California AHMCT Research Center
University of California at Davis
California Department of Transportation

APPLICATION ISSUES FOR HIGHWAY MAINTENANCE AND CONSTRUCTION - DYNAMICS AND CONTROL OF HYDRAULIC SYSTEMS AND DEPLOYMENT OF LINUX BASED SYSTEMS

Xin Feng
Bahram Ravani
Steven A. Velinsky

AHMCT Research Report
UCD-ARR-04-06-30-02

Final Report of Contract
IA 65A0049 Task Order 01-5

June 30, 2004

This report has been prepared in cooperation with the State of California, Business and Transportation Agency, Department of Transportation and is based on work supported by Interagency Agreement 65A0049, Task Order 01-5 through the Advanced Highway Maintenance and Construction Technology Research Center at the University of California at Davis.

ABSTRACT

The Advanced Highway Maintenance and Construction Technology (AHMCT) Research Center has been developing robotic equipment and machinery for highway maintenance and construction operations. It is a cooperative venture between the University of California at Davis and the California Department of Transportation (Caltrans). The research and development projects have the goal of increasing safety and efficiency of roadwork operations through the appropriate application of automation solutions. This report describes the outcome of a two-part study aimed at specific issues for highway maintenance and construction application of advanced technologies.

In the first part of the report, we investigate Linux as an operating system for control systems. Linux and its source code are freely available and programmers all over the world contribute to its development. In development for just one decade, Linux now offers first rate graphical desktop and integrated development environments, in addition to its advanced and comprehensive server and network facilities. We focus on using Linux to integrate a control system, for its graphical user interface and its communication advantages. The second part of this research has dealt with the development of methods for generation of smooth trajectories of robotic end-effectors and adding scientific bases to some of the application specific developments in the driver assisted in snow plowing project. Several scientific contributions have been made in robotic motion design and path planning that can be used to program robotic systems for smooth motions in highway applications such as crack sealing or painting of roadway markings.

EXECUTIVE SUMMARY

The Advanced Highway Maintenance and Construction Technology (AHMCT) Research Center has been developing robotic equipment and machinery for highway maintenance and construction operations. It is a cooperative venture between the University of California at Davis and the California Department of Transportation (Caltrans). The research and development projects have the goal of increasing safety and efficiency of roadwork operations through the appropriate application of automation solutions. This report describes the outcome of a two-part study aimed at specific issues for highway maintenance and construction application of advanced technologies.

In the first part of the report, we investigate Linux as an operating system for control systems. Linux and its source code are freely available and programmers all over the world contribute to its development. In development for just one decade, Linux now offers first rate graphical desktop and integrated development environments, in addition to its advanced and comprehensive server and network facilities. We focus on using Linux to integrate a control system, for its graphical user interface and its communication advantages. We develop some ready-to-use C++ classes for serial port communication, which feature simplicity and efficiency and are multithreaded, event-driven and platform-independent. We also demonstrate how to develop programs with an interactive graphical user interface for testing serial port devices and integrating these control devices into a complete control system. We also investigate and demonstrate remote robot control over the Internet via Web services and grid computing. This study confirms that Linux is a valuable and fully capable operating system for control. Examples and experiments are given in this report including full source code and tutorials, with which one can easily start deploying Linux.

The second part of this research has dealt with the development of methods for generation of smooth trajectories of robotic end-effectors and adding scientific bases to some of the application specific developments in the driver assisted in snow plowing project. Several scientific contributions have been made in robotic motion design and path planning that can be used to program robotic systems for smooth motions in highway applications such as crack sealing or painting of roadway markings. Design principles have also been developed for implementation of a mechatronics type magnetic sensing technology to replace analogue magnetometers for sensing of lateral positions of vehicles in snow plowing operations. In addition, a stochastic driver model has been developed that would be suitable for modeling driver behavior in snow and ice operations. This model can be used as a basis for simulating driver steering in snow plowing operations and can be used for design of driver assist systems.

TABLE OF CONTENTS

DISCLAIMER / DISCLOSURE

The research reported herein was performed as part of the Advanced Highway Maintenance and Construction Technology (AHMCT) Research Center, within the Department of Mechanical and Aeronautical Engineering at the University of California, Davis and the Division of Research and Innovation of the California Department of Transportation. It is evolutionary and voluntary. It is a cooperative venture of local, state and federal governments and universities.

The contents of this report reflect the view of the author(s) who is (are) responsible for the facts and accuracy of the data presented herein. The contents do not necessarily reflect the official views of the STATE OF CALIFORNIA or the FEDERAL HIGHWAY ADMINISTRATION and the UNIVERSITY OF CALIFORNIA. This report does not constitute a standard, specification, or regulation.

Part I - Deploying Linux for Highway Equipment Development

1. Introduction

With its combination of power, solid stability, low cost (free) and source code openness, Linux has a bright future as a popular operating system. The Advanced Highway Maintenance and Construction Technology (AHMCT) Research Center has been deploying the newest operating systems (OS), such as Windows NT, Windows 2000, and QNX, to make our prototype equipment reliable, cost effective and user friendly (Feng, et al., 1998, 2002, 2004; Bennett et al., 2003; Velinsky et al., 2003). Despite its advantages, compared to the current mainstream Windows OS, the deployment of Linux requires more effort due to its deficient documentation and development environment. So far, we have yet to give serious attention to this new trend in OS technology. As such, in this research project, we conduct a thorough study of Linux and its implementation in our equipment development – in both desktop and embedded environments.

1.1 Linux Operating System

Linux is a UNIX-like, 32-bit operating system. It has the following unique strengths:

- It is made from scratch and based on the most recent achievements of operating system technology. Therefore, it does not have the inherited problems of legacy operating systems.
- It has a modular, scalable and microkernel architecture, which allows it to work with a wide range of processors (Intel, Alpha, PowerPC…), peripherals and other hardware.
- It has thorough memory protection and is currently the most reliable and stable OS for PC and one of the most stable OS in general.
- It is the most network/Internet friendly OS.
- It is free and all the source codes are open.
- It is now supported by large corporations such as IBM.

Linux was born as a networking OS and grew with the Internet. It provides all the necessary networking and Internet facilities to allow programmers throughout the world to contribute to it. Unlike other Unix operating systems that require expensive, proprietary hardware to run on, Linux runs on standard PC hardware and others such as PowerPC, Alpha, etc.

1.2 Real-time and Embedded Linux

Today's complex control systems, such as those found in the AHMCT Research Center, demand the dependability, responsiveness, and guarantees of hard real-time capabilities. For some embedded systems, Linux is usable just as it stands. It presents a good alternative to Windows or DOS for applications without real-time requirements or for those with real-time requirements that are met with dedicated hardware or a dedicated processor.

Sufficient progress has been made in the implementation of Linux as a real-time system. There are two general approaches to real-time Linux: the POSIX approach and the kernel approach. The former method adds to Linux the POSIX.1b (or IEEE 1003.1b) real-time extensions and the latter implements a simple real-time kernel underneath the operating system (Epplin, 1997).

1.3 Linux Application Development

One of the major facts that made Windows dominate PC OS is its huge developer base. Integrated development tools and their documentation are numerous; e.g., Visual Basic, Visual C/C++, Delphi, and Boland C Builder, to name but a few. Without adequate applications or tools for building applications, an OS can never be of any practical value.

In this research, we examine the programming tools available for Linux, including C, C++ compilers, and toolkits such as Qt and scripting languages such as Perl, Awk and Sed. Most of these tools are freely downloadable from the Internet. Ultimately, we intend to provide AHMCT students and engineers with the most powerful and easy to use development environment as well as some templates from which to start.

1.4 Linux Driver Development

Connecting external hardware to DOS-based PCs usually requires no more than a simple Basic program. In comparison, connecting Linux-based computers to external hardware for data logging and/or control can be problematic because the operating system requires special device drivers to connect any device to the computer. Furthermore, developing a device driver for an operating system is always very difficult (Hassan, 2001).

In this research, we avoid this difficulty by using stand-alone control devices that use serial port communication or a network interface to connect to a host computer. Later in this report, some sample code for serial port and TCP/IP communication will be provided.

1.5 Embedded Linux

Linux is not just for standard desktop PCs. It is modular, very scalable and flexible. It is also very efficient. A floppy disk is adequate to run a Linux router for DSL or cable modem, for example, see http://www.zelow.no/floppyfw. A full Linux kernel takes only a few hundred kilobytes of memory. This makes Linux very attractive for embedded devices that require small footprints (Epplin, 1997). The gum-sized single board computer shown in Figures 1-1 and 1-2 runs Linux, as do the Sharp Zaurus series PDAs (see Fig. 1-3).

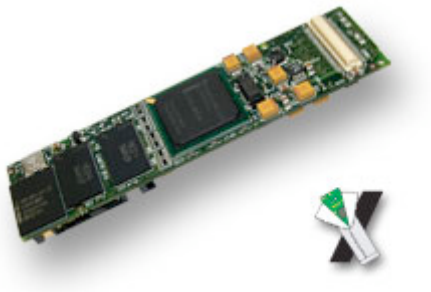

Figure 1-1 Tiny Gumstix Runs Linux

Figure 1-2 Gumstix in Box

Figure 1-3 Sharp Zaurus Linux Handheld

A good Embedded OS is very important to us, as most of our equipment is truck-based. In this research, we examine the possibility of embedding Linux into our robot controllers and other embedded controllers, together with the real-time Linux effort described earlier.

In summary, we investigate Linux as a cost effective, high performance and reliable platform for our equipment development, both for graphical user interface purposes and real-time control.

2. Linux Operating System

Our research started with the study of Linux as a general operating system. We investigated Linux as a server and a desktop, and its implementation to our research center as an alternative OS to Windows 2000. We examine Linux's advantages in this regard by comparing the new set of Linux servers to our existing Windows 2000 servers.

2.1 History

Linus Torvalds started the kernel of Linux in 1991 when he was a student at the University of Helsinki in Finland. He released the initial version for free on the Internet for others to participate in its development. The initial version was inspired from the Minix operating system – a free and very simple Unix that runs on PCs. Linux was started as a Unix for 386-based PCs and it was successful because:

- At that time, Unix was already a mature operating system.
- PC hardware was low cost and the technology and the market were ready to explode.
- The Internet was about to boom.

Unix is one of the most popular operating systems worldwide. It was the first OS that was written entirely in C, not the hard-to-work-with assembly languages. Most students study and use Unix when they are in school and therefore they typically support Unix following graduation to a work environment.

However, most Unix systems run on expensive proprietary hardware. Some Unix run on PC hardware, but the OS costs more than the hardware, which is thus self-defeating. Although most Unix distributors give away Unix to universities for free, the cost is high for businesses. Unix is very difficult to manage and requires very experienced administrators. Again, this is not a big problem for universities, as computer science students are a source of such expertise. At the time of Linux' inception, PC hardware was very cheap, but the major operating systems for them, the Windows 95 or Windows NT, had not yet matured, and they crashed frequently. Therefore, porting Unix to PC hardware was a natural thought. BSD (Berkeley Unix) is one of those efforts and it played an important role in Linux – most Linux utilities are ported from BSD.

The Internet has been the primary contributor to the success of Linux. On one hand, Linux deploys over the Internet to gain developers worldwide; on the other hand, Linux provides a reliable and low cost (free) platform to boost the Internet and services like HTTP, FTP, email, DNS, etc. With such a cross movement, both Linux and the Internet have generated great change in the last ten years or so. Now Linux is a complete Unix clone, able to run anything that Unix can, plus it has Windows-like graphical user interfaces.

2.2 Linux Advantages

Like Unix, Linux is a complete multitasking, multi-user, scalable and stable operating system. It complies with a number of Unix standards, such as IEEE POSIX.1, System V, etc. Much free Unix software can be compiled and run on Linux without any changes.

One of the greatest strengths of Linux is, of course, its networking capability. Linux implements fully TCP/IP networking, including all the network services, such as WWW, FTP, Telnet, NNTP and SMTP. The best of them: routing, firewall, NAT and VPN support is included in the kernel; so Linux can be easily configured as a firewall and router for secure private network, locally or remotely. For example, the floppyfw is one of these projects that allow a router with advanced firewall-capabilities to fit on one single floppy disc (http://www.zelow.no/floppyfw).

The key feature of Linux is the so-called "Open Source" movement. Other than relying upon a single corporation to develop and maintain a software package, Open Source allows anyone to contribute openly to the software. Being able to access the source code, Linux developers can easily fix a bug or a security hole, add a driver to new hardware, or customize Linux to fit a special application such as a top box or DSL router as mentioned above.

The unbeatable feature of Linux: it is free – free to get, free to use, free to work on and free to distribute. Not just the OS is free; all Linux distributions (RedHat, ManDrake, SuSE, etc.) include many free applications from office productive packages, database engines to Internet servers and clients - pretty much everything needed for computing. By installing Linux, PC manufacturers can sell fully functional PCs without having to pay premier fees for a proprietary operating system such as Windows XP.

Before MS Windows 2000 and Apple Macintosh OS X, Linux was the only personal computer operating system that was virtually crash free. Linux machines usually run for months to years without having to be rebooted or reset. In addition to its reliability, Linux is very efficient. A Linux server requires minimum hardware resources to provide first-class network services. Usually a very old 80386 with 16MB memory is adequate to run a 7/24 Linux server smoothly.

3. Linux Server

In order to provide a research and real implementation example, we set up a Linux server as the only server (without Windows 2000) in our new lab at Second Street, Davis, California. There is only one DSL connection in this new location, so we needed a server to share this connection for multiple computers and to make a secure connection to campus. The server needs to provide many challenging network tasks, including routing, firewall, VPN (virtual private network), wireless station, file and printer sharing, etc.

3.1 Gateway and Routing

A DSL router provides high-speed Internet access in this facility. Unlike the computers on the UC-Davis campus that all connect directly to the Internet, this lab has only one Internet connection – a DSL line with only a single static IP address. To allow all the computers in this lab to share this DSL connection, a special router is required. Linux was a pioneer on providing such a special private-to-public gateway and it is still the leader in this field. Compared to a Windows 2000–based router, Linux is far more flexible and reliable. Its routing method, ipchains, is not specific to protocol and network interface and is more advanced than the NAT (network address translation) method used by Windows 2000.

3.2 Firewall

This Linux server provides a firewall to prevent unauthorized access from the Internet to our laboratory facilities. The ipchains that Linux uses for routing has all the capabilities for one to set up either the simplest or most complicated firewall rulings. Due to concerns relative to hackers and viruses, each and every computer must be protected by a firewall before it is connected to the Internet. Windows 2000 does not have this function build-in and third party add-ons are inadequate.

3.3 Virtual Private Network

To connect computers in the noted facility and those on campus, we need a secure network channel. Formerly, a dedicated private line was required, which was very expensive (about $400 per month) and very slow (only 19.9kb or so). With the new VPN technology, a secure channel can be created over the Internet. It is as secure as a real private line (everything that goes through the channel is 128-bit encrypted) and much faster (as fast as the Internet connection speed). And it costs nothing, because it uses the free Internet. By using Linux, even expensive VPN equipment can be saved and an old 80386 will be adequate.

For the implementation of Microsoft's PPTP, there is both a server and client for Linux (http://pptpclient.sourceforge.net/). This Linux server keeps a PPTP connection to one of the servers in our research center on campus and such a connection makes a secure tunnel between the two subnets, i.e., the two subnets are virtually local. This Linux server also runs a PPTP server for both Windows and Linux clients, so our staff and students, at home or on the road, can easily make a VPN connection to the new lab and access resources there, conveniently and securely.

3.4 Wireless Station

A Linux server provides far more flexibility than a dedicated access-point and significantly less expensive. A Linux wireless station can also be easily upgraded and this is important in light of the rapid evolution of wireless technology.

Linux's kernel supports most popular wireless cards such as Lucent and Netgear cards. There is a wireless management toolkit for wireless settings such as wireless modes (infrastructure, point-to-point, etc.), encryption modes (64 or 128 bit), power saving mode, etc, (http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html).

3.5 DHCP Server

By running a DHPC server on this Linux server, all the computers in the new lab are network-wise plug-and-play. This significantly reduces the network administration difficulties, as most of these computers are portable. In addition to many other features that Windows 2000 lacks, the Linux DHCP server allows IP addresses to be assigned based on a MAC address and therefore each NIC can keep its own IP without the hassle of a dynamic IP.

3.6 File and Printer Sharing

Linux can share files and printers with Windows systems through the SAMBA service, either as a server or as a client. Linux supports most printers and more printer drivers are added daily.

Another Linux server had been set earlier mainly as a wireless station/gateway for our conference room. Beside, there is at least one Linux desktop in our center, used for MATLAB and other research purposes. Among the many Linux distributions, we found RedHat 6.2 has the best support from the open source society. Therefore, all our Linux PCs deploy RedHat. We have found that Linux can do all and more than Windows 2000 and requires far less system resources. The Linux server in our new lab, for example, is a 486 PC with only 32MB RAM, which cannot run Windows 2000. Linux is also far more reliable and has far less down time than Windows 2000. The newly set system has run continuously for its first month with only a 4-hour down time for a NIC (network interface card) replacement. While a Windows 2000 server costs $500 or more, Linux is free.

These initial Linux experiments and implementations are promising. Linux provides a very economical and powerful alternative solution to the network services our research needs: file sharing, printer sharing, firewall, WWW server, mail server, VPN server, etc. Although it is more difficult to set up, Linux is much more flexible than Windows 2000 and many "how-to" write-ups can be found on websites to help the configurations. In the following sections, we will study Linux as an OS for automatic control.

4. Linux for Control

For automatic control, there are two areas within which Linux can contribute, those being the application of task-specific devices and their integration. In our case, we prefer to the latter, i.e., use commercially available devices to build customizable control systems and use Linux to integrate these devices into a full system. Figure 4-1 shows an example of one of our embedded systems.

4.1 Linux for System Integration

Task-specific control devices, such as I/O devices and motion controllers, require high reliability, which is sometimes difficult to attain, and are time consuming to develop. Developing these low-level hardware devices and the corresponding required real-time Linux OS to use them are the focus of our work. Specifically, our focus is on how to use Linux to integrate these devices to make a solid-stable, high-performance and loyalty-free system (Feng et al., 2002).

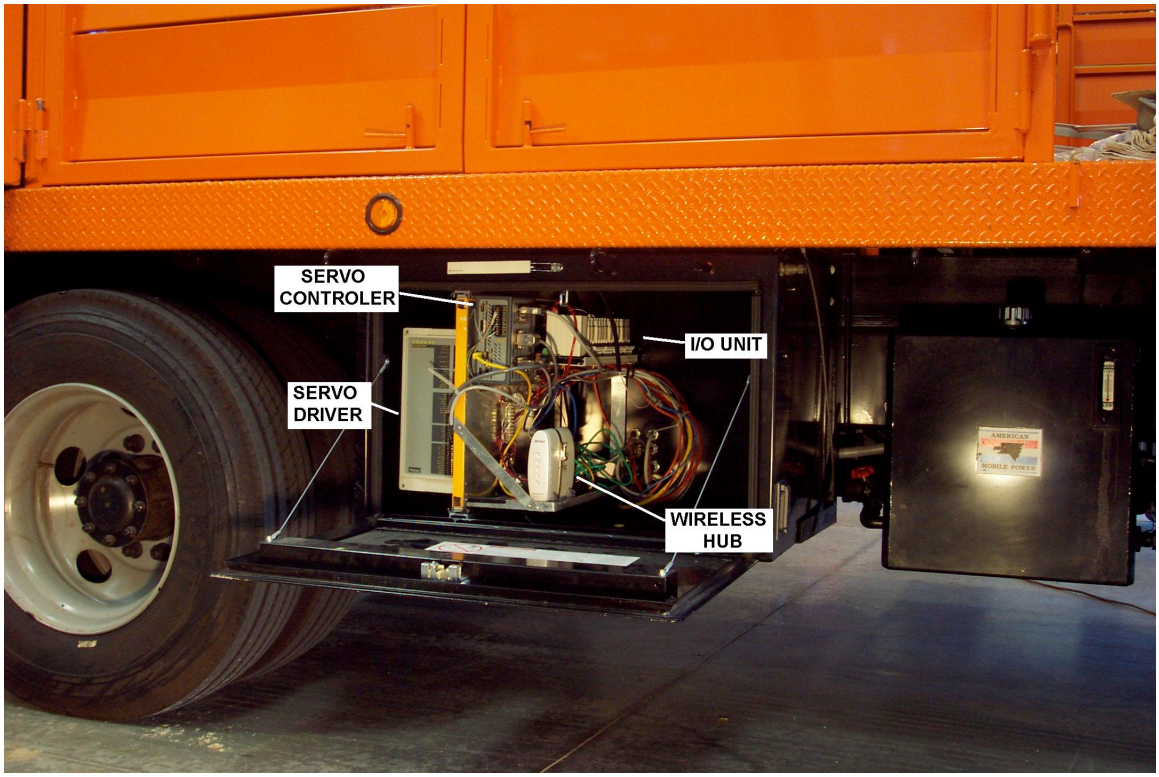

Figure 4-1 An Embedded Control System Integrated by a Wireless Laptop

In order to build a control system, there are two types of hardware to choose from: add-on cards and stand-alone devices. Add-on cards share two major disadvantages:

- The Linux kernel must have a device driver and a user application module that interfaces to the card.

- Add-on cards are not service friendly – the computer must be shutdown and its case must be opened to install them.

There is a special issue with a commercial hardware driver for Linux. Under the open source agreement, manufacturers are required to make the driver's source code open, which is contrary to their desire. Thus, some manufacturers simply cannot provide a Linux driver for their hardware in order to protect their technology. Writing a hardware driver is always very difficult especially for developers who are not the owners of the specific hardware. With stand-alone devices, such as those with serial or Ethernet ports, Linux does not need a driver for them and can communicate with them via the standard serial port device driver or TCP/IP stack. Therefore, stand-alone, especially Ethernet based, I/O devices are very desirable.

Among the stand-alone I/O hardware vendors, Opto22 stands out as the best choice especially for their Linux friendly products (Nakatani, 2001). They now offer both Ethernet and wireless LAN I/O units and provide very good Linux support including demonstration utilities with source code. For motion control, Motion Engineering Inc. (MEI) has the most comprehensive C/C++ software libraries, including Linux support (http://www.motioneng.com). Motion controllers usually use proprietary programming languages, which are hardware specific and are thus not portable. These languages also lack the good structure found in object-oriented languages like C/C++ and Java. Therefore, being able to program MEI motion controllers in C/C++ other than assembly-like proprietary languages is a significant advantage.

4.2 Research Testbed

The testbed, shown in Fig. 4-2, for this research consists of mainly two 400 MHz PCs, one runs Linux and one runs Windows XP. A crossover serial cable connects them for serial port communication tests. In some of the examples in this research, the serial cable is connected between the COM1 and COM2 on the Linux box, so both sending and receiving can be tested conveniently in a single program. A MEI motion controller is installed into the Linux box and it controls a 4-axis TA9000 motion control development system, to demonstrate some real robot control implementations. A video camera is also provided for easier remote development and control.

Figure 4-2 Testbed for Linux Development

5. Linux Software Development

As noted earlier, our focus is on using Linux to integrate a control system. This includes hardware integration and software development. In the last section, we discussed the hardware side, which is relatively simple. Here we will investigate the software development side, which, based on our experience, is far more important.

5.1 KDevelop and Qt

Compared to the software development of Win32 - the most popular platform for PC, there are two main differences:

- Like Unix, much programming can be done with shell scripts. Win32 does offer scripting, but it is still far less powerful and flexible than Linux.
- Unlike other platforms, Linux offers no choices to a C++ programmer when it comes to compilers; gcc (the GNU Compiler Collection) is the only C++ compiler for Linux. For Win32, one can use either Borland C++ or Visual C++ to build applications.

Linux scripts are easy to write compared to writing C/C++ code. This is good for AHMCT students who are primarily mechanical engineering majors. However, scripts are mainly oriented for network configuration and text manipulation and they run slower than binary code, which is a concern with control. Also, scripts are not well-structured languages and should not be used to build large software packages beyond small utilities.

For developing GUI applications, there are two GUI desktop environments to deal with: KDE and GNOME. KDE looks and feels better and it is based on a C++ API rather than a C API, and its development seems to be progressing faster and more smoothly. KDE is built upon Qt, a C++ application framework that was not free to developers and is now free for noncommercial applications.

Qt is a multiplatform C++ GUI toolkit, created by Trolltech in early 1996 (Dalheimer, 2002). It is object-oriented, easily extendable, and allows true component programming. It provides comprehensive functionality for building applications with graphical user interfaces. With Qt, developers can easily design applications for the following platforms:

- MS/Windows – 95, 98, NT4.0, ME, 2000, and XP
- Unix/X11 – Linux, Sun Solaris, HP-UX, Compaq Tru64 UNIX, IBM AIX, SGI IRIX and a wide range of others
- Macintosh – Mac OS X
- Embedded – Linux platforms with frame buffer support.

Qt's multiplatform support is done on the source code level; the code is compiled before running. Java code runs on top of virtual machines, which is usually not as efficient as native binary code. For control, efficiency is a major concern and Qt is favorable because it generates native binary code on different platforms. Qt, as a full package itself, provides a good IDE (integrated development environment) and WYGIWYG (what you get is what you see) programming style, very much like the MS

Visual Basic and the MS Windows Forms found in the latest Visual C++ .NET. Based on Qt, KDevelop provides an even more comprehensive IDE for Linux.

For those who are familiar with MS Visual Studio's IDE, KDevelop IDE looks and feels much the same. It supports development of programs under the various GUI operating environments (KDE, GNOME and Qt) as well as console applications. It also has a decent programmer's editor and integrated debugger. Thus, in this work, we have selected KDevelop to develop our control applications for Linux.

5.2 Desktop Management System for Control

As mentioned, there are mainly two popular desktop management systems for Linux GUI applications: GNOME and KDE. We have found that both are slow and use too much memory. For example, a 32MB/133MHz laptop that runs Win98 smoothly is too slow for KDE application. Thus, we had to identify an efficient desktop environment for control applications. Among other window managers, we have found that the ICEWM is the most efficient, as well as being easy to use and setup with adequate features. Both the KDevelop IDE and the applications developed under it run well within ICEWM. Although it is not as straightforward as KDE, it is still configurable by editing its configuration files. Therefore, ICEWM should be considered as the first choice for control applications.

6. KDevelop and Qt

6.1 KDevelop

The KDevelop IDE, a C/C++ integrated development environment for Unix/Linux, is very much like MS Visual Studio. By following the new project wizard of KDevelop, for example, we successfully developed our first Linux GUI program in just a few minutes – it is just as easy to use as Visual Studio. The KDevelop IDE is publicly available under the GPL and supports KDE/Qt, GNOME, plain C and C++ projects. Most of the features of Visual Studio can be found in KDevelop, such as:

- It has a New Application Wizard – KAppWizard, shown in Fig. 6-1, which generates complete, ready-to-go sample applications. This saves significant time and effort to get a new project started.
- It is integrated with all the development tools needed for C++ programming, including compiler, linker, KDbg, automake, autoconf.
- It allows class browsing and file management.
- It has a built-in dialog editor for easy creation of user interfaces.
- It has a class generator for creation of new classes to be integrated into the current project.

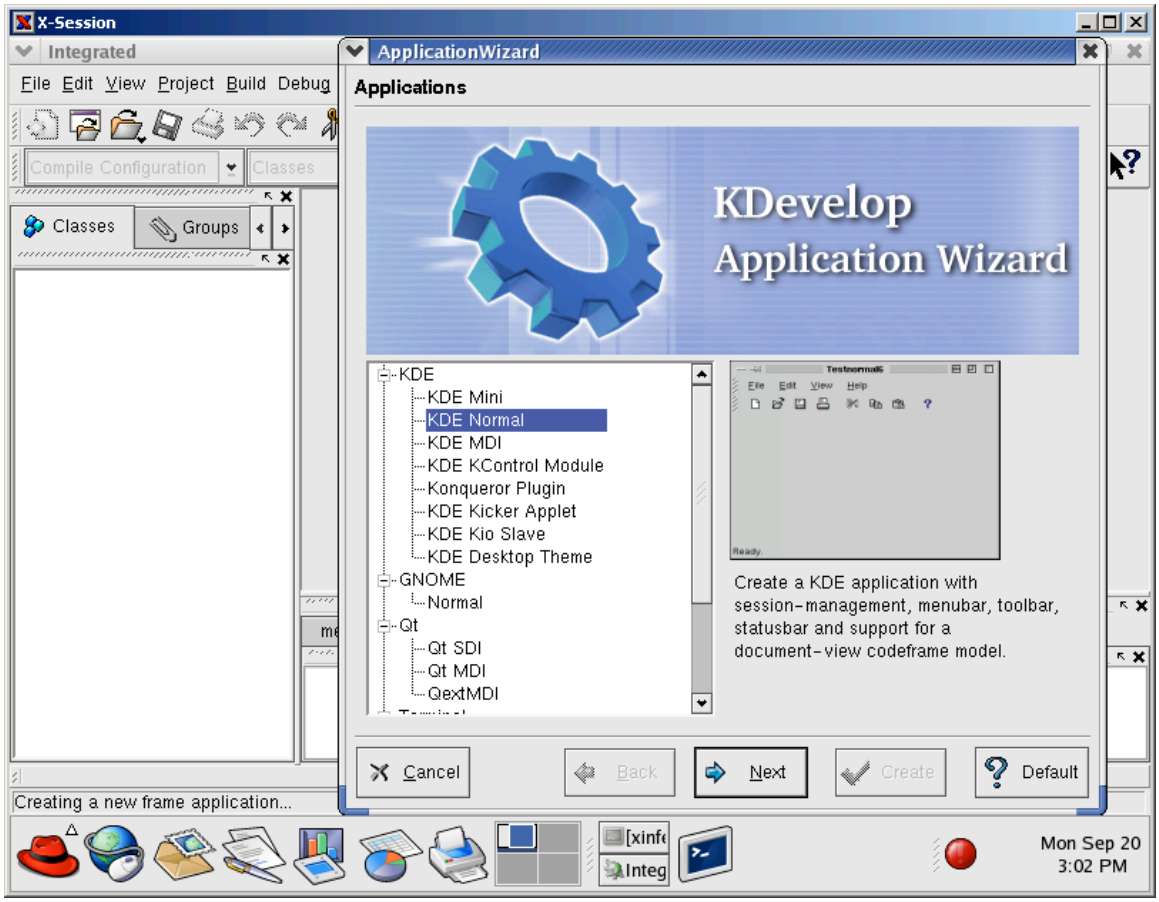

Figure 6-1 KDevelop App Wizard

One of the unique features of KDevelop is that it allows the adding of other program needs to the "Tools" menu. This is necessary for a Linux IDE, because, unlike the monopoly of Visual Studio, Linux development is diversified.

6.2 Qt Programming

We have chosen KDevelop as our IDE (integrated development environment) for Linux. KDevelop is based on Qt, a middle ware between C++ source code and a C++ compiler. In order to program for serial port and general interactive GUI control interfaces in KDevelop, we must handle Qt and its programming first.

Unlike Java and MS .NET, which requires virtual machines on top of an OS, Qt does not require any run-time library to support the compiled binary code. Therefore, Qt is very suitable for real-time and embedded control programming on different platforms. Qt currently supports Win32, Unix/Linux and Mac and the same source code previously written can be compiled on these platforms. There is also an embedded version of Qt, Qt/Embedded, and a handheld version, Qtopia, which is fully based on Qt/Embedded and has been implemented in commercial PDAs such as the SHARP Zaurus series (Dalheimer, 2002).

Qt is very much like MS MFC (Main Foundation Classes), with similar classes to support GUI objects, such as QString (vs. CString), QPainter (vs. CDC), QObject (vs. CObject), QTimer (vs. CTimer), etc. As one can see, these classes even share the same names (just change the letter "C" to "Q"). Meanwhile, KDevelop (Qt based) is very much like VC++ (MFC based) and, for example, supports the same Doc-View architecture. Event handling such as mouse input and screen painting is almost the same. However, the documentation of KDevelop is still very poor and it took quite a long time before we successfully programmed an interactive interface that updates the screen based on the user's mouse input. Qt has very good documentation though and, once we get into KDevelop, the Qt-based programming is relatively easy.

Without writing any C/C++ code and with a few mouse clicks, KDevelop's App Wizard easily generated a graphical window as shown in Fig. 6-2. This application has been named "HelloAhmct."

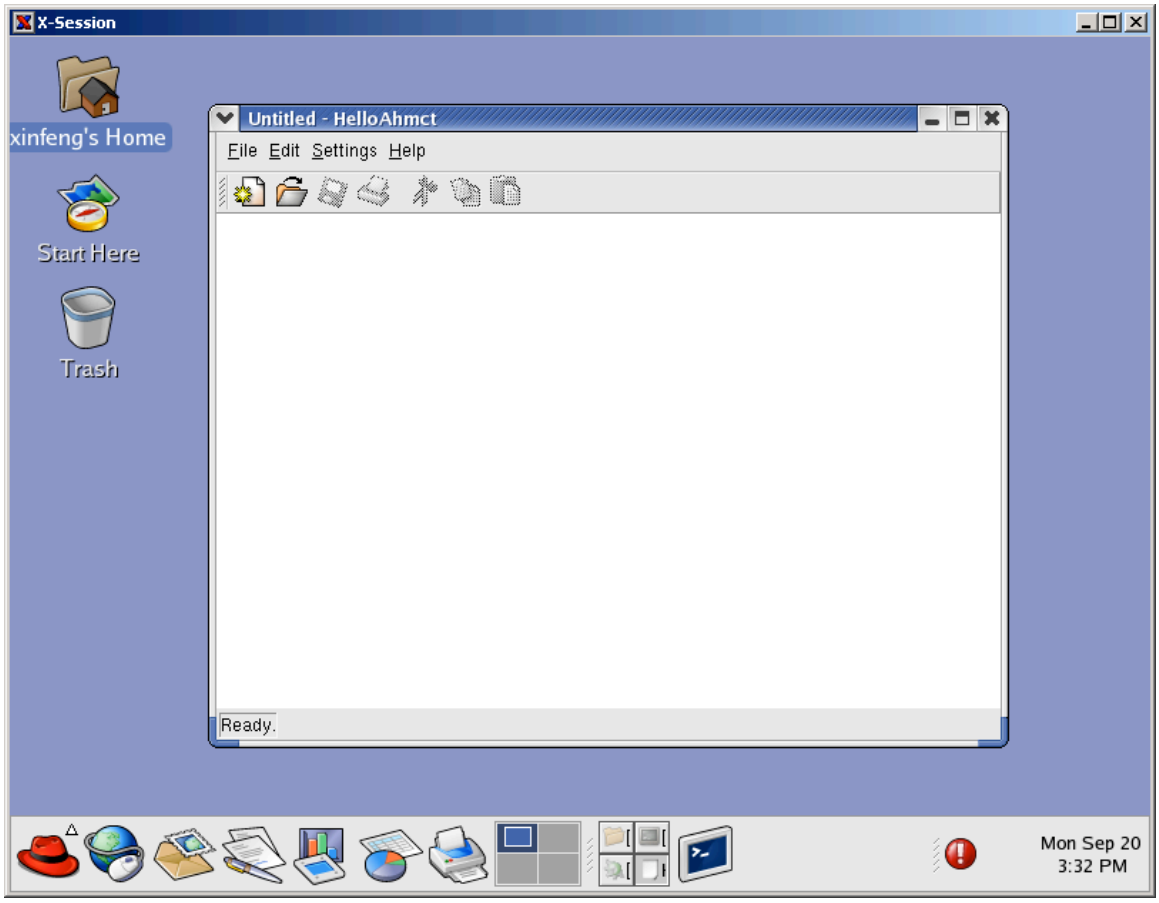

Figure 6-2 The Empty Application Generated Automatically by App Wizard

To let this application do something, we added a member function to the HelloAhmctView class generated automatically by App Wizard. This member function, paintEvent, overrides the base paintEvent function and prints "Hello, AHMCT!" out to the screen (see Fig. 6-3). This function can be added easily using a class wizard and all the code is only few lines and fairly simple, i.e.:

```
Void HelloAhmctView::paintEvent (QPaintEvent *e)
{
   QWidget::paintEvent( e );
   QPainter painter;
   painter.begin ( this );
   painter.drawText (100, 100, "Hello AHMCT!" );
   painter.end ();
}
```

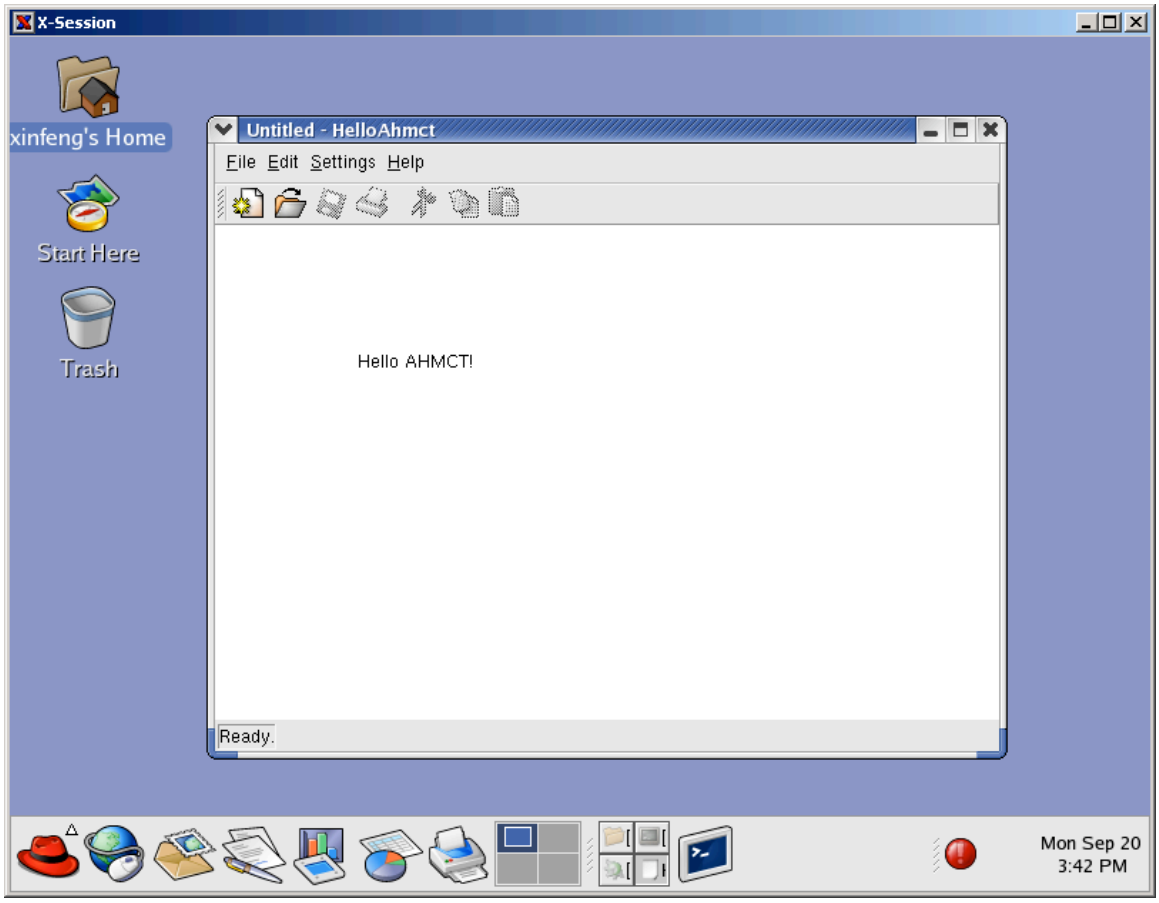

Figure 6-3 Add Screen Printing to the Empty Application

In the following sections, more details on interactive graphics programming with KDevelop and Qt will be provided.

7. Serial Port and Basic Qt Programming

Many control programs involve serial port communication, because serial port (RS-232, etc.) is still the most popular communication interface for control applications. Therefore, we choose serial port programming as the example to show both how to programm in Linux and how to deal with hardware devices. At the end, we can provide a serial port C++ class, which is very useful by itself and very valuable for others to start with.

7.1 Serial Port in Linux

Like other Unix platforms, Linux treats serial ports (COM1-4) and other system devices as files. That is, the serial ports, /dev/ttyS0 - /dev/ttyS3, can be opened, accessed and closed just like ordinary files.

As a serial port is treated just like an ordinary file, its programming is quite simple. We successfully used the following simple code to receive characters from the serial port and then send the characters back between COM1 and 2 of a Linux system.

```
/*************************************************************************
                          main.c  -  serial port communication
                             -------------------
    begin                : Sun Jul 20 12:19:06 PDT 2003
    copyright            : (C) 2003 by Xin Feng
    email                : xinfeng@ucdavis.edu

*************************************************************************/


#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <termios.h>

int open_port(void)
{
  int fd;
  fd = open("/dev/ttyS1", O_RDWR | O_NOCTTY | O_NDELAY);
  if (fd == -1)
  {
    fprintf(stderr, "open_port: Unable to open /dev/ttyS0 -%s\n",
strerror(errno));
  }
  return (fd);
}
```

```
int main(int argc, char *argv[])
{
  //printf("Hello, world!\n");
  int mainfd=0;
  char chout;
  struct termios options;

  mainfd = open_port();

  fcntl(mainfd, F_SETFL, FNDELAY);

  tcgetattr(mainfd, &options);
  cfsetispeed(&options, B2400);
  cfsetospeed(&options, B2400);
  //options.c_ispeed = B9600;
  //options.c_ospeed = B9600;
  options.c_cflag |= (CLOCAL | CREAD);
  options.c_cflag &= ~PARENB;
  options.c_cflag &= ~CSTOPB;
  options.c_cflag &= ~CSIZE;
  options.c_cflag |= CS8;
  options.c_cflag &= ~CRTSCTS;
  options.c_cflag &= ~(ICANON | ECHO | ISIG);
  tcsetattr(mainfd, TCSANOW, &options);
  /* clear buffer*/
  read(mainfd, &chout, sizeof(chout));
  chout = 0;

  while (1)
  {
    read(mainfd, &chout, sizeof(chout));

    if (chout != 0) {
      printf("Got %c. \n", chout);
      write(mainfd, &chout, sizeof(chout));
      //chout=0;
    }
    chout=0;
    usleep(20000);
  }

  close(mainfd);

  return EXIT_SUCCESS;
  //return 0;
}
```

7.2 Some Notes

As serial ports are, of course, special files, there are some special issues here, especially regarding the ports configurations, as follows:

- Like any other files, serial port access is controlled by permission policy. By default, only the superuser (root) can access serial ports and permission must be

changed to allow the normal user to read from and write to serial ports. For example, “chmod a+rw /dev/ttyS0” allows any user to access COM1.

- Port configuration is set by the termios structure. It is necessary to always enable CLOCAL and CREAD, so the program does not own the port, and thus the serial interface driver will read incoming bytes.
- Never enable input echo if the device or computer connected to is already echoing characters.

7.3 Test

We conducted the test between a Windows PC and a Linux box with a serial cable as noted in Fig. 7-1. The Windows PC sends text and the Linux PC receives.

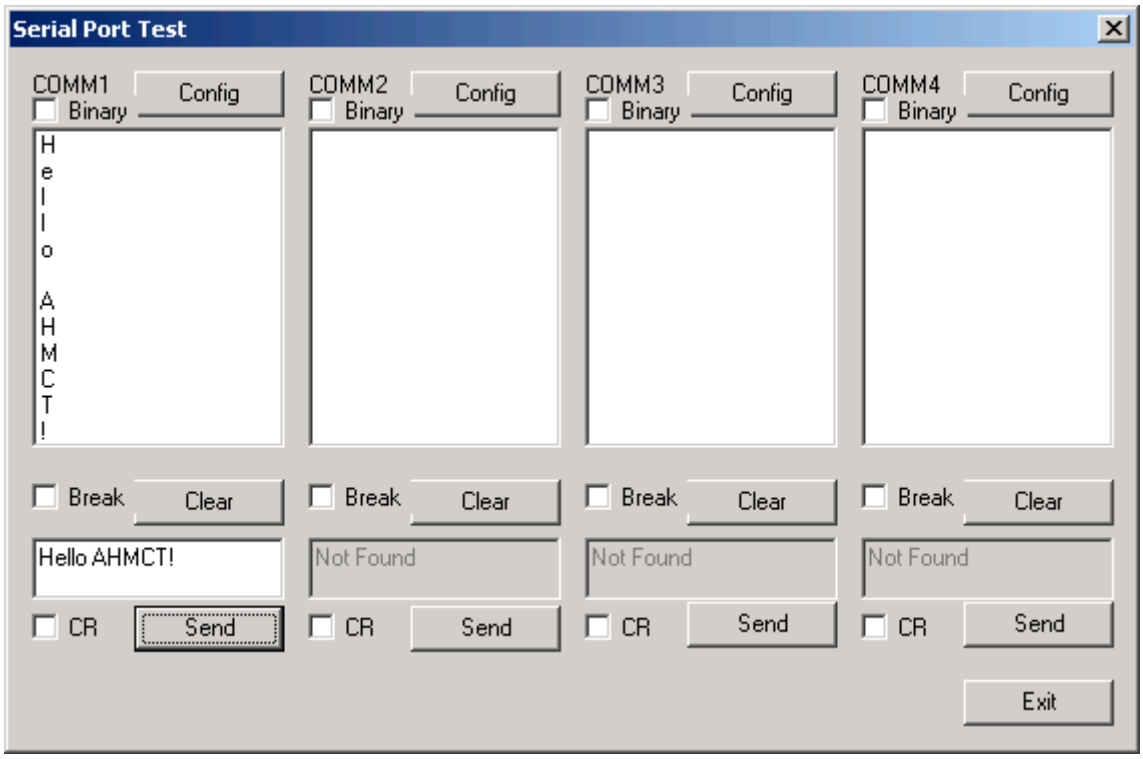

Figure 7-1 Send to Serial Port from Windows

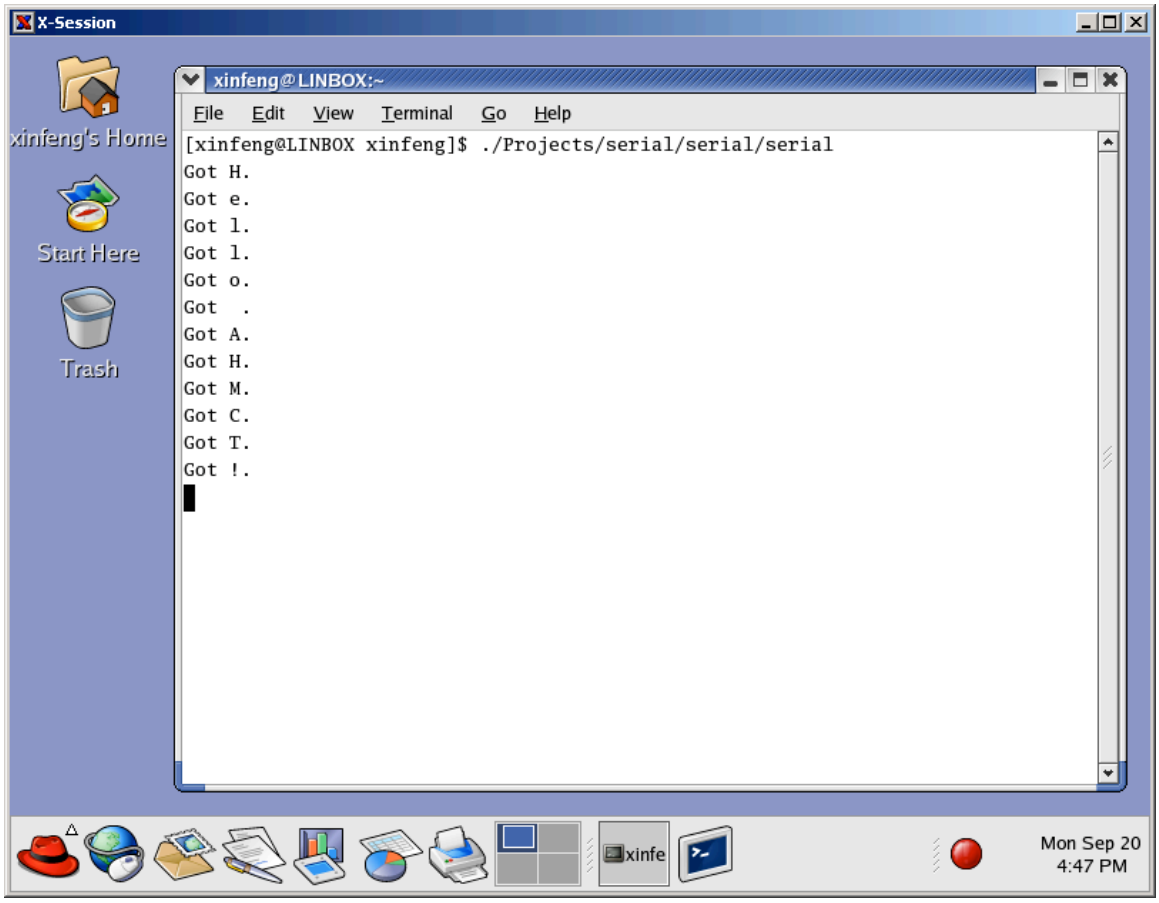

Figure 7-2 Receive from Serial Port on Linux

In the above program (Fig. 7-2), characters are received and then printed out to the console. Things get much more complicated if the characters received are to be sent to a graphical user interface. Qt programming will then be involved. However, Qt does not have a class for serial ports yet, because serial port access is a platform specific issue. At least two objects are required: one handles the background communication and the other handles the graphical presentation. There are two methods for the background communication handling: using a timer or using a thread. These issues are addressed in the next section. We shall also try to make serial port programming object-oriented and provide a general class for any application that uses a serial port.

8. Linux Security

The Linux PC used for this research is behind a firewall and was initially completely secure. Later. a new programmer (consultant) was added to the project and we opened one port through the firewall for him to access this Linux PC remotely. This opened a path for a hacker to enter our system and resulted in the loss of a significant amount of our work. This incident taught us a valuable lesson on Linux's security and thus, here we take a break to address this important issue.

Unlike Windows, which was mainly developed for personal use, Linux was born for network and servers that allow the remote use by multiple users. Meanwhile, Linux, like any UNIX, has the most powerful scripting tools. Therefore, a hacker can easily use the network services and their security holes to break into a Linux system. And, once a hacker breaks into a Linux system, he/she can do almost anything easily. Therefore, security is a very important issue, especially when using Linux for control and the potential damage far beyond file loss.

Based our experience, to follow are the two most effective methods for Linux protection on a network:

1) Always put a Linux box behind a firewall and do not open any port. Telnet (21) and XDMCP (177 etc.) ports are the most dangerous ones, although they are very convenient for remote access.

If remote access has to be given, use a Windows 2000 server (for its terminal service feature) or Windows XP (for its remote desktop feature) instead, because they are far more secure and have encryption built-in. Then use them, also behind the same firewall, to access Linux boxes. In addition to the security aspect, this method also brings two side benefits. First Windows' terminal service is very efficient, far more responsive and practically usable than any X-based remote access when graphics are involved. X-based remote access through dial-up or even broadband is simply not practical – it is too slow. In this configuration, the Linux box is actually accessed locally. Second, Windows terminal service and remote desktop are interrupt-proof. If a network interrupt happens during a remote access session, the user can simply log in again once the network is recovered and continue the session from where it was left with nothing lost. Putting the security issue aside, only these two side benefits alone are enough reason to access a Linux box through a Windows 2000 or XP server.

2) Stop all unnecessary services. The more network services running, the more security holes they may generate. Without the right service, a hacker has no way to break in. In addition to Telnet and X, FTP, MAIL, HTTP and NFS are also the risky services.

Although one can check and apply security patches frequently, but these cannot be relied upon. An administrator can never keep up with hackers who continuously scan ports for security holes. Meanwhile, many security patches generate even more new holes! Therefore, the above noted two aspects are the most essential.

9. KDevelop and Interactive Programming

After the hacking incident discussed, we had to regenerate the serial communication program. Our goal is to build a serial communication C++ class and build a graphical program to do serial port testing. This allows us to cover both the hardware and GUI handling, which are the most essential aspects of control programming. This requires Qt programming under the KDevelop environment, and particularly we concentrated on the most essential GUI programming – screen-printing and mouse handling. It is noted that the KDevelop, V3.0 was available which is more polished than V2.0 and programming it is even easier.

9.1 KDevelop

As described above, starting a new project is as easy as with MS Visual Studio with just a few clicks. The new application is then built and executed without any problem. So in just a few minutes, we have a fully functional graphical application without having to write any code; e.g., see Figs. 6-1 to 6-3. This is as good as VS. On the down side, the building process is painfully longer than VS. It is as slow as using VS on an old 133Mhz/32MB PC. KDevelop authors still have considerable optimization work necessary such as implementing the pre-compiled head file technique used in VS that saves unnecessary re-compiling for unchanged large head files.

We then attempted to print a text, say, "Hello AHMCT!" on the application's screen. In VS, adding a message handler for a mouse-click, for example, is easy. One simply selects from a message list and adds a handler to it. In KDevelop, one must manually override the right virtual function that is specific to each message. This is not direct for new Qt programmers because they usually do not know which virtual function to look for. To make this matter even worse, documentation is almost non-existent. After some searching, we found that the virtual function for screen-printing is painEvent(). Then things are very similar to VS. To follow is the additional code needed for the application that is auto-generated by KDevelop:

```
Void MyKdeView::paintEvent( QPaintEvent *e)
{
        QWidget::paintEvent( e );
        QPainter painter;
        painter.begin(this);
        painter.drawText(100,100,"Hello AHMCT!");
        paintr.end();
}
```

See Fig. 6-3 for the actual screen output.

9.2 Interactive User Interface

We then add the interactive user interface to the application: print "Hello AHMCT!" when the mouse is clicked. The virtual functions for mouse events are mousePressEvent(), mouseMoveEvent(), mouseReleaseEvent(), etc. Naturally, we

moved the above code from the paintEvent() function into the mousePressEvent() and things worked out as expected. There is one problem though: as soon as the screen is redrawn, all the printed text is gone. To fix this problem, however, turned out to be quite complicated.

First, we need a buffer, APixmap, to store the drawing and initialize this buffer with white color. Then all the drawing should be done to this buffer rather than directly to the screen. During re-painting of the screen, the paintEvent() function simply copies this buffer to the screen and does nothing else.

Second, we need to handle one more event – resizeEvent(), to make sure that the size of the buffer matches that of the actual screen whenever the window size is changed.

Third, we need to copy the buffer to the screen after each drawing, so the screen gets updated immediately.

Here is the code for handling mouse event and screen-printing:

```
/****************************************************************
    helloahmctview.cpp  -  mouse event and paint
                             -------------------
    begin                : Mon Sep 20 15:20:54 PDT 2004
    copyright            : (C) |YEAR| by Xi2004
    email                : xinfeng@ucdavis.edu

****************************************************************/


// include files for Qt
#include <qprinter.h>
#include <qpainter.h>

// application specific includes
#include "helloahmctview.h"
#include "helloahmctdoc.h"
#include "helloahmct.h"
#include "string.h"

HelloAhmctView::HelloAhmctView(QWidget *parent, const char *name) :
QWidget(parent, name)
{
  setBackgroundMode(PaletteBase);

  buffer.fill (white);

}

HelloAhmctView::~HelloAhmctView()
{
}

HelloAhmctDoc *HelloAhmctView::getDocument() const
{
  HelloAhmctApp *theApp=(HelloAhmctApp *) parentWidget();
```

```
  return theApp->getDocument();
}

void HelloAhmctView::print(QPrinter *pPrinter)
{
  QPainter printpainter;
  printpainter.begin(pPrinter);

  // TODO: add your printing code here

  printpainter.end();
}
/** No descriptions */
void HelloAhmctView::paintEvent(QPaintEvent *e){

  QWidget::paintEvent(e);
  QRect r = e->rect();
  bitBlt(this, r.x(), r.y(), &buffer, r.x(), r.y(), r.width(),
r.height());

  //QPainter painter;
  //painter.begin(this);
  //painter.drawText(100,100,"Hello AHMCT!");
  //painter.end();
}
/** No descriptions */
void HelloAhmctView::mousePressEvent(QMouseEvent *e){
  QPainter painter;
  painter.begin(&buffer);
  painter.drawText(e->pos(), "Hello AHMCT!");
  char s[30];
  sprintf (s, "(%d,%d)", e->x(), e->y());
  painter.drawText(e->x(), e->y()+15, s);
  painter.end();
  bitBlt(this, 0,0, &buffer, 0,0, width(), height());
}
/** No descriptions */
void HelloAhmctView::resizeEvent(QResizeEvent *e){
  QWidget::resizeEvent(e);

  int w = width() > buffer.width()?width():buffer.width();
  int h = height() > buffer.height()?height():buffer.height();

  QPixmap tmp(buffer);
  buffer.resize(w,h);
  buffer.fill(white);
  bitBlt(&buffer, 0,0, &tmp, 0,0, tmp.width(),tmp.height());
}
```

Figure 9-1 shows the actual screen output. “Hello AHMCT!” is printed at the position where the mouse is located on the screen at the time it is clicked. Also, the mouse position is printed out.

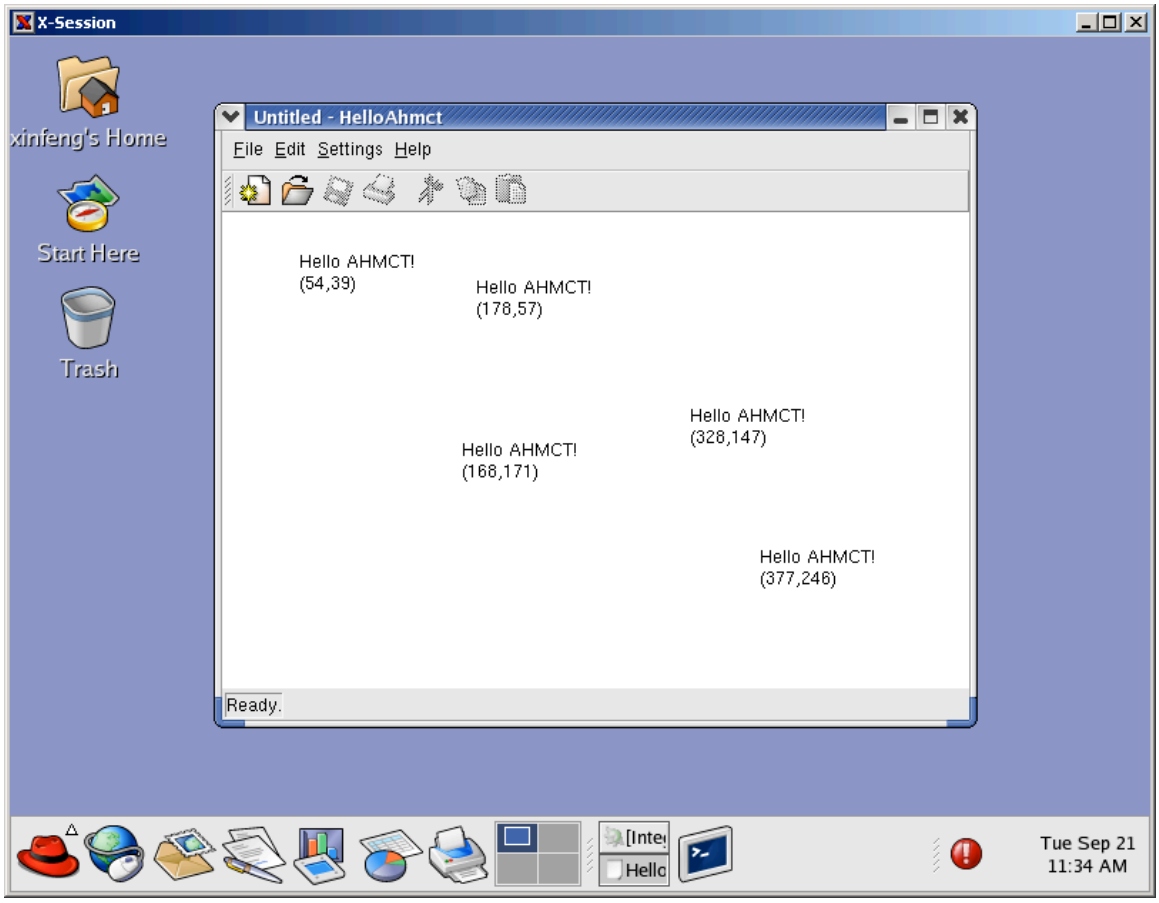

Figure 9-1 Interactive User Interface

We then entered into interactive graphical programming. As the documentation is very poor, it took us a fair amount of time to resolve issues. The only good resources available are the Qt reference page found on www.trolltech.com and the "KScrible" tutorial found on www.kdevelop.org. On a positive note, things are quite similar to VS.

10.A C++ Class for Serial Port Communication

The KDevelop 3.0 was use following after the hacking described earlier. The more we used KDevelop 3.0, the more we liked it – it is a significant improvement to V2.0 and it is now quite mature and as good as MS Visual Studio.

With this very strong programming environment, we finally started the development of a general C++ class for serial port communication. As far as we are aware, there is no such class yet for Linux. This is not surprising since most of the Linux developers still use C.

10.1 Serial Port C++ Class

KDevelop 3.0 makes it fairly easy to create a new class. We simply create a new project by choosing the Terminal/C++ template from the KDevelop Application Wizard. Once this project is created, we then right click on the Classes item in the browsing window and select "new class". Giving the name of the new class, "CSerial", we have the new class stored in cserial.h and cserial.cpp files in the same manner as with MS Visual Studio.

Then we add class members to this class based on the serial port C code we developed earlier. We design this CSerial to hide all the very complicated details of serial communication under Linux. With CSerial, one can simply open a serial port with the Open() function and initiate it with Init(), and the communication is established. Then one uses the Read() and Write() functions to receive and send data from and to the port. When done, one closes the communication with the Close() function.

10.2 Test Program

An example program is provided to show how to use this class. This simple sample opens ttyS0 to read the serial port and ttyS1 to send. Figure 10-1 shows the actual screen output while the Linux PC's COM1 and COM2 are connected with a crossover cable. When "Hello AHMCT!" is typed in, the characters of this string are sent to COM2 (ttyS1). The characters are received from COM1 (ttyS0) and are printed out on the screen.

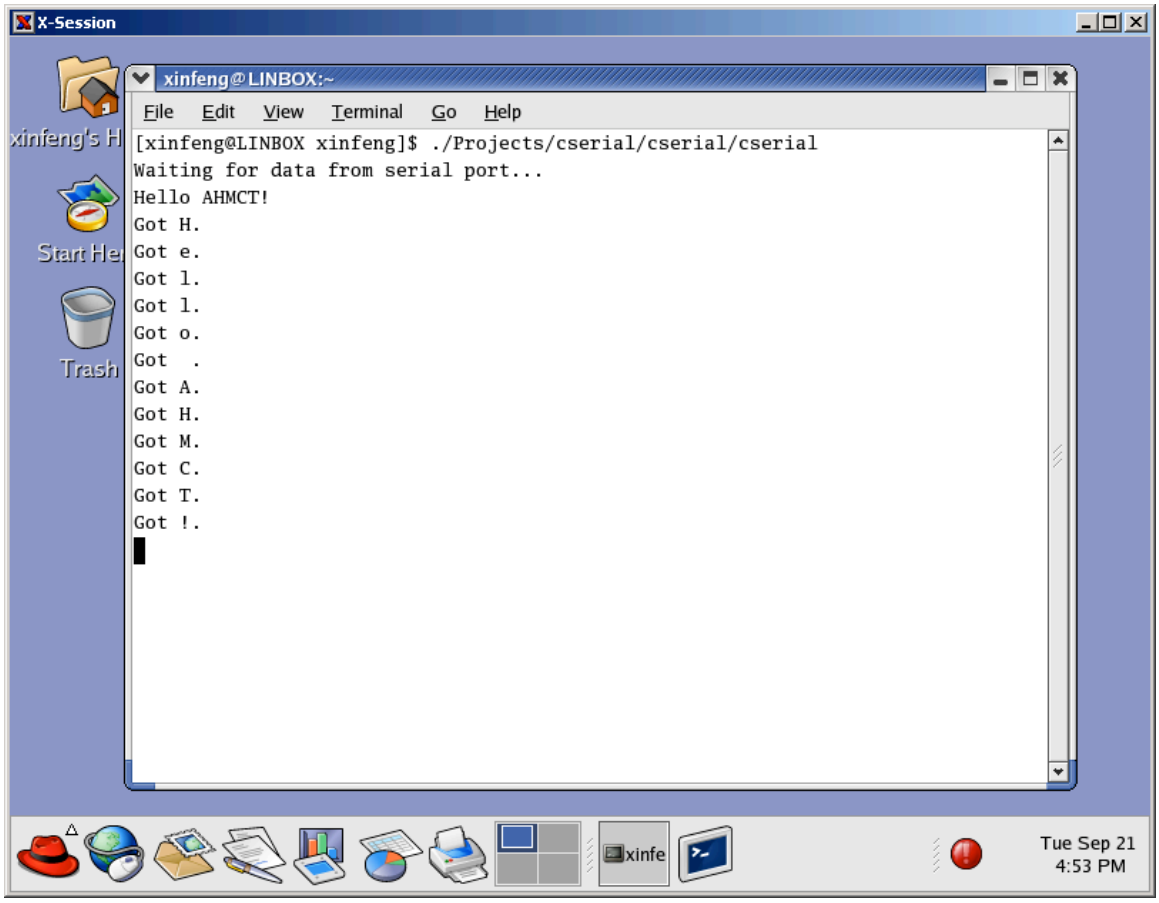

Figure 10-1 Sending and Receiving Using the C++ class

To follow is the source code of the test program:

```
/***************************************************************************
    main.cpp  -  Test terminal for CSerial class
                             -------------------
    begin                : Sun Jul 20 20:30:05 PDT 2003
    copyright            : (C) 2003 by Xin Feng
    email                : xinfeng@ucdavis.edu

***************************************************************************/

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <iostream.h>
#include <stdlib.h>
#include <unistd.h>
#include "cserial.h"

#define RETURN '\n'

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

```
{
  char chout;
  //cout << "Hello, World!" << endl;
  cout << "Waiting for data from serial port..." << endl;
  CSerial sp,sp2;
  sp.Open("/dev/ttyS0");
  sp.Init();
  sp2.Open("/dev/ttyS1");
  sp2.Init();

  chout=0;
  while (1)
  {
    chout = (char)getchar();
    if(chout != RETURN) sp2.Write(&chout);
    chout =0;
    sp.Read(&chout);
    if (chout != 0) {
      printf("Got %c.\n", chout);
      //sp.Write(&chout,sizeof(chout));
    }
    chout=0;
    usleep(20000);
  }

  sp.Close();

  return EXIT_SUCCESS;
}
```

As one can see, by using a C++ class, the complicated serial port communication programming becomes quite simple and easy. Without using such a class, it would be hard for above simple program to handle elegantly and neatly two serial ports with so few lines of code. A flowchart of the code is given in Fig. 10-2.

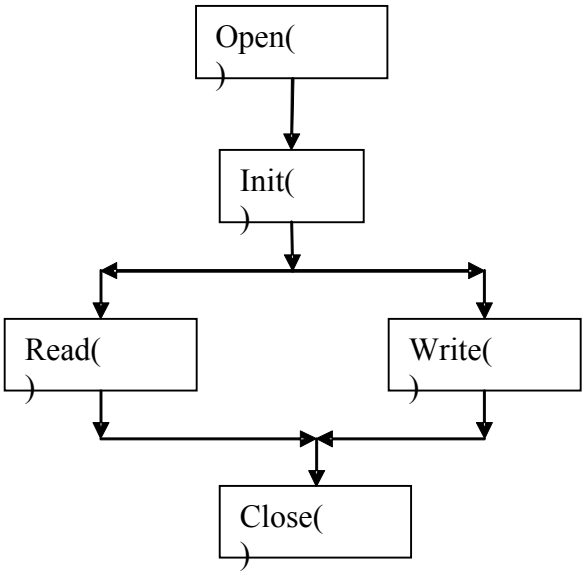

Figure 10-2 Flowchart of Using the C++ Class

10.3 The Source Code

To follow are the cserial.h and cserial.cpp files that made up the CSerial class.

```
/***************************************************************************
                          cserial.h  -  class for serial port
                             -------------------
    begin                : Sun Jul 20 2003
    copyright            : (C) 2003 by Xin Feng
    email                : xinfeng@ucdavis.edu

***************************************************************************/


#ifndef CSERIAL_H
#define CSERIAL_H


/**
  *@author Xin Feng
  */

class CSerial {
public:
      CSerial();
      ~CSerial();
  /** No descriptions */
  int Open(const char *sPort);
  /** No descriptions */
  int Init();
  /** No descriptions */
  int Write(char *c,int nBytes=1);
  /** No descriptions */
  int Read(char *c);
  /** No descriptions */
  int Close();
protected: // Protected attributes
  /**  */
  int m_nPort;
};

#endif

/***************************************************************************
                          cserial.cpp  -  class for serial port
                             -------------------
    begin                : Sun Jul 20 2003
    copyright            : (C) 2003 by Xin Feng
    email                : xinfeng@ucdavis.edu

***************************************************************************/

#include "cserial.h"
#include <stdio.h>
#include <string.h>
```

```
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <termios.h>

CSerial::CSerial(){
}
CSerial::~CSerial(){
}
/** No descriptions */
int CSerial::Open(const char *sPort){
  m_nPort = open(sPort, O_RDWR | O_NOCTTY | O_NDELAY);

  if (m_nPort == -1)
  {
    fprintf(stderr, "Open: Unable to open serial port - %s\n",
strerror(errno));
  }

  return m_nPort;
}
/** No descriptions */
int CSerial::Init(){
  struct termios options;
  fcntl(m_nPort, F_SETFL, FNDELAY);

  tcgetattr(m_nPort, &options);

  cfsetispeed(&options, B9600);
  cfsetospeed(&options, B9600);

  options.c_cflag |= (CLOCAL | CREAD);
  options.c_cflag &= ~PARENB;
  options.c_cflag &= ~CSTOPB;
  options.c_cflag &= ~CSIZE;
  options.c_cflag |= CS8;

  options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);

  options.c_iflag &= ~INPCK;
  options.c_iflag |= (IGNPAR | ISTRIP);
  options.c_iflag &= ~(IXON | IXOFF | IXANY);

  options.c_oflag &= ~OPOST;

  return tcsetattr(m_nPort, TCSANOW, &options);
}
/** No descriptions */
int CSerial::Write(char* c, int nBytes){
  return write(m_nPort, c, nBytes);
}
/** No descriptions */
int CSerial::Read(char *c){
  return read(m_nPort, c, sizeof(c));
}
/** No descriptions */
int CSerial::Close(){
```

```
  return close(m_nPort);
}
```

In the next section, we will add thread or process support to this class, so CSerial itself can handle the communication in the background. This will simplify further the communication programming in applications and make the communication more efficient with the least performance impact to the user interface. We will also develop a Qt widget based on this class to be used in GUI applications.

11.A multithreaded C++ Class for Serial Port Communication

To deploy Linux for control, communication, especially serial communication, is critical. However, so far we have not found a Unix/Linux C++ class for this and no serial communication code been developed in a multithreaded way. Therefore, we developed such a reusable and object oriented C++ class, the CSerial, in the last section. Now, we make it multithreaded; so it will be more efficient and have the least performance impact to the user interface.

11.1 The Source Code

The following code is reflective of the multithreaded capability.

```
/***************************************************************************
    cserial.h  -  multithreaded class for serial port
                             -------------------
    begin                : Sun Jul 20 2003
    copyright            : (C) 2003 by Xin Feng
    email                : xinfeng@ucdavis.edu

***************************************************************************/


#ifndef CSERIAL_H
#define CSERIAL_H

#define SERIAL_PORT_BUFFER 1000

#include <qthread.h>
#include <qobject.h>
/**
  *@author Xin Feng
  */

class SDATA {
  public:
  int length;
  char data[SERIAL_PORT_BUFFER + 1];
};

class CSerial : public QThread{
public:
      CSerial();
      ~CSerial();
  CSerial(QObject* o) { pParent = o;}
  /** No descriptions */
  int Open(const char *sPort);
  /** No descriptions */
  int Init();
  /** No descriptions */
  int Write(const char *c,int nBytes=1);
  /** No descriptions */
  int Read(char *c);
  /** No descriptions */
```

```
  int Close();
protected: // Protected attributes
  /**  */
  int m_nPort;
  void run();
  QObject* pParent;
};

#endif

/***************************************************************************
    cserial.cpp  -  multithreaded class for serial port
                             -------------------
    begin                : Sun Jul 20 2003
    copyright            : (C) 2003 by Xin Feng
    email                : xinfeng@ucdavis.edu

***************************************************************************/

#include "cserial.h"
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <termios.h>

#include <qapplication.h>

CSerial::CSerial(){
}
CSerial::~CSerial(){
}
/** No descriptions */
int CSerial::Open(const char *sPort){
  m_nPort = open(sPort, O_RDWR | O_NOCTTY | O_NDELAY);

  if (m_nPort == -1)
  {
    fprintf(stderr, "Open: Unable to open serial port - %s\n",
strerror(errno));
  }

  return m_nPort;
}
/** No descriptions */
int CSerial::Init(){
  struct termios options;
  //fcntl(m_nPort, F_SETFL, FNDELAY);
  fcntl(m_nPort, F_SETFL, 0);

  tcgetattr(m_nPort, &options);

  cfsetispeed(&options, B9600);
  cfsetospeed(&options, B9600);
```

```
  options.c_cflag |= (CLOCAL | CREAD);
  options.c_cflag &= ~PARENB;
  options.c_cflag &= ~CSTOPB;
  options.c_cflag &= ~CSIZE;
  options.c_cflag |= CS8;

  options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);

  options.c_iflag &= ~INPCK;
  options.c_iflag |= (IGNPAR | ISTRIP);
  options.c_iflag &= ~(IXON | IXOFF | IXANY);

  options.c_oflag &= ~OPOST;

  return tcsetattr(m_nPort, TCSANOW, &options);
}
/** No descriptions */
int CSerial::Write(const char* c, int nBytes){
  return write(m_nPort, c, nBytes);
}
/** No descriptions */
int CSerial::Read(char *c){
  //int byte;
  //ioctl(m_nPort, FIONREAD, &byte);
  return read(m_nPort, c, SERIAL_PORT_BUFFER);
}
/** No descriptions */
int CSerial::Close(){
  return close(m_nPort);
}

void CSerial::run()
{

  SDATA data;
  for(;;)
  {
    //c=0;
    //qDebug("Ping!");
    data.length=Read(data.data);
    if (data.length>0) {
      //c[n+1]='\0';
    //if(c != 0)
    //{
      QCustomEvent ce((QEvent::Type)(QEvent::User +100),&data);
      QApplication::sendEvent(pParent,&ce);
    }
    //usleep(20000);
  }

}
```

11.2 Some Notes

- This C++ class, CSerial, is generic and is built with Qt. Therefore it can be implemented to all major platforms including Windows, Macintosh, Unix, etc., in addition to Linux.
- It is based on the QThread, Qt's multithreading base class.
- It uses QCustomEvent, Qt's base class for customized events, to notify the main application and pass the data.
- A simple class, SDATA, is built to hold communication data. This class has only two members: a character array and its length. Therefore, CSerial can handle both ASCII and binary communications.

Considering this new class, one finds that the main difference is the run() member function. This virtual function of QThread allows running anything in the thread. In this case, the thread checks for new data and, if new data arrived, it notifies the serial port consumer with a customized event and passes the data via the event.

11.3 Test

QThread and QCustomEvent are very well written; therefore CSerial is fairly simple (only couple lines total are required) and easy to use. An application with graphical user interface is developed to test this class in real serial communication. Based on the test results, this class is indeed easy to use, efficient and reliable. It is much simpler and better than the MFC serial class we have used previously in every aspect. The test results are shown in Figs. 11-1 and 11-2.

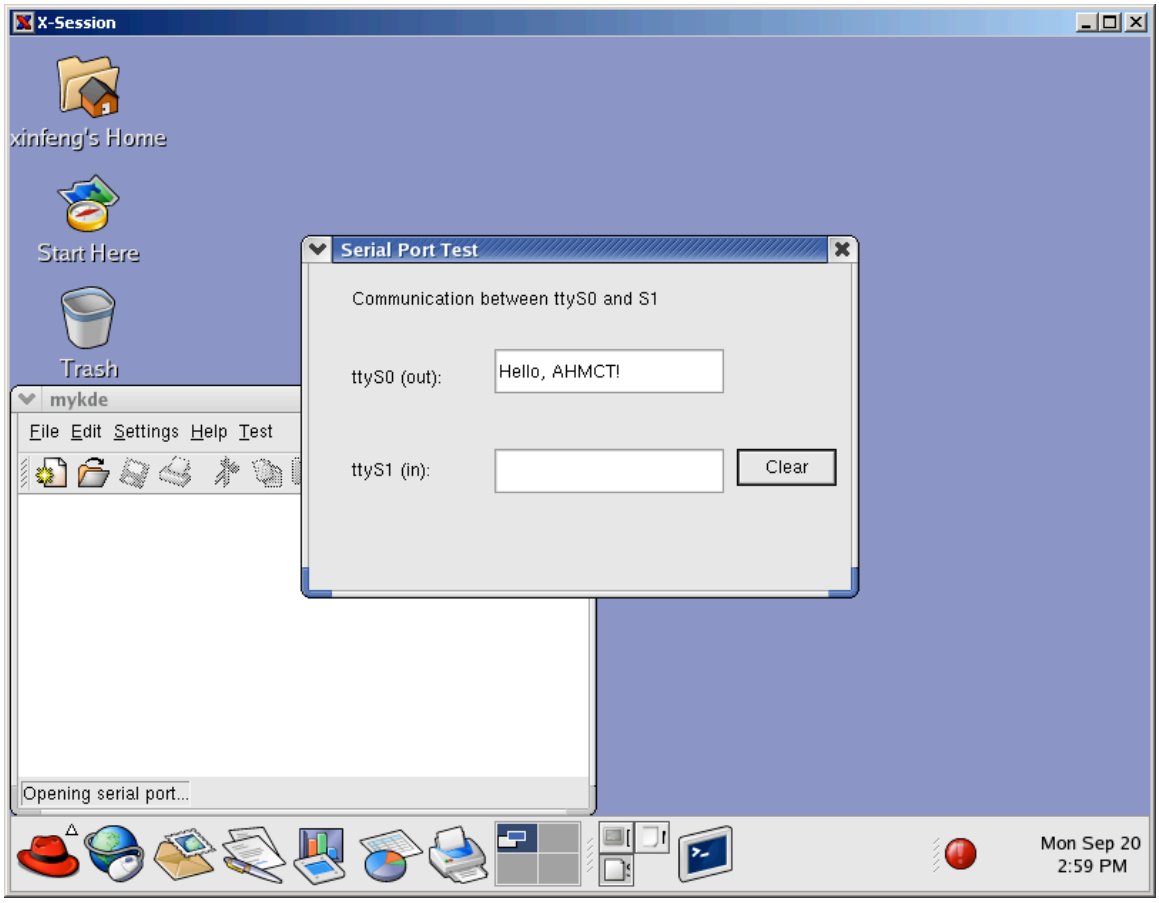

Figure 11-1 Input to be Sent to Serial Port

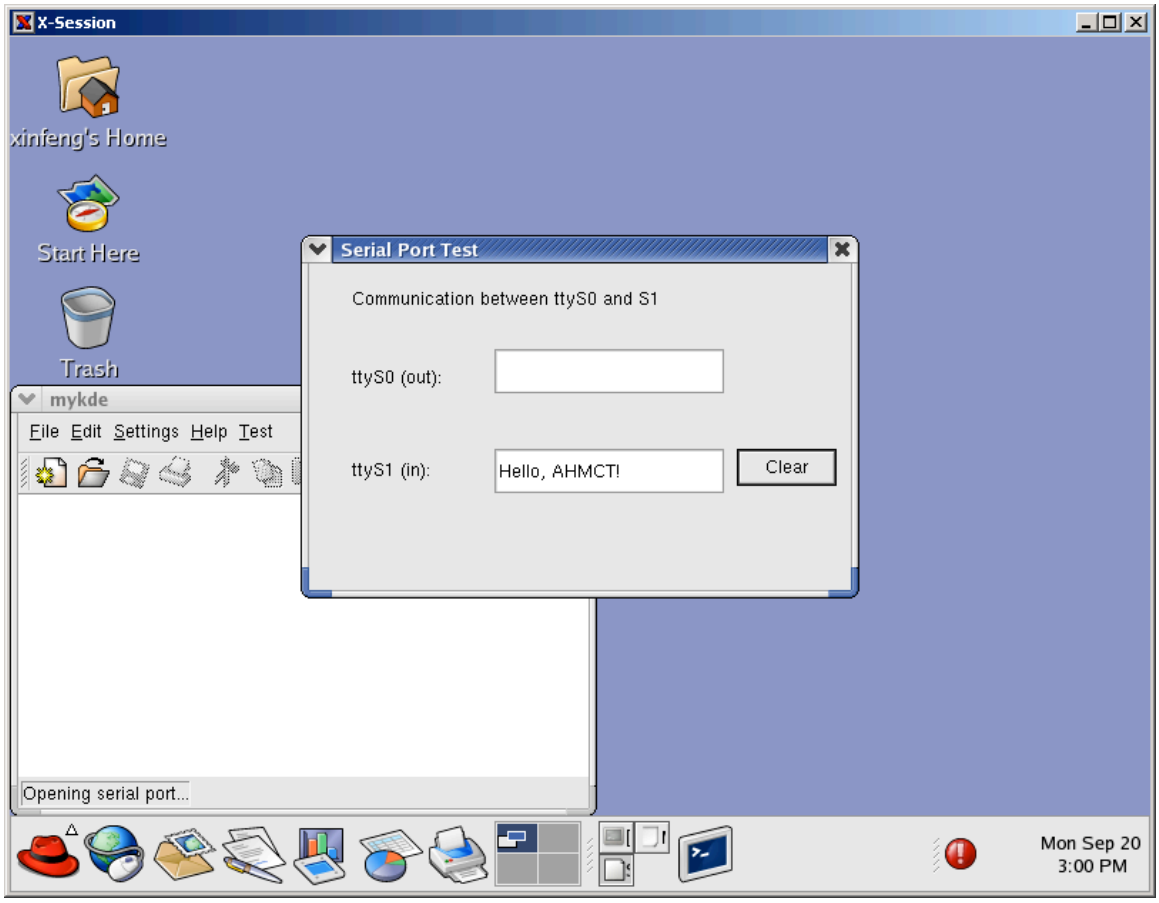

Figure 11-2 Output Received from Serial Port

So far, we have addressed control using Linux over a serial communication. With this reusable, object-oriented, multi-platform and multithreaded C++ class, one can easily control anything that has a serial port interface.

12.More on GUI Programming

In order to build a graphical user interface to test the multithreaded CSerial class, graphical programming using Qt is studied in additional depth. Poor documentation has made this somewhat difficult. However, once a basic understanding is developed, its use is actually better than MS Visual Studio 6.0 and as good as the latest MS Visual Studio .NET.

12.1 Menu

In Visual Studio, adding a menu or a menu item is fairly easy and can be done with a few mouse clicks. In Qt/KDevelop, however, this is still requires text. In order to add a popup "Test" menu and an item "Serial" under Qt, the following code must be added manually into the initMenuBar() function of the main app class as:

```
pTestMenu = new KPopupMenu();
menuBar()->insertItem(i18n("&Test"), pTestMenu);
testSerial->plug(pTestMenu);
testSerial->plug(toolbar());
```

The first two lines are straightforward; they create a new popup object and insert it into the menu bar. After this, "Test" will appear on the menu bar. When clicked, it will pop up the submenu items that lie under it. The last two lines add "Serial" under "Test" to the toolbar, respectively. "testSerial" is declared as a KAction object, which actually handles the action when the "Serial" submenu or the toolbar icon is clicked. This object is created in the initActions() function of the main app class: as

```
testSrial = new KAction(i18n("&Serial"), "testserial", 0, this, SLOT(slotTestSerial()),
actionCollection());
```

What this really does is connect the action to a slot, the slotTestSerial(). In slotTestSerial(), a dialogue window is started as:

```
TestSerialDlg tsd;
Tsd.exec();
```

In summary, this work allows one to start a dialogue window from a menu, as shown in Figs. 12-1 and 12-2.

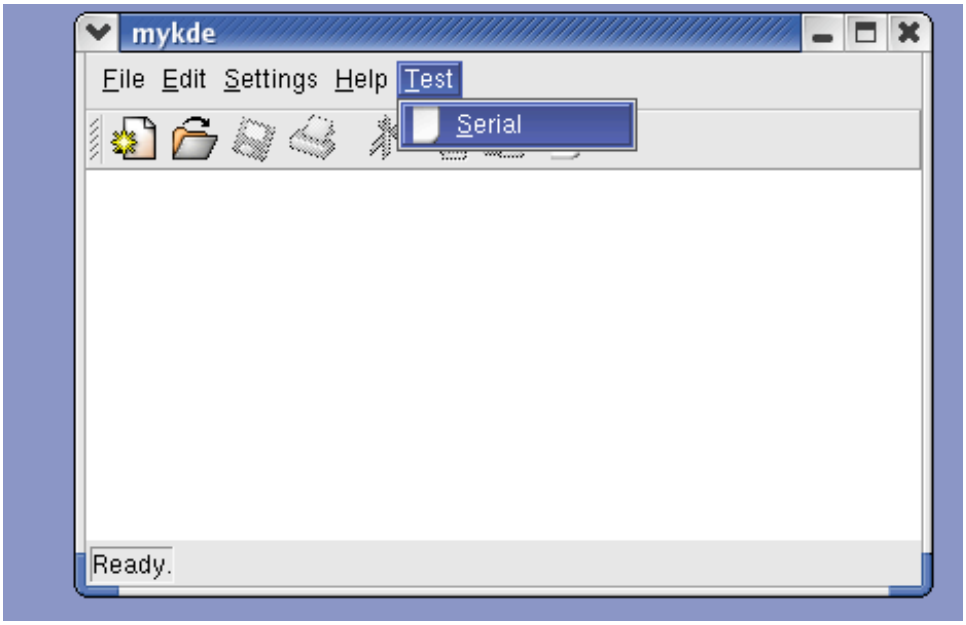

Figure 12-1 The Added Menu Items

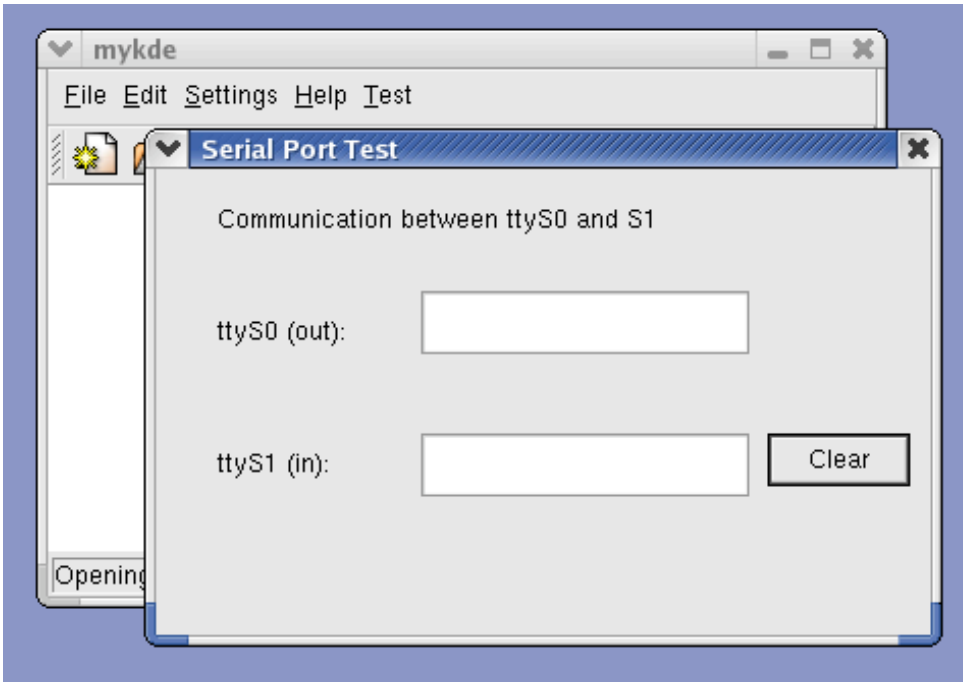

Figure 12-2 The Dialog Window popped out by the Menu

12.2 Toolbar

In last section, an action has been added to the toolbar. We now need to design a new icon, an image in PNG format. KDevelop provides an icon editor, the KIconEdit, for this job. Its file name must be told when the action object is created; in this case it is "testserial" and the associated code is:

testSrial = new KAction(i18n("&Serial"), "testserial", 0, this, SLOT(slotTestSerial()), actionCollection());

This line shows "Open the serial port test dialog" when the mouse pointer is moved on to the icon, as shown in Fig. 12-3 and the associated code is:

testSerial->setStatusText(i18n("Open the serial port test dialog"));

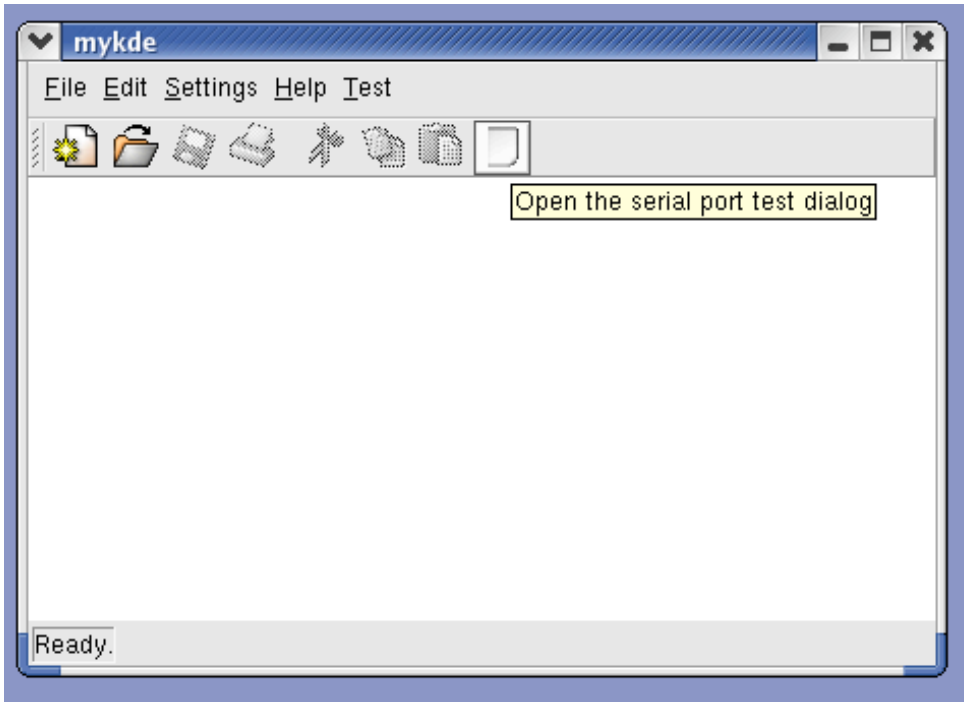

Figure 12-3 Added Toolbar icon

12.3 Dialog, Widgets, Signal and Slot

In Qt/Kdevelop, all graphics objects, including dialogs, are widgets. A widget has its own window and can take user inputs. Widgets communicate with each other in the so-called "Signal and Slot" scheme.

Qt/KDevelop provides a WYSIWYG tool, Qt Designer, for creating a dialog. Actually, one can create a whole application within Qt Designer. In our dialog, there is a text box for taking in text that is to be sent to serial port 1 (ttyS0), another text box to show text received from serial port 2 (ttyS1), and a clear button to clear the second test box as shown in Fig. 12-4.

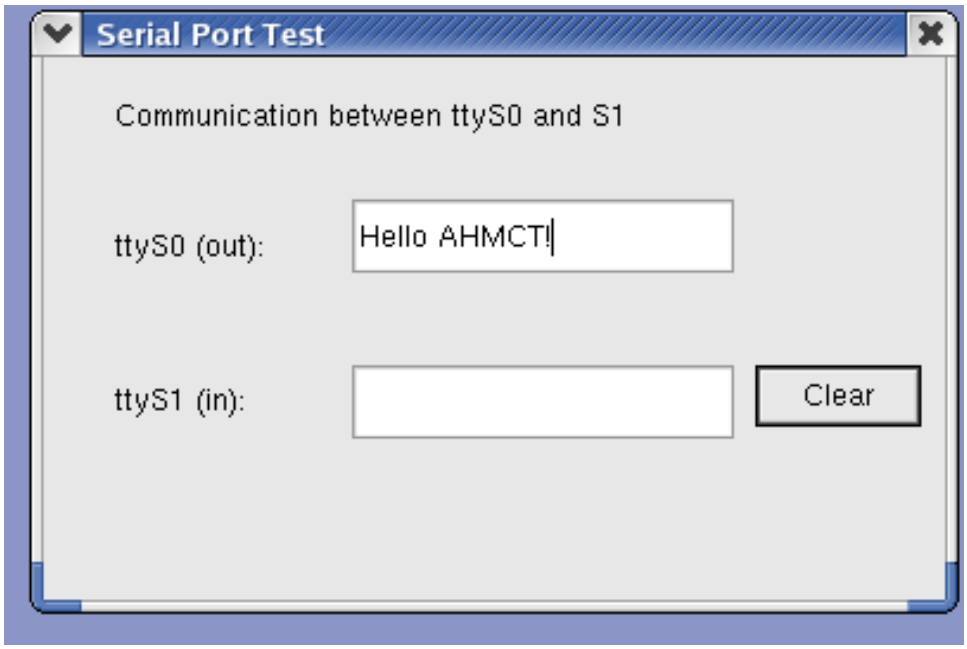

Figure 12-4 The Dialog

When this dialog is initialized, it creates two threads for ttyS0 and ttyS1, respectively:

```
Void TestSerialDlg::init()
{
 pThread1 = new CSerial(this);
 pThread1 -> Open("/dev/ttyS0");
 pThread1 -> Init();
 pThread1 -> start();

 pThread2 = new CSerial(this);
 pThread2 -> Open("/dev/ttyS1");
 pThread2 -> Init();
 pThread2 -> start();
}
```

When the return key is hit, the LineEdit1_returnPressed slot will handle that signal - the text in the first box is sent to ttyS0:

```
void TestSerialDlg::LineEdit1_returnPressed()
{
 sReceived = "";
 LineEdit2->clear();
 QString s = LineEdit1->text();
 pThread->Write(s, s.length());
 LineEdit1->clear();
}
```

When ttyS1 receives the text, the second thread posts a QCustomEvent. This dialog will respond to this event with the following code – show the received text in the second box:

```
Void TestSerialDlg::customEvent( QCustomEvent *e)
{
 SDATA* sd = (SDATA*)e->data();
 Char* s = sd->data;
 S[sd->kength] = '\0';
 sReceived.append(s);
 LineEdit2->setText(sReceived);
}
```

When the clear button is clicked, this signal is sent to the PushButton1_clicked() slot, which clears the receiving buffer and the second box:

```
Void TestSerialDlg::PushButton1_clicked()
(
 sReceived="";
 LineEdit2->clear();
}
```

In summary, Qt/KDevelop handles graphical objects elegantly with widgets. All widgets communicate with each other with signals and slots, which is neater and more efficient than messages used in Visual C++.

12.4 3D Programming

In addition to 2D graphics programming, we also studied interactive 3D programming under Linux (Windows and Macintosh as well). Unfortunately, we have not found a good solution yet. We do not want to use OpenGL because it is not based on C++. We looked into more than a dozen of 3D toolkits and related windowing systems - Glut, Glt, Glui, FLTK, Mesa, Quesa, WxWindows, VTK, to name a few.

Unfortunately, although there are so many toolkits available, all have serious shortcomings. We finally narrowed down our selection to Glt and VTK; both are open source and support multiple platforms. Glt is relatively simple, but it has no available documentation (Jones and Snyder, 2003). VTK is very comprehensive and has some books about it, but it is designed mainly for scientific visualization and not for control purposes (Kitware, 2003). Both Glt and VTK do not have a windowing system built-in for interactive GUI. In the future, we need to find a good solution of adding a windowing system such as Qt to Glt and VTK to allow interactive 3D control. There are some solutions for integrating Qt with VTK, Glt and Glui, but, again, all have serious problems.

13.Networking and Web Services

Thus far, a comprehensive study on serial port communication programming thorugh Linux, from the graphical user interface to the control hardware, has been performed. As long as the control device uses a serial port interface, one can easily develop a working control system from the sample code provided so far.

Obviously, at this point, everything is networked together. Most control devices, including motion controllers, I/O controllers, and single board computers, come with Ethernet or WiFi as a standard interface. It must be noted though that most of these devices still come with serial ports as well and many still use a serial port for initial configuration or firmware upgrading.

All of these Ethernet devices come with a C++ class, or a Visual Basic ActiveX control (usually the control is built from the C++ class) that can be used to communicate to the device. Therefore, there is no need to build a C++ class like the CSerial class we built for serial port devices.

The new challenge relates to how to offer the control services over the Internet/Intranet, as now everything is networked. Web services are the easiest ways to provide a cross-network interface to existing code.

13.1 Web Services

On Windows, Web services are a new feature of the .NET Framework that lets client code access software components across networks using standard protocols such as SMTP and HTTP (Templeman and Olsen, 2003). Unlike other software components such as COM that are difficult to program, one does not need much specialized knowledge to write Web services. It is often very easy to provide a Web service interface to existing code. Web services use standard Internet data formats and protocols for communication, so it is much easier to build distributed control systems. Writing a remote client in Java (or any language) to access some Windows-based components written in C++, for example, is fairly easy.

There are many ways to create Web services in Windows; they can be based on ASP.NET or ATL (Active Template Library). There is no need to know ASP.NET and ATL, Visual Studio .NET hides all the details about them and one can create a Web service with just few mouse clicks. For control applications, ATL is the choice because it uses traditional C/C++ (unmanaged) that most control code still uses.

In order to investigate the possibilities of Web services for control, Web service for robot control is studied. The robot comprises a MEI 8-axis motion controller and a 4-axis TA9000 motion control development system, as described earlier. The Web service provides an interface for remote clients to access (control) the robot.

13.2 Robot Control C++ Class

First we need a C++ class for the robot, as the Web service will eventually call it upon client request. We build the robot control class, Controller, based on MEI's motion

software library. This class offers some very common robot control functions such as initialization, move, stop, etc. as follows:

```
// controller.h: interface for the Controller class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_CONTROLLER_H__6F4C78B6_FC02_4B1E_95B7_6EFBF5EF4F24__IN
CLUDED_)
#define
AFX_CONTROLLER_H__6F4C78B6_FC02_4B1E_95B7_6EFBF5EF4F24__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

//#if defined(MEI_RCS)
//static const char MEIAppRCS[] = "RCSHeader";
//#endif

//controller defines, count starts from 0
#define     MOTION_NUMBER     (0)  //default Motion Supervisor
#define     AXIS_NUMBER (8)         //total axes the controller handles
or in concern

//motion defines
#define AXIS_COUNT      (0)         //default axis count for a motion

#define MOTION_TYPE     (MPIMotionTypeTRAPEZOIDAL)     //default  motion
type

#define     VELOCITY    (1000)            //default velocity
#define ACCELERATION    (5000)      //default acceleration
#define     DECELERATION      (5000)      //default deceleration

#include "stdmpi.h"
//#include "stdmei.h"

class Controller
{
public:
      void Reset();
      void  Stop(long  *axisNum,  long  axisCount  =  AXIS_COUNT,  long
motionNum = MOTION_NUMBER);
      long  IsIdle(long  *axisNum,  long  axisCount  =  AXIS_COUNT,  long
motionNum = MOTION_NUMBER);
      double GetAxisPositionActual(long axis = 0);
      //Trapezoidal motion
      void MoveTrap(double *position, long *axisNum, long axisCount =
AXIS_COUNT, double velocity = VELOCITY, double accel = ACCELERATION,
double decel = DECELERATION, long motionNum = MOTION_NUMBER);
      void SetMotorAmpEnable(long number = 0, long ampEnable = TRUE);
      void Cleanup();
      void Init();
//    Controller();
      virtual ~Controller();
```

```
      Controller(int argc = 1, char *argv[] = NULL );

protected:
     MPIMotion CreateMotion(long *axisNum, long axisCount = AXIS_COUNT,
long motionNum = MOTION_NUMBER);
      MPIControl control;
      MPIControlType    controlType;
      MPIControlAddress controlAddress;
};

#endif
//  !defined(AFX_CONTROLLER_H__6F4C78B6_FC02_4B1E_95B7_6EFBF5EF4F24__INC
LUDED_)
```

13.3 ATL Web Service

We started the Web service project by following the Celsius–Fahrenheit converter example that is readily available. Then we simply add functions to control the robot such as controller initialization, motors enabling, robot move, stop, getting robot position, etc.

```
// NeesRbtAtl.h : Defines the ATL Server request handler class
//
#pragma once

#include "..\controller\controller.h"


typedef struct _RBT_POS {
      double X;
      double Y;
      double Z;
      double T;
} RBT_POS;

namespace NeesRbtAtlService
{
// all struct, enum, and typedefs for your webservice should go inside
the namespace

// INeesRbtAtlService - web service interface declaration
//
[
      uuid("FC710917-475B-4F74-B2A9-1D5FB50915AA"),
      object
]
__interface INeesRbtAtlService
{
      // HelloWorld is a sample ATL Server web service method.  It
shows how to
      // declare a web service method and its in-parameters and out-
parameters
      [id(1)] HRESULT HelloWorld([in] BSTR bstrInput, [out, retval]
BSTR *bstrOutput);
      // TODO: Add additional web service methods here
```

```
	[id(2)] HRESULT ConvertF2C([in] double dFahr, [out, retval]
double* dCels);
	[id(3)] HRESULT ConvertC2F([in] double dCels, [out, retval]
double* dFahr);
	[id(4)] HRESULT InitController();
	[id(5)] HRESULT EnableMotors([in] bool bEnable);
	[id(6)] HRESULT Move([in] double dX, [in] double dY, [in] double
dZ, [in] double dT);
	[id(7)] HRESULT Stop();
	[id(8)] HRESULT IsIdle([out, retval] long* lIdle);
	[id(9)] HRESULT GetAxisPosition([in] long lAxis, [out, retval]
double* dAxis);
	[id(10)] HRESULT GetRbtPos([out, retval] RBT_POS* posRbt);
};


// NeesRbtAtlService - web service implementation
//
[
	request_handler(name="Default", sdl="GenNeesRbtAtlWSDL"),
	soap_handler(
		name="NeesRbtAtlService",
		namespace="urn:NeesRbtAtlService",
		protocol="soap"
	)
]
class CNeesRbtAtlService :
	public INeesRbtAtlService
{
//protected:
//	Controller* m_pController;
public:
	// This is a sample web service method that shows how to use the
	// soap_method attribute to expose a method as a web method
	[ soap_method ]
	HRESULT HelloWorld(/*[in]*/ BSTR bstrInput, /*[out, retval]*/
BSTR *bstrOutput)
	{
		CComBSTR bstrOut(L"Hello ");
		bstrOut += bstrInput;
		bstrOut += L"!";
		*bstrOutput = bstrOut.Detach();

		return S_OK;
	}
	// TODO: Add additional web service methods here
	[ soap_method ]
	HRESULT ConvertF2C(/*[in]*/ double dFahr, /*[out, retval]*/
double* dCels)
	{
		if (dCels == 0) return E_POINTER;

		*dCels = ((dFahr - 32) *5) / 9.0;
		return S_OK;
	}

	[ soap_method ]
```

```
        HRESULT ConvertC2F(/*[in]*/ double dCels, /*[out], retval]*/
    double* dFahr)
        {
              if (dFahr == 0) return E_POINTER;

              *dFahr = (9/5.0 * dCels) + 32;
              return S_OK;
        }

        [ soap_method ]
        HRESULT InitController()
        {
              Controller controller;
              controller.Reset();
              return S_OK;
        }

        [ soap_method ]
        HRESULT EnableMotors(bool bEnable)
        {
              Controller controller;
              for(int i=0; i<=AXIS_NUMBER; i++) {
                    controller.SetMotorAmpEnable(i, bEnable);
              }
              return S_OK;
        }

        [ soap_method ]
        HRESULT Move(double dX, double dY, double dZ, double dT)
        {
              double dPos[4] = { dX, dY, dZ, dT };
              long lAxisNum[4] = { 0, 1, 2, 3 };

              Controller controller;
              controller.MoveTrap(dPos, lAxisNum, 3);
              return S_OK;
        }

        [ soap_method ]
        HRESULT Stop()
        {
              long axis[4] = {0,1,2,3};
              Controller controller;
              controller.Stop(axis, 3);
              return S_OK;
        }

        [ soap_method ]
        HRESULT IsIdle(long* lIdle)
        {
              long axis[4] = { 0,1,2,3 };
              Controller controller;
              *lIdle = controller.IsIdle(axis, 3, 0);
              return S_OK;
        }

        [ soap_method ]
```

```
        HRESULT GetAxisPosition(long lAxis, double* dAxis)
        {
              Controller controller;
              *dAxis = controller.GetAxisPositionActual(lAxis);
              return S_OK;
        }

        [ soap_method ]
        HRESULT GetRbtPos(RBT_POS* posRbt)
        {
              Controller controller;
              posRbt->X = controller.GetAxisPositionActual(0);
              posRbt->Y = controller.GetAxisPositionActual(1);
              posRbt->Z = controller.GetAxisPositionActual(2);
              posRbt->T = controller.GetAxisPositionActual(3);
              return S_OK;
        }

}; // class CNeesRbtAtlService

} // namespace NeesRbtAtlService
```

13.4 Web Services Client

In order to test this Web service and the robot over the Internet, we also need to build a Web service consumer (client). In Visual Studio .NET, allowing an application to access a Web service is very simple; all that is needed is to add to the project a Web reference. Once the reference is added, Visual Studio .NET creates a local head file, WebService.h, based on the remote Web services description. After that, using those functions provided by the Web service is no different than using local functions. The programmer can ignore anything about Web service and just focus on the functions. For example, the following code is for enabling the motors on the remote site:

```
private: System::Void checkBoxMotorEna_CheckedChanged(System::Object *
sender, System::EventArgs *  e)
              {
                    NeesRbtAtlService* ps = new NeesRbtAtlService();
                    ps->EnableMotors(checkBoxMotorEna->Checked);
              }
```

Note that everything is normal C++ (managed) and it is totally transparent to the client programmer. The above code is activated when the “Enable Motors” check box is checked, as shown in Fig. 13-1. The client program, wrote with the new Windows Forms, and provided buttons and progress bars for the user to control the remote robot and see its status graphically.

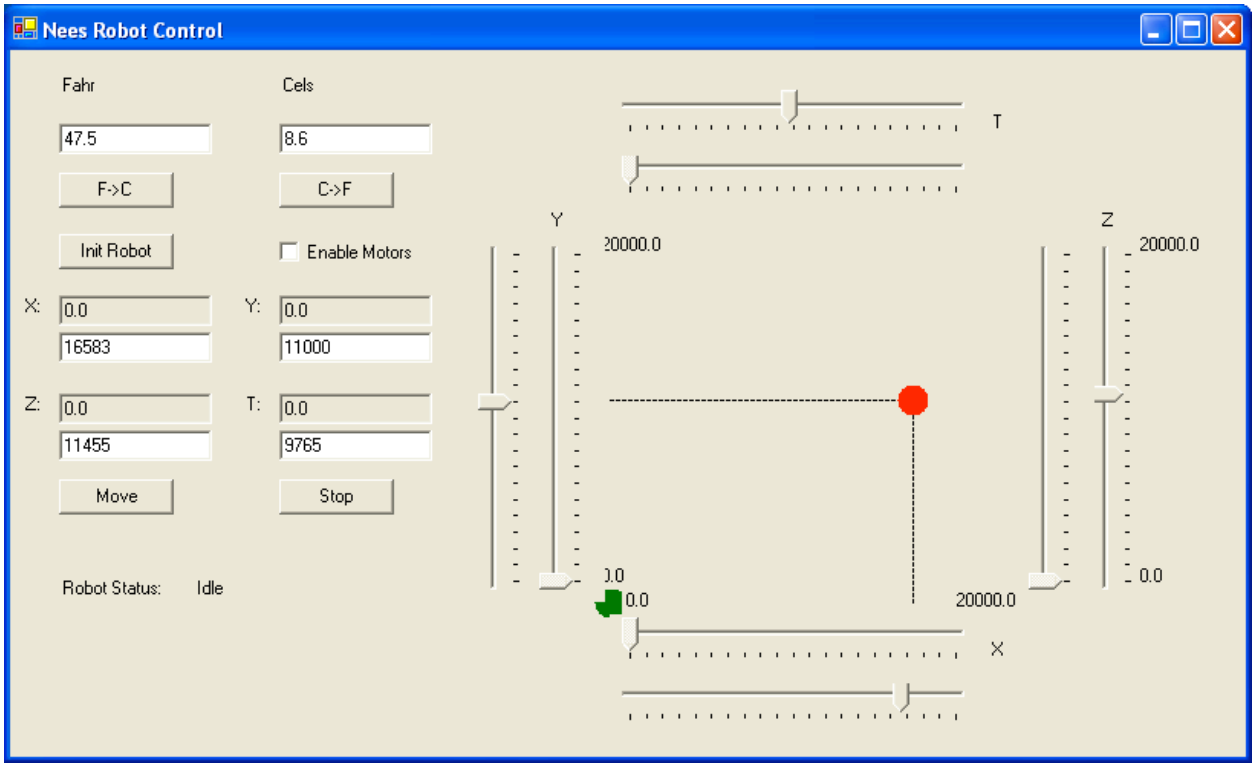

Figure 13-1 Web Service Client for Remote Robot Control

14. Globus, Linux Web Services and Beyond

14.1 Globus

In the last section, we experimented with Web services and robot control in Windows. In the Linux world, there are several available web services such as IBM Dynamic eBusiness and Sun ONE. Among them, we have found Globus Alliance to be the best for the following reasons:

- It has a better and more general architecture, the so-called Grid Computing, which comprises web services and other distributed computing.
- It is a research and development project; web services are focused more on business.
- It already has a very good open source toolkit – Globus Toolkit, for easily building Grids applications.

The Globus Toolkit contains a set of services and software libraries to support Grids and Grid applications. It includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. Compared to web services, a key feature of Grids is the resource management, which is very important for distributed and coordinated control. For example, it is possible to allow operation of a robot from several operators in different countries at the same time.

14.2 NTCP

In a project we are involved, the NEES (National E) project, we use a special protocol for remote robot operation, the NTCP protocol. NTCP stands for NEESGrid Teleoperated Control Protocol. It is based on the Grids idea and is built on the Globus Toolkit described above. NTCP includes both server and client. On the server side, it deploys the "plugin" idea; so one can easily develop a plugin for a specific control task and plug it into a NTCP server. On the client side, it provides APIs (application programming interfaces), the NTCPHelper library. We downloaded NTCP and OGSA (the latest Globus Toolkit). By following the NTCP client sample provided by NEES, we successfully compiled the NTCP server and a sample client. The client was able to access the server's built-in dummy plugin.

14.3 NTCP C plugin

In the last section, we successfully deployed NTCP (NEESGrid Teleoperated Control Protocol, a very good implementation of Globus Toolkit) with a JAVA plugin and client. However, almost all control hardware, such as motion controllers, still come with C code (drivers and sample applications) as opposed to JAVA. Therefore, the C plugin that is included in the NEESPop package is the one required for control and it needs further investigation.

We first tried the dummy and sample C plugins found in the CVS package, but we could not get them compiled. Therefore, we ended up installing the entire NEESPop package, which is huge and takes about 1GB storage space on RedHat 9. The installation was very time consuming and required several installation attempts. After the installation, we modified and compiled successfully the sample C plugin into a shared

object (SO). We let the NTCP server load this SO via the gateway (JAVA) plugin (all C plugins are loaded this way, i.e., through a JAVA gateway). We then made some necessary modifications to the JAVA client that was used in last section and it communicated successfully to the sample C plugin: connected to the server (in non-secure mode), opened a session, proposed a transaction, executed the move (for 30 seconds), checked the status and queried the control point.

14.4 Step-by-Step

Here we list the steps that we went through to make the C plugin to work, as there is no related documentation yet:

1) Follow the "hello world" sample and install at least the nclient CVS package. This write up gives an illustartion of NTCP and gets new users into the NTCP protocol with the least effort.
2) Install NeesPop. The dummy sample C plugin found in the CVS package cannot be compiled. Before installation, perl-Compress-Zlib-1.16-8.i386.rpm and perl-Archive-Tar-0.22-26.noarch.rpm must be installed. Without these two RPM packages, nstallation will fail. During the installation, NeesPop, MySql causes most of the trouble, mainly on passwords and computer name.
3) The sample C plugin is found under: /neesgrid-2.2-neespop/components/ntcp/BUILD/sample_plugin-1.0/sample_plugin.c. To compile it, follow the last two pages of "NTCP Sample Plugin Documentation." Then run "gpt-build -force gcc32dbgpthr" to get the "libntcp_sample_plugin_gcc32dbgpthr.so" compiled and installed to $GLOBUS_LOCATION/lib.
4) Edit "/$GLOBUS_LOCATION/server-config.wsdd". Change the ntcpBackendFactory parameter to: "org.nees.ntcp.plugins.gateway.GatewayPluginFactory".

5) Add "org.nees.ntcp.plugins.gateway.module" parameter right after it and set the value to: "$GLOBUS_LOCATION/lib/libntcp_sample_plugin_gcc32dbgpthr.so".

6) Note that $GLOBUS_LOCATION should be changed to real path (/usr/local/nees/opt/grid). These parameters tell NTCP to load the C gateway and the sample C plugin.
7) Start the NTCP server under $GLOBUS_LOCATION by "ant startContainer -Dservice.port=8090." The NTCP server will load the C gateway java code, which then loads the actual C plugin object.
8) Make the following changes to the nclient:
    a) Change the control point name "dummy" to "result1," "result2" or "result3" when calling getControlPoint function. The sample C plugin uses these control point names by default.
    b) After the "//Check the results" and before "//Need a loop until...", add:
    do {
    } while (trans.getState() != TransactionStateType.terminated);

This is because the default execution of the simulation lasts 30 seconds.

14.5 Test

Now start nclient by "ant run." The resultant screen is as shown in Fig. 14-1.

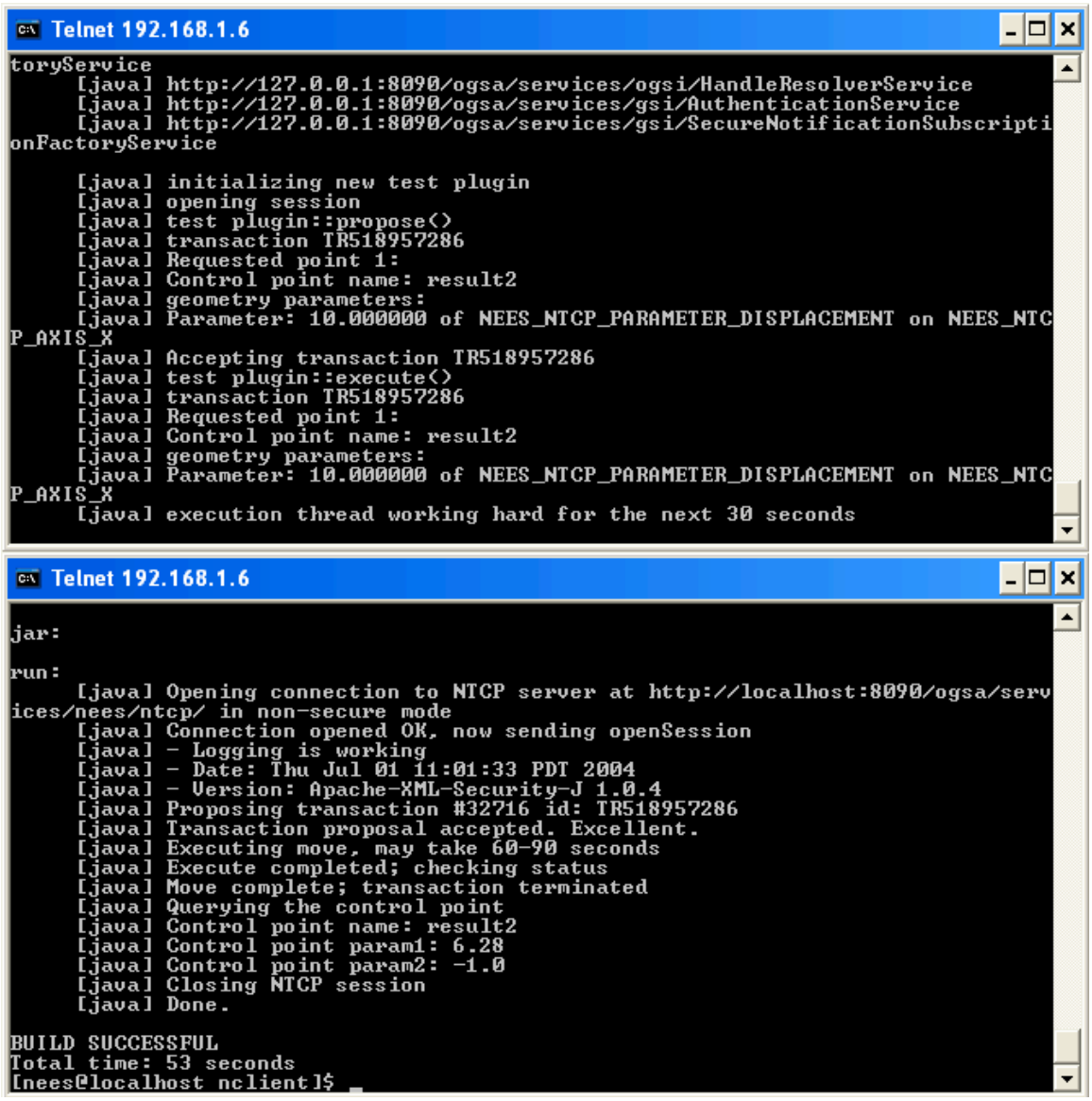

Figure 14-1 NTCP C Plugin Experiment

The upper terminal shows the server's output; the bottom terminal shows the client's output. So far, we have addressed control with Linux over a network. With this working C plugin, it is then fairly easy to add any control code, such as motion and robot control code, to do real-world remote control.

15. Summary

15.1 Conclusions

In this research project, we investigated Linux as an alternative operating system for general research and control applications. Linux and its source code are freely available and programmers all over the world contribute to its development. In just one decade, Linux now offers a first rate graphical desktop and integrated development environments, in addition to its advanced and comprehensive server facilities. We focused on using Linux to integrate a control system, for its graphical user interface and communication capabilities. We developed some ready-to-use C++ classes for serial port communication, which feature simplicity and efficiency and are multithreaded, event-driven and platform independent. We also demonstrated how to develop programs with an interactive graphical user interface for testing serial port devices and integrating these control devices into an implemented control system. We also investigated and demonstrated remote robot control over the Internet via Web services and grids computing. This study confirms that Linux is now a valuable and fully capable operating system for control, with mature desktop management systems and integrated development environments. Examples and experiments are given in this report including full source code and tutorials with which one can easily start deploying Linux.

15.2 Source Code

All the source code for all the projects described in this report are available in the company CD.

The Content of the CD:

NeesRobot – the NeesRobot solution (Visual Studio .NET)
- Compiled – ready to run executable codes
- Controller – the robot control C++ class
- NeesRbtAsp – ASP.NET version of Web service for robot control
- NeesRbtAtl – ATL version of Web service for robot control
- NeesRbtCtl – Web service client for robot control
- Rbt – Local robot control for VS.NET
- Rbt – Local robot control for VS 6.0

ORCA – Serial port communication for Windows

Projects – Linux projects
- cserial – C++ class for serial port communication
- glt – 3D programming with GLT
- helloahmct – Interactive user interface with KDevelop
- mykde – multithreaded and GUI programming
- MyQt – shows how simple a Qt program can be
- Myvtk – 3D programming with VTK
- Rh9kde – a simple test KDevelop projects under RedHat 9 environment
- serial – basic serial programming

References

Bennett, D.A., Feng, X., and Velinsky, S.A., Robotic machine for highway crack sealing, Transportations research record – journal of the transportation research board 1827, pp 18-26, 2003

Dalheimer, M. K., Programming with Qt, O'Reilly, c2002

Dalheimer, M.K., Embedded development with Qt/embedded, Dr. Dobb's journal, March 2002

Epplin, J., RT-Linux as an embedded operating system, Embedded Syst. Programming, Oct. 1997

Epplin, J., Linux as an embedded operating system, Embedded Syst. Programming, Oct. 1997

Feng, X., Bennett, D.A., and Velinsky, S.A., Embedded PC control system for a highway crack sealing machine, Proc. Of the ASME design technical conference, paper #DETC98/CIE-5531, 1998

Feng, X., Velinsky, A. S., and Hong, D., Integrating embedded PC and internet technologies for real-time control and imaging, vol. 7, no. 1, March, 2002

Feng, X., Mathurin R., and Velinsky, S. A., Practical, interactive and object-oriented machine vision for highway crack sealing, in press, 2004

Hassan, W., Writing real-time device drivers for telecom switches, part 1, Embedded Linux Journal, issue 05, 2001

Hassan, W., Writing real-time device drivers for telecom switches, part 2, Embedded Linux Journal, issue 06, 2001

Jones, H. and Snyder, M., Robotic control & 3D DUIs, Dr. Dobb's journal, January 2003

Kitware, Inc., The VTK user's guide, Kitware, Inc., 2003

Nakatani, B., Automating the physical world with Linux, part 1: control automation, ELJonline, http://www.linuxdevices.com/articles/AT2399207183.html, May, 2001

Nakatani, B., Automating the physical world with Linux, part 2: expanding control automations, http://www.linuxdevices.com/articles/AT3916001067.html, July, 2001

Templeman, J. and Olsen, A., Microsoft visual C++ .NET step by step, Microsoft press, 2003

Velinsky, S.A., Feng, X., and Bennett, D.A., Operator controlled, vehicle-based crack sealing machine, Heavy vehicle systems, vol. 10, no. 3, 2003

Part II - Methods for Generation of Smooth Trajectories for Robotic Systems

Summary

This research has dealt with the development of methods for generation of smooth trajectories of robotic end-effectors. Such trajectories are used for several highway applications such as robotic crack sealing and painting of roadway markings. The emphasis of this research has been on developing scientific methods that can provide the basis for more practical application oriented uses in highway maintenance. The aim has been to develop the scientific fundamentals while the implementations can be made in more task specific projects within the AHMCT (Advanced Highway Maintenance and Construction Technology) research center. A second aim has been to develop methods that can be published in peer reviewed conferences and journals to add scientific visibility to AHMCT and its research and to attract top quality students through the visibility provided by these publications for highway maintenance research. In these regards several scientific methods have been developed and published in scientific conferences and journals. This work in addition to its original aim has allowed development of scientific fundamentals for one application-specific project, namely the snow plow project within the AHMCT research center. In this regard, design principles have been developed for implementation of a mechatronics type magnetic sensing technology to replace analogue magnetometers for sensing of lateral positions of vehicles in snow plowing operations. In addition, a stochastic driver model has been developed that would be suitable for modeling driver behavior in snow and ice operations. This model can be used as a basis for simulating driver steering in snow plowing operations and can be used for design of driver assist systems.

Trajectory Generation Methods for Robotic Systems Based on Curve Type Algorithms

For robotic systems to be programmed to perform any tasks, smooth trajectories need to be generated. Such trajectories involve specifying the end effecter location at many discrete key or control positions and then finding a way to have the robotic system move such that the end-effector smoothly passes through these positions. This is especially important in Highway Maintenance applications such as crack sealing or robotic sign stenciling where roadway letter markings have special smoothness requirements. In other applications such as programming the long reach robotic arm of a snooper truck for bridge inspection, the smoothness of the trajectory is important to make the motion of the end-effector bucket comfortable for the workers riding in the bucket and performing the bridge inspection.

In this research several new methods have been developed to generate smooth trajectories for robotic systems. These methods have a broad applicability not only to robotic systems but also to other problems in computer graphics and animation. The problem with development of smooth trajectories for robotic end-effectors is however associated with the curved nature of the underlying space resulting in non-linear methods

for trajectory generation. This part of the research has emphasized development of methods based on algebra of linear spaces as compared to non-linear algebra of curved spaces. The methods developed have broad applicability to many problems including robot motion and trajectory planning as well as key framing in computer graphics and animation. Two papers have been published as a result of this part of the research [1, 2].

The first paper [1] have dealt with the development of two algorithms for the interpolation of given positions of a moving body by a smooth and fair motion, such that chosen feature points of the moving system run on smooth and fair paths. An algorithm is outlined which rely on known interpolatory variational subdivision for curves and on registration techniques from Computer Vision. For the numerical solution of the arising optimization problems a geometric method based on instantaneous kinematics is used. The result has been a computational method for motion design and trajectory generation that can be used in robotic applications.

The second paper [2] have dealt with the following problem that arises in robot motion planning: given N positions or key frames of a moving body like the end-effector of a robot at time instances $t_{i,}$ compute a smooth rigid body motion (or robot trajectory) that interpolates or approximates the given positions. A new algorithm is presented for this problem that can be considered as a transfer principle from curve design algorithms to motion design. It is proved in the paper that using this new algorithm the motion or robot trajectory generated has the same smoothness of the curve design algorithm employed. This means that high degree of continuity can be achieved and this important in highway maintenance applications such a painting of roadway markings involving letters for STOP, Pedestrian Crossing and so on. The results obtained in this research motivated additional research developments. These are discussed in a third paper [3] that has been published.

This third paper [3], dealt with the following problem which arises in robot motion planning, NC machining and computer animation: Given are a fixed surface and N positions of a moving surface such that the positions are in point contact with the fixed surface. Compute a smooth and fair Euclidean gliding motion of the first surface phi on the second surface which interpolates (or approximates) the given positions at time instances t(i). First we generalize interpolatory variational subdivision algorithms for curves to curves on surfaces. Second we study an unconstraint motion design algorithm which we then extend to the main contribution of this paper, an algorithm for the design of a motion constraint by a contacting surface pair. Both motion design algorithms use a feature point representation of the moving surface, subdivision algorithms for curves, instantaneous kinematics, and ideas from line geometry. Geometric methods are used for the numerical solution of the arising optimization problems. The results have applications in robot trajectory planning when the end-effector is constrained to move near or on a surface such as a roadway surface.

Trajectory Generation Methods for Robotic Systems Based on Path or Motion Error Functions

A second line of research pursued in this work has taken a totally different approach from the previously described method for development of smooth trajectories for robotic systems. This approach has tried to design motions using an optimized error function.

This has led to the development of a new approach for defining a metric for rigid body displacements [4]. The metric is defined locally based on a mapping of spatial displacements. The mapping is from one hundred years ago and is due to Eduard Study who mapped each position of a rigid body onto a point on a quadric, now called the Study quadric. This quadric is a six-dimensional rational hyper-surface, embedded in a seven-dimensional projective real space, called Study's soma space. In this work the Study's quadric is used to define a new metric for rigid body displacements based on an optimized local mapping of the quadric. The local mappings of the quadric are achieved using stereographic projections, resulting in an affine space where the Euclidean definition of a metric can be used for rigid body displacements and techniques from design of curves and surfaces can be directly utilized for motion design. The results are illustrated by examples. It has applications for mechanism design and robot motion and trajectory planning. This approach can be used to design smooth robot trajectories based on Chaikens method. It can also be used for design of mechanisms and a paper has been published [5] for this second application of the approach.

Scientific Fundamentals for the Snow Plow Project

The scientific results obtained under the first part of this research as described above motivated other application specific fundamental research and development at AHMCT research center. One such development was related to the design of a mechatronic sensing system for magnetic sensing for vehicle guidance. A system was developed and implemented for the snow plow project that used digital technology to replace the analogue magnetometers for sensing of vehicle lateral guidance using pavement embedded magnets. In this work the underlying scientific principles of this digital sensing technology was developed and published in a scientific journal [6].

In a second effort, the research was aimed at developing a driver assistance system for the snow plowing operations. It was desired to develop scientific fundamentals for simulating drivers so that design ideas for a driver assistance system can be tested on a computer before costly hardware development and field testing. This requires development of a driver model that would be suitable for modeling driver behavior in snow and ice operations. This work therefore has developed a stochastic form of a human driver model which can be used for simulating vehicle guidance and control.

The human motor-control function is complex and can be affected by factors such as driver's training and experience, fatigue, road conditions, and attention. The variations in these effects become more pronounced in hazardous driving conditions such as in snow and ice. An example of such driving conditions is snow removal operation in highway maintenance, where the use of a stochastic driver model seems to be more desirable. This work evaluated and extended existing models of a human driver including stochastic or statistical considerations related to differences in drivers' experiences and their conditions as well as variations in the effect of disturbances such as plowing forces. The aim was to develop a simulation environment that can be used in design and evaluation of driver assistance systems for snow removal operation in an Intelligent Transportation System (ITS) environment. The results have been accepted for publication as a technical paper in the Journal of Dynamic Systems, Measurement and Controls. Since the paper

has not yet appeared in the journal, it is included here in Appendix A entitled: A STOCHASTIC FORM OF A HUMAN DRIVER STEERING DYNAMICS MODEL

References

1. SUBDIVSION ALGORITHMS FOR MOTION DESIGN BASED ON HOMOLOGOUS POINTS, M. Hofer, H. Pottmann, and B. Ravani, *Advances in Robot Kinematics, ed. By J. Lenarcic and F. Thomas, Proc. Of ARK conference 2002, pp. 235-244.*
2. FROM CURVE DESIGN ALGORITHMS TO THE DESIGN OF RIGID BODY MOTIONS, M. Hoefer, H. Pottmann, and B. Ravani, *The Visual Computer,* 20 (2004); pp. 279-297.
3. GEOMETRIC DESIGN OF MOTIONS CONSTRINED BY A CONTACTING SURFACE PAIRS, M. Hofer, H. Pottmann, and B. Ravani, J. of Computer Aided Geometric Design, Vol. 20, 2003, pp. 523-547.
4. LOCAL METRICS FOR RIGID BODY DISPLACEMENTS, J. K. Eberharter, and B. Ravani, ASME *Transactions J. of Mechanical Design,* Vol. 126, Sept. 2004, pp. 805-812.
5. MOTION APPROXIMATION USING A STEREOPGRAPHIC PROJECTION OF STUDY'S SOMA SPACE, J. K. Eberharter and B. Ravani, *CD ROM Proc. Of 12th Romansy Conf, Montreal, Canada*, April 2004.
6. A MECHATRONICS SENSING SYSTEM FOR VEHICLE GUIDANCE AND CONTROL, S. Donecker, T. A. Lasky, and B. Ravani, *IEEE/ASME Trans. On Mechatronics,8 (4):500-510, Dec. 2003.*

Appendix A

A STOCHASTIC FORM OF A HUMAN DRIVER STEERING DYNAMICS MODEL

Magomed Gabibulayev
Researcher, Ph.D.*, *gabibulayev@ucdavis.edu
and Bahram Ravani
Fellow of ASME*, *professor*, *bravani@ucdavis.edu

Advanced Highway Maintenance & Construction Technology (AHMCT) Research Center
Department of Mechanical & Aeronautical Engineering
University of California - Davis

ABSTRACT

This work develops a stochastic form of a human driver model which can be used for simulating vehicle guidance and control. The human motor-control function is complex and can be affected by factors such as driver's training and experience, fatigue, road conditions, and attention. The variations in these effects become more pronounced in hazardous driving conditions such as in snow and ice. One example of such driving conditions is snow removal operation in highway maintenance, where the use of a stochastic driver model seems to be more desirable. This work evaluates and extends existing models of a human driver including stochastic or statistical considerations related to differences in drivers' experiences and their conditions as well as variations in the effect of disturbances such as plowing forces. The aim is to develop a simulation environment that can be used in design and evaluation of driver assistance systems for snow removal operation in an Intelligent Transportation System (ITS) environment.

INTRODUCTION

Development of a proper model of driver steering dynamics is important in many applications including assessment of highway safety and design and evaluation of advanced automotive technologies. Advanced automotive technologies have been playing a key role in the development of Intelligent Transportation Systems (ITS). Much of the reported work on driver steering dynamic models has involved using deterministic models to predict or simulate integrated driver vehicle responses in steering maneuvers. These models typically do not take into account differences in drivers' experiences and their conditions nor do they consider non-deterministic variations in road conditions. Stochastic variations in road conditions can become important in simulating driving under hazardous conditions such as in snow, ice, and fog. Furthermore, when there are more than just tire interactions between the vehicle and the roadway such as in simulating the steering dynamics of a vehicle used in snow plowing operations, stochastic modeling of road inputs can become important.

In our research in ITS, we have been working on the development of Advanced Snow Plowing Vehicles (ASPV) for highway maintenance applications. In this ITS work, we have been collaborating with the California Partners for Advanced Transit and Highways (PATH) program at the University of California at Berkeley where their vehicle control technology based on pavement embedded magnetic sensors [25] is being utilized to assist the snow plow operator for driver-in-the-loop guidance of the snow plow for lane keeping in snow whiteout conditions. The whiteout conditions occur in the initial plowing of the roadway when snow covers all roadway including roadway edges and shoulders whitening out all delineations and road boundaries and making it difficult for drivers to maintain their lanes especially around curves. As part of this collaborative effort, a driver assistance system for steering and lane keeping has been developed and incorporated into several snow plowing trucks for use in winter maintenance applications in California and Arizona. For evaluating different design alternatives and improvements, we have also been working on the development of a simulation environment that would allow analysis of steering dynamics of the snow plow vehicle and the human-in-the-loop driver assistance system. Such a system requires a model of the human driver steering dynamics. Because of the tedious task of controlling a snow plow in the hazardous conditions of snow and ice as well as variations in the road conditions, we decided that a stochastic model of the human driver would be more suitable for this application.

This paper discusses our theoretical work on developing a stochastic form of a driver-in-the-loop dynamic model for vehicle guidance and driver simulation. The benefit of such a model is that the differences in drivers' experiences and their conditions can be modeled statistically. In addition, variations encountered due to road conditions in plowing operations such as encountering ice or snow packs on the road can be modeled as a stochastic input to the vehicle dynamic model. Previous works on developing a human driver model for steering dynamics have mostly focused on developing deterministic models.

One of the first driver models originated from work for aerospace industry as a representation of pilot behavior, introduced by McRuer et al. [1,2]. Fig. 1 shows the driver model, which contains feedback loops for lateral position, Y_V, and heading angle ψ, from vehicle model G_V. These loops act through a compensatory term G_{DC} for lane position control; a feed-forward term G_{DA} to anticipate changes in the upcoming path; and a final term G_{DP}, which represents precognitive control.

Another driver model was proposed by Hess and Modjtahedzadeh in [3-6] (Fig. 2). This model also originated from work for aerospace industry as a representation of pilot behavior. In Fig. 2, the element G_{NM} represents a second order neuromuscular system of driver's arms. The element G_{P_1} has, as its input, the output of the neuromuscular system, $\delta(t)$ which is the driver's steering input to the vehicle. The element G_{P_2} receives its input from the output of G_{P_1}. Both of these elements represent feedback of variables, derived from the motion of human limbs and muscle tissue, and are referred as "proprioceptive" feedback elements. The element G_L is a time delay

element, representing human signal processing delays. The elements G_{NM}, G_{P_1}, G_{P_2} and G_L represent high frequency compensation to the vehicle model. The element G_C is the PD controller, which represents low frequency compensation to the vehicle model. Controller G_C is realized by a human driver and its output signal $u_1(t)$ represents a visual guidance cue. The term G_I is so-called "preview" element, including a first order filter and time advance factor, which represents the advancement of input, corresponding to a single constant "look-ahead distance" or "aim point" depending on a particular forward speed. Look-ahead distance is considered to be about 70 ft at 20-30 mph, which corresponds to about 1.5 seconds in time.

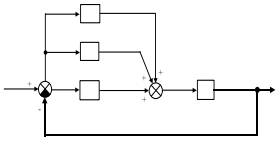

Fig. 1. McRuer's et al. compensatory/anticipatory/precognitive driver model

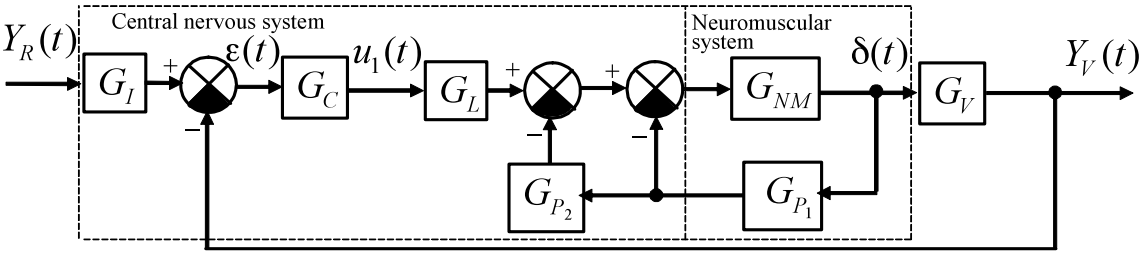

Fig. 2. Hess and Modjtahedzadeh's driver model

In Fig. 3, an optimal preview model G_{prev} [7] internalizes a vehicle model G_V as part of the driver response to the upcoming road path. The element G_{DLY} represents a sub-model of driver delay effect. Driver delay and preview time are selected, and optimal control techniques are used to generate control gains.

An optimal control model proposed in [8] is a combination of some of the features of [2] and [7]. The model provides a basis for predicting the effects of attention degradation on manual tracking performance from an increase in noise in operator response.

There are also neural network approaches to driver modeling for vehicle control [9], where driver model steering angle output is mapped from vision-based road views.

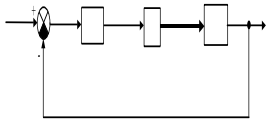

Fig. 3. MacAdam's optimal driver model

In [10] and [11] a driver model is considered in the form of a second order *Z*-transfer function

$$G_d(z) = \frac{b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{1}$$

with a delay of one sampling interval equal to 0.1 second.

Fig. 4 shows Ukawa's [12] driver model. The element $G_c(s)$ represents the transfer function of the PD controller. The element $G_{act}(s)$ represents an automatic steering actuator, which is treated as "an instrument which rotates the front wheel around the kingpin by hydraulic power steering with servomotor mounted on the steering shaft". The element $G_{act}(s)$ is estimated by "dead time plus second order lag element". The element $G_{\psi\delta}(s)$ represents the transfer function between the yaw angle ψ (output) and the steering angle δ (input). The term *L* is a so-called "predict-distance" (about 20 m).

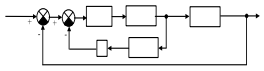

Fig. 4. Ukawa's driver model

There are some similarities among the reported driver models for vehicle simulation – all of them include, for example, time delay elements to account for human signal processing delays, they also include compensators to make the system stable and eliminate positioning error. For this work we have chosen the deterministic driver model of Hess and Modjtahedzadeh's [5] (we shall refer to this as the H-M model). We have selected this model since it allows modeling of proprioceptive feedback in the human motor control function. The approach presented here, however, is general and can be used to make most existing driver models stochastic. The H-M model is, therefore, used only

as an example base model for the development of the finalized stochastic model, where fluctuation of parameters around their average (expected) values (due to factors, such as training, experience, and fatigue) are considered. In the absence of such fluctuations, the model reduces to that of Hess and Modjtahedzadeh's [5], which has been validated with experimental data.

USE OF STOCHASTIC GAINS AND TIME CONSTANTS IN THE DRIVER MODEL

In this section we consider the possibility of having some fluctuation of gains and time constants in the system with some expected (average) values and standard deviations. The following assumptions are employed:

- All stochastic parameters and signals of the system are considered to be normally distributed. This assumption is valid in most cases according to the Central Limit theorem in statistics [13, 14, 16], which states that the sum of a large number of independent observations from the same distribution has, under certain general conditions, an approximate normal distribution. Moreover, the approximation steadily improves as the number of observations increases. For a complete statistical analysis of a system with normally distributed parameters and signals, only two statistical moments of each parameter and signal need to be considered: expected value(s) (first order moment) and auto-covariance function (second order moment). In cases, where the probability distributions of parameters or signals significantly differ from the normal distribution, more statistical moments (third, fourth, and higher order moments) have to be used in the statistical analysis.
- Only the range of change of three standard deviations for each parameter and signal around its expected value is considered. All values of parameters and signals out of this range are ignored. For normal distributions, 99.7% of all values of each parameter fall into this range [13, 14, and 16], i.e. only one out of four hundred possible values of each parameter (signal) fall out of this range. This assumption is widely used in statistics to simplify analysis. In other words, we assume that all normally distributed parameters and signals have minimum (expected value minus three standard deviations) and maximum (expected value plus three standard deviations) values.
- A statistical linearization of non-linear elements does not significantly influence the accuracy of the results. A statistical linearization differs from a traditional linearization, for which small deviations from a point of linearization are assumed. In general, when statistical linearization is applied, a memory-less non-linear element, is replaced by two time-varying coefficients – one of them is used for determining expected values (usually denoted as $k_0(t)$), another one is for determining centered values (usually denoted as $k_1(t)$). In the case of multiplication of two stochastic signals, the non-linear element is replaced with a time-varying input of expected values $F_0(t)$ of this multiplication and two coefficients $k_1(t)$ and $k_2(t)$ for each multiplied signal respectively for determining centered values. These parameters ($F_0(t)$, $k_0(t)$, $k_1(t)$, and $k_2(t)$) vary in time in such a way that in most cases the

system's behavior is very close to that of the non-linear one. This property of the statistical linearization has been demonstrated in the literature [18, 19, and 23]. Another effect of such a replacement is that a spectral density of the output signal of a non-linear element[1] is approximated to be proportional to the spectral density of the input signal of the non-linear element. This approximation works well in most cases for systems having linear elements with filtering properties, so that high-frequency changes of the spectral density of the output signal of the real nonlinear element are filtered out. There are some linear elements with pronounced filtering properties in the system considered in this work (G_{NM}, G_{P_1}, G_{P_2}, G_V); therefore, the statistical linearization is assumed to work well.

- Stochastically changing parameters are considered to be uncorrelated, i.e. cross-covariance functions of these parameters are considered to be equal to zeros. In case if any correlations are discovered among the parameters, their cross-covariance functions have to be involved in the calculations.
- Expected values of multiplication of non-zero-mean and zero-mean stochastic signals are considered to be negligibly small in comparison with the expected values of the multiplied non-zero-mean signal. This assumption was confirmed by simulation results related to this work.
- When the Spectral Methods (see [13, 14, 17-19, 23]) in $L^2[0,T]$ domain are applied, the solution is considered to converge with a limited number of coefficients of the Fourier decomposition for each signal and element in the block diagrams. When solving matrix algebraic equations, the inverse matrices are assumed to exist and Fourier series are assumed to converge. Normally, a small number (4-8) of Fourier coefficients is enough for the solution to converge. In rare cases this number needs to be increased several times. Changing the number of Fourier coefficients can also resolve the issue of existence of the inverse matrices since by changing this number, matrix dimensions also change. Very often bad convergence happens only on the right bound of the considered time interval [13] and eliminating last several points of calculations can resolve the problem. In this work we used 128 Fourier coefficients (the Basis of Walsh functions is used) because taking this particular number of coefficients does not complicate a problem at all when a computer is used for simulations. No convergence problems were encountered in this work.
- A two-degree-of freedom bicycle model is considered to be sufficient for lateral vehicle control studies. Small wheel angle approximation is considered here and the forward speed of the vehicle is assumed to be constant. The method of statistical analysis developed here will work for more complex models; however, the bicycle model is assumed to be sufficient here, and supports clear exposition of the method.

If a gain K_y has its average value $m_{K_y} = 0.4$, minimum value $K_{y\min} = 0.2$ and maximum value $K_{y\max} = 0.6$, we can find its standard deviation with the probability of 99.7%, assuming that we have normal distribution of this parameter, as follows [13]

[1] Spectral densities are used for determining variances of signals of stationary systems only; two-dimensional auto-covariance functions have to be used for analysis of non-stationary systems.

$$\sigma_{K_y} \cong \frac{\left|K_{y\max} - m_{K_y}\right|}{3} = \frac{\left|m_{K_y} - K_{y\min}\right|}{3} = \frac{\left|0.6 - 0.4\right|}{3} = 6.67 \cdot 10^{-2}. \quad (2)$$

Similarly, having information about minimum and maximum values of any normally distributed parameter of the system, we may use (2) for determining a standard deviation of that parameter, assuming that its expected value is half of the sum of its maximum and minimum values.

It is well-known that any stochastic parameter can be represented as the sum of its expected (mean) value(s) and its centered (zero-mean) value(s) [13-19]. For example, for the gain K_y we may write

$$K_y(t) = m_{K_y}(t) + \overset{\circ}{K}_y(t), \quad (3)$$

In this equation, $m_{K_y}(t)$ is the expected value of the gain $K_y(t)$, and $\overset{\circ}{K}_y(t)$ is the centered value of the gain $K_y(t)$. Note the difference of this representation with one of the linearization with small deviations around the point of the linearization. Equation (3) is not a result of a linearization, but a representation of a stochastically changing parameter K_y. In the representation (3), zero-mean values $\overset{\circ}{K}_y(t)$ are not necessarily small. For centered values we will use letters with symbol "$\circ$" on the top. In general, expected values may change with time. Processes and parameters with time-varying expected values and/or standard deviations are called "non-stationary".

Consider a truck such as a snowplow as shown in Fig. 5. The control system modeling the lateral movement of this vehicle in its lane of travel, using the H-M driver model, is shown in Fig. 6. Modeling lateral displacement of a vehicle in its lane of travel is important in simulating automatic or driver assisted lane keeping systems or in developing automatic control systems for power assisted lane keeping purposes while driving. In Fig. 6, elements of the driver model are described by the following differential equations:

Compensator G_C:

$$u_1(t) = K_y\left(T_3 \frac{d}{dt}\varepsilon(t) + \varepsilon(t)\right), \quad (4)$$

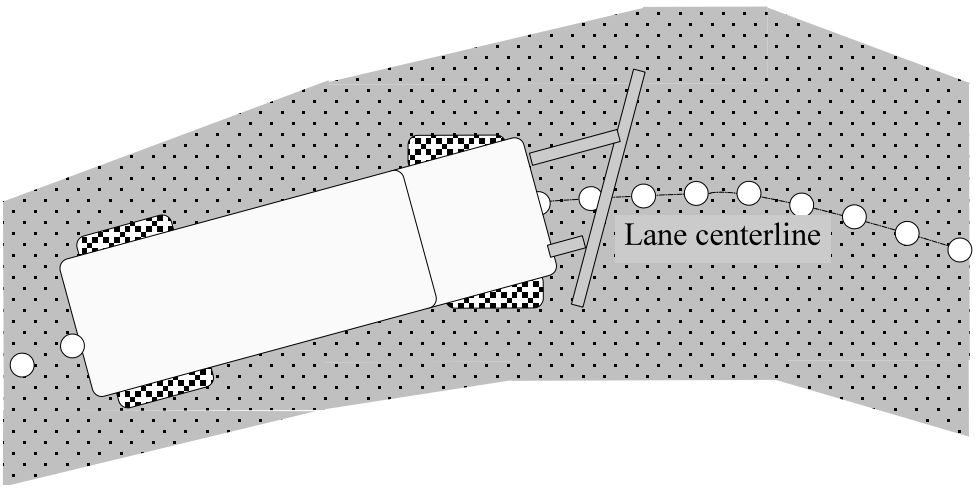

Fig. 5. Top view of snowplowing operation

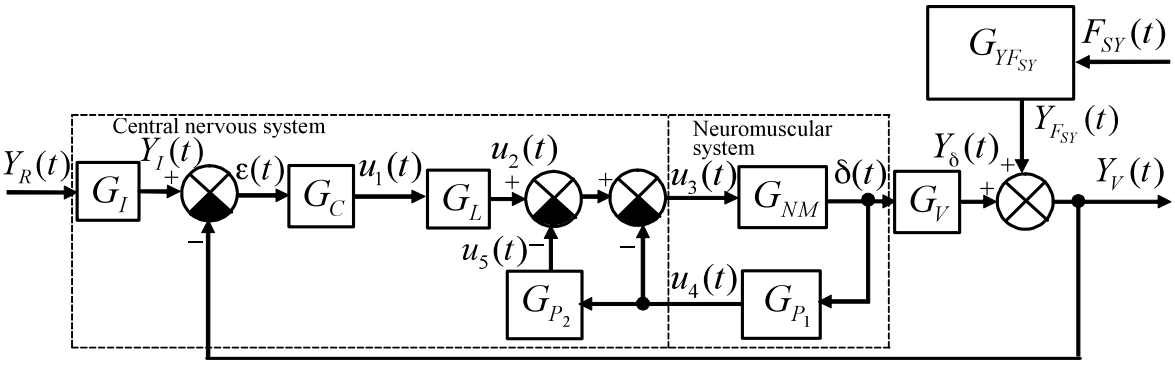

Fig. 6. Control system for lateral displacement of the snowplow with Hess-Modjtahedzadeh's driver model

Time delay element G_L (a third order Pade approximation is used with a time delay τ_0 of 0.15 *s*):

$$\begin{aligned} &1\cdot 10^{-4}\frac{d^3}{dt^3}u_2(t)+8\cdot 10^{-3}\frac{d^2}{dt^2}u_2(t)+0.267\frac{d}{dt}u_2(t)+3.55u_2(t)= \\ &-1\cdot 10^{-4}\frac{d^3}{dt^3}u_1(t)+8\cdot 10^{-3}\frac{d^2}{dt^2}u_1(t)-0.267\frac{d}{dt}u_1(t)+3.55u_1(t), \end{aligned} \tag{5}$$

Element G_{NM}, describing the neuromuscular system of driver's arms:

$$\frac{d^2}{dt^2}\delta(t)+2\xi\omega_n\frac{d}{dt}\delta(t)+\omega_n^2\delta(t)=\omega_n^2 u_3(t), \tag{6}$$

Proprioceptive element G_{P_1} :

$$\frac{d}{dt}u_4(t)+\frac{1}{T_1}u_4(t)=K_1\frac{d}{dt}\delta(t), \tag{7}$$

Proprioceptive element G_{P_2} :

$$\frac{d}{dt}u_5(t)+\frac{1}{T_2}u_5(t)=K_2u_4(t), \tag{8}$$

Inertial part of the preview element G_I :

$$T_4\frac{d}{dt}Y_{Ii}(t)+Y_{Ii}(t)=Y_R(t), \tag{9}$$

In this last equation, $Y_{Ii}(t)$ is the output signal of the inertial part of the preview element G_I (note that in Fig. 6, the term $Y_I(t)$ is the full output signal of the preview element). Time advance part of the preview element with a time advance factor τ_p [5] can be implemented just by shifting the calculation results for the lateral displacement of the vehicle (and other signals in the closed loop) τ_p seconds back in time, considering that events happen τ_p seconds earlier. We will use this approach in the next section to implement this time advance element, which acts contrary to the time delay element.

In Fig. 6, $F_{SY}(t)$ is a lateral component of the force acting on the plow from the snow (see also Fig. 16 and Table 1 in the next section). Plowing force changes stochastically due to having ice or snow packs on the road and non-ideal road surface conditions. The term $G_{YF_{SY}}$ represents the element describing how the disturbance force $F_{SY}(t)$ effects the lateral displacement of the vehicle. Element G_V describes the vehicle model, relating the steering wheel angle to the lateral displacement of the vehicle. Differential equations describing elements G_V and $G_{YF_{SY}}$ are represented in the next section.

When the gain $K_y(t)$ is stochastic as given by equation (3), it can be replaced in the system (see Fig. 6 and equation (4)) by the equivalent block diagram form shown in Fig. 7.

Having any stochastic parameter in a closed loop makes all the signals in the loop stochastic. Therefore, an input signal $\varepsilon(t)$ of the gain block along with the output signal $\varepsilon_{out}(t)$ will be stochastic. To be able to deal with time-varying stochastic parameters in the system, we have to work with differential equations (or their block diagram representations [13, 14, 18, 19]) of each element in the block diagram (see Fig. 6), not with transfer functions since the Laplace transformation cannot be applied to most time-varying systems[2]. In this work, we use the Discrete

[2] The Laplace transformation works with limited classes of time-varying differential equations, having, for example, only polynomial or exponential coefficients; equations after transformation are not algebraic as in case of the Laplace

Fast Fourier Transform (DFFT) to transfer differential equations (or their block diagram representations), describing each element in the block diagram of Figure 6, to spectral characteristics in $L^2[0,T]$ domain, which can be used for analysis of both linear time invariant and time-varying systems. In other words, each element in Fig. 6 is replaced by its spectral characteristic obtained from the differential equation describing the element (time-varying or time-invariant) and spectral methods are used for simulations, i.e. we find statistical characteristics (expected values and covariance functions) of the lateral displacement of the vehicle in the spectral domain (we can also find statistical characteristics of any signal in the closed loop) using block diagram transformations in this domain similar to those applied to transfer functions (for serial, parallel and feedback connections of elements). Applying the Inverse Discrete Fast Fourier Transform (IDFFT) allows us to transfer the spectral characteristics of the expected values and covariance functions of signals to the time domain, so that we are able to trace in time the expected values of the lateral displacement of the vehicle and all possible changes of it around the expected values with the effect of changing internal driver parameters and plowing forces.

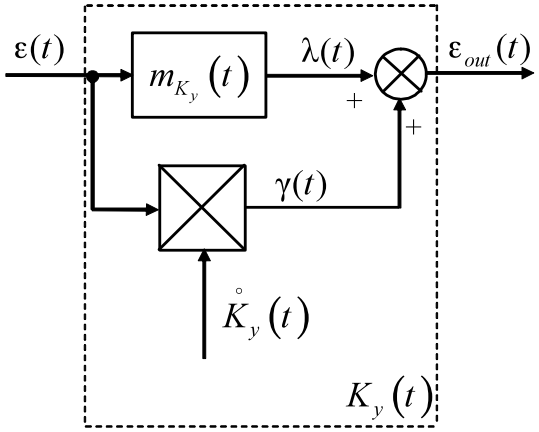

Fig. 7. Equivalent block diagram representation of a stochastic gain K_y

As we can see from Fig. 7, there is a multiplication of two stochastic processes $\varepsilon(t)$ and $\overset{\circ}{K}_y(t)$ in the system, which we will call as "product non-linearity". According to the statistical theory applied in control systems (see, for example, [16]), for statistically linearized memory-less non-linearity $X(t) = \chi(Y_1, Y_2, ..., Y_n)$ with several inputs (see Fig. 8 and 9), output signal of the nonlinear element $X(t)$ can be represented in the following form

$$X(t) \cong F_0 + \sum_{i=1}^{n} k_i \overset{\circ}{Y}_i(t). \tag{10}$$

In this equation, the statistical characteristic F_0 represents the expected values of the output signal $X(t)$ of the nonlinear element, $\overset{\circ}{Y}_i(t)$ ($i = 1,...,n$) – inputs of the nonlinear element and k_i - coefficients of the statistical linearization for *i*'s input, respectively. F_0 and k_i can be found from the following equations (see, for example, [16])

---

transformation of linear time invariant differential equations; therefore, this transformation is rarely used for time-varying systems.

$$F_0 = E\left[\chi\left(Y_1, Y_2, ..., Y_n\right)\right], \tag{11}$$

$$\sum_{i=1}^{n} k_i E\left[\overset{\circ}{Y}_i(t)\overset{\circ}{Y}_j(t)\right] = E\left[\chi\left(Y_1, Y_2, ..., Y_n\right)\overset{\circ}{Y}_j\right]. \tag{12}$$

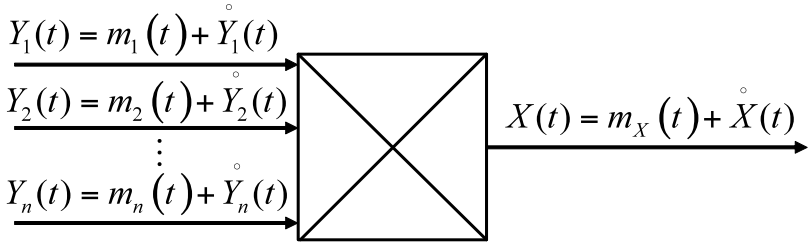

Fig. 8. A block diagram representation of a product non-linearity with *n* inputs

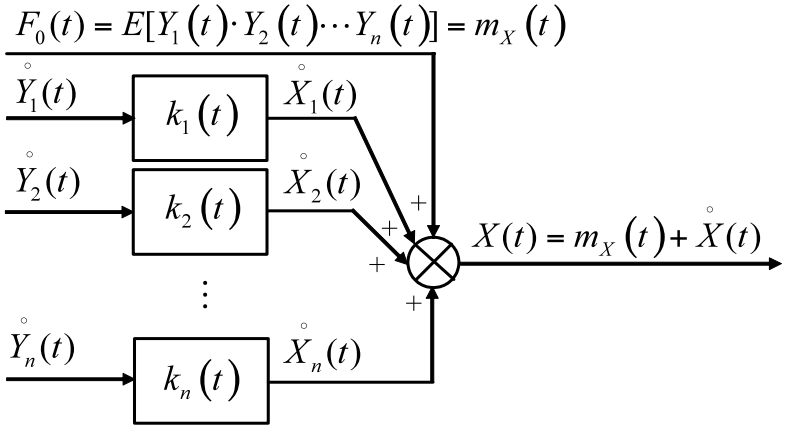

Fig. 9. A block diagram representation of a statistically linearized product non-linearity with *n* inputs

For a multiplication of two stochastic signals we will have

$$F_0 = E\left[\chi\left(Y_1, Y_2\right)\right] = E\left[Y_1(t)Y_2(t)\right] = E\left[\overset{\circ}{Y}_1(t)\overset{\circ}{Y}_2(t)\right] + m_{Y_1}(t)m_{Y_2}(t) = D_{Y_1Y_2}(t) + m_{Y_1}(t)m_{Y_2}(t), \tag{13}$$

$$k_1(t) = m_{Y_2}(t), \qquad k_2(t) = m_{Y_1}(t), \tag{14}$$

where $D_{Y_1Y_2}(t)$ is a cross-variance of the $Y_1(t)$ and $Y_2(t)$.

In our case (see Fig. 7, and note that $\overset{\circ}{K}_y(t)$ is a zero-mean signal) $Y_1(t) = \varepsilon(t)$, and $Y_2(t) = \overset{\circ}{K}_y(t)$, therefore,

$$F_0 = D_{\varepsilon K_y} + m_{Y_1}(t)m_{Y_2}(t) = D_{\varepsilon K_y}, \; k_1(t) = m_{Y_2}(t) = 0, \; k_2(t) = m_{Y_1}(t) = m_\varepsilon(t), \tag{15}$$

where $D_{\varepsilon K_y}$ is a cross-variance of signals $\varepsilon(t)$ and $\overset{\circ}{K}_y(t)$.

Note that the expected value $m_\gamma(t)$ of the signal $\gamma(t)$ (see Fig. 7) is

$$m_{\gamma}(t) = E\left[\gamma(t)\right] = E\left[\varepsilon(t)\overset{\circ}{K}_y(t)\right] = E\left[\overset{\circ}{\varepsilon}(t)\overset{\circ}{K}_y(t)\right] = D_{\varepsilon K_y}(t). \tag{16}$$

This is the expected value of the product of the zero-mean function $\overset{\circ}{K}_y(t)$ and non-zero-mean function $\varepsilon(t)$ - in most cases this product is negligibly small in comparison with the expected value of the multiplied non-zero-mean signal and can be neglected, so that $F_0 \cong 0$.

Now let us consider a driver model with three stochastic parameters K_y, T_3 and T_4 with some expected values and standard deviations. The gain K_y is related to the power (driver's motor control function), which the driver can apply to the steering wheel by his hands. Variations in this gain are influenced by the driver's performance and conditions such as fatigue. The time constant T_3 is related to the preview distance or visual cues used by the driver. It is a function of the vehicle speed and driver's aim point on the roadway. The driver steers the vehicle towards this aim point. It is influenced by the driver's experience and the roadway conditions. The time constant T_4 plays the role of the delay effect for the road preview by the driver. This is related to the driver's sensory perception of the road conditions. In winter maintenance, snow, rain, and fog, as well as unexpected emergency events can cause variations and changes in the time constants T_3 and T_4.

It should be pointed out that the emphasis of the paper has been on developing a stochastic form of a driver model. The model is only the first step towards providing a method for accounting for individual differences in drivers. Much more research would be needed to have the parameters associated with individual drivers to be explicit in the model. We hope that this would be something that can be achieved in the future.

When all the stochastic parameters and signals in the system are normally distributed, only two statistical moments of each parameter and signal have to be used for stochastic analysis: expected value(s) and auto-covariance (or autocorrelation) function. If distribution(s) of signals is (are) not Gaussian, then more statistical moments have to be involved in calculations, which might complicate getting results. Expected values, variances and auto-covariance function of any stochastic parameter or input signal $Y(t)$ can easily be determined from experimental data [13, 17] as follows (suppose we have N experimental results):

$$m_Y^*(t_i) = \frac{1}{N}\sum_{k=1}^{N} y_k(t_i) \tag{17}$$

– expected value of signal $Y(t)$ at time t_i,

$$D_{YY}^*(t_i) = \frac{1}{N-1}\sum_{k=1}^{N}\left(y_k(t_i) - m_Y^*(t_i)\right)^2 \tag{18}$$

– variance of signal $Y(t)$ at time t_i,

$$R^*_{YY}(t_i,t_j)=\frac{1}{N-1}\sum_{k=1}^{N}\left(y_k(t_i)-m^*_Y(t_i)\right)\left(y_k(t_j)-m^*_Y(t_j)\right) \tag{19}$$

− auto-covariance function of signal $Y(t)$ for times t_i and t_j.

If exact experimental data are not available, but there is information about minimum and maximum values of a stochastic parameter or a signal and if expected value(s) of the parameter or the signal is (are) close to average value(s), then we can say that we have a normally distributed parameter or signal. In many cases we can approximate auto-covariance function by six more commonly used ones [13, 17] (see equation (2) for estimation of standard deviations):

$$R_{YY}(t_1,t_2)=D_{YY}e^{-\alpha|t_2-t_1|}, \tag{20}$$

$$R_{YY}(t_1,t_2)=D_{YY}e^{-\alpha|t_2-t_1|}\cos\left(\beta(t_2-t_1)\right), \tag{21}$$

$$R_{YY}(t_1,t_2)=D_{YY}e^{-\alpha|t_2-t_1|}\left(\cos\left(\beta(t_2-t_1)\right)+\frac{\alpha}{\beta}\sin\left(\beta(t_2-t_1)\right)\right), \tag{22}$$

$$R_{YY}(t_1,t_2)=D_{YY}e^{-\alpha^2|t_2-t_1|^2}, \tag{23}$$

$$R_{YY}(t_1,t_2)=D_{YY}e^{-\alpha^2|t_2-t_1|^2}\cos\left(\beta(t_2-t_1)\right), \tag{24}$$

$$R_{YY}(t_1,t_2)=D_{YY}e^{-\alpha|t_2-t_1|}\left(1+\alpha|t_2-t_1|\right), \tag{25}$$

where α and β − the so-called "fading" parameters, choosing which depends on how fast a stochastic parameter or signal $Y(t)$ changes in time.

Similar to equation (3), time constants T_3 and T_4 can be represented as follows:

$$T_3(t)=m_{T_3}(t)+\overset{\circ}{T}_3(t), \tag{26}$$

$$T_4(t)=m_{T_4}(t)+\overset{\circ}{T}_4(t). \tag{27}$$

Since the gain K_y was already separated from the compensator G_C, what is left of the compensator G_C can be written in the form of the following time-varying differential equation:

$$\varepsilon_{out_1}(t) = T_3(t)\frac{d\varepsilon_{in_1}(t)}{dt} + \varepsilon_{in_1}(t). \tag{28}$$

A combination of equations (26) and (28) can be introduced in a block diagram form as shown in Fig. 10, where D represents a differentiator. This is similar to block diagram representation of the gain K_y.

In developing a block diagram representation of the inertial part of the element G_I, it is more convenient to define a stochastic natural frequency of this element rather than using a time constant $T_4(t)$. This natural frequency is defined by:

$$\omega_4(t) = 1/T_{44}(t) = m_{\omega_4}(t) + \mathring{\omega}(t), \tag{29}$$

The block diagram representation of the combination of equation (29) and the differential equation describing the inertial part of the element G_I, namely

$$\frac{d(Y_{Ii}(t))}{dt} + \omega_4(t)Y_{Ii}(t) = \omega_4(t)Y_R(t), \tag{30}$$

is shown in Fig. 11, where the block element with the sign $\int$ represents an integrator.

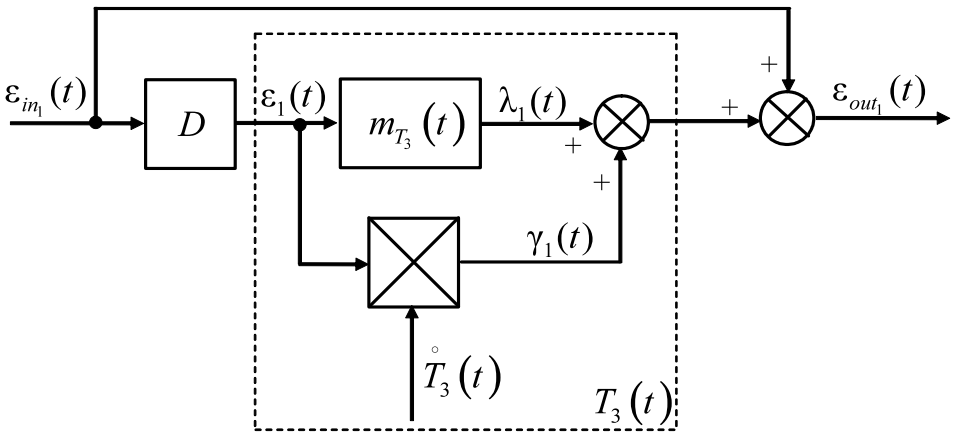

Fig. 10. Equivalent block diagram for a combination of (26) and (28)

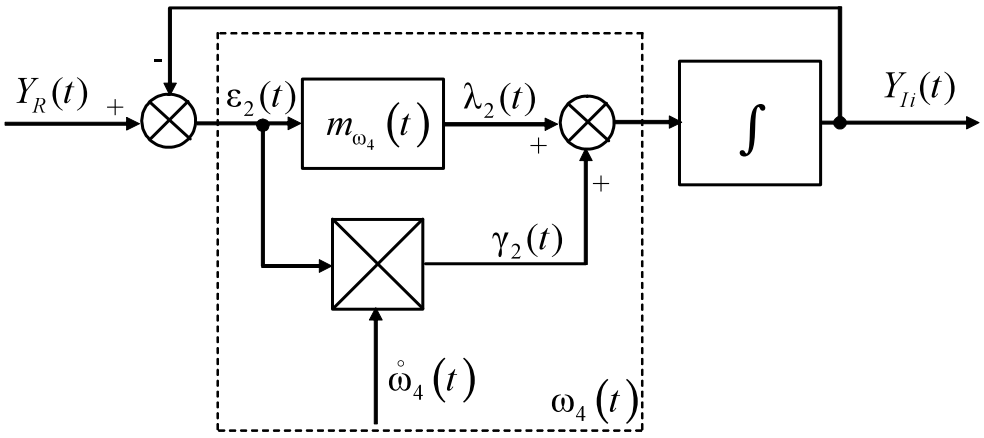

Fig. 11. Equivalent block diagram for a combination of (29) and (30)

As we can see from Figures 7, 10, and 11, a "product non-linearity" appears for every stochastic parameter in the system. Each "product non-linearity" has to be linearized according to statistical linearization procedure, described so far.

After statistical linearization of all "product non-linearities", two different block diagrams have to be used for simulation – one of them is for expected values of the signals, another one is for centered values of the signals. The block diagram for expected values is shown in Fig. 12 with stochastic parameters equal to their expected values. In Fig. 12, an element G_E is an equivalent block diagram representation of a part of the block diagram shown in Fig. 6, combining the elements G_L, G_{NM}, G_{P_1}, G_{P_2} and G_V. Element G_{I_a} represents a time advance part of the preview element with the time advance factor τ_p. If we consider constant expected values of stochastic parameters, the expected values of the lateral displacement of the vehicle will be the same as that of the H-M driver model. The method of statistical analysis, developed in this work, allows considering time-varying expected values of driver and vehicle parameters since a transformation from differential equations to spectral characteristics is used. Fig. 13 shows the equivalent block diagram for expected values of the signals with respect to spectral characteristics. Fig. 14 shows a block diagram for centered values of the signals. Fig. 15 shows the equivalent block diagram for centered values of the signals with respect to spectral characteristics.

To make sure that the representations (3), (26), (27), and (29) of stochastic parameters, and statistical linearization of product nonlinearities are valid for large deviations of these parameters around their expected (average) values; simulations were done for the element described by equation (30). The expected values and variances of the output signal of this element were obtained by two methods: 1) Monte Carlo method (20,000 runs of the program), and 2) Method, developed in this work, which authors call "A spectral method of analysis of stochastic non-linear non-stationary systems with the use of statistical linearization" (one run of the program). A relative error for the expected values came out to be less than 1%. A relative error for the upper boundaries was less than 2%. A relative error for the lower boundaries was less than 3%.

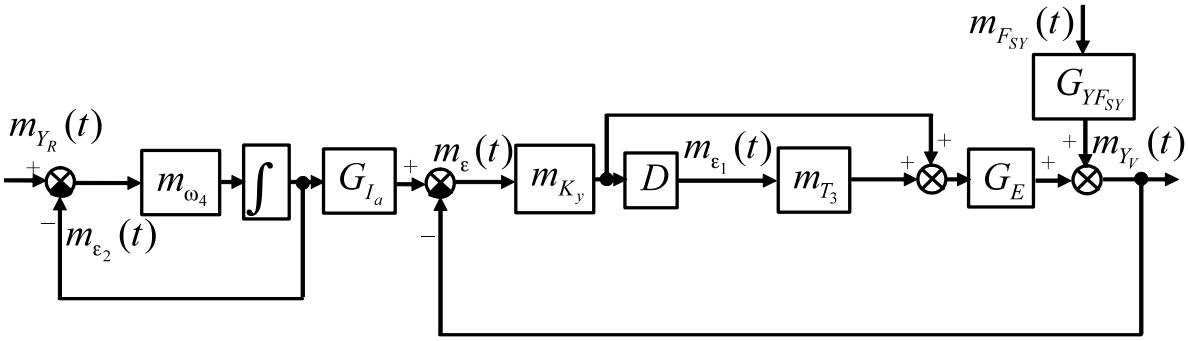

Fig. 12. A block diagram for expected values of signals

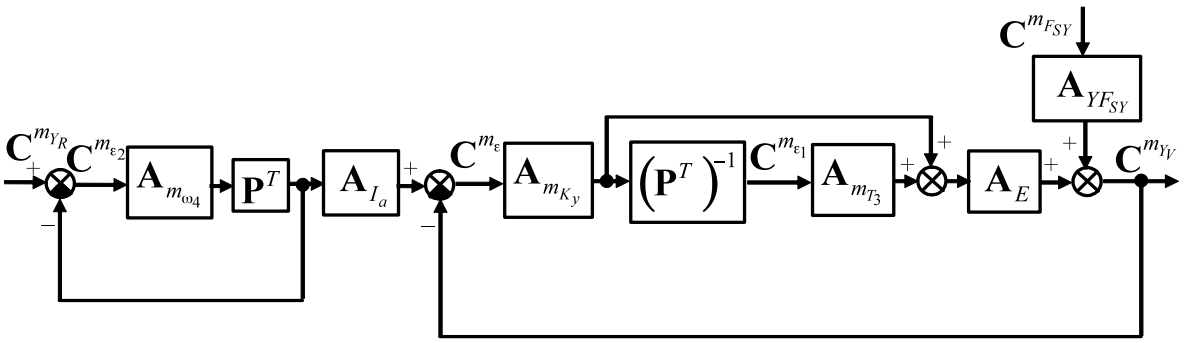

Fig. 13. Equivalent block diagram for expected values of signals with respect to spectral characteristics

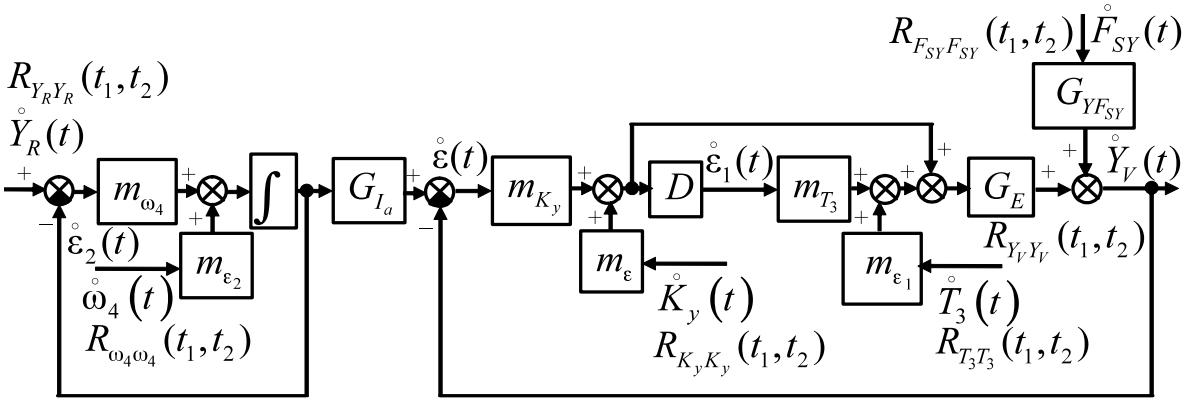

Fig. 14. A block diagram for centered values of signals

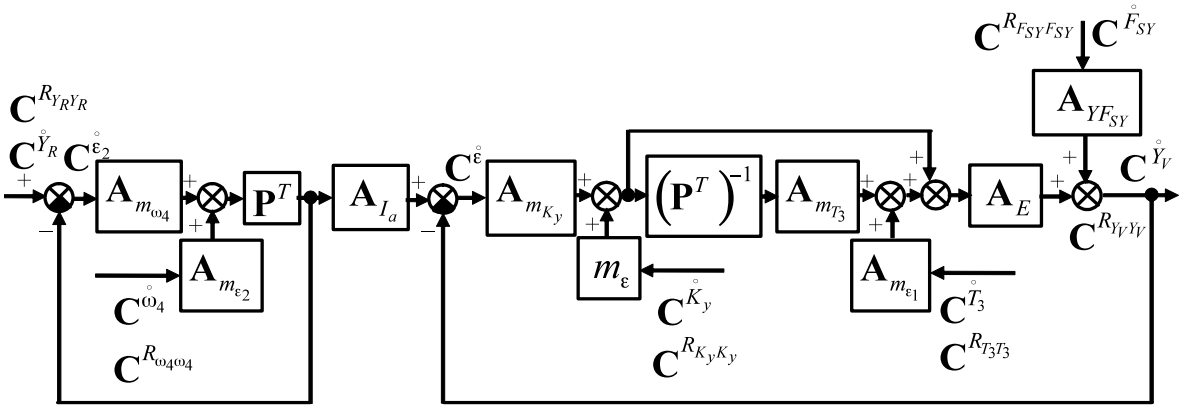

Fig. 15. Equivalent block diagram for centered values of signals with respect to spectral characteristics

In order to find the expected values and auto-covariance function of the lateral displacement of the vehicle, a method with the following three major steps is applied:

Step 1: Discrete Fast Fourier Transform (DFFT) is used to find spectral characteristics (sets of Fourier coefficients) of statistical characteristics (expected values and auto-covariance functions) of all signals and elements of the system (see Figures 12 and 14). In other words, block diagrams for expected and centered values have to be transformed to equivalent block diagrams with respect to spectral characteristics (see Fig. 13 and 15), applying the Discrete Fast Fourier Transform (each centered value corresponds to its auto-covariance function, which has to be transformed to spectral form).

Step 2: Spectral characteristics of the expected values and auto-covariance function of the lateral displacement of the vehicle are determined. This can be done using the following two basic simple equations for determining auto-covariance functions $R_{XX}(t_1,t_2) = E[\{X(t_1) - E(X(t_1))\} \bullet \{X(t_2) - E(X(t_2))\}]$ and the spectral characteristics $\mathbf{C}^{m_X}$ of the expected values $m_X(t) = E[X(t)]$ [9-11] of the output signals $X(t)$ of the linear elements (or systems):

$$R_{XX}(t_1,t_2) = \mathbf{\Phi}^T(t_1)\ \mathbf{A}\mathbf{C}^{R_{YY}}\mathbf{A}^T\mathbf{\Phi}(t_2), \tag{31}$$

$$\mathbf{C}^{m_X} = \mathbf{A}\mathbf{C}^{m_Y}, \tag{32}$$

where $\mathbf{A}$ is the spectral characteristic of the linear element (or system); $\mathbf{C}^{R_{YY}}$ is a square matrix of the Fourier coefficients of the discrete decomposition of the auto-covariance function $R_{YY}(t_1,t_2)$ of the input signal $Y(t)$ using orthonormal basis functions $\mathbf{\Phi}(t)$; and $\mathbf{C}^{m_Y}$ is the spectral characteristic of the expected values $m_Y(t)$ of the input signal $Y(t)$.

Before applying equations (31) and (32) for determining auto-covariance function and the expected values of lateral displacement of the vehicle, block diagram transformations in the spectral domain (see Fig. 13 and 15) [13, 14, 16−19] and the superposition principle have to be used to simplify calculations. The equations for determining spectral characteristics of the expected values and auto-covariance function of the lateral displacement of the snowplow are as follows:

For the spectral characteristic of the expected values:

$$\mathbf{C}^{m_{Y_V}} = \mathbf{A}_{MR}\mathbf{C}^{m_{Y_R}} + \mathbf{A}_{MF_{SY}}\mathbf{C}^{m_{F_{SY}}}, \tag{33}$$

where $\mathbf{A}_{MR} = \mathbf{A}_{m_\varepsilon m_{Y_V}}\left(\mathbf{I} + \mathbf{A}_{m_\varepsilon m_{Y_V}}\right)^{-1}\mathbf{A}_{Ia}\mathbf{P}^T\mathbf{A}_{m_{\omega 4}}\left(\mathbf{I} + \mathbf{P}^T\mathbf{A}_{m_{\omega 4}}\right)^{-1}$,

$\mathbf{A}_{m_\varepsilon m_{Y_V}} = \mathbf{A}_E\left(\mathbf{A}_{m_{T_3}}\left(\mathbf{P}^T\right)^{-1} + \mathbf{I}\right)\mathbf{A}_{m_{K_y}}$,

$\mathbf{A}_E = \mathbf{A}_V\mathbf{A}_{NMP_1P_2}\mathbf{A}_L$,

$\mathbf{A}_{NMP_1P_2} = \mathbf{A}_{NM}\left(\mathbf{I} + \mathbf{A}_{P_1}\mathbf{A}_{NM}\right)^{-1}\left(\mathbf{I} + \mathbf{A}_{P_2}\mathbf{A}_{P_1}\mathbf{A}_{NM}\left(\mathbf{I} + \mathbf{A}_{P_1}\mathbf{A}_{NM}\right)^{-1}\right)^{-1}$,

$\mathbf{A}_{MF_{SY}} = \left(\mathbf{I} + \mathbf{A}_V\mathbf{A}_{NMP_1P_2}\mathbf{A}_L\left(\mathbf{A}_{m_{T_3}}\left(\mathbf{P}^T\right)^{-1} + \mathbf{I}\right)\mathbf{A}_{m_{K_y}}\right)^{-1}\mathbf{A}_{YF_{SY}}$,

$\mathbf{I}$ is the identity matrix with the dimensions equal to the number of Fourier coefficients of each signal and element of the system (128 is considered in this paper).

For the spectral characteristic of the auto-covariance function:

$$\begin{aligned}\mathbf{C}^{R_{Y_VY_V}} &= \mathbf{A}_{D_{K_y}}\mathbf{C}^{R_{K_yK_y}}\left(\mathbf{A}_{D_{K_y}}\right)^T + \mathbf{A}_{D_{T_3}}\mathbf{C}^{R_{T_3T_3}}\left(\mathbf{A}_{D_{T_3}}\right)^T + \mathbf{A}_{D_{\omega 4}}\mathbf{C}^{R_{\omega 4\omega 4}}\left(\mathbf{A}_{D_{\omega 4}}\right)^T \\ &\quad + \mathbf{A}_{D_{F_{SY}}}\mathbf{C}^{R_{F_{SY}F_{SY}}}\left(\mathbf{A}_{D_{F_{SY}}}\right)^T,\end{aligned} \tag{34}$$

where $\mathbf{A}_{D_{K_y}} = \mathbf{A}_V\mathbf{A}_{NMP_1P_2}\mathbf{A}_L\left(\mathbf{A}_{m_{T_3}}\left(\mathbf{P}^T\right)^{-1} + \mathbf{I}\right)\left(\mathbf{I} + \mathbf{A}_{m_{K_y}}\mathbf{A}_{NMP_1P_2}\mathbf{A}_L\left(\mathbf{A}_{m_{T_3}}\left(\mathbf{P}^T\right)^{-1} + \mathbf{I}\right)\right)^{-1}\mathbf{A}_{m_\varepsilon}$,

$\mathbf{A}_{D_{T_3}} = \mathbf{A}_V\mathbf{A}_{NMP_1P_2}\mathbf{A}_L\left(\mathbf{I} + \left(\mathbf{A}_{m_{T_3}}\left(\mathbf{P}^T\right)^{-1} + \mathbf{I}\right)\mathbf{A}_{m_{K_y}}\mathbf{A}_V\mathbf{A}_{NMP_1P_2}\mathbf{A}_L\right)^{-1}\mathbf{A}_{m_{\varepsilon 1}}$,

$\mathbf{A}_{D_{\omega 4}} = \mathbf{A}_V\mathbf{A}_{NMP_1P_2}\mathbf{A}_L\left(\mathbf{A}_{m_{T_3}}\left(\mathbf{P}^T\right)^{-1} + \mathbf{I}\right)\mathbf{A}_{m_{K_y}}\left(\mathbf{I} + \mathbf{A}_V\mathbf{A}_{NMP_1P_2}\mathbf{A}_L\left(\mathbf{A}_{m_{T_3}}\left(\mathbf{P}^T\right)^{-1} + \mathbf{I}\right)\mathbf{A}_{m_{K_y}}\right)^{-1}$

$\cdot\,\mathbf{P}^T\mathbf{A}_{m_{\omega 4}}\left(\mathbf{I} + \mathbf{P}^T\mathbf{A}_{m_{\omega 4}}\right)^{-1}\left(\mathbf{A}_{m_{\omega 4}}\right)^{-1}\mathbf{A}_{m_{\varepsilon 2}}$,

$\mathbf{A}_{D_{F_{SY}}} = \left(\mathbf{I} + \mathbf{A}_V\mathbf{A}_{NMP_1P_2}\mathbf{A}_L\left(\mathbf{A}_{m_{T_3}}\left(\mathbf{P}^T\right)^{-1} + \mathbf{I}\right)\mathbf{A}_{m_{K_y}}\right)^{-1}\mathbf{A}_{Y_{F_{SY}}}$.

Note that the expected values and standard deviations of parameters K_y, T_3, T_4 and other system parameters can vary with time. The spectral methods can easily deal with time-varying expected values and standard deviations of parameters and signals, and time-varying coefficients of differential equations. The beauty of the method, developed in this work is in that equations (33) and (34) will be the same whether the system parameters are constant or time-varying.

The spectral characteristic of each element of the system must be determined differently depending on whether the parameters of that element are constant or time varying. For example, if the expected value m_{K_y} of the gain $K_y(t)$ is constant, the

spectral characteristic of the element $\mathbf{A}_{m_{K_y}}$ (see Fig. 13 and 15) can be determined as follows [13]:

$$\mathbf{A}_{m_{K_y}} = m_{K_y}\mathbf{I}, \tag{35}$$

where as the spectral characteristic of this element when the expected value $m_{K_y}(t)$ of this gain is time-varying must be determined as [13]:

$$\mathbf{A}_{m_{K_y}} = \left(\tilde{\mathbf{C}}^{m_{K_y}}\right)^T, \tag{36}$$

where $\left(\tilde{\mathbf{C}}^{m_{K_y}}\right)^T$ is the so-called "spectral characteristic of the multiplier" (square matrix) [13], formed by an isomorphism of the spectral characteristic $\mathbf{C}^{m_{K_y}}$ (column vector) of the expected value $m_{K_y}(t)$ of the gain $K_y(t)$. In the next section, some examples of spectral characteristics of the elements are described by the linear time-invariant differential equations. If an element is described by the linear time-varying differential equation, a block diagram representation of this differential equation or another technique [13] can be used for determining its spectral characteristic.

Step 3: Inverse Discrete Fast Fourier Transform (IDFFT) is applied to equations (31) and (32) to find the expected values and auto-covariance function of the lateral displacement of the vehicle in the time domain.

After these three steps, both the expected values and standard deviations of the lateral displacement of the vehicle can be obtained. Note that the variance $D_{XX}(t)$ of the signal $X(t)$ can be determined from the auto-covariance function $R_{XX}(t_1,t_2)$ at $t_1 = t_2$ ($R_{XX}(t_1,t_2)\big|_{t_1=t_2} = R_{XX}(t,t)$) and the standard deviation $\sigma_{XX}(t)$ of the signal $X(t)$ can be determined as $\sqrt{D_{XX}(t)}$.

Advantages of the developed method of stochastic analysis include:

- The method allows handling nonlinear non-stationary time-varying systems and time-varying regimes of time invariant systems. This is the main difference between the method, developed here, and traditional method of statistical linearization, which was initially developed for studying only steady-state regimes of stationary systems (constant expected values and standard deviations). When applying the spectral method, developed here, not only driver parameters can be considered stochastic and having time-varying expected values and standard deviations, but also the vehicle parameters can be considered being stochastic and having time-varying expected values and standard deviations;
- The linear part of the system can be high order (hundreds) with time-dependent terms;

- A system can have several nonlinear memory-less elements. For most types of nonlinear memory-less elements, an iteration procedure [23] needs to be involved for determining the expected values and standard deviations of signals;
- The differential equations are transferred to matrix algebraic equations, solutions for which are easier to find;
- The method can be used for solving inverse problems when the output signal of the system is known and a proper input needs to be found at a given model of the system. If both the exact input and output signals are known from the experiment, the system parameters can also be identified by applying the spectral method.
- The method is good for engineers due to simplicity of its use.

Shortcomings of the developed method of stochastic analysis include:

- The solution might not always converge. This happens mostly because of two reasons: 1) sometimes the inverse matrices do not exist when one is trying to solve matrix algebraic equations. In this case, another way of solving matrix equations or changing the number of Fourier coefficients (changing matrix dimensions) might help; 2) consideration of a limited number of coefficients of the Fourier decomposition for each signal and element in the block diagrams (or differential equations). Steps described on page 7 of this paper need to be taken to improve a convergence process.
- In the absence of linear elements with filtering properties in a nonlinear system, the statistical linearization of nonlinear memory-less elements might give only rough estimation of the statistical characteristics of signals.

SIMULATION RESULTS

In order to show the effect of stochastic variation of certain driver parameters on simulation of vehicle steering dynamics, we apply the result of this work to simulation of a control system for lateral displacement of a snowplowing vehicle. Since the focus of this paper is on the driver rather than vehicle model, we use a simple two degree-of-freedom (slip speed v and yaw rate ω) bicycle model for the vehicle [20] with plowing effects modeled simply by a single force applied to the plowing blade in front of the vehicle as shown in Fig. 16. For steady-state plowing operations, we assume that the vehicle forward speed u remains constant. Vehicle system parameters are shown in Table 1. The current method will work for more complex models, e.g. [21, 22]; however, the so-called bicycle model is sufficient to show the applicability of the developed method. Equations of motion of the vehicle are described in detail in [23].

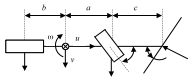

Fig. 16. Two degree-of-freedom vehicle model including plow blade

Table 1. Vehicle System Parameters

Symbol	Value	Unit	Description
a	2.06	m	Distance from C.G. to front axle
b	1.94	m	Distance from C.G. to rear axle
c	1.48	m	Distance from front axle to the plow blade
m_V	11460	kg	Vehicle mass
n_{St}	15	-	Steering system gear ratio
I_Z	14000	$kg \cdot m^2$	Yaw moment of inertia
F_{Y_1} , F_{Y_2}		N	Front and rear lateral tire forces respectively
F_S		N	Force acting on the plow from the snow
F_{SY}		N	Lateral component of the force F_S
u	11.1	m/s	Vehicle forward speed
v		m/s	Vehicle lateral speed
$C_{F\alpha_1}$	55200	N/rad	Front tire steady-state cornering stiffness
$C_{F\alpha_2}$	78995	N/rad	Rear tire steady-state cornering stiffness
α_1, α_2		rad	Front and rear tire slip angles, respectively
δ		rad	Driver steer angle
β	45	$degrees$	Plow blade angle
ψ		rad	Yaw angle
ω		rad/s	Yaw rate

For the parameters of Table 1, a differential equation describing the vehicle G_V in Figure 6 (between the steering wheel angle δ (input signal) and the lateral displacement of the vehicle) can be obtained from the state-space representation of equations of motion of the vehicle (see [23]; note that tire forces are estimated as $F_{Y_i} = 2C_{F_{\alpha_i}} \alpha_i$, i =1,2):

$$\frac{d^4}{dt^4} Y_\delta(t) + 8.95 \frac{d^3}{dt^3} Y_\delta(t) + 19.77 \frac{d^2}{dt^2} Y_\delta(t) = 0.642 \frac{d^2}{dt^2} \delta(t) + 5.07 \frac{d}{dt} \delta(t) + 28.99 \delta(t) \ . \tag{37}$$

A spectral characteristic $\mathbf{A}_V$ of the element G_V can be determined using the spectral characteristic of the integrator $\mathbf{P}$ [13, 14, 18, and 19] as:

$$\mathbf{A}_V^x = \mathbf{I} + 8.95\,\mathbf{P}^T + 19.77\,\mathbf{P}^T\mathbf{P}^T, \tag{38}$$

$$\mathbf{A}_V^y = 0.642\,\mathbf{P}^T\mathbf{P}^T + 5.07\,\mathbf{P}^T\mathbf{P}^T\mathbf{P}^T + 28.99\,\mathbf{P}^T\mathbf{P}^T\mathbf{P}^T\mathbf{P}^T, \quad \mathbf{A}_V = \left(\mathbf{A}_V^x\right)^{-1}\mathbf{A}_V^y. \tag{39}$$

A differential equation describing the element $G_{YF_{SY}}$ between the disturbance force $F_{SY}(t)$ and the lateral displacement of the vehicle can also be obtained from the state-space representation of equations of motion of the vehicle (see [23]):

$$\begin{aligned}&\frac{d^4}{dt^4}Y_{F_{SY}}(t) + 8.95\frac{d^3}{dt^3}Y_{F_{SY}}(t) + 19.77\frac{d^2}{dt^2}Y_{F_{SY}}(t) = \\ &-8.726\cdot10^{-5}\frac{d^2}{dt^2}F_{SY}(t) - 7.541\cdot10^{-4}\frac{d}{dt}F_{SY}(t) - 6.415\cdot10^{-3}F_{SY}(t).\end{aligned} \tag{40}$$

A spectral characteristic $\mathbf{A}_{YF_{SY}}$ of the element $G_{YF_{SY}}$ can be determined as follows:

$$\mathbf{A}_{YF_{SY}}^x = \mathbf{I} + 8.951\,\mathbf{P}^T + 19.77\,\mathbf{P}^T\mathbf{P}^T, \tag{41}$$

$$\mathbf{A}_{YF_{SY}}^y = -8.726\cdot10^{-5}\mathbf{P}^T\mathbf{P}^T - 7.541\cdot10^{-4}\mathbf{P}^T\mathbf{P}^T\mathbf{P}^T - 6.415\cdot10^{-3}\,\mathbf{P}^T\mathbf{P}^T\mathbf{P}^T\mathbf{P}^T, \tag{42}$$

$$\mathbf{A}_{YF_{SY}} = \left(\mathbf{A}_{YF_{SY}}^x\right)^{-1}\mathbf{A}_{YF_{SY}}^y. \tag{43}$$

Driver parameters (see equations (4)–(9)) were chosen based on recommendations in [5] as follows:

$$\begin{aligned}&k = 2,\ K_1 = 1,\ K_2 = 10,\ T_1 = 2.5s,\ \tau_0 = 0.15s,\ \xi_n = 0.707,\ \omega_n = 10\ rad/s, \\ &T_2 = 0.1s,\ \tau_p = 1.5s,\ K_y = 1,\ T_3 = 2.5s,\ T_4 = 1.4s.\end{aligned} \tag{44}$$

These parameters are considered to be expected values. Three parameters are considered to vary stochastically: K_y, T_3 and T_4. Interpretations of these parameters were described in previous section. Deviations of up to 50% around their expected values are considered for these parameters, which were estimated based on data in [11], comparing the Bode plots of the open-loop systems with different driver parameters. Note that the Bode plots have been used for parameter estimation only. The Laplace transformation has not been used for estimation of expected values and standard deviations of signals. Spectral Methods have been used for this purpose. This is because the Spectral Methods can be applied for statistical analysis of both time-varying systems and time-varying regimes of time-invariant systems.

Deviations of parameters K_y, T_3 and T_4 are described by the following approximations of auto-covariance functions:

$$R_{K_yK_y}(t_1,t_2) = D_{K_yK_y}\,e^{-\gamma_1|t_1-t_2|}, \tag{45}$$

$$R_{T_3T_3}(t_1,t_2) = D_{T_3T_3}\, e^{-\gamma_2|t_1-t_2|}, \tag{46}$$

$$R_{\omega_4\omega_4}(t_1,t_2) = D_{\omega_4\omega_4}\, e^{-\gamma_3|t_1-t_2|}, \tag{47}$$

where $D_{K_yK_y} = 2.78\cdot 10^{-2}$, $D_{T_3T_3} = 16s^2$, $\omega_4 = 1/(T_4)$, $D_{\omega_4\omega_4} = 0.0142\ s^{-2}$, $\gamma_1 = \gamma_2 = \gamma_3 = 0.1\ s^{-1}$.

We are interested in the dynamics of the snowplow during a lane change maneuver (see Figures 12−15). This maneuver can be described by the following expression (displacement is in meters, time is in seconds)

$$m_{Y_R}(t) = Y_R(t) = \begin{cases} 0, & \text{for } 0 \le t < 2.36, \\ 2\sin\left((t-2.36)/1.8\right), & \text{for } 2.36 \le t \le 5.35, \\ 2, & \text{for } t > 5.35, \end{cases} \tag{48}$$

As we mentioned above, plowing force changes stochastically due to plow encountering ice or snow packs on the road and non-ideal road surface conditions. Changes of expected values of the plowing force (in Newton's) are approximated by

$$m_{F_{SY}}(t) = \begin{cases} 400, & \text{for } 0 \le t < 2.36, \\ 400 + 200\sin\left((t-2.36)/0.91\right), & \text{for } 2.36 \le t \le 5.35, \\ 400, & \text{for } t > 5.35, \end{cases} \tag{49}$$

The expected values of the plowing force before the lane change maneuver are considered to be constant and equal to the expected values of the plowing force after the maneuver, which makes sense. Changes of the expected values of the plowing force during the lane change maneuver are approximated by a half sine wave with a maximum of 600 N in the middle of the maneuver. If experimental data on the expected values of the plowing force become available, then the sine wave can easily be replaced by any experimentally determined time-varying function.

Auto-covariance function of the plowing force is approximated as

$$R_{F_{SY}F_{SY}}(t_1,t_2) = D_{F_{SY}F_{SY}}\, e^{-\gamma_4|t_1-t_2|}, \tag{50}$$

where $D_{F_{SY}F_{SY}} = 5777\ N^2$, and $\gamma_4 = 0.2\ s^{-1}$.

Fig. 17 shows the expected values (dotted line), upper (dashed line) and lower (dash-dotted line) bounds of all possible deviations of lateral displacement of the snowplow around the expected values (with the probability of 99.7%) when having only one stochastic gain K_y and the plow blade off the ground. The expected values correspond to normal driving conditions. As we can see from Fig. 17, having just one stochastic parameter in the system makes significant changes of up to 15% to the lateral displacement of the vehicle. Since the gain K_y is related to driver's performance, the

stochastic modeling allows simulation of driver's steering control under a range of drivers.

Fig. 18 shows the expected values (dotted line), upper (dashed line) and lower (dash-dotted line) bounds of all possible deviations of lateral displacement of the snowplow around the expected values having two stochastic parameters: gain K_y and time constant T_3 (the plow blade is off the ground). As we can see, having two stochastic parameters in the system increases deviations of the lateral displacement of the vehicle to more than 25%. Again, using stochastic modeling for these two parameters allows simulating steering dynamics for a range of drivers' performance and visual cues.

Fig. 19 shows the expected values (dotted line), upper (dashed line) and lower (dash-dotted line) bounds of all possible deviations of lateral displacement of the snowplow around the expected values having three stochastic parameters: gain K_y, time constants T_3 and T_4 (the plow blade is off the ground). Having three stochastic parameters in the system increases deviations of the lateral displacement of the vehicle to more than 30%. The addition of third stochastic parameter adds variations mostly due to differences in driver's perception time.

Fig. 20 shows the results for the lateral displacement of the snowplow with three stochastic parameters K_y, T_3, T_4, and the plow blade on the ground. As we can see, there is a steady-state error and more deviations at steady-state due to effect of a stochastic plowing force. As we can see from Fig. 17−20, the gain K_y has the most effect (up to 15%) on changes of the lateral displacement of the vehicle. This means that deviation in driver's performance in terms of experience and fatigue can play a more significant role as compared to small deviations related to driver's preview or sensory perception and visual cues in controlling a vehicle in snow removal operations. One conclusion is that training and proper scheduling that would minimize degradation in driver's performance can play a significant role in safe road maintenance operations.

Figures 21 and 22 show the results for the steering wheel angle with three stochastic parameters K_y, T_3, and T_4, and the plow off and on the ground respectively. As we can see, at 50% stochastic change of all three parameters, the steering wheel angle deviates for up to 80% during the maneuver, which is significant. Obviously, as stochastic variations in all the parameters are super-imposed, deviations of other parameters will lead to more deviations of the lateral displacement of the vehicle and actual lateral displacement could be far from the desired one so that the vehicle would go off the travel lane to an opposite lane, or hit the highway barrier in the shoulder. If the reference signal is also stochastic, it can easily be taken into consideration. This will also increase deviations of the lateral displacement of the vehicle.

It should be noted that in all the computations a closed set of Walsh functions [24] was used. For each signal, we used 128 expansion terms (coefficients of the Fourier decomposition) for the calculations. This was done in MATLAB®. Stochastic parameters are considered to have normal distributions. Calculation time was less than 2 seconds on a Pentium 4 computer.

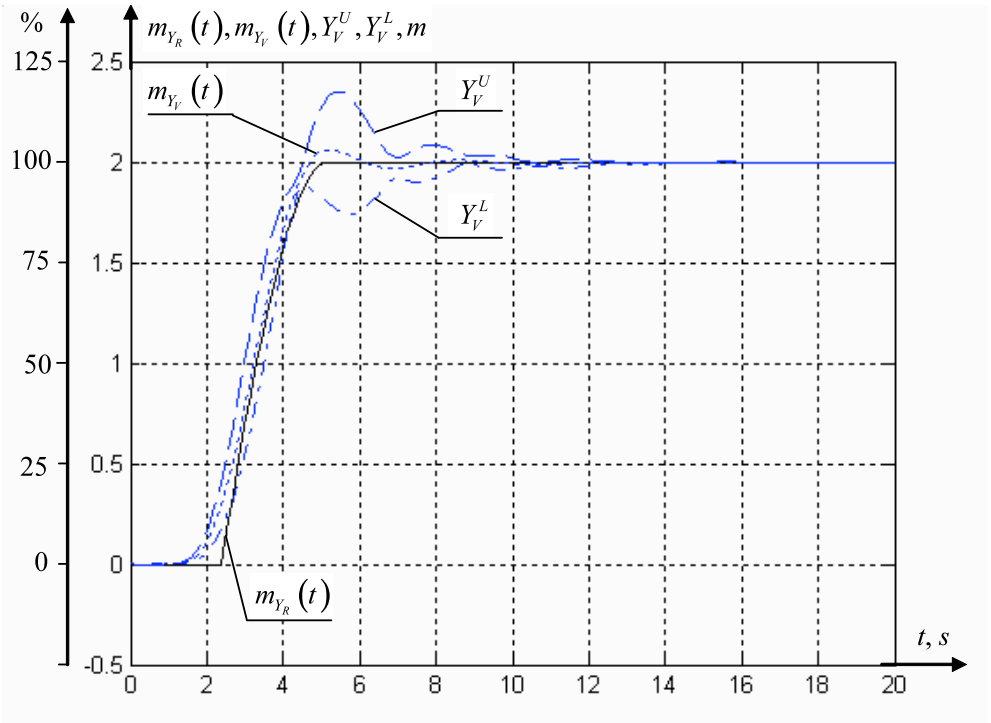

Fig. 17. Lane change maneuver with only one stochastic gain K_y (plow blade off the ground)

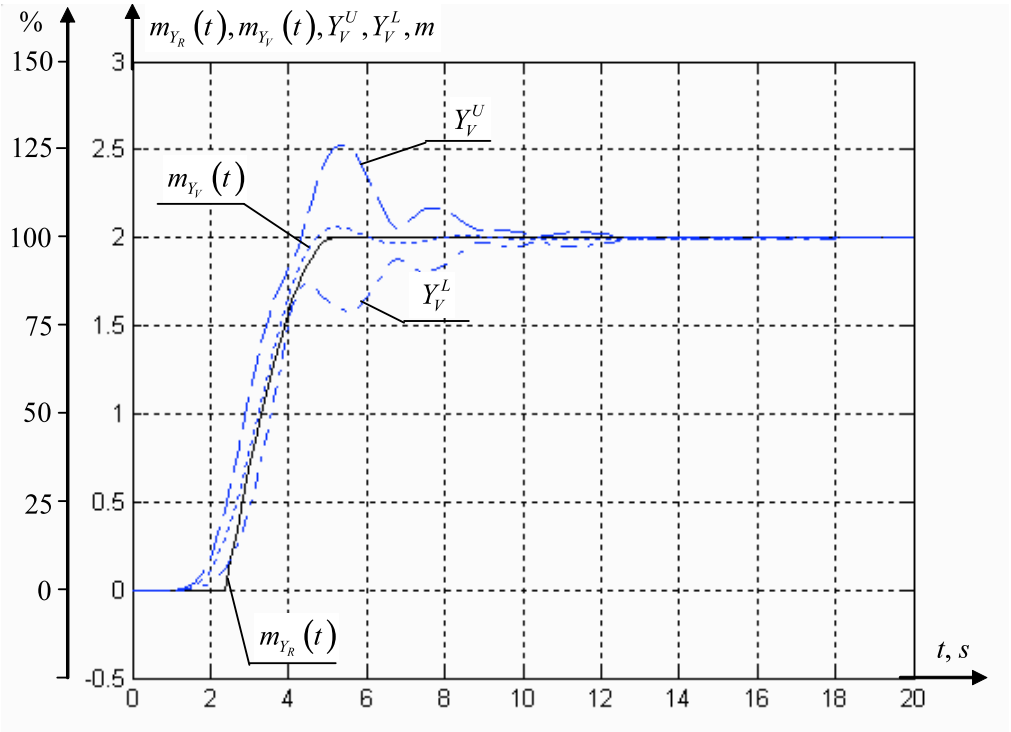

Fig. 18. Lane change maneuver with two stochastic parameters: K_y and T_3 (plow blade off the ground)

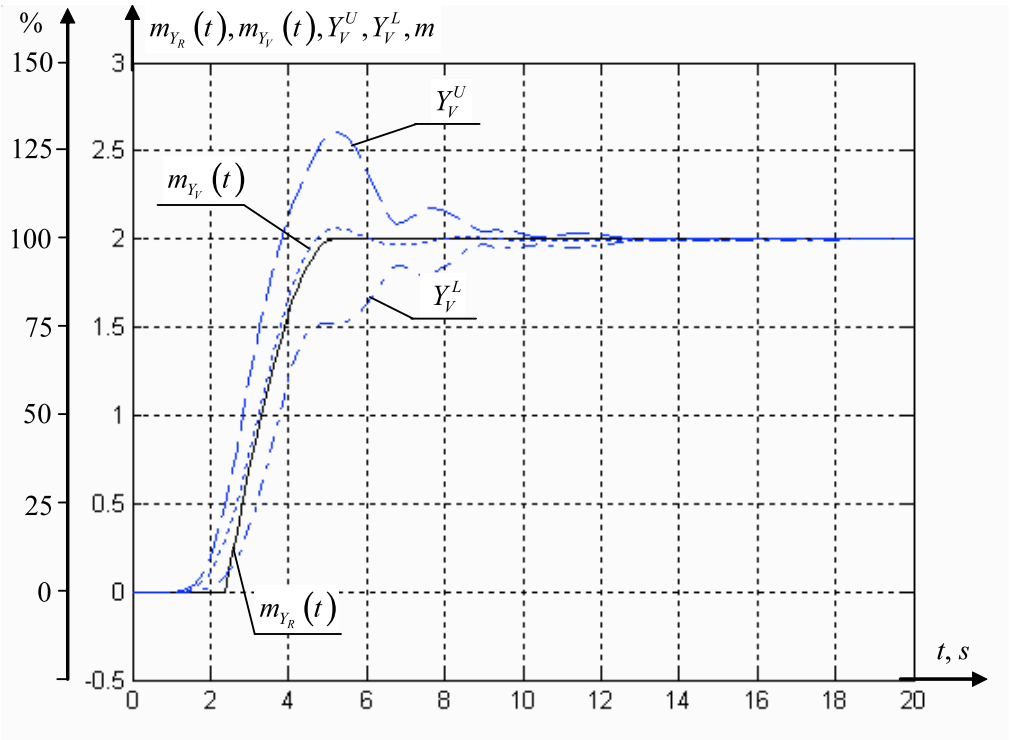

Fig. 19. Lane change maneuver with three stochastic parameters: K_y, T_3, and T_4 (plow blade off the ground)

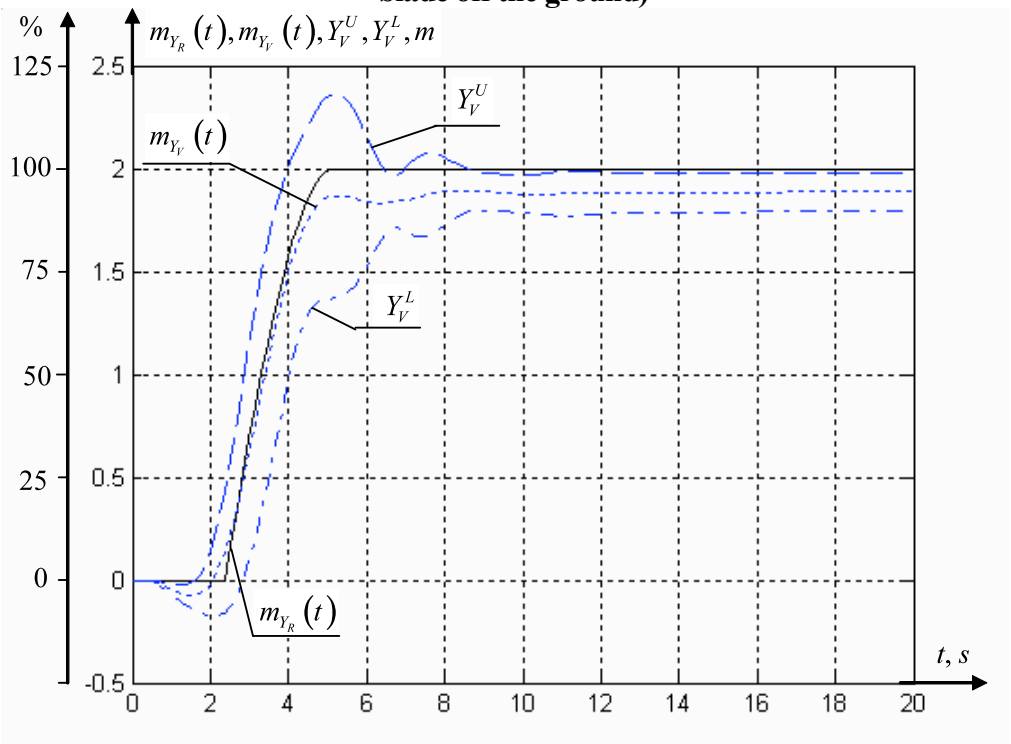

Fig. 20. Lane change maneuver with stochastic parameters K_y, T_3, and T_4 (plow blade on the ground)

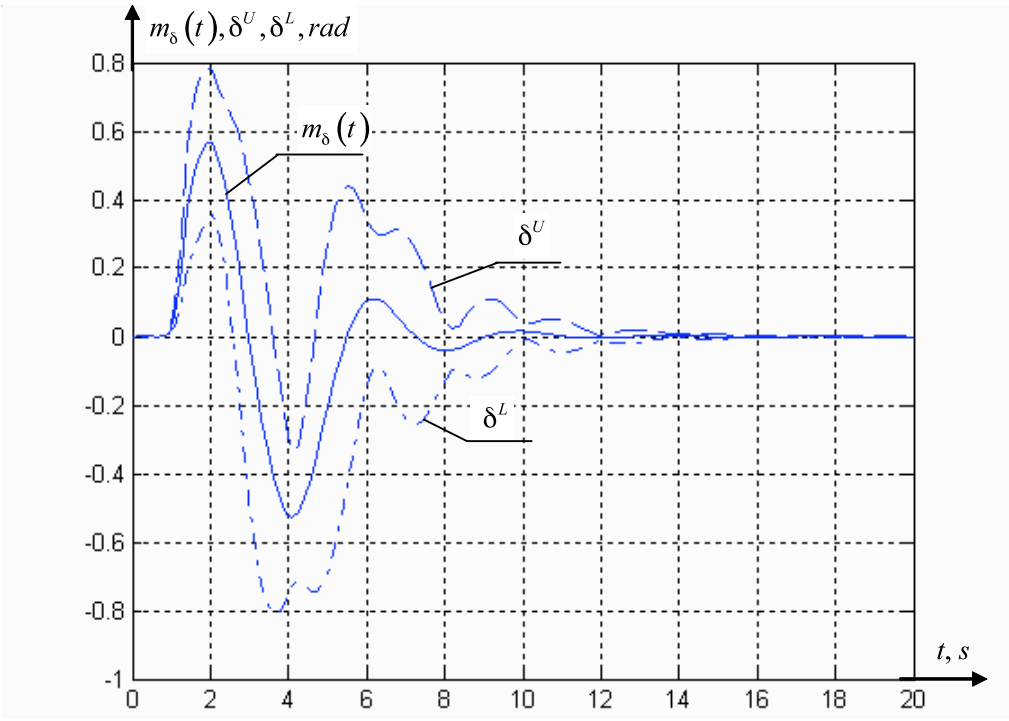

Fig. 21. Steering wheel angle: expected values $m_\delta(t)$; upper boundary δ^U; lower boundary δ^L (plow blade off the ground)

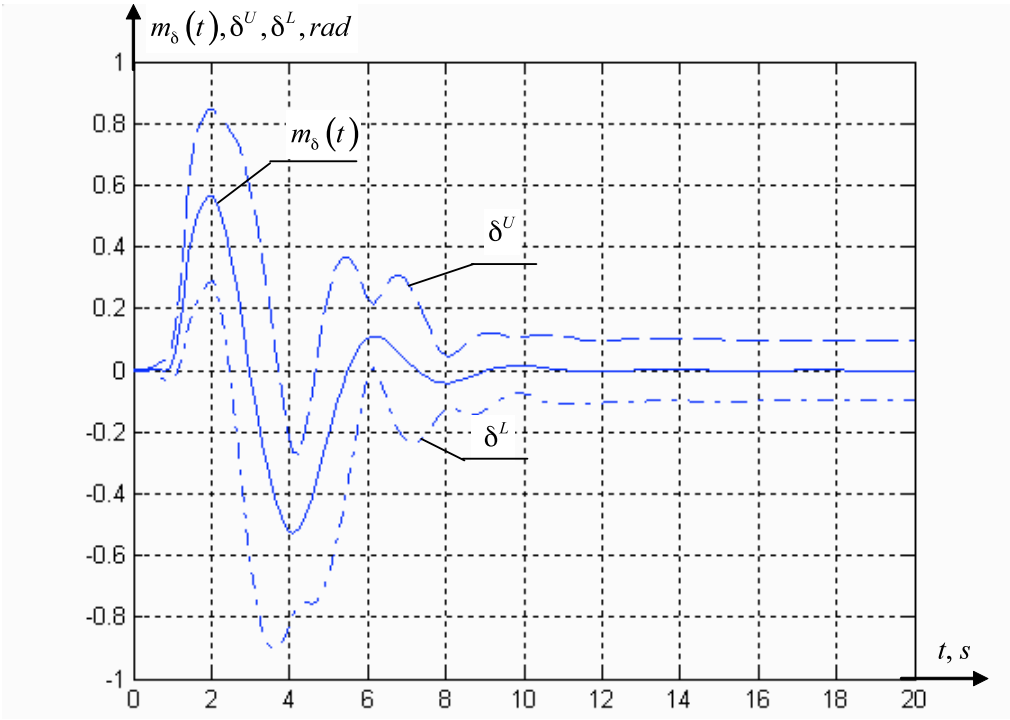

Fig. 22. Steering wheel angle : expected values $m_\delta(t)$; upper boundary δ^U; lower boundary δ^L (plow blade on the ground)

Note that the expected values and/or standard deviations of stochastic parameters might also be changing with time. The method, developed here, can easily deal with any time-varying expected values and standard deviations of system parameters. In other words, it can handle time-varying differential equations. This can be demonstrated by a simple example. Suppose that the expected values and variances of the key parameters (K_y, T_3, and T_4) change during the steering maneuver as follows:

$$m_{K_Y}(t) = \begin{cases} 1, & \text{for } 0 \le t < 2.36, \\ 1 - 0.4\sin\left((t-2.36)/0.91\right), & \text{for } 2.36 \le t \le 5.35, \\ 1, & \text{for } t > 5.35, \end{cases} \tag{51}$$

$$m_{T_3}(t) = \begin{cases} 2.5, & \text{for } 0 \le t < 2.36, \\ 2.5 - \sin\left((t-2.36)/0.91\right), & \text{for } 2.36 \le t \le 5.35, \\ 2.5, & \text{for } t > 5.35, \end{cases} \tag{52}$$

$$m_{\omega_4}(t) = \begin{cases} 0.714, & \text{for } 0 \le t < 2.36, \\ 0.714 - 0.56\sin\left((t-2.36)/0.91\right), & \text{for } 2.36 \le t \le 5.35, \\ 0.714, & \text{for } t > 5.35, \end{cases} \tag{53}$$

$$D_{K_Y K_Y}(t) = \begin{cases} 2.78\cdot 10^{-2}, & \text{for } 0 \le t < 2.36, \\ 2.78\cdot 10^{-2} + 1.11\cdot 10^{-2}\sin\left((t-2.36)/0.91\right), & \text{for } 2.36 \le t \le 5.35, \\ 2.78\cdot 10^{-2}, & \text{for } t > 5.35, \end{cases} \tag{54}$$

$$D_{T_3 T_3}(t) = \begin{cases} 16, & \text{for } 0 \le t < 2.36, \\ 16 + 6.4\sin\left((t-2.36)/0.91\right), & \text{for } 2.36 \le t \le 5.35, \\ 16, & \text{for } t > 5.35, \end{cases} \tag{55}$$

$$D_{\omega_4 \omega_4}(t) = \begin{cases} 0.0142, & \text{for } 0 \le t < 2.36, \\ 0.0142 + 5.67\cdot 10^{-3}\sin\left((t-2.36)/0.91\right), & \text{for } 2.36 \le t \le 5.35, \\ 0.0142, & \text{for } t > 5.35, \end{cases} \tag{56}$$

The expected values and variances of the stochastic parameters before the lane change maneuver are considered to be constant and equal to the expected values of the stochastic parameters after the maneuver. Changes of the expected values of these parameters during the lane change maneuver are approximated by half sine waves with a minimum of 40% change in the middle of the maneuver. If it is determined experimentally that the expected values and variances of these parameters behave differently, the approximations above can easily be replaced by any experimentally determined time-varying functions.

Fig. 23 shows the lane change maneuver with the time-varying expected values of the stochastic parameters K_y, T_3, and T_4, time-varying variances of all three parameters, and the plow blade off the ground. Comparison with the results in Fig. 19 shows that the expected values of the lateral displacement of the vehicle significantly changes during the maneuver. Deviations of the lateral displacement of the vehicle also become more significant during the maneuver, increasing to more than 50% in comparison with the desired lateral displacement of the vehicle. Fig. 24 shows the lane change maneuver with the plow blade on the ground. Comparison with the results in Fig. 20 shows significant changes of the lateral displacement of the vehicle during the maneuver.

Figures 25 and 26 show the results for the steering wheel angle with three stochastic parameters K_y, T_3, and T_4, and the plow blade off and on the ground, respectively. As we can see, at 50% stochastic change of all three parameters, the steering wheel angle deviates for more than 100% during the maneuver, which is more significant than in case of constant expected values of these parameters.

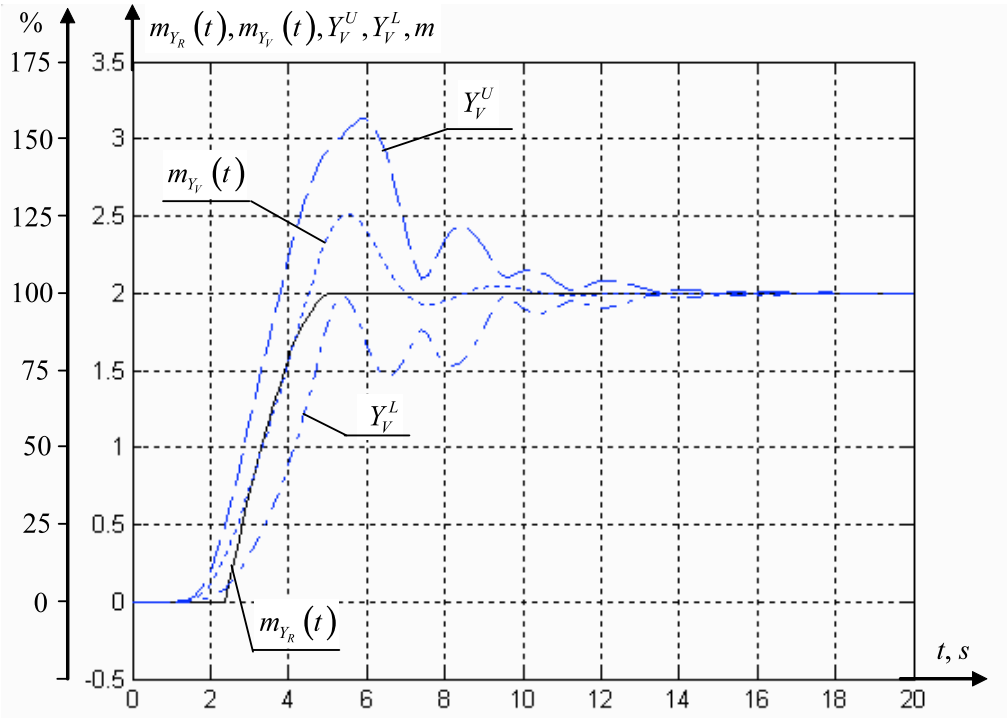

Fig. 23. Lane change maneuver with stochastic parameters K_y, T_3, and T_4 (plow blade off the ground), time-varying expected values and variances of these parameters

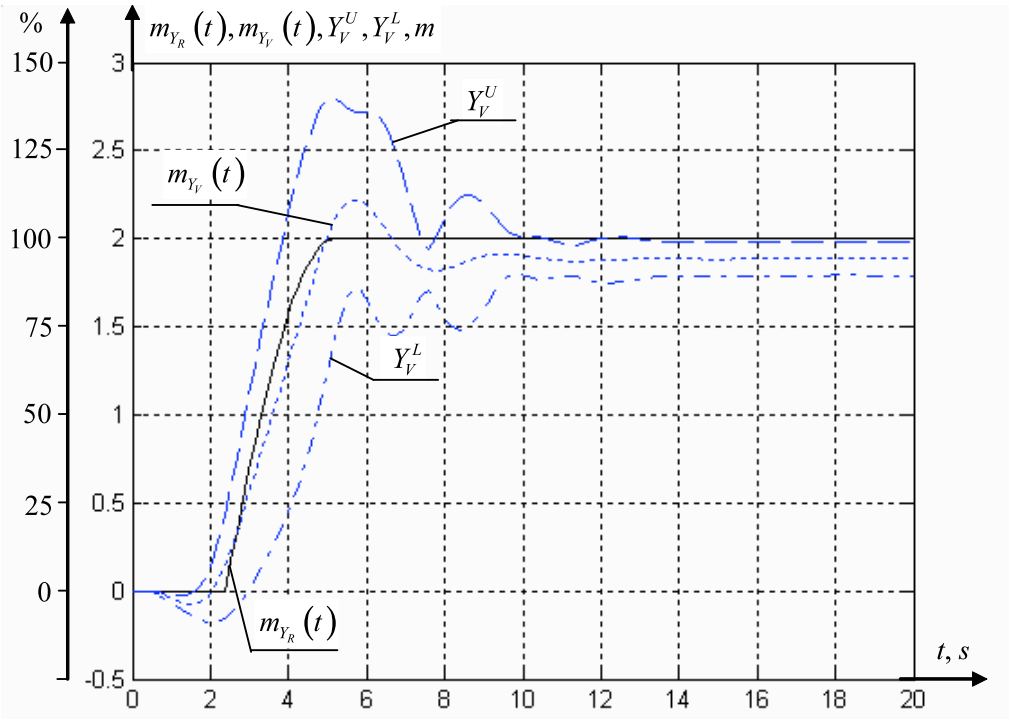

Fig. 24. Lane change maneuver with stochastic parameters K_y, T_3, and T_4 (plow blade on the ground) and time-varying expected values and variances of these parameters

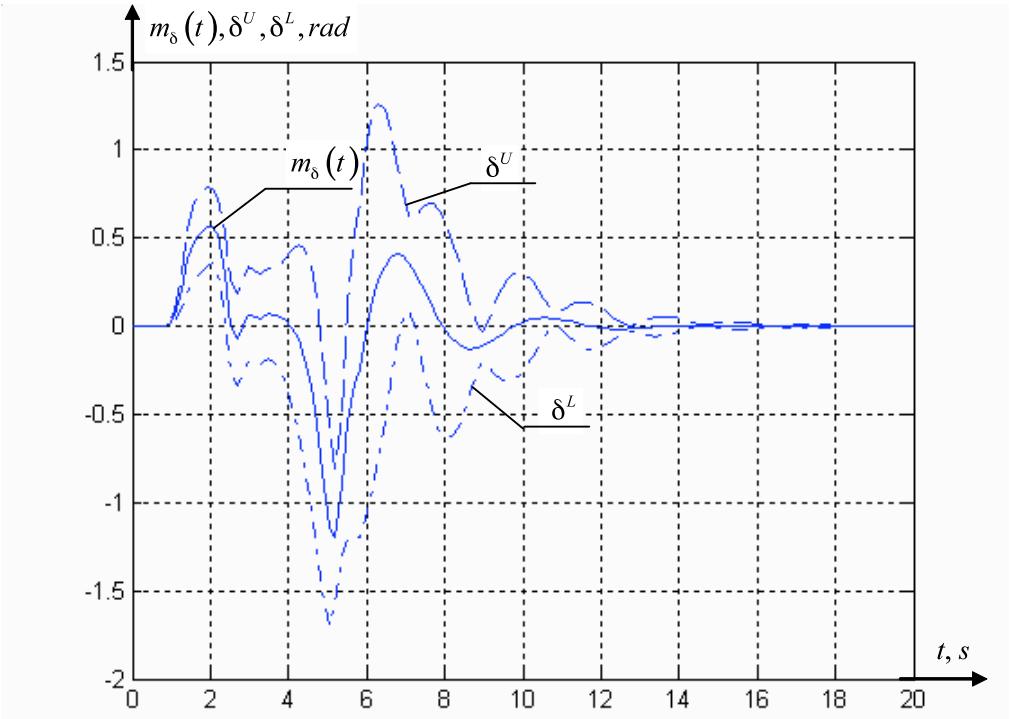

Fig. 25. Steering wheel angle : expected values $m_\delta(t)$; upper boundary δ^U; lower boundary δ^L (plow blade off the ground)

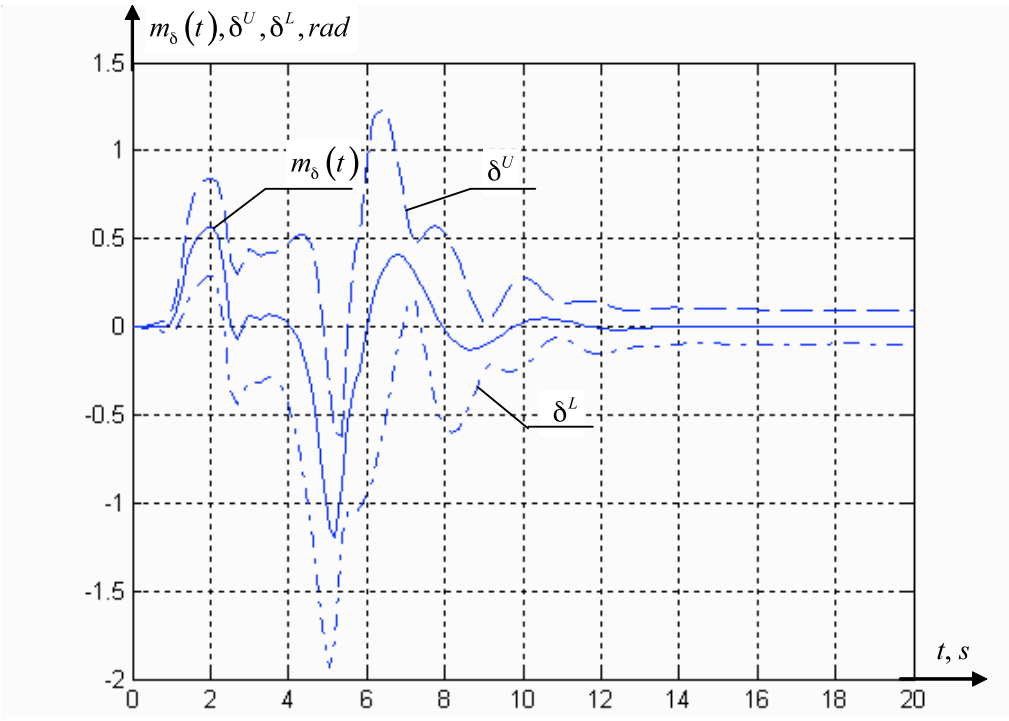

Fig. 26. Steering wheel angle: expected values $m_\delta(t)$; upper boundary δ^U; lower boundary δ^L (plow blade on the ground)

COMPARISON WITH EXPERIMENTAL DATA

The stochastic driver model developed in this paper reduces to the well-known H-M [5] model if there are no parameter variations. The H-M model [5] has been validated with average experimental data for a compact car [5]. However, the H-M model gives only the expected (average) values of the lateral position of the vehicle. Here, in addition to estimation of the expected values of the lateral position of the vehicle, we will show the comparison of all possible values of the lateral position of the vehicle, obtained from the stochastic model, developed here, with the experimental data of Reid and Solowka [26] for obstacle avoidance maneuver of a truck.

Reid and Solowka [26] describe their experiment on obstacle avoidance maneuver as follows: "The obstacle avoidance maneuver required the drivers to steer around an obstruction, which could suddenly appear from the road edge. The obstacle consisted of a falling pole, which when down (it took 1 *s* to fall) obstructed 80% of the roadway lane along which the vehicle was being driven. The vehicle's forward speed was maintained at 50 km/h by a cruise control. The drivers were instructed to steer around any obstacle which might appear, by pulling into the left-hand lane and then returning to the right-hand lane as quickly as possible. Because any one of 27 poles could be selected at random by the experimenter to fall on any run, the driver was never certain when the event would occur. In addition, 1/3 of all runs down the course contained no event thus further increasing the uncertainty". Four drivers participated in this study. A total of 32 runs were performed by drivers on a truck. Parameters of the truck used are shown in Table 2.

Table 2. Vehicle System Parameters

Symbol	Value	Unit	Description
a	3.48	m	Distance from C.G. to front axle
b	1.35	m	Distance from C.G. to rear axle
m_V	20500	kg	Vehicle mass
n_{St}	36	-	Steering system gear ratio
I_Z	71790	$kg \cdot m^2$	Yaw moment of inertia
u	13.8	m/s	Vehicle forward speed

The bicycle vehicle model was used here for simulations. The differential equation, describing the vehicle G_V is as follows:

$$\frac{d^4}{dt^4} Y_V(t) + 12.86 \frac{d^3}{dt^3} Y_V(t) + 23.92 \frac{d^2}{dt^2} Y_V(t) = 0.748 \frac{d^2}{dt^2} \delta(t) + 3.864 \frac{d}{dt} \delta(t) + 39.75 \delta(t). \quad (57)$$

A spectral characteristic $\mathbf{A}_V$ of the vehicle, corresponding to this differential equation is:

$$\mathbf{A}_V^x = \mathbf{I} + 12.86\, \mathbf{P}^T + 23.92\, \mathbf{P}^T \mathbf{P}^T, \quad (58)$$

$$\mathbf{A}_V^y = 0.748\, \mathbf{P}^T \mathbf{P}^T + 3.864\, \mathbf{P}^T \mathbf{P}^T \mathbf{P}^T + 39.75\, \mathbf{P}^T \mathbf{P}^T \mathbf{P}^T \mathbf{P}^T, \quad \mathbf{A}_V = \left(\mathbf{A}_V^x\right)^{-1} \mathbf{A}_V^y. \quad (59)$$

Parameters (44) are considered to be expected values for the driver model. Only time advanced factor is chosen to be different and equal to 0.85 *s*. Three parameters are considered to vary stochastically: K_y, T_3 and T_4. Deviations of up to 50% around their expected values are considered for these parameters with approximations of auto-covariance functions (45)−(47). The expected values $m_{Y_{VEXP}}(t)$ of the experimental data have been used as a desired input signal for the model. Initial lateral displacement of the vehicle at each run of the experiment was not equal to zero and was different at each run; therefore, the desired input signal was modeled as having the expected values and some standard deviation around the expected values. The following auto-covariance function was used to account for different initial lateral displacements of the vehicle at the beginning of each run of the experiment

$$R_{Y_R Y_R}(t_1, t_2) = D_{Y_R Y_R}\, e^{-\gamma_5 |t_1 - t_2|}, \quad (60)$$

where $D_{Y_R Y_R} = 4.23 \cdot 10^{-3} m^2$, $\gamma_5 = 5 s^{-1}$.

The experimental data are shown in Figure 27. The solid line represents average (expected) values $m_{Y_{VEXP}}(t)$ of the lateral displacement of the vehicle. The dashed line represents the upper boundary of the lateral displacement of the vehicle $Y_{V_{EXP}}^U$ (average value plus standard deviation). The dash-dotted line represents the lower boundary of the lateral displacement of the vehicle $Y_{V_{EXP}}^L$ (average value minus standard deviation).

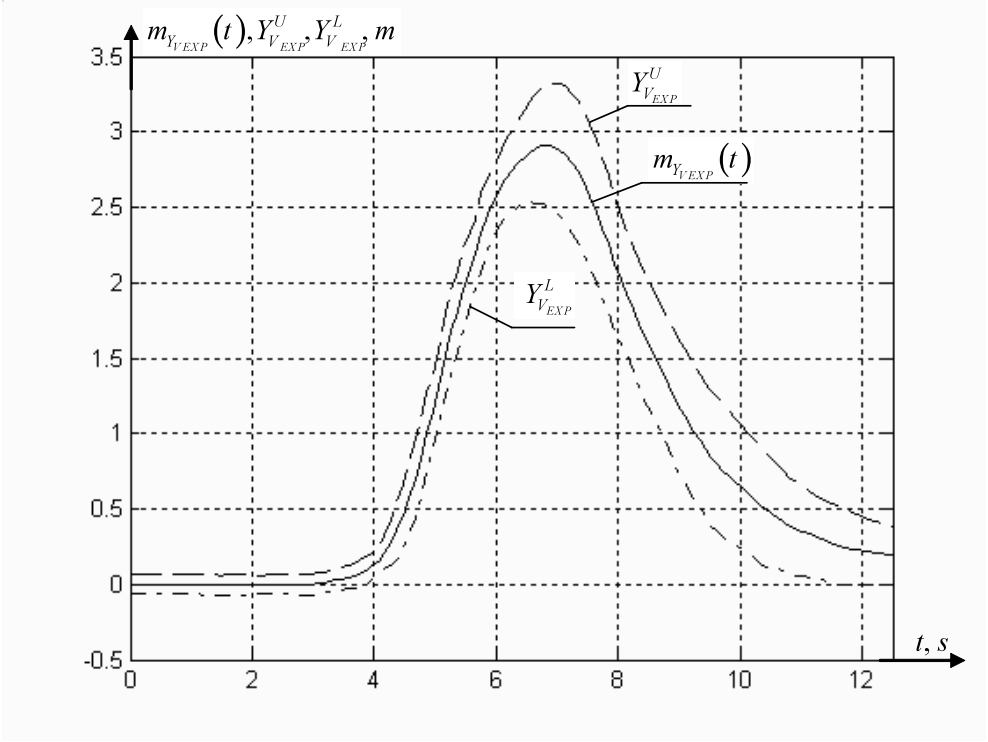

Fig. 27. Experimental data on the lateral displacement of the vehicle: $m_{Y_{VEXP}}(t)$ - expected values; $Y^U_{V_{EXP}}$ - upper boundary; $Y^L_{V_{EXP}}$ - lower boundary

The results obtained from the model are shown in Figure 28. The solid line represents average (expected) values $m_{Y_{VMOD}}(t)$ of the lateral displacement of the vehicle. The dashed line represents the upper boundary of the lateral displacement of the vehicle $Y^U_{V_{MOD}}$ (average value plus standard deviation). The dash-dotted line represents the lower boundary of the lateral displacement of the vehicle $Y^L_{V_{MOD}}$ (average value minus standard deviation).

Fig. 29 shows both the experimental data and the results obtained from the model. As we can see from Fig. 29, the stochastic model developed here gives a good estimate of the expected values and standard deviations of the lateral displacement of the vehicle. Maximum error for the expected values of the lateral displacement of the vehicle is about 5%. Maximum error for the upper boundaries of the lateral displacement of the vehicle is also about 5%. Maximum error for the lower boundaries of the lateral displacement of the vehicle is about 10%.

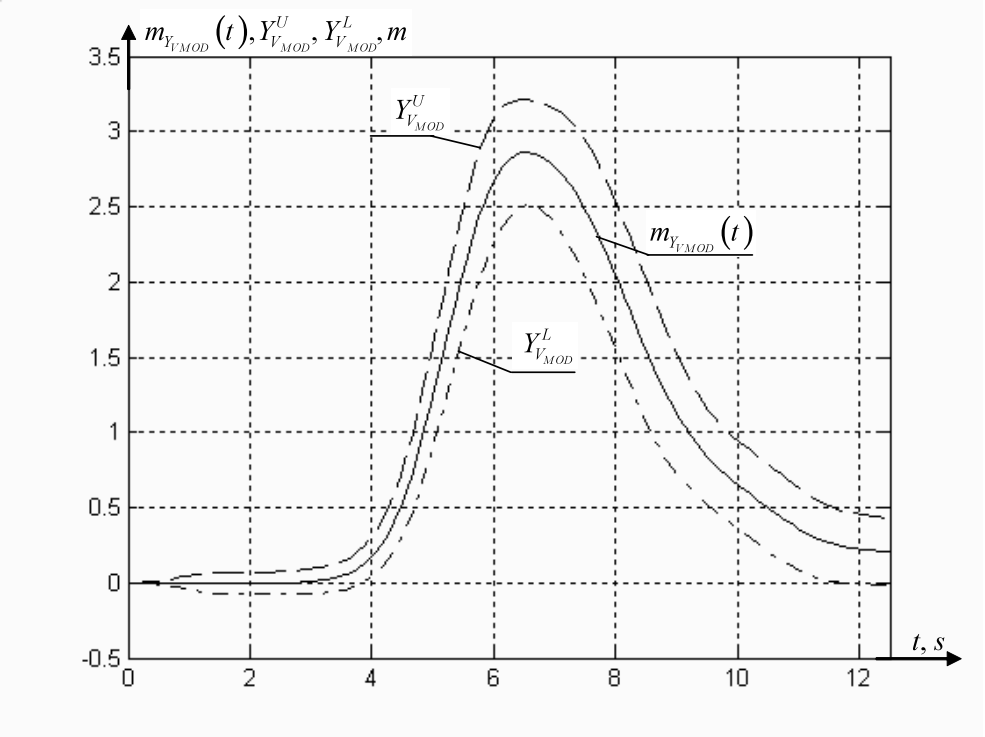

Fig. 28. Model results on the lateral displacement of the vehicle: $m_{Y_{VMOD}}(t)$ - expected values; $Y^U_{V_{MOD}}$ - upper boundary; $Y^L_{V_{MOD}}$ - lower boundary

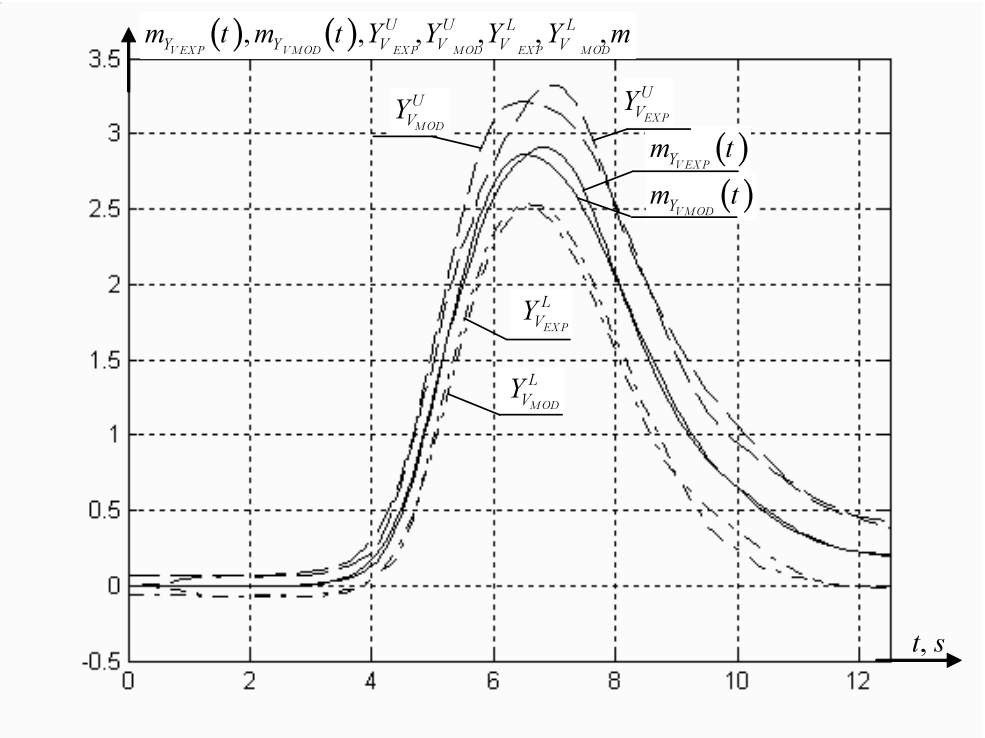

Fig. 29. Model results and the experimental data

CONCLUSIONS

A new stochastic form of a human driver model for vehicle simulation has been introduced in this paper. This model takes into account fluctuations of drivers' parameters around their average (expected) values due to factors such as their training, experience, perception, fatigue, and visual cues. This model through one simulation of steering maneuvers allows prediction of steering dynamics for a spectrum of drivers and road conditions based on mean and standard deviations.

The stochastic modeling is performed on an existing human driver model in the literature. However, the methodology and the approach of this paper can be used with any of the existing driver models to model stochastic variations of parameters. Moreover, any compensator (driver in our case) or plant (vehicle in our case) in any control system may be represented in a stochastic form given that there is information about parameter fluctuations. The new method of stochastic analysis, used in this work is based on the Fourier discrete decomposition of time-varying signals and systems using an orthonormal set of Walsh functions. It combines statistical linearization with spectral methods, facilitating handling of time-varying systems and transient regimes of linear time invariant (LTI) systems. Fourier discrete decomposition allows transforming differential equations to matrix forms, for which solutions can easily be obtained. The current model includes nonlinear elements (product nonlinearities). The method for estimation of statistical characteristics of the system can easily be applied to systems with other types of nonlinear memory-less elements. The linear part of the system can be of high orders (hundreds) with time-dependent terms. For specialized lower speed applications, such as simulating snow removal operations, it is shown that stochastic variations of parameters can produce significant lateral displacements. Such deviations can lead to lane departures and should be properly modeled in developing training and operational procedures for the drivers as well as in designing driver steering assistance systems for such highly specialized applications.

ACKNOWLEDGMENTS

This work was supported by the California Department of Transportation (Caltrans) New Technology and Research Program, through the Advanced Highway Maintenance and Construction Technology (AHMCT) Research Center at the University of California-Davis under contract IA65A0068.

REFERENCES

1. D. McRuer and D. Weir, "Theory of manual vehicular control", *Ergonomics*, 1969, V. 12, pp. 599-633.

2. D. McRuer, R. Allen, D. Weir, and R. Klein, "New Results in Driver Steering Control Models", *Human Factors*, 1977, V. 19, No. 4, pp. 381-397.

3. A. Modjtahedzadeh and R. Hess, "A Model of Driver Steering Dynamics for Use in Assessing Vehicle Handling Qualities", *Advanced automotive technologies*: *American Society of Mechanical Engineers*, New York, N.Y., 1991, pp. 41-56.

4. R. Hess and A. Modjtahedzadeh, "A Control Theoretic Model of Driver Steering Behavior", *IEEE Control Systems Magazine*, August 1990, pp. 3-8.

5. A. Modjtahedzadeh, R.A. Hess, "A Model of Driver Steering Control Behavior for Use in Assessing Vehicle Handling Qualities", *Journal of Dynamic Systems, Measurement, and Control*, 1993, V. 115, pp. 456-464.

6. R. Hess, "Model for Human Use of Motion Cues in Vehicular Control", *Journal of Guidance*, V. 13, 1990, pp. 476-482.

7. C. MacAdam, "Application of an Optimal Preview Control for Simulation of Closed-loop Automobile Driving", *IEEE Trans. Syst., Man., Cybern.,* 1981, V. SMC-11, pp. 393-399.

8. W. Levison, "A Model for Mental Workload in Tasks Requiring Continuous Information Processing", in *Mental Workload: Its Theory and Measurement*, N. Moray, Ed. New York: Plenum, 1979.

9. C. Thorpe, M. Hebert, T. Kanade, and C. Shafer, "The New Generation System for the CMU Navlab", in *Vision-Based Vehicle Guidance*, Berlin, Germany: Springer-Verlag, 1992.

10. Liang-Kuang Chen and A. Galip Ulsoy, "Identification of a Driver Steering Model, and Model Uncertainty, From Driving Simulator Data", *Journal of Dynamic Systems, Measurement, and Control*, 2001, V. 123, pp. 623-629.

11. Liang-Kuang Chen and A. Galip Ulsoy, "Design of a Vehicle Steering Assist Controller Using Driver Model Uncertainty", *International Journal of Vehicle Autonomous Systems*, 2002, V. 1, No. 1, pp. 111-132.

12. H. Ukawa, H. Idonuma, and T. Fujimura, "A Study on the Autonomous Driving System of Heavy Duty Vehicle", *International Journal of Vehicle Autonomous Systems*, 2002, V. 1, No. 1, pp. 45-62.

13. K.A. Pupkov, N.D. Egupov, and A.I. Trofimov, G.G. Sebryakov, S.I. Nikolayenko, A.K. Karyshev, M.O. Gabibulayev, M.Y. Adkin, "Statistical Methods of Analysis, Synthesis and Identification of Nonlinear Automatic Control Systems", Moscow, Russia: MSTU Publishers, 1998, 564 p.

14. K.A. Pupkov, A.I. Barkin, E.M. Voronov, N.D. Egupov, V.G. Konkov, V.N. Pilishkin, V.I. Sivtsov, A.I. Trofimov, N.V Faldin, Y.P. Kornushin, M.O. Gabibulayev, L.T. Milov, S.V. Lapin, D.V. Melnikov, D.A. Akimenko, A.M. Makarenkov, A.K. Karyshev, S.I. Nikolayenko, Y.V. Slekenichs, V.I. Krasnoshechenko, "Analysis and Statistical Dynamics of Automatic Control Systems", Moscow, Russia: MSTU Publishers, V. 1, 2000, 747p.

15. Josef L. Zeman, "Approximate Analysis of Stochastic Processes in Mechanics", Udine, Italy: Wien-New York, 1972, 86 p.

16. Y.M. Astapov, B.C. Medvedev, "Statistical Theory for Automatic Control Systems", Moscow, Russia: Nauka Publishers, 1982, 304 p.

17. A.N. Dmitriyev, N.D. Egupov, A.M. Shestopalov, and Y.G. Moiseyev, "Computer-oriented methods for computations and design of telecommunication and control systems", Moscow, Russia: Radio i sviaz Publishers, 1990, 272 p.

18. A.I. Trofimov, N.D. Egupov, A.N. Dmitriyev, "Computer-oriented Theoretical Methods for Automatic Control Systems", Moscow, Russia: Energoatomizdat Publishers, 1997, 653 p.

19. K.A. Pupkov, N.D. Egupov, V.G Konkov, L.T. Milov, A.I. Trofimov,S.V. Lapin, A.K. Karyshev, M.O. Gabibulayev, Z.G. Shirokova, A.N. Burlakin, K.I. Zhelnov, D.B. Birulin, "Methods of Analysis, Synthesis and Optimization for Non-Stationary Automatic Control Systems", Moscow, Russia: MSTU Publishers, 1999, 683 p.

20. P.J.TH Venhovens and K. Naab, "Vehicle Dynamics Estimation Using Kalman Filters", *Vehicle System Dynamics*, 1999, 32, pp. 171-184.

21. D. Karnopp and B. Jang, "Simulation of Vehicle and Power Steering Dynamics Using Tire Model Parameters Matched to whole Vehicle Experimental Results", *Vehicle System Dynamics*, 2000, 33, pp. 121-133.

22. D. Margolis and T. Shim, "Using μ Feedforward for Vehicle Stability Enhancement", *Vehicle System Dynamics*, 2001, 35, pp. 103-119.

23. M. Gabibulayev, B. Ravani, and Ty A. Lasky, "Stochastic Modeling for Lateral Control in Snowplow Operations", *9th World Congress on Intelligent Transport Systems*, Chicago, IL, October 2002.

24. J. L. Walsh, "A Closed Set of Normal Orthogonal Functions," *American Journal of Mathematics*, 1923, 45, pp. 5-24.

25. H. S. Tan, J. Guldner, S. Patwardhan, C. Chen, and B. Bougler, "Development of an Automated Steering Vehicle Based on Roadway Magnets – A Case Study of Mechatronic System Design", *IEEE/ASME Transactions on Mechatronics*, Vol. 4, No. 3, September 1999.

26. L.D. Reid, E.N. Solowka, "A systematic Study of Driver Steering Behaviour" *Ergonomics*, 1981, Vol. 24, No. 6, 447-462.