scope in Figure 6.26)

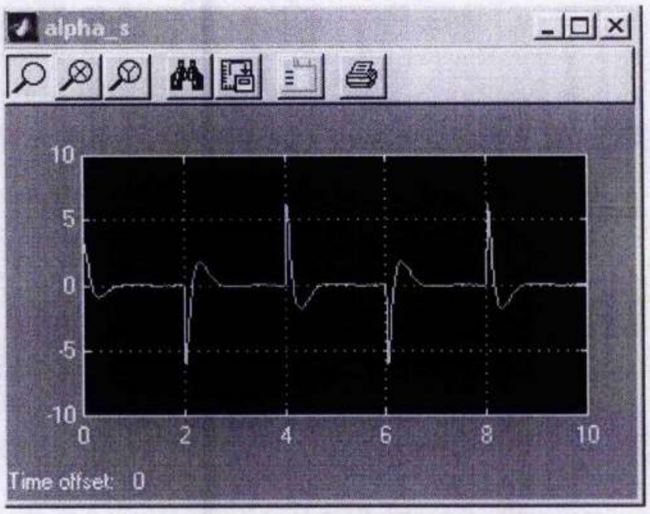


Figure 6.29: Output of alpha in LQR control

6.4.4 Real Time Control Program in Linux

A different controller was adopted in this application therefore the major difference between this project and previous application is the control algorithm. In `do_control()` function in lqrstrain_gage.c:

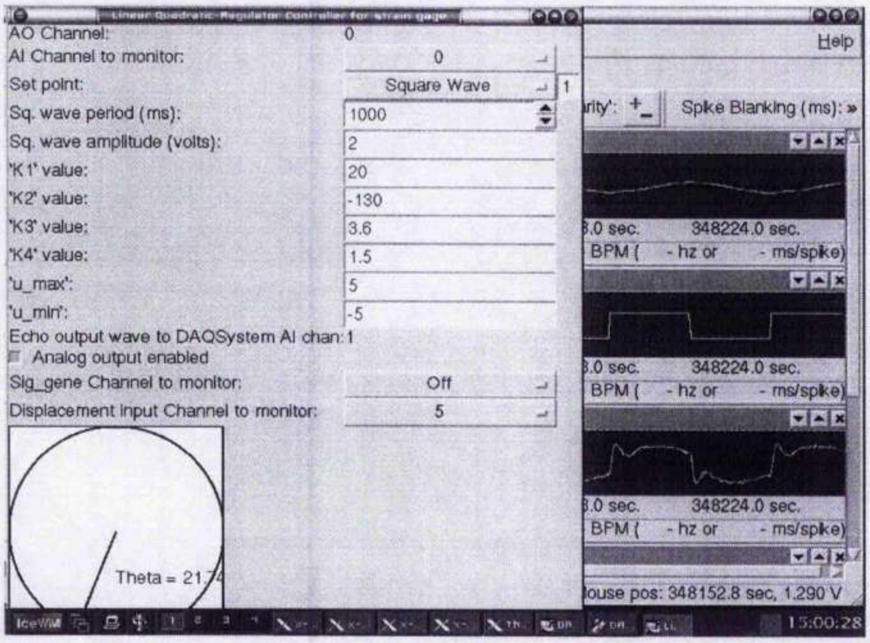
```
static void do_control (MultiSampleStruct * m)
{
  /** Formula is u = v-kx
      Status Feedback
  */
  ......
  sample = rtlab_get_sample_by_chan(shm->ai_chan, m);

  if (!sample) y = 0; /* AI monitoring is off
                           OR the channel they want to monitor
                           is not found, so ignore vAI term.. */
  else  y = sample->data; /* sample is not NULL, use vAI term..  */

14    theta = y*D2R/s_ser;
15
16    ......
17    sample = rtlab_get_sample_by_chan(shm->Displacement_inp_channel,
18  m);
19    if (!sample) alpha = 0; /* AI monitoring is off
20                               OR the channel they want to monitor
21                               is not found, so ignore vAI term.. */
22    else  alpha = sample->data*0.0254/L; /* Change unit from inch to
23  metre, then compute alpha  */
24    ......
25    shm->feedback_angle = (theta+alpha)*R2D;
26
27    /* Compute control */
28
29    u = (vDesired*D2R*shm->k1/s_ser - (shm->k1*theta_f + shm-
30  >k2*alpha_f + shm->k3
31  *theta_fd + shm->k4*alpha_fd))*0.1;
32
33    /** Now, optionally echo it BACK to an AI channel for the UI */
34    sample = rtlab_get_sample_by_chan(shm->echo_to_ai, m);
35    if (sample) sample->data = vDesired;
36
37    sample = rtlab_get_sample_by_chan(2, m);
38    if (sample) sample->data = theta_fd;
39
40    sample = rtlab_get_sample_by_chan(3, m);
41    if (sample) sample->data = u;
42    rtlab_data_write(&ctx, u);
43  }
```

Lines 7 to 23 get the voltage values from two of the analog input channels and convert the values into those in the unit of radians. Lines 29 to 31 compute the control. Lines 34 to 41 echo the values to different channels for observation

In terms of graphical window, K1, K2, K3 and K4 are showed as control parameters instead of K, Ti, Td, and Tf, as shown in Figure 6.30.

6.4.5 Results

There are four control parameters in the plugin window: K1, K2, K3 and K4. Same as previous applications, three selections for setpoints are available in the application: fixed position, square wave, and any wave type generated by the signal generator. The channel for displacement feedback is set to channel 5 here, and the user can set it to any other spare channels.


Linear Quadratic Regulator Controller for strain gage

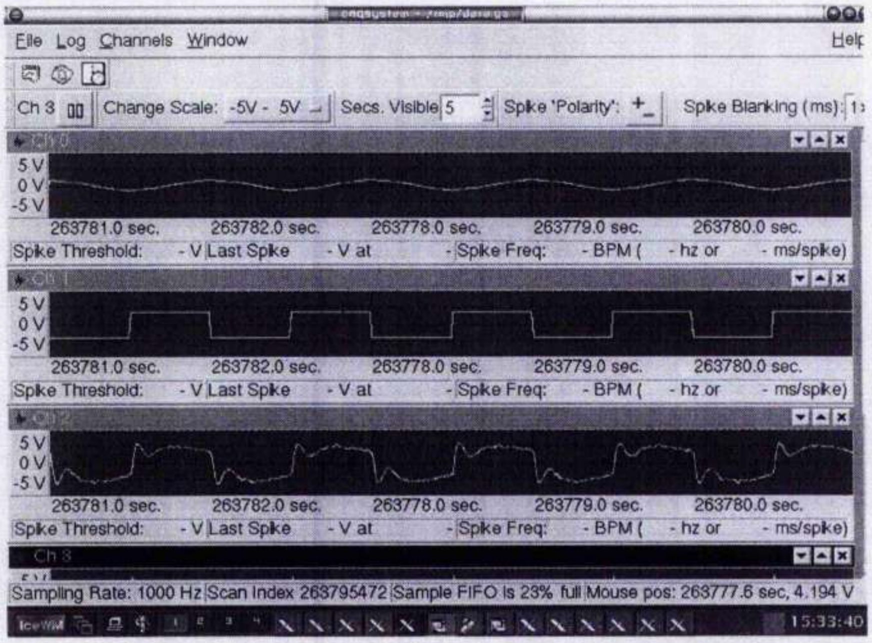
```
AO Channel:                                 0
AI Channel to monitor:                      0
Set point:                                  Square Wave   1
Sq. wave period (ms):                       1000
Sq. wave amplitude (volts):                 2
'K1' value:                                 20
'K2' value:                                 -130
'K3' value:                                 3.6
'K4' value:                                 1.5
'u_max':                                    5
'u_min':                                    -5
Echo output wave to DAQSystem AI chan:1
Analog output enabled
Sig_gene Channel to monitor:                Off
Displacement Input Channel to monitor:      5
Theta = 21.74
```

```
.0 sec.      348224.0 sec.
BPM (   - hz or    - ms/spike)
.0 sec.      348224.0 sec.
BPM (   - hz or    - ms/spike)
.0 sec.      348224.0 sec.
BPM (   - hz or    - ms/spike)
ouse pos: 348152.8 sec, 1.290 V
```

Figure 6.30 Plugin window of lqrstrain_gage

As shown in Figure 6.31, Channel 0 shows the angular position of the motor servo.

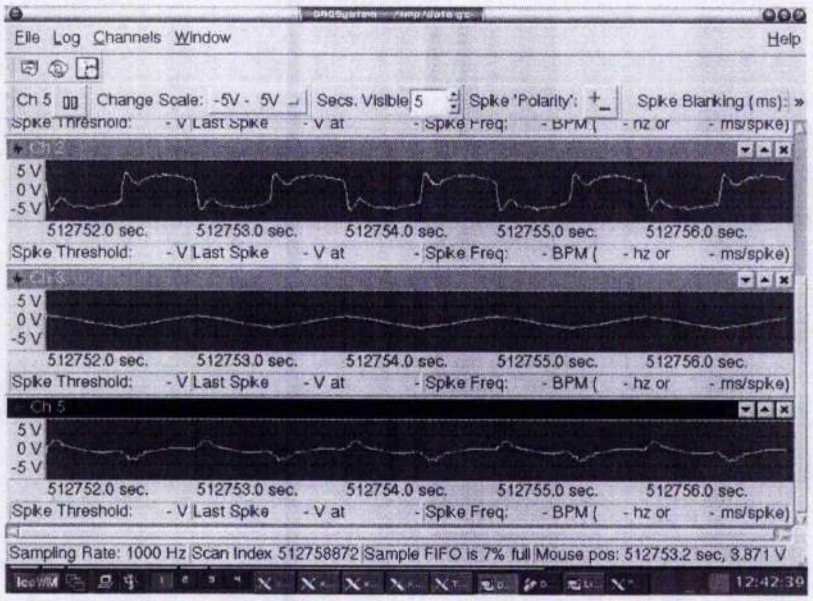
Its original scale is (-176, 176). Channel 1 is the setpoint, and Channel 2 is the angular velocity of the motor servo. The original range is (-880 deg/s, 880 deg/s).



Sampling Rate: 1000 Hz Scan Index 263795472 Sample FIFO is 23% full Mouse pos: 263777.6 sec, 4.194 V

Figure 6.31 Control results of a square wave (1)

Channel 3 is the relative angular tip position of the straingage to the whole plant system, its original range is (-176 deg, 176deg). Channel 5 is the displacement of the tip, original scale (-12.7 cm, 12.7cm) (Figure 6.32).



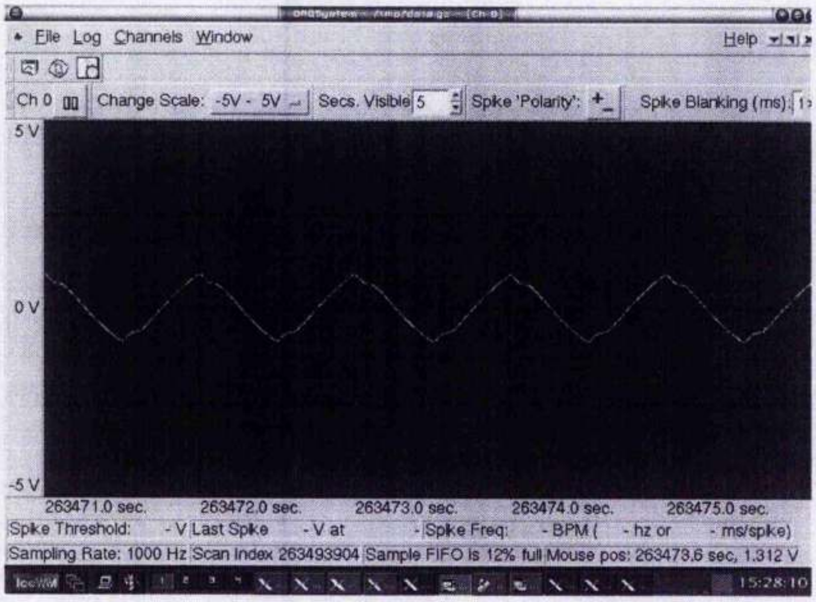
Ch 5 | Change Scale: -5V - 5V | Secs. Visible 5 | Spike 'Polarity': + | Spike Blanking (ms):

Spike Threshold: - V Last Spike - V at - Spike Freq: - BPM (- hz or - ms/spike)

Sampling Rate: 1000 Hz Scan Index 512758872 Sample FIFO is 7% full Mouse pos: 512753.2 sec, 3.871 V

Figure 6.32 Control results of a square wave (2)

Following are the enlarged graphs for each channel:



Ch 0 | Change Scale: -5V - 5V | Secs. Visible 5 | Spike 'Polarity': + | Spike Blanking (ms):

Spike Threshold: - V Last Spike - V at - Spike Freq: - BPM (- hz or - ms/spike)

Sampling Rate: 1000 Hz Scan Index 263493904 Sample FIFO is 12% full Mouse pos: 263473.6 sec, 1.312 V

Figure 6.33 Angular position of motor servo SRV02 (theta)

The setpoint is the same as that in Chapter 6.2 (Figure 6.14) and Chapter 6.3 (Figure 6.22).

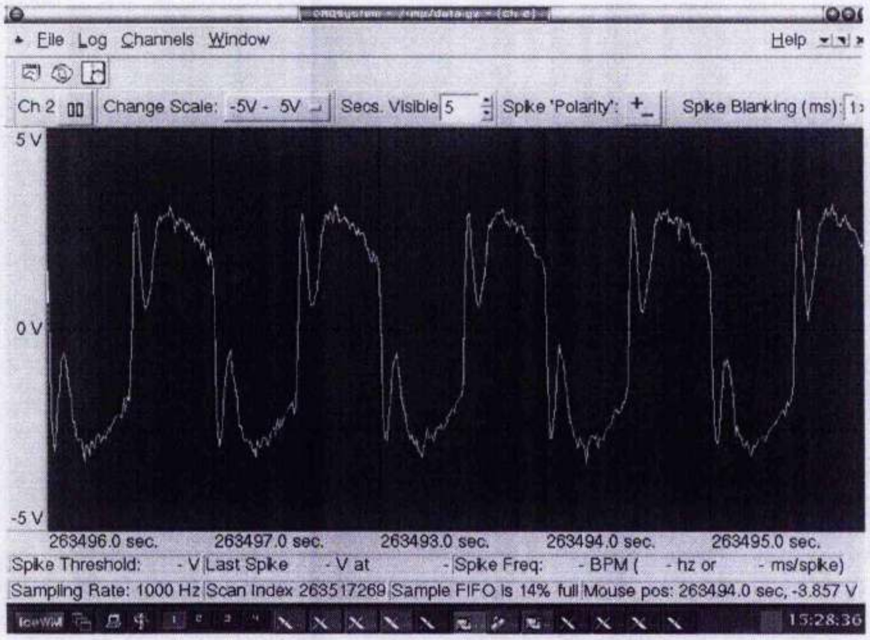
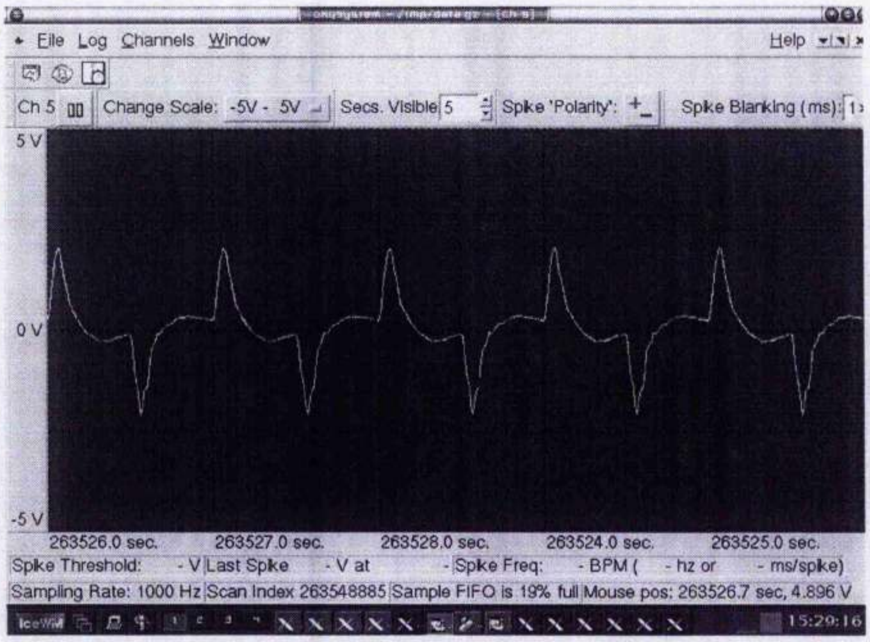
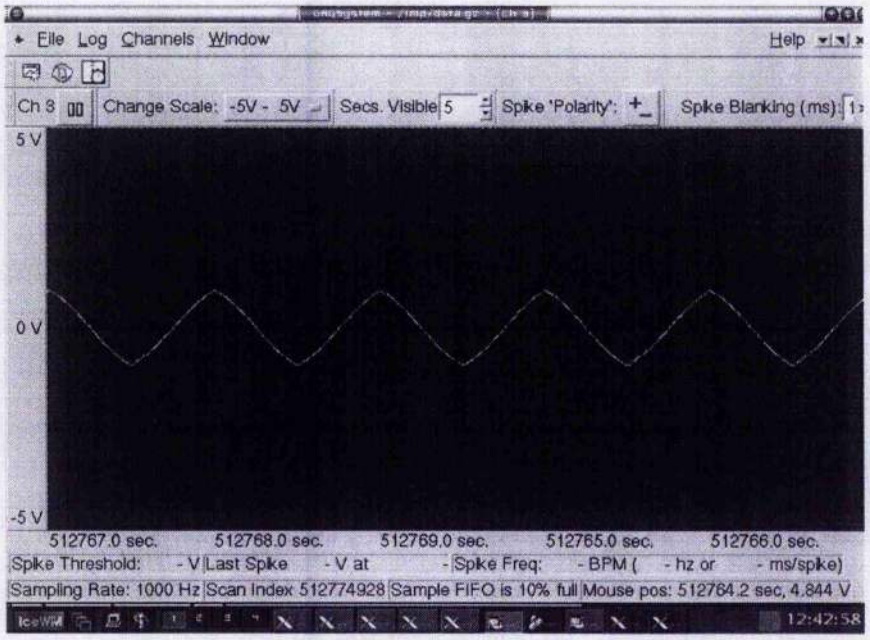


Figure 6.34 Angular velocity of motor servo SRV02



```
Ch 5 | Change Scale: -5V - 5V | Secs. Visible 5 | Spike 'Polarity': +_ | Spike Blanking (ms): 1
Spike Threshold: - V|Last Spike - V at - |Spike Freq: - BPM ( - hz or - ms/spike)
Sampling Rate: 1000 Hz|Scan Index 263548885|Sample FIFO is 19% full|Mouse pos: 263526.7 sec, 4.896 V
```

Figure 6.35 Displacement of the tip of the straingage (alpha)



```
Ch 3 | Change Scale: -5V - 5V | Secs. Visible 5 | Spike 'Polarity': +_ | Spike Blanking (ms): 1
Spike Threshold: - V|Last Spike - V at - |Spike Freq: - BPM ( - hz or - ms/spike)
Sampling Rate: 1000 Hz|Scan Index 512774928|Sample FIFO is 10% full|Mouse pos: 512764.2 sec, 4.844 V
```

Figure 6.36 Relative angular tip position to the whole system (theta+alpha)

The system is found to perform well. It is very sensitive to variations in low pass filter cutoff frequency, sampling rate and derivative feedback gains. The system can be tuned by changing the gains K2 and K4, bigger K2 causes larger overshoot and too much K4 will lead to instability.

6.5 Summary

In this chapter four control applications with different controllers and user interfaces are presented. One simple command-line mode program which combines COMEDI with RTAI LXRT, and three applications in the form of RTLab plugins in which different controllers are used based on the complexity of the applications.

According to the results shown in different sections. A tradition PID control may provide a short settling time and a small overshoot when a single-input single-output object (motor servo) is controlled. However it is very difficult to have a steady response for a single-input multi-output system (flexible link with motor servo) with the same controller, therefore a LQR controller is introduced in chapter 6.4 and it provides decent stability, overshoot and settling time.

Chapter 7 Conclusions and Discussion

7.1 Why Linux?

Aside from Windows, Linux is probably currently the most popular operating system in the world. It is being adopted more and more in modern industries due to its huge potential and good real time capability. We chose Linux as the platform in this project as its features of ([30]):

- Multitasking—It allows several tasks running at the same time.
- Multiuser—Several users on the same machine at the same time.
- Multiprocessor—SMP support is available on the Intel and SPARC platforms.
- Multithreading—Multiple independent threads are allowed to run in a single memory space.
- POSIX job control.
- Multiple virtual consoles—Linux allows several independent login sessions through the console, and the user can switch between them via hotkeys.

Linux supports several common filesystems such as Minix, Xenix, and all the common system V filesystems. It provides memory protection between processes, and this ensures the system would not be brought down by just one program, further more, Linux is open source software, which means its source code is opened to every user.

7.2 Why RTAI?

As introduced in Chapter 2, RTLinux and RTAI are two of the most popular major real time Linux branches available currently. Both of them have similar scheduling methods and API functions however RTAI is supported better today and it works better with COMEDI, therefore it is chosen to provide real time performances in this project.

7.3 COMEDI Drivers

There are a numbers of drivers provided in COMEDI which could be found in $COMEDIDIR$/comedi/drivers/, including the driver of MultiQ3 board used in this project. However the driver of AD512 card is not provided with current COMEDI distributions. A driver program was thus written to carry out this function.

One thing to note, COMEDI provides two different libraries, Comedilib and Kcomidilib, Comedilib is a user-space library while Kcomedilib is a Linux kernel module that provides the same interface as Comedilib in kernel space, i.e. for real-time tasks. When using Comedilib, include comedilib.h as head file but when using Kcomedilib, insert kcomedilib.o into kernel. Misusage of the COMEDI library may cause "unresolved symbol" errors during compiling.

7.4 RTLab

As introduced in chapter 5, RTLab is an ongoing project to develop a general-purpose, open-source, hard real-time experiment interface software system ([19]). All applications in RTLab exist in the form of RTLab plugin modules. In order to get RTLab compiling and running properly, users need to have ([21], [22]):

- A Real Time Linux variant, either RTLinux v3.1 (or later) or RTAI 24.1.9 (or later). For RTLinux, shared memory driver, POSIX standard IO, and floating point support should be enabled, and for RTAI, floating point support, POSIX API, RT memory manager, and FIFOs need to be enabled when configuring.
- Comedi0.7.65 or later with comedilib or kcomedilib. Enable real time support, of course.
- Qt 3.x. The GUI software
- Suitable compiler. GNU g++-2.96 and gcc 2.95 are recommended for best results as C++ and regular C compiler. If RTAI is used with RTLab g++-2.95 and gcc-2.95, user had better try g++-2.96 and gcc-2.95 instead.

When running RTLab, the following modules are required to be loaded into the kernel:

- For RTLinux

kcomedilib, comedi, mbuff, rtl_fifo, rtl_sched, rtl_posixio, rtl_time, rtl

- For RTAI

comedi, kcomedilib, rtai, rtai_sched, rtai_pthread, rtai_fifos, rtai_shm

During experiments, the error message "unresolved symbols init_z_apps > and free_z_apps" appeared when only the modules mentioned above were loaded. The problem was solved when another module rtai_lxrt was inserted into Kernel.

7.5 Real Time Solutions

Users can either achieve real time control in hard real time or soft real time. Hard-real time controlling gives a Linux process, or a pthread (POSIX thread) hard real time execution capability allowing full kernel pre-emption, while soft-real time controlling gives processes standard Linux behaviour and, allows other processes to run at the same time, this being forbidden in hard real time. Different scheduling methods are used for different real time solutions. Refer to Chapter 4 for more details.

Chapter 8 Future Work

Towards a better understanding and research in the field of real time control under Linux, there are a numbers of topics which could be considered for further research:

(1) RTAI + COMEDI solutions.

The integration of COMEDI under RTAI can be developed using RTAILAB and MATLAB. Release of 2.24.11 of RTAI allows the integration of COMEDI drivers in SIMULINK schemas and to generate code for (X)RTAILAB. If there is no MATLAB/SIMULINK/RTW, the same thing can be done under Scilab/Scicos also. In this case, the unofficial release of RTAI (2.24.12prel) and two of other files, rt_scilab.tgz and a modified xrtailab.cpp are needed. In this project COMEDI + RTAI LXRT was adopted and other ways may be explored in future works.

(2) Math functions in RTLab.

Some of the math functions are not available in RTLab because they live in the math library which is a user space library. One possible solution to use them in kernel space is linking libm.a to the custom module by ar x user/lib/libm.a, get dozens of .o files out then link all of them to the custom module. However this will cause some errors of undefined symbols as fputs(), fprint(), etc. Therefore it might be worthwhile to create of stripped down version libm and release it with the RTLab resources, so that the people wishing to use libm can easily do so without extra effort.

(3) Qt Designer in RTLab.

The graphical interface software Qt provides a very convenient and powerful tool named Qt Designer by which user can build their own graphs simply by clicking the mouse in the graphical interface of Qt Designer, instead of writing hundreds lines of laborious scripts. All the custom widgets will be put into one .ui file by Qt Designer and compiled later. Although this kind of .ui file cannot be picked up properly by the build system in RTLab, user can compile them manually. That means, create a .ui file by Qt Designer then convert it to a .cpp file then compile it. An alternative way is changing around the RTLab build system to make it as painless as possible to create plugins with the designer. Thus adding plugins to RTLab will be much easier.

(4) Mouse movement feedback in RTLab plugin.

In this project, the setpoints are input in RTLab plugin by inputting them in plugin graphical interface through keyboard. Further work can be done using a mouse event function to feedback the current position of the mouse, then connecting it to another function in which the setpoint is replaced by the position of the mouse. The setpoint will change each time user click the left (or right, depends on the settings) mouse button on certain part of the graphical interface. The plant will then be directed to the next supposed control position, and all of these could be available for observation in the graphical interface. Similarly, It is possible to add other mouse events in RTLab such as mouseup, mousedown, mousemove, etc, by modifying the corresponding code in .cpp file.

(5) Separate plugin windows in RTLab.

User can have as many windows as required in RTLab plugin. Everything is this project is in one window, however. Include Wtype_TopLevel in Wflags if a top-level window is needed, or alternatively make the window's parent '0' (or root level). If the user wants the plugin window to also have an entry in the 'Window' menu of DAQsystem, call these methods on the daq_system for instance:

```
int  windowMenuAddWindow(QWidget *w); /* returns window id */
void windowMenuRemoveWindow(int window_id);
```

Call the first one when constructing the plugin therefore the window gets an entry in the DAQSystem window menu, and call the second one when removing the plugin so that its entry is removed from the window menu.

(6) Encoders in RTLab.

The DAQ card subdevices used in RTLab to input and output voltages currently are analogue input and output channels. It is also possible to use encoders (provided by some cards) instead. The encoders are used to measure positions of moving axes in a machine by counting pulses generated by the motion. The software transforms these pulses into linear or rotational displacement. The difference between occupying encoders and analogue input subdevices is the encoder reduces the dead zone effect when plant (such as servo) is moving (or rotating). By use of encoders, user can not only reduce the dead zone effect but also acquire additional measurements. This has not been included in the updated version however it seems to be in progress.

Method (1), (2)and (6) are generally ongoing by other developers and method (3), (4) and (5) are the ways in which to take the work of this thesis further.

References

1. Franklin Gene F., Powell J.David and Emami-Naeini Abbas, "Feedback Control of Dynamic Systems", 3rd, Addison-Wesley Publishing Company Inc., 1994, Massachusetts, USA.

2. Welsh Matt and Kaufman Lar, "Running Linux", 2nd Ed, O'Reilly & Associates Inc., 1996, California, USA.

3. Dutton Ken, Thompson Steve and Barraclough Bill, "The Art of Control Engineering", Addison Wesley Longman Limited, 1997, Massachusetts, USA.

4. "Posix Threads Programming", training material from Lawrence Livermore National Laboratory, http://www.llnl.gov/computing/tutorials/workshops/workshop/pthreads/MAIN.html, last visited 21 September 2004.

5. Salzman Peter Jay and Pomerantz Ori, "The Linux Kernel Module Programming Guide", 2001, http://www.faqs.org/docs/kernel/, last visited 20 June 2005.

6. "DIAPM RTAI Programming Guide 1.0", Lineo Inc., 2000, Utah, USA.

7. "MATLAB—High-Performance Numeric Computation and Visualization

Software (For UNIX Workstations) User's Guide", The MATH WORKS, Inc., 1992, Massachusetts, USA.

8. "SIMULINK—A Program for Simulating Dynamic Systems (For the X Window System^{TM})", The MATH WORKS, Inc., 1992, Massachusetts, USA.

9. Hekman Jessica Perry, "LINUX IN A NUTSHELL", O'Reilly & Associates Inc., 1997, California, USA.

10. "DIAPM RTAI – Beginner's Guide", The RTAI Development Team, 2002, http://www.aero.polimi.it/~rtai/documentation/articles/guide.html, last visited 19 June 2005.

11. Schleef David, Hess Frank and Bruyninckx Herman, "Comedi Documentation—The Control and Measurement Device Interface handbook", 2003, http://www.comedi.org/doc/index.html, last visited 20 June 2005.

12. Culianu Calin, "RTLab_plugin.h", Retrieved from RTLab distribution package, http://www.rtlab.org/download.jsp, last visited 20 June 2005.

13. Tan Haoqiang, "C Language Programming", Tsing Hua University Press, 1988, Beijing, PRC.

14. Zou Siyi, "Linux Designing and Application", Tsing Hua University Press, 2002, Beijing, PRC.

15. Hu Shousong, "Automatic Control Theory", Defense Industry Press, Beijing, 1998, PRC.

16. Yu Changguan, "Modern Control Theory", Haerbin Polytechnic University press, 1998, Haerbin, PRC.

17. Henderson Bryan, "Linux Loadable Kernel Module HOWTO", 2003, http://www.tldp.org/HOWTO/Module-HOWTO/, last visited 20 June 2005.

18. Apkarian Jacob, "A Comprehensive and Modular Laboratory for Control Systems Design and Implementation", Quanser Consulting Inc., 1994, 1997, Ontario, USA.

19. Culianu Calin and Christini David J., "Real-Time Experiment Interface System: RTLab", 2002, http://www.rtlab.org/NEBC_2003_Paper.pdf, last visited 18 June 2005.

20. "Rotary Flexible Link with SRV02", Quanser Consulting Inc., Ontario, USA.

21. Culianu Calin, "How to Install RTLab", Retrieved from RTLab distribution package, 2002, http://www.rtlab.org/download.jsp, last visited 20 June 2005.

22. Culianu Calin, "How to Compile RTLab", Retrieved from RTLab distribution package, 2002, http://www.rtlab.org/download.jsp, last visited 20 June 2005.

23. "RTAI_KCOMEDI_LXRT", 2002,

http://cvs.rtai.org/index.cgi/etna/comedi_lxrt/README?rev=1.12, last visited 06 August 2003.

24. Dozio Lorenzo and Mantegazza Paolo, "RTAI-Lab", 2002, http://cvs.rtai.org/index.cgi/stromboli/rtailab/README?rev=1.10&content-type=text/x-cvsweb-markup, last visited 06 August 2004.

25. Dozio E. Bianchi, L. and Mantegazza P., "A Hard Real Time Support for LINUX", 2002. Retrieved from RTAI distribution package, http://download.gna.org/rtai/ , last visited 20 June 2005.

26. "AD512 data acquisition cards user's manual", HUMUSOFT s.r.o., 1997, Czech Republic.

27. "MultiQ-3™ Programming Manual", Quanser Consulting Inc., Ontario, USA.

28. Bird Tim, "Comparing Two Approaches to Real Time Linux", 2000, http://www.linuxdevices.com/articles/AT7005360270.html, last visited 19 June 2005.

29. Ripoll Ismael, "RTLinux versus RTAI", 2002, http://bernia.disca.upv.es/rtportal/comparative/rtl_vs_rtai.html, last visited 20 June 2005.

30. Johnson Michael K.,"Linux Information Sheet", 1998, http://www.tldp.org/HOWTO/INFO-SHEET.html, last visited 20 June 2005.

31. "The Makefile", webpage from Opus Software Inc., http://www.opussoftware.com/tutorial/TutMakefile.htm, last visited 20 June 2005.

32. Schleef David, "Hardware_Driver.HOWTO", http://www.comedi.org/download.php. Retrieved from Comedi distribution package. Last visited 20 June 2005.

33. The trolltech, "Product Overview", http://www.trolltech.com/products/index.html, last visited 20 June 2005.

34. Hadi Saadat, "EE-479 Digital Control System Project 1 Flexible Link", http://people.msoe.edu/~saadat/1%20Flexible_Link_Project.pdf, last visited 19 June 2005.

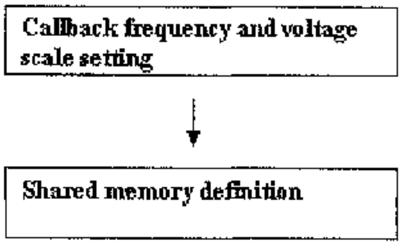
Appendices

Appendix A

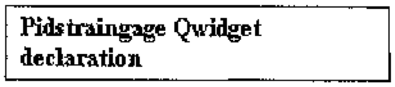
Block Diagram and pseudo code of the programs for SIMO-PID control project (Pidstraingage) and SIMO-LQR (Lqrstraingage) control project.

Pidstraingage program:

pidstraingage.h:

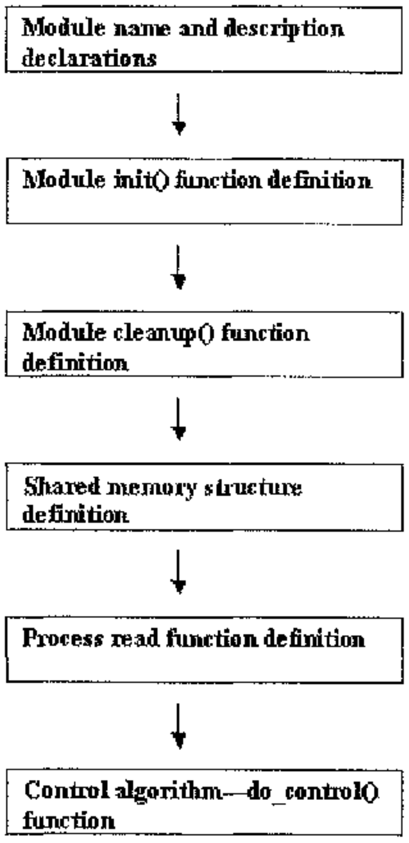


pidstraingage_private.h:



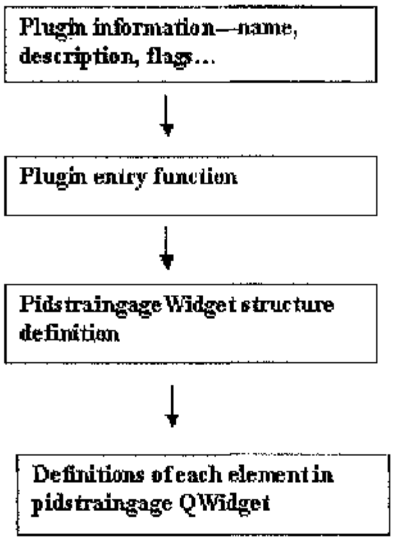
Appendix A

pidstraingage.c:


Module name and description declarations
Module init() function definition
Module cleanup() function definition
Shared memory structure definition
Process read function definition
Control algorithm—do_control() function

Appendix A

pidstraingage.cpp:


Plugin information—name, description, flags...
Plugin entry function
PidstraingageWidget structure definition
Definitions of each element in pidstraingage QWidget

Pseudo code of do_control() function (in pidstraingage.c):

```
/**
 * This plugin does the following:
 *
 * A Proportional + integral + derivative controller
 * Kernel code:
 *      1. Generates different waves with the counterpart periods and
amplitudes.
 *     2. Reads ADC channel shm->ai_chan and calls the voltage 'y'.
 *     3. Computes u = k*(w-y) + u/(1+sTi) - sTd/(1+sTf) y.
 *     4. Writes u to DAC channel shm->ao_chan.
 *
 * GUI (pidstrain_gage.cpp):
 *     Simple GUI to change parameters on-line
 */

static void do_control (MultiSampleStruct * m)
{
   constants
```

Appendix A

```
Declare phase length, sampling rate, and amplitude of Square wave.

Check if the plugin is ready to run

Check the operation wish to perform, input wave type selection:
0—Fixed position, designated through graphical interface.
1—Square wave, amplitude and frequency are designated through
graphical interface.
2—wave from signal generator, modify wave parameter via adjusting
signal generator


If (the input is a square wave
Get the phase length and frequency of the wave
Compute the desired voltage value
  }

Else if (the input is signal from signal generator){
Sample from signal generator

Check if the input voltage is in the valid range
}

else{
Check if the input voltage is in the valid range
Input signal is a fixed value in relation to the angular position
of the motor servo
}

Read input voltage value from analog input channel

Check if AI monitoring is off or the channel they want to monitor
is not found

Get error value through dividing desired voltage value by input
voltage value

Compute control. Get angular position and angular velocity of motor
servo

Filtering
```

```
  Clip

  Optionally echo desired voltage value back to an AI channel for the
  UI

  Echo angle velocity back to channel 2 so that it can be observed
  in Graphical interface.
  Output control signal

  Save output
}
```

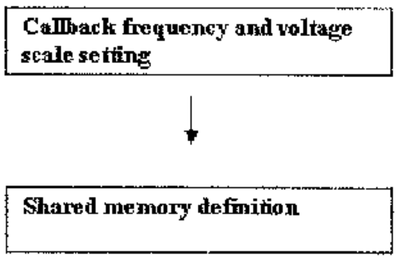
Graphical interface layout setting (in pidstraingage.cpp):

```
  Plugin name, description and entry function declaration
  extern "C" {

  Declare some stuff needed by plugin engine, these symbols are read
  by libdl/dlsym()*/

  ds_plugin_ver is DS_PLUGIN_VER

  Flag is Plugin::RequiresRTLab

  Plugin name is PLUGIN_NAME

  Plugin description is
  "A Proportional + integral + derivative controller\n"
  "Kernel code:\n"
  "    1. Generates different waves with the counterpart periods and
amplitudes.\n"
  "   2. Reads ADC channel shm->ai_chan and calls the voltage 'y'.\n"
  "   3. Computes u = k*(w-y) + u/(1+sTi) - sTd/(1+sTf) y.\n"
  "   4. Writes u to DAC channel shm->ao_chan.\n"
  "\n\n"
  "GUI (pidstrain_gage.cpp):"
  "    Simple GUI to change square  wave and PID parameters on-line",
  * author = "Xiaoyu Duan",
  * requires =
     "piddxy.o be loaded into the kernel. P+I+D control";

  Plugin * entry(QObject *o)
  {

/*  This is a top-level widget, and the parent is root */

  Issue warning message if plugin loading failed.

  Show the widget
  }

};

  Store    some    widgets    that    we    need    pointers    to    for
(dis)connectSignals()   and updateStats()
  struct Pidstrain_gageWidgets
{
  Declare all the elements needed in this plugin
};

/*  Plugin constructor */
Pidstrain_gage::Pidstrain_gage(DAQSystem *d)
  : QWidget(d, PLUGIN_NAME, Qt::WType_TopLevel), ds(d)
{
  Attach to shared memory

  Declare new plugin widget

  Build GUI

  Connect Signals

  Set caption name
}

/*  Plugin destructor */
Pidstrain_gage::~Pidstrain_gage()
{
  Set wave_on to 0
  Set wave_type to -1
  Set channel number for signal generator to -1

  Detach shared memory
  Delete plugin widget; just delete the struct, not the actual widget
}

  Return plugin name
  Return plugin description

/*  Setting up the main layout */
void Pidstrain_gage::buildGUI()
{
  Create new layout
  Create shared memory controller constant
  Get number of analogue input channels
  Add AO Channel QLabel to the main layout
  Add AI Channel QLabel and QComboBox to the main layout
  Add wave type QLabel and QComboBox to the main layout
  Add wave period QLabel and QSpinBox to the main layout
  Add wave amplitude QLabel and QLineEdit to the main layout
  Add K value QLabel and QLineEdit to the main layout
  Add Ti value QLabel and QLineEdit to the main layout
  Add Td value QLabel and QLineEdit to the main layout
  Add Tf value QLabel and QLineEdit to the main layout
  Add maximum voltage QLabel and QLineEdit to the main layout
  Add minimum voltage QLabel and QLineEdit to the main layout
  Add Echo output wave to DAQSystem AI chan QLabel to the main layout
  Add wave on QcheckBox to the main layout
  Add input channel number for signal generator QLabel and QcomboBox
to the main layout
  Add input channel number for displacement feedback Qlabel and
QcomboBox to the main layout
  Set text in each element
}
/* Connect signals with slots */
void Pidstrain_gage::connectSignals()
{
  Connect each signal to the counterpart slot
}
```

```
/* -----------------------------------------------------------------------
--------
   Slots below set parameters in the shared memory region for
notifying the real-time process
-----------------------------------------------------------------------
------*/
SetAIChannel slot declaration
SetWaveType slot declaration
SetGene slot declaration
SetDisp_inp slot declaration
SetAngle slot declaration
SetPeriod slot declaration
SetAmplitude slot declaration
SetK slot declaration
SetTi slot declaration
SetTd slot declaration
SetTf slot declaration
SetU_max slot declaration
SetU_min slot declaration
SetAIEcho slot declaration
SetAO slot declaration
```

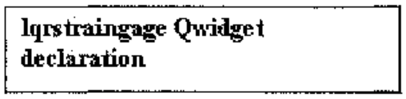
Lqrstraingage program:

lqrstraingage.h:

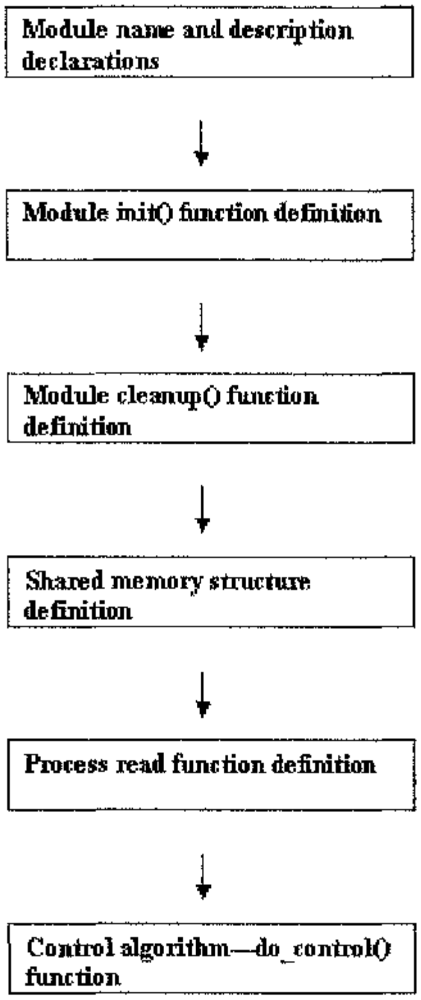


Appendix A

lqrstraingage_private.h:

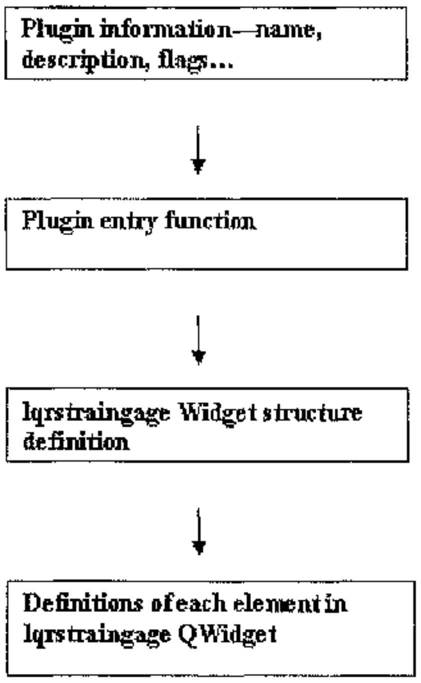


lqrstraingage.c:


Module name and description declarations
Module init() function definition
Module cleanup() function definition
Shared memory structure definition
Process read function definition
Control algorithm—do_control() function

Appendix A

lqrstraingage.cpp:


Plugin information—name, description, flags...
Plugin entry function
lqrstraingage Widget structure definition
Definitions of each element in lqrstraingage QWidget

Pseudo code of do_control() function(in pidstraingage.c):

```
/**
 * This plugin does the following:
 *
 * A Linear Quadratic Regulator controller for straingage
 * Kernel code:
 *     1. Generates different waves with the counterpart periods and
amplitudes.
 *     2. Reads ADC channel shm->ai_chan and calls the voltage 'y'.
 *     3. Computes u = v-kx.
 *     4. Writes u to DAC channel shm->ao_chan.
 *
 * GUI (lqrstrain gage.cpp):
 *     Simple GUI to change parameters on-line
 */
```

Appendix A

```
static void do_control (MultiSampleStruct * m)
{
  /** Formula is u = v-kx
      Status Feedback
  */

   constants
  Declare phase length, sampling rate, and amplitude of Square wave.

  Check if the plugin is ready to run

  Check the operation wish to perform, input wave type selection:
  0—Fixed position, designated through graphical interface.
  1-Square wave, amplitude and frequency are designated through
  graphical interface.
  2—wave from signal generator, modify wave parameter via adjusting
  signal generator


  If (the input is a square wave
  Get the phase length and frequency of the wave
  Compute the desired voltage value
    }

  Else if (the input is signal from signal generator){
  Sample from signal generator

  Check if the input voltage is in the valid range
  }

  else{
  Check if the input voltage is in the valid range
  Input signal is a fixed value in relation to the angular position
  of the motor servo
  }

  Read input voltage value from analog input channel

  Check if AI monitoring is off or the channel they want to monitor
  is not found
  Get error value through dividing desired voltage value by input
  voltage value

  Compute control. Get angular position and angular velocity of motor
  servo

  Filtering

  Clip

  Optionally echo desired voltage value back to an AI channel for the
  UI

  Echo angle velocity back to channel 2 so that it can be observed
  in Graphical interface.
  Output control signal

  Save output
}
```

Graphical interface layout setting (in lqrstraingage.cpp):

```
  Plugin name, description and entry function declaration
  extern "C" {

  Declare some stuff needed by plugin engine, these symbols are read
  by libdl/dlsym()*/

  ds_plugin_ver is DS_PLUGIN_VER

  Flag is Plugin::RequiresRTLab

  Plugin name is PLUGIN_NAME

  Plugin description is
  "A Linear Quadratic Regulator controller\n"
  "Kernel code:\n"
  "    1. Generates different waves with the counterpart periods and
amplitudes.\n"
```

```
  "    2. Reads ADC channel shm->ai_chan and calls the voltage 'y'.\n"
  "    3. Computes u = v-kx.\n"
  "    4. Writes u to DAC channel shm->ao_chan.\n"
  "\n\n"
  "GUI (lqrstrain gage.cpp):"
  "    Simple GUI to change square wave and LQR parameters on-line",
    * author = "Xiaoyu Duan",
    * requires =
       "lqrstrain_gage.o be loaded into the kernel. LQR control";

  Plugin * entry(QObject *o)
  {

/*  This is a top-level widget, and the parent is root */

  Issue warning message if plugin loading failed.

  Show the widget
  }

};

  Plugin * entry(QObject *o)
  {

/*  This is a top-level widget, and the parent is root */

  Issue warning message if plugin loading failed.

  Show the widget
  }

};

  Store some widgets that we need pointers to for
(dis)connectSignals()   and updateStats()
  struct Lqrstrain_gageWidgets
{
  Declare all the elements needed in this plugin
};
```

Appendix A

```
/*  Plugin constructor */
Lqrstrain_gage::Lqrstrain_gage(DAQSystem *d)
  : QWidget(d, PLUGIN_NAME, Qt::WType_TopLevel), ds(d)
{
  Attach to shared memory

  Declare new plugin widget

  Build GUI

  Connect Signals

  Set caption name
}

/*  Plugin destructor */
Lqrstrain_gage::~Lqrstrain_gage()
{
  Set wave_on to 0

  Set wave_type to -1

  Set channel number for signal generator to -1

  Detach shared memory

  Delete plugin widget; just delete the struct, not the actual widget
}

  Return plugin name
  Return plugin description


s/* setting up the main layout */

void Lqrstrain_gage::buildGUI()
{
  Create new layout

  Create shared memory controller constant
  Get number of analogue input channels

  Add AO Channel QLabel to the main layout

  Add AI Channel QLabel and QComboBox to the main layout

  Add wave type QLabel and QComboBox to the main layout

  Add wave period QLabel and QSpinBox to the main layout

  Add wave amplitude QLabel and QLineEdit to the main layout

  Add Ki value QLabel and QLineEdit to the main layout

  Add K2 value QLabel and QLineEdit to the main layout

  Add K3 value QLabel and QLineEdit to the main layout

  Add K4 value QLabel and QLineEdit to the main layout

  Add maximum voltage QLabel and QLineEdit to the main layout

  Add minimum voltage QLabel and QLineEdit to the main layout

  Add Echo output wave to DAQSystem AI chan QLabel to the main layout

  Add wave on QcheckBox to the main layout

  Add input channel number for signal generator QLabel and QcomboBox
to the main layout

  Add input channel number for displacement feedback Qlabel and
QcomboBox to the main layout

  Set text in each element
}

void Lqrstrain_gage::connectSignals()
{
  Connect each signal to the counterpart slot
}
```

Appendix A

```
/* --------------------------------------------------------------------
-------
   Slots below set parameters in the shared memory region for
notifying the
   real-time process
----------------------------------------------------------------------
-------*/

SetAIChannel slot declaration

SetWaveType slot declaration

SetGene slot declaration

SetDisp_inp slot declaration

SetAngle slot declaration

SetPeriod slot declaration

SetAmplitude slot declaration

SetK1 slot declaration

SetK2 slot declaration

SetK3 slot declaration

SetK4 slot declaration

SetU_max slot declaration

SetU_min slot declaration

SetAIEcho slot declaration

SetAO slot declaration
```

Appendix B

Comedi driver for AD512 card:

```
/*
   module/ad512.c
   hardware driver for AD512 data acquisition card.


   This program is free software; you can redistribute it and/or
modify
   it under the terms of the GNU General Public License as published
by
   the Free Software Foundation; either version 2 of the License, or
   (At your option) any later version.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

 */
/*
Driver:ad512.o
Description:[HUMUSOFT] AD512 data acquisition card
Author: Xiaoyu Duan
Status: unknown
Devices:[HUMUSOFT]ad512

*/

#include <linux/comedidev.h>

#include <linux/ioport.h>


#define AD512_SIZE 16
```

Appendix B

```
/*
 * AD512 port offsets
 */
#define AD512_ADLO 0
#define AD512_ADHI 1
#define AD512_DA0LO 0
#define AD512_DA0HI 1
#define AD512_DA1LO 2
#define AD512_DA1HI 3
#define AD512_DACTRL 4
#define AD512_ADSTAT 5
#define AD512_ADCTRL 5
#define AD512_DIN 7
#define AD512_DOUT 7

/*
 * flags for STATUS register
 */
#define AD512_STATUS      0x80
#define AD512_TIMEOUT     30


static int ad512_attach(comedi_device *dev,comedi_devconfig *it);
static int ad512_detach(comedi_device *dev);
static comedi_driver driver_ad512={
      driver_name:      "ad512",
      module:           THIS_MODULE,
      attach:           ad512_attach,
      detach:           ad512_detach,
};
COMEDI_INITCLEANUP(driver_ad512);

struct ad512_private{
      lsampl_t ao_readback[2];
};
#define devpriv ((struct ad512_private *)dev->private)

static int ad512_ai_insn_read(comedi_device *dev,comedi_subdevice *s,
      comedi_insn *insn, lsampl_t *data)
{
      int i,n;
```

```
        int chan;
        unsigned int hi, lo;

        chan = CR_CHAN(insn->chanspec);
        outb((chan | 0x48),
                dev->iobase+AD512_ADCTRL);

        for(i = 0; i < AD512_TIMEOUT; i++) {
                if(!(inb(dev->iobase+AD512_ADSTAT) & AD512_STATUS))
                        break;
        }
        if(i==AD512_TIMEOUT)return -ETIMEDOUT;

        for(n=0;n<insn->n;n++){
                hi = inb(dev->iobase + AD512_ADHI) & 0xff;
                lo = inb(dev->iobase + AD512_ADLO) & 0xff;
                data[n] = (((hi << 8) | lo) + 0x800) & 0xfff;
        }

        return i;
}

static int ad512_ao_insn_read(comedi_device *dev, comedi_subdevice
*s,
        comedi_insn *insn, lsampl_t *data)
{
        int i;
        int chan = CR_CHAN(insn->chanspec);

        for(i=0;i<insn->n;i++){
                data[i]=devpriv->ao_readback[chan];
        }

        return i;
}

static int ad512_ao_insn_write(comedi_device *dev, comedi_subdevice
*s,
        comedi_insn *insn, lsampl_t *data)
{
        int i;
```

```
      int chan = CR_CHAN(insn->chanspec);

      for(i=0;i<insn->n;i++){
            outw((data[i] & 0xfff), dev->iobase+AD512_DA0LO+2*chan);
            outb(0x48, dev->iobase+AD512_DACTRL);

            devpriv->ao_readback[chan] = data[i];
      }

      return i;
}

/*
   options[0] - I/O port
   options[1] - irq
   options[2] - number of encoder chips installed
 */

static int ad512_attach(comedi_device * dev, comedi_devconfig * it)
{
  int result = 0;
  int iobase;
      comedi_subdevice *s;

    iobase = it->options[0];
    printk("comedi%d: ad512: 0x%04x ", dev->minor, iobase);
    if (check_region(iobase, AD512_SIZE) < 0) {
      printk("comedi%d: I/O port conflict\n", dev->minor);
      return -EIO;
    }

      request_region(iobase, AD512_SIZE, "ad512");
      dev->iobase = iobase;

      dev->board_name = "ad512";
      dev->n_subdevices = 2;
      result = alloc_subdevices(dev);
      if(result<0)return result;

      result = alloc_private(dev,sizeof(struct ad512_private));
      if(result<0)return result;
      s = dev->subdevices + 0;
      /* ai subdevice */
      s->type = COMEDI_SUBD_AI;
      s->subdev_flags = SDF_READABLE;
      s->n_chan = 8;
      s->insn_read = ad512_ai_insn_read;
      s->maxdata = 0xfff;
      s->range_table = &range_bipolar5;

      s = dev->subdevices + 1;
      /* ao subdevice */
      s->type = COMEDI_SUBD_AO;
      s->subdev_flags = SDF_WRITABLE;
      s->n_chan = 2;
      s->insn_read = ad512_ao_insn_read;
      s->insn_write = ad512_ao_insn_write;
      s->maxdata = 0xfff;
      s->range_table = &range_bipolar5;

  return 0;
}


static int ad512_detach(comedi_device * dev)
{
  printk("comedi%d: ad512: remove\n", dev->minor);

  if (dev->iobase) { release_region(dev->iobase, AD512_SIZE); }

  return 0;
}
```

Appendix C

test.c:

```
/* This file is a part of an example showing how Qt works in
conjunction with RTLab
 * Copyright (C) 2004 Xiaoyu Duan
 */
/**
 * Example RTLab plugin -- Kernel side.
 * This plugin does the following:
 * Kernel code:
 *   Writes a 2 volts signal to DAC channel 0 when receiving a signal
of any value but 0 from ADC Channel 0.
 *
 * GUI (test.cpp):
 *    Simple GUI to change AI online.
 */
#include "rtlab_kmodule.h"
/** Kernel-side defs for test plugins */
#include "test.h"

#define MODULE_NAME "test"

MODULE_AUTHOR("Xiaoyu Duan");
MODULE_DESCRIPTION(MODULE_NAME ": An example showing how Qt works in
conjunction with RTLab.  Made as an example of a simple RTLab
plugin\n$Id: test.c 18/11/2004 16:12:55Z Xiaoyu $");

int init(void);  /**<  data structures and register callback */
void cleanup(void); /**< Cleanup.. */

static int init_shared_mem(void);

/* The callback called by rtlab core every millisecond... */
static void do_control (MultiSampleStruct * m);
```

```
/* Called whenever the /proc/rtlab/test proc file is read */
static int proc_read (char *, char **, off_t, int, int *, void
*data);

module_init(init);
module_exit(cleanup);

/*-----------------------------------------------------------------------
---------
  Some private 'global' variables...
-----------------------------------------------------------------------
--------*/
/* NB: This module needs at least a 1000 hz sampling rate!
   It will fail if that is not the case at module initialisation,
   and may produce undefined results if that is not the case while
   the module is running. */
static TestShm *shm = 0;
static const int REQUIRED_SAMPLING_RATE = 1000;
static struct proc_dir_entry *proc_ent = 0;
static struct rtlab_comedi_context ctx = {0, 0, 0, 0, 0};
/*-----------------------------------------------------------------------
--------*/

int init (void)
{
  int retval = 0;

  if (rtp_shm->sampling_rate_hz < REQUIRED_SAMPLING_RATE) {
    printk(MODULE_NAME ": cannot start the module because sampling
rate of "
      "rtlab is not %d hz! "MODULE_NAME" *requires* a %d Hz rate "
      "on the RT loop for its own internal simplicity.  The current "
      "rate that rtlab is looping at is: %d",
      REQUIRED_SAMPLING_RATE, REQUIRED_SAMPLING_RATE,
      (int)rtp_shm->sampling_rate_hz);
      return -ETIME;
  }

  if ( (retval = rtp_register_function(do_control)) /* register
callback */
      || (retval = init_shared_mem())
      /** Tell rtlab core to call the callback at this rate.. */
      || (retval = rtp_set_callback_frequency(do_control,
REQUIRED_SAMPLING_RATE))
      /**  the rtlab_comedi_context convenience struct.. */
      || (retval = rtlab_init_ctx(&ctx, COMEDI_SUBD_AO, 0, 0.0,
AREF_GROUND))
      || (retval = rtp_activate_function(do_control)) /* turn callback
on */
      )
     {
       cleanup();
       return retval;
     }

  proc_ent = create_proc_entry(MODULE_NAME, S_IFREG|S_IRUGO,
rtlab_proc_root);
  if (proc_ent)  /* if proc_ent is zero, we silently ignore... */
    proc_ent->read_proc = proc_read;

  return retval;
}

void cleanup (void)
{
  if (proc_ent)
    remove_proc_entry(MODULE_NAME, rtlab_proc_root);

  rtp_deactivate_function(do_control);
  rtp_unregister_function(do_control);
  if (shm)  { rtos_shm_detach(shm); shm = 0; }
}

static int init_shared_mem(void)
{
  shm =
    (TestShm *) rtos_shm_attach (TEST_SHM_NAME,
                                          sizeof(TestShm));
  if (! shm)  return -ENOMEM;

  memset(shm, 0, sizeof(TestShm));
  shm->period_milliseconds = 1000;
  shm->wave_on = 0;
  shm->magic = TEST_SHM_MAGIC;

  return 0;
}

static int proc_read (char *page, char **start, off_t off, int count,
                      int *eof,  void *data)
{
  PROC_PRINT_VARS;

  PROC_PRINT("%s Module\n\n"
             "magic: %x\n"
             "ao_chan: 1\n"
             "ai_chan: 1\n"
             "period_milliseconds: %d\n"
             "wave is on?: %s\n",
             MODULE_NAME,
             shm->magic, shm->period_milliseconds, shm->wave_on ?
"Yes" : "No");
  PROC_PRINT_DONE;
}
/**
 * This function does the following:
 *   Kernel code:
 *   Writes a 2 volts signal to DAC channel 1 when receiving a signal
of any value but 0 from ADC Channel 1.
 *
 *  This function is called by rtlab's core... see
rtp_register_function()
 */
static void do_control (MultiSampleStruct * m)
{
  double vOut, vAI;
  SampleStruct *sample;

  if (!shm->wave_on) return;

  sample = rtlab_get_sample_by_chan(1, m);
  if (!sample) vAI = 0; /* AI monitoring is off
                           OR the channel they want to monitor
                           is not found, so ignore vAI term.. */

  /* our output voltage */
  else vOut = 2866; /* sample is not NULL, output 2 volts (2 * 2048/5
+ 2047 = 2866 ).   */

  rtlab_data_write(&ctx, vOut);


}
```

test.cpp:

```
/*
 * This file is a part of an example showing how Qt works in
conjunction with RTLab
 *
 * Copyright (C) 2004 Xiaoyu Duan
 */
#include <qwidget.h>
#include <qlayout.h>
#include <qlabel.h>
#include <qfont.h>
#include <qstring.h>
#include <qgroupbox.h>
#include <Qtimer.h>
#include <qcheckbox.h>
#include <qbuttongroup.h>
#include <qradiobutton.h>
#include <qspinbox.h>
#include <qcombobox.h>
#include <qhbox.h>
#include <qvbox.h>
#include <qlineedit.h>
#include <qvalidator.h>
#include <qscrollbar.h>
#include <qmenubar.h>
#include <qpopupmenu.h>
#include <qmessagebox.h>
```

```
#include <qfiledialog.h>
#include <qfile.h>
#include <Qtextstream.h>
#include <qpen.h>
#include <qcolor.h>

#include <set>

#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>

#include "common.h"
#include "daq_system.h"
#include "shm.h"
#include "ecggraph.h"
#include "plugin.h"
#include "exception.h"
#include "tempspooler.h"
#include "test.h"

#include "test_private.h"
#include "searchable_combo_box.h"
#include "plugin_utility.h"

#define RCS_VERSION_STRING  "$Id: test.cpp 18/11/2004 16:37:22Z
Xiaoyu $"

using namespace std;

#define PLUGIN_NAME "Testing Program"

extern "C" {

  /* Stuff needed by plugin engine... these symbols are read by
libdl/dlsym()*/
  int ds_plugin_ver = DS_PLUGIN_VER;
  int flags = Plugin::RequiresRTLab;

  const char * name = PLUGIN_NAME,
             * description =
  "A simple reference plugin that does the following:\n"
  "Kernel code:\n"
  "  Writes a 2 volts signal to DAC channel 0 when receiving a signal
of any value but 0 from ADC Channel 0.\n"
  "GUI (test.cpp):"
  "   Simple GUI to change AI online.",
    * author = "Xiaoyu Duan.",
    * requires =
       "test.o be loaded into the kernel.  Analog input and output.";
  const char * kmodules = "test.o";

  Plugin * entry(QObject *o)
  {

    /* Top-level widget.. parent is root */
    DAQSystem *d = dynamic_cast<DAQSystem *>(o);

    Assert<PluginException>(d, PLUGIN_NAME " Load Error",
                            "The " PLUGIN_NAME " plugin can only be
used in "
                            "conjunction with daq_system!  Sorry!");
    Test *g = new Test(d); ;
    if (g) g->show();
    return g;
  }

};

/* Store some widgets that we need pointers to for
(dis)connectSignals()and updateStats() */
struct TestWidgets
{
  QSpinBox *period_milliseconds;
  QCheckBox *wave_on;
};

Test::Test(DAQSystem *d)
```

```
  : QWidget(d, PLUGIN_NAME, Qt::WType_TopLevel), ds(d)
{
  // attach to Shm
  shm = PluginUtility::shmAttach<TestShm>(TEST_SHM_NAME,
                                          TEST_SHM_MAGIC,
                                          PLUGIN_NAME,
                                          "test.o");

  widgets = new TestWidgets;

  buildGUI();
  connectSignals();

  setCaption(name());
}

Test::~Test()
{
  shm->wave_on = 0;
  PluginUtility::shmDetach(shm);
  delete widgets; /* just deletes the struct, not the actual widgets
*/
}

const char *Test::name() const { return ::name; }
const char *Test::description() const { return ::description; }


void Test::buildGUI()
{
  QGridLayout *layout = new QGridLayout(this);

  const ShmController & rtlab_shm = ds->shmController();
  int n_ai_chans =
rtlab_shm.numChannels(ComediSubDevice::AnalogInput), i;

  layout->addWidget(new QLabel("AO Channel:", this), 0, 0);
  layout->addWidget(new QLabel("1",  this), 0, 1);
  layout->addWidget(new QLabel("AI Channel to monitor:", this), 1,
0);
  layout->addWidget(new QLabel("1",  this), 1, 1);
  layout->addWidget(new QLabel("Sq. wave period (ms):", this), 2, 0);
  widgets->period_milliseconds = new QSpinBox(10, 10000, 1, this);
  layout->addWidget(widgets->period_milliseconds, 2, 1);

  widgets->wave_on = new QCheckBox("Analog output enabled", this);
  layout->addMultiCellWidget(widgets->wave_on, 3, 3, 0, 1);

  widgets->period_milliseconds->setValue(shm->period_milliseconds);
  widgets->wave_on->setChecked(shm->wave_on);
}

void Test::connectSignals()
{
  connect(widgets->period_milliseconds, SIGNAL(valueChanged(int)),
          this, SLOT(setPeriod(int)));
  connect(widgets->wave_on, SIGNAL(toggled(bool)), this,
SLOT(setAO(bool)));
}

/* -----------------------------------------------------------------------
-------
   Slots below set parameters in the shared memory region for
notifying the
   real-time process
------------------------------------------------------------------------
------*/
void Test::setPeriod(int period)
{
  if (period > 0)  shm->period_milliseconds = period;
}
void Test::setAO(bool on)
{
  shm->wave_on = on;


}
```

Mainpro.c:

```
/*
  COPYRIGHT (C) 2003 Xiaoyu Duan (xduan@mech.gla.ac.uk)
  This library is free software;   you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.
  */
  #include <stdio.h>
  #include <math.h>
  #include <unistd.h>
  #include <sys/types.h>
  #include <sys/mman.h>
  #include <sys/stat.h>
  #include <fcntl.h>
  #define KEEP_STATIC_INLINE   /* undef this to use libcomedilxrt */
  #include <rtai_lxrt_user.h>
  #include <rtai_lxrt.h>
  #include <rtai_comedi_lxrt.h>  /* comment this when using
  libcomedilxrt */
  #include <rtai_comedi_lxrtlib.h>  /* include this to use
libcomedilxrt */
  #define ts 0.002        /* time interval */
  #define kp 0.025
  #define kd -0.007
  #define k  35.2
  #define s  0.00284      /* sensitivity of motor servo */
  #define f  25           /* cutoff frequency */
  #define SAMPLE_LENGTH  10000
  #define SECOND  1000000000
  #define MSG_DELAY 1000

  double t = 0.00;
  double theta = 0.00, theta_desired = 0.00;
  double theta_fp = 0.00, theta_f = 0.00;
  double theta_fd = 0.00, theta_p = 0.00, volts = 0.00;
  int second;
  char *subdevice_types[] = {
         "unused",
         "analog input",
          "analog output",
          "digital input",
          "digital output",
          "digital I/O",
          "counter",
          "timer",
          "memory",
          "calibration",
          "processor"
  };

  double control_algrithm(double volts)
  {
  double af, bf, omega, volts_in, volts_out;
  omega = 2*M_PI*f;
  count = 0;
  volts_in = (volts-4095)*5.00/4096; /* convert binary value to
actual voltage value */
  theta = volts_in/s;   /* convert voltage to degree */
  theta_fp = theta_f;   /*save previous filtered output */
  theta_p = theta;   /* save old sample */
  af = (omega*ts)/(omega*ts+2);
  bf = (omega*ts-2)(omega*ts+2);
  theta_f = af*(theta_f+theta_p)-bf*theta_fp;  /* filtering */
  if (theta_f>176.00){
    theta_f = 176.00;
  }else if(theta_f<-176.00)
   theta_f = -176.00;
  }
  theta_fd = (theta_f-theta_fp)/ts;
  /*  theta_desired = 5.00*sin(M_PI*t/5); give desired theta here,
use this if sine wave is desired. */
  t = t+ts;
  volts_out = kp*(k*theta_desired theta_f)-kd*theta_fd; /* compute
output voltage value */
  if(volts_out>5.00)
     volts_out = 5.00;
  else if(volts_out<-5.00)
     volts_out = -5.00;
  volts_out = (volts*2048/5.00)+2047;  /* convert it back to binary
value*/
```

```
  return volts_out;
   }

  int main(int argc, char **argv)
  {
  RT_TASK *comedi_task;
  void *dev;
  int i, n_subdevs, type;
  double theta_temp;
  int subdev_ai, subdev_ao, subdev_dio;
  lsampl_t data;
  int n, nch, semcnt;
  char name[50];
  SEM *sem;
  Unsigned long mbx_name = nam2num("MBX");  /* use mail box to
achieve Interprocess communication here */
  MBX *mbx;

  /* set schedule priority */
  struct sched_param mysched;
  if (sched_setscheduler(0, SCHED_FIFO, &mysched)==-1){
  puts("ERROR IN SETTING THE SCHEDULER UP");
  perror("error");
  exit(0);
  }

  if(!comedi_task = rt_task_init(nam2num("COMEDI", 1, 0, 0))){
printf("CANNOT INIT COMEDI TASK\n");
  exit(1);
  }

  mbx = rt_get_adr(mbx_name);  /* recognize mailbox */
  sem = rt_sem_init(nam2num("SEM"), 0);
  rt_set_oneshot_mode();
  start_rt_timer(0);
  second = nano2count(SECOND);
  mlockall(MCL_CURRENT | MCL_FUTURE);
  dev = comedi_open("/dev/comedi0");
  printf("\n OVERALL INFO:\n");
  printf("   Version code : 0x%06x\n",
comedi_get_version_code(dev));
```

```
  rt_comedi_get_board_name(dev, name);
  printf("   Board name   : %s\n", name);
  rt_comedi_get_driver_name(dev, name);
  printf("   Driver name  : %s\n", name);
  printf("   Number of subdevices : %d\n", n_subdev =
comedi_get_n_subdevices(dev));

  for (i = 0; i < n_subdevs; i++) {
  printf("\n Subdevice : %d\n", i);
  type = comedi_get_subdevice_type(dev, i);
  printf(" Type : %d (%s)\n", type, subdevice_types[type]);
  printf(" Number of channels : %d\n", nch =
comedi_get_n_channels(dev, i));
  printf(" Maxdata : %d\n", comedi_get_maxdata(dev, i, 0));
  printf(" Number of ranges : %d\n", comedi_get_n_ranges(dev, i,
0));
  }
  subdev_ai = comedi_find_subdevice_by_type(dev, COMEDI_SUBD_AI, 0);
  subdev_ao = comedi_find_subdevice_by_type(dev, COMEDI_SUBD_AO, 0);
  subdev_dio = comedi_find_subdevice_by_type(dev, COMEDI_SUBD_DIO,
0);

  printf("\n Start control ...");
  fflush(stdout);
  comedi_lock(dev, subdev_ai);

  for(n = 0; n<SAMPLE_LENGTH; n++){
  rt_comedi_wait_timed(sem, nano2count(100000), &semcnt);
  rt_mbx_receive_timed(mbx, &theta_temp, sizeof(theta_temp),
  nano2count(MSG_DELAY));  /*receive desired theta value via mailbox
*/
  printf("name: %x, address%p \n", mbx name, mbx);
  printf("RECEIVED THETA_DESIRED = %f", theta_temp);
  theta_desired = theta_temp;
  comedi_data_write(dev, subdev_ao, 0, 0, AREF_GROUND, &data);
  data = control_algrithm(data);
  comedi_data_write(dev, subdev_ao, 1, 0, AREF_GROUND, data);
  rt_sleep(0.002xsecond);
  }

  comedi_data_write(dev, subdev_ao, 0, 0, AREF_GROUND, 2048);
  comedi_data_write(dev, subdev_ao, 1, 0, AREF_GROUND, 2048);
  printf(" OK.\n");
  comedi_unlock(dev, subdev_ai);
  comedi_close(dev);
  rt_sem_delete(sem);
  stop_rt_timer();
  rt_task_delete(comedi_task);
  return 0;
}
```

Mailboxpro.c:

```
/*
COPYRIGHT (C) 2003 Xiaoyu Duan (xduan@mech.gla.ac.uk)

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2 of the License, or (at your option) any later version.
This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.
*/
#include <stdio.h>
#include <stdiolib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/user.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sched.h>

#define KEEP_STATIC_INLINE   /* undef this to use libcomedilxrt */
#include <rtai_lxrt_user.h>
#include <rtai_lxrt.h>
#define MSG_DELAY 2000000000

int main(int argc, char **argv[])
```

```
{
  unsigned long runtsk_name = nam2num("RUNTSK");
  unsigned long mbx_name = nam2num("MBX");
  double theta_desired;
  int count;
  RT_TASK *runtsk;
  MBX *mbx;
  struct sched_param mysched;
  if (sched_setscheduler(0, SCHED_FIFO, &mysched)==-1){
  puts("ERROR IN SETTING THE SCHEDULER UP");
  perror("error");
  exit(0);
  }
  mlockall(MCL_CURRENT | MCL_FUTURE);


  if(!runtsk = rt_task_init( runtsk_name, 0, 0, 0))){
  printf("CANNOT INIT MAILBOXPRO \n");
  exit(1);
  }

  rt_set_oneshot_mode();
  start_rt_timer(nano2count(1000000000);
  if(!(mbx=rt_mbx_init(mbx_name, 1)));
    printf("CANNOT CREAT MAILBOX %/x\n", mbx_name);
    exit(1);
  }
  printf("name: %/x, address:%/x. \n", mbx_name, mbx);
  count=5; /* It is possible to enter desired value for 5 times. */

  while(count){
  printf("PLEASE ENTER DESIRED THETA\n");
  scanf("%lf", &theta_desired);
    rt_mbx_send(mbx, &theta_desired, sizeof(theta_desired));
    printf("DESIRED THETA IS %f \n", theta_desired);
    rt_sleep(nano2count(DELAY));
    count--;
  }

  stop_rt_timer();
  rt_mbx_delete(mbx);
```

```
rt_task_delete(runtsk);
return 0;
}
```