A Test Bench System for Remote Manipulation Using Standard Technologies

Daisei Fukazawa*, Alberto Palacios Pawlovsky**

Toin University of Yokohama, Faculty of Engineering.
Aoba-ku, Kurogane-cho 1614, Yokohama, Kanagawa, 225-8502, Japan
*sw1@joy.ocn.ne.jp **palacios@cc.toin.ac.jp

Abstract—This work describes a small system we used to test remote manipulation using standard technologies. The system's hardware is built with off-the-shelf parts and the RCX (Robot Command eXplorer) one-board processor and some parts of the RIS (Robotics Invention System) of LEGO. The system's software used the Linux OS (Operating System) and software and routines freely available. With our system it is possible to remotely control a small PC (Personal Computer) camera and obtain real-time video feedback through the same page controlling the camera movements.

Keywords—remote, manipulation, internet, Linux, PERL, LEGO, NQC.

I. Introduction

Internet and wireless communication technologies are evolving at a very fast pace in recent years. Faster and cheaper internet services are bringing permanent connection internet services to our homes. Technologies as ADSL (Asymmetric Digital Subscriber Line) and similar technologies are pushing the Internet out of its traditional environment and with the help of other standards it will be present someday in any electrical appliance around us[1][2][3]. Cellular phones' text-based mail services have popularized the use of e-mail even among users that do not own a computer. Last year the number of cellular phone users in Japan exceeded the number of users accessing the internet through a personal computer. The number of internet users is growing so fast that some countries are in their way to adopt the latest level of the Internet Protocol (IP) IPv6 (Internet Protocol Version 6) and that is now included as part of IP support in many products including major computer operating systems[4]. Cellular phones services are entering in their third generation and real-time video services are becoming available in some countries. Smaller and more capable operating systems are also appearing in some electronic gadgets that will bring the functionality of personal computers to PDAs (Personal Digital Assistant) and cellular phones. This new generation of cellular phones that handle images and have certain internet facilities will let us operate remotely machinery and apparatus that otherwise would need the presence of a person. With this kind of cellular phones we could expect to control web-based surveillance cameras, put an eye on our children and made visual connections to friends and colleagues in the other side of the world. To test the capabilities that are at our hands now, we built a small remote controlled camera by gathering and assembling widely available software programs and off-the-shelf pieces of hardware. The remote controlled camera we built is controlled from any computer with an internet connection and a capable browser. This work describes the means to assemble a similar system. We made also some tests to check the possibility of accessing this system from a cellular phone.

Fig. 1. Remote controlled camera: main components.

II. Test Bench System Overall Overview

The test bench system of this paper evolved from a work done to develop new experiments for a laboratory course of microcomputer applications that use the RIS (Robot Invention System) of LEGO [5]. The initial goal was to control toy robots through a Web page. The RIS kit has a one-board microcontroller board that is the commercial version of a programmable brick developed at the MIT Media laboratory [6]. The microcontroller used in this board is a Hitachi H8 with 32KB of RAM and provides three inputs and three outputs. The outputs could be used to control any of the motors provided with the RIS and related expansion kits. The inputs can handle signals from a variety of sensors (light-sensors, touch sensors, etc.). This kit let the students build near real prototypes in very short times and without the need of special instruments (soldering irons, pliers, etc.). It is being used in many institutions as a means for fostering interest in engineering and for laboratory courses with very tight time schedules. An article describing these and other uses of the RIS can be found in [7]. The center of our system is a PC (personal...
Computer) acting as a server that provides access to the Web page that control the camera (Fig. 2). The camera used is a small IO Data's CCD camera connected to a personal computer through a video capture board (IO Data's GV-VCP2M/PCI). The PC holding the camera is a desk top computer running the Linux operating system (Turbo Linux version 7). The mechanism used to move the camera was built using parts of the RIS kit. The control of the camera is performed through a web page that has a video pane and four-direction link-like control text. Clicking on the text activate small programs that pass commands from the internet browser to the mechanism controlling the movement of the camera. This is realized using CGI (Common Gateway Interface) and small PERL (Practical Extraction and Report Language) programs that communicate with the one-board RCX computer controlling the motors moving the camera. There are many ways of building the mechanism supporting and moving the camera so we will not detail them here. Because the camera is connected trough a cable is not possible to make it move to any direction without a limitation in the displacement (angle) in each move. In the assembling shown in Figure 1 one rotation sensor (not part of the RIS) was used to measure and control the amount of movement of the camera to avoid exceeding the limits impose by the hardware moving the camera. This also permitted us to add one more control to position the camera to its home position.

Fig. 2. A view of the home page controlling the camera.

A. Test Bench System Hardware

The test bench system consists of a personal computer (PC), parts of the RIS kit and a CCD camera (Fig. 1).

B. Test Bench System: Software

The setting of the PC's side software does not need any special manipulation. They are those usually employed to set a Web server. The only tricky setting that sometimes could lead to annoying problems is the identification of the serial port used by the IR (Infra Red) tower of the RIS kit. Sometimes and depending of the OS version in use will be necessary to set this port manually in super user mode using,

```
setserial/dev/ttySXX_irqyy
```

Where X could be 1,2 or 3 and the yy irq number will depend of the devices connected to your PC. They their port and interrupt number could be read from,

```
/proctty/driver/serial
```

provided you are using Turbolinux. With the IR tower connected to the PC just find the port that is handling it (look for the device with transmission rate of 2400 bauds).

B.1 PC/RCX communication’s Software

The communication between the PC and the RCX could be implemented in many ways. There are many programs to talk to the RCX and send it commands [8]. Since we needed to control the RCX via a Web page we decided to use CGI and PERL to write the commands to control the motors attached to the RCX. One program that served us as a start point was the one of [9] that is written in PERL and let us communicate with the RCX using commands. The protocol associated with sending a "message" to the RCX is pretty simple. Bit encoding is 2400 baud, NRZ, 1 start, 8 data, odd parity, 1 stop bit. At the packet level, all packets has a common header

```
0x55 0xff 0x00
```

(sometimes aa ff 00 is used) followed by the data body (opcode bytes and their inverted values)

```
D1 ~ D1 D2 ~ D2 ... Dn ~ Dn
```

a checksum byte (and its inverted value)

```
C ~ C
```

where, C = D1 + D2 + ... Dn. This bit encoding and basic packet structure of the serial protocol was originally described by Dave Baunm in a message posted to the lego-robotics mailing list [10]. For example:

```
55 ff 00 17 08 11 ee 08 07
```

is a packet sending the message "0x11" to the RCX. This scheme, of having an equal number of zero and one bits, allows a receiver to compensate for a constant signal bias simply by subtracting the average signal value. Note that the header also has an equal number of ones and zeros. This characteristic of the header alerts the receiver before the real data arrives. Each message opcode is either a request (PC to RCX) or a reply (RCX to PC). Requests have 0x80 bit cleared, replies have the 0x80 bit set. The reply corresponding to a given request is the request’s complement and vice-versa. Opcodes come in pairs; that is, there are two opcodes to specify each request, and there are two opcodes to specify each reply. The opcodes has been studied (reverse engineered) and details can be found in [11]. Two simple programs written in PERL to start the motor connected to the A port of the RCX are shown in Figs. 3 and 4.

Fig. 3 shows a program that when run sends two datum, a command and an argument to the RCX. This listing shows in details each step of forming a packet of data to be transmitted from the PC to the RCX and could be easily modified to handle input at run time.
#!/usr/bin/perl
#
#motorOn.pl
#didactic version#
#
$D1=hex(21);
$nD1= ($D1)^(255)
$D2=hex(81);
$nD2= ($D2)^(255);
$C=($D1+$D2)%256;
$nC= ($C)^(255);
pack(C*,0x55,0xff,0x00,$D1,$nD1,$D2,$nD2,$C,$nC);

Fig. 3. Detailed PERL program to turn on one motor.

Transmission opening and closing have been abbreviated for clarity. A complete and at the same time more compact program realizing the same function is shown in Fig. 4. In this program data has been calculated manually and each opcode byte is directly written in the pack function line. It is possible to build in this way a Web page with links to a bunch of dedicated programs controlling the movement of small toy-robots built with the RIS kit. To make them do more complicated tasks it is necessary to assemble some programs in a library and call it from the specialized programs. This approach, of using a library, also makes possible the passing of arguments at run time.

#!/usr/bin/perl
#
#motorOn.pl
#complete and compact version#
#
$tty = "/dev/ttyS1";
system("/bin/stty -F $tty speed 2400 raw");
open(TTY,">$tty") || die "open $tty failed.
print TTY pack("C*",0xff,0x55,0x00,0x21,0xde,
0x81,0x7e,0xa2,0x5d);
close(TTY);

Fig. 4. One more program to turn on a motor of the RCX.

However, to build a remote controlled camera with image refreshing capabilities it is necessary to assemble a few more programs.

B.2 Web and Camera’s Software

To make the camera function with the Linux OS one has to install manually its bttv driver. The driver we used was bttv-0.6.4.h.tar.gz. New drivers could be downloaded from [12][13]. Installation is very simple, just follow the directions accompanying the drivers. One possible problem could appear due to the permissions of the installed files. Before using the driver check that /dev/dev0 file’s permission is set to 666. To make possible the refreshing of the image coming from the camera it is necessary to save it in a file and load it through a CGI program. For this we used the CGI w3cam program of [14]. This program is now at version 0.7.2 (we used version 0.6.5). After unpacking and building the required files just put them in the cgi directory of your server. For moving the camera we built a library file that parse data coming from the page showing the images taken with the camera. Fig. 5 details the CGI program to move the camera upwards. The programs to move the camera downwards, and to the left and to the right differs from it only in the line passing parameters to the PERL program handling the communication with the RCX. The arguments passed to the sendcommand.pl program gives direction of movement of the camera.

#!/usr/bin/perl
require "cgiLib.cgi";
&out_html;
system("perl sendcommand.pl 4 ");
exit 0;
sub out_html{
&pr('Content-Type: text/html',"n",
'\<html\><head>',
'\<title\>ugokuzo kamera</title>',
'\</head\>',
'\<body bgcolor="white">',
'\<input type="image" ',
'\<li> <a href="/cgi-bin/up.cgi">UP</a>',
'\<li> <a href="/cgi-bin/down.cgi">DOWN</a>',
'\<li> <a href="/cgi-bin/right.cgi">RIGHT</a>',
'\<li> <a href="/cgi-bin/left.cgi">LEFT</a>',
'\<BR>',
'\RCX CAMERA',
'\</form\>',
'\<img src="/cgi-bin/w3cam.cgi?size=320x240 &color=1&id=1004357802&refresh=0.10&usleep=0 &freq=0&mode=plain&input=composite&norm=ntsc &quality=60">','</body></html>\});

Fig. 5. CGI program to move the camera upwards.

This approach used to display the images taken with the camera avoid the flicker that otherwise will appear in changing from one image to another (new) one. The above CGI program handles the parameters and pass them to the sendcommand.pl program every time one of the link-like text (UP, DOWN, RIGHT, LEFT) controlling the camera is clicked (Fig.2). The other CGI programs controlling the other links are similar to the one shown in Fig.5, they vary only in the parameters passed to the sendcommand.pl program. Fig.6 shows the sendcommand.pl PEARL program that communicates with the RCX and chooses the task to run the motors that move the camera. It also uses a small PEARL library program that contains one function (subroutine) to pack the information to be sent to the RCX. The toggling of the 0x80 bit of the opcode previously sent to the RCX is not set/reset by the CGI programs. It is handled by the NQC (Not Quite C [15]) program that is permanently loaded in the RCX and moves the camera. This NQC program shown in Fig. 7 has four tasks (one for each direction of movement) and one dummy task that alle-
violate us of the toggling of the 0x80 bit in the opcodes (main is also one more dummy task).

```perl
#!/usr/bin/perl
do 'rcxLib.pl';
if( 1 == $ARGV[0]) {
   &trx(1,0x71, 1);
   &trx(1,0x71, 5);
}
elsif(2 == $ARGV[0]) {
   &trx(1,0x71, 2);
   &trx(1,0x71, 5);
}
elsif(3 == $ARGV[0]) {
   &trx(1,0x71, 3);
   &trx(1,0x71, 5);
}
elsif(4 == $ARGV[0]) {
   &trx(1,0x71, 4);
   &trx(1,0x71, 5);
}
else { &trx(1,0x71, 5); }
```

Fig. 6. PERL program controlling the RCX.

The NQC program shown in Fig.7 is downloaded to the RCX and acts according to the parameters passed by the CGI program of Fig.6. After running one non-trivial task (moving the camera) the dummy task is also run, this changes the 0x80 bit before the reception of a new command. Other programs could be easily derived if one takes as reference part of the information in the transmission program shown in [16].

III. Conclusion

This work has detailed the implementation of a small test bench system that controls a PC camera through a Web page. There are other similar works we found about. One details the implementation of a remote controlled camera that moves in a rail [17]. Another work we came across implements a more sophisticated version with almost the same hardware used in our work [18][19]. It is evident from these works and the system shown in this paper that many technologies are becoming more easy to use for our benefit. We are studying now some ways to control non IP-aware appliances through a Web browser.

Acknowledgments

We would like to thank here to the students of the microcomputer applications course of spring 2001 for pointing us to many references and sources when developing some remotely controlled toy-robots.

References


```c
Figure 7. NQC program to move the camera.
```


```
```