AFRL-RI-RS-TR-2011-079

RESEARCH AND DEVELOPMENT OF COLLABORATIVE ENVIRONMENTS FOR COMMAND AND CONTROL

UNIVERSITY OF ALABAMA

MAY 2011

FINAL TECHNICAL REPORT

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■**UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

AFRL-RI-RS-TR-2011-079 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/	/s/
PETER A. JEDRYSIK Work Unit Manager	JULIE BRICHACEK, Chief Information Systems Division Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* **OMB No. 0704-0188**

1. REPORT DATE *(DD-MM-YYYY)* May 2011	**2. REPORT TYPE** Final Technical Report	**3. DATES COVERED** *(From - To)* July 2008 – December 2010

4. TITLE AND SUBTITLE RESEARCH AND DEVELOPMENT OF COLLABORATIVE ENVIRONMENTS FOR COMMAND AND CONTROL	**5a. CONTRACT NUMBER** N/A
	5b. GRANT NUMBER FA8750-08-1-0227
	5c. PROGRAM ELEMENT NUMBER N/A
6. AUTHOR(S) Jingyuan Zhang	**5d. PROJECT NUMBER** NASA
	5e. TASK NUMBER NG
	5f. WORK UNIT NUMBER 04
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Alabama, 301 Rose Administration Bldg., Tuscaloosa, AL 35487-0001	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RISB, 525 Brooks Road, Rome NY 13441-4505	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFRL/RI
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-RI-RS-TR-2011-079

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. PA# 88ABW-2011-0667
Date Cleared: 18 FEBRUARY 2011.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This Final Technical Report discusses the accomplishments of a research effort to develop technology that enables more effective collaboration among computer system operators in a Command and Control environment where both conventional desktop systems and wall-sized displays are used. A Bluetooth LED pen system was developed to allow multiple operators to interact with a wall-size display simultaneously. A capability called mice/cursors without borders was developed which allows a user to move a mouse into another computer, interact with it, and perform copy and paste operations between two computers. To facilitate the development of multi-pen/cursor applications in Java, a Java package was developed to help subscribe to the service. Also, commercial-off-the-shelf frame grabbers have been used with some custom software developed under this project to achieve desktop sharing.

15. SUBJECT TERMS
LED pen tracking, large screen interaction, desktop sharing

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** PETER A. JEDRYSIK
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	62	**19b. TELEPHONE NUMBER** (*Include area code)* N/A

Table of Contents

List of Figures

List of Tables

Executive Summary

In a command and control room, there are usually many computers. Some of them, especially those with a wall-sized display, are for group use whereas the others such as desktops or laptops are for individual use. In this project, a set of hardware and software has been developed to make efficient use of these computers, and to foster collaboration. For a user standing in front of a wall-sized display/computer, a pen-type interaction device is preferred. For that purpose, a Bluetooth LED pen is developed. The Bluetooth LED pen has its own processing and communication capability. It has multiple buttons that can be used to simulate different mouse buttons. Each Bluetooth LED pen has a unique ID. A pen with a unique ID can find a lot of applications. For example, a user can copy an item for one wall-sized computer and paste it to another wall-sized computer.

For a user sitting before a desktop, a mouse is a preferred interaction device. However, a mouse is limited within the computer to which it is attached. In this project, mice/cursors without borders has been developed which allows the user to move a mouse into another computer, interact with another computer, and perform copy and paste operations between two computers.

Multiple interaction devices such as LED pens and/or mice can stay on a single display; the software developed can provide a service to distribute the events generated by these LED pens and/or mice to any application. By subscribing to the service, a truly multi-pen/cursor application can be developed. To facilitate the development of multi-pen/cursor applications in Java, a Java package has been developed to help subscribe to the service.

Desktop sharing is a useful operation in a command and control room. In this project, commercial-off-the-shelf products, Epiphan's DVI2USB™ frame grabbers have been used to achieve desktop sharing. There are several advantages of using DVI2USB™ frame grabbers to perform desktop sharing. First, the desktop to be shared can run any operating system. Second, it is fast. It is done by hardware to capture the desktop and to transport the captured frames to the destination. Finally, there is no network connection required between source and destination computers.

1. Introduction

A command and control room can consist of wall-sized computer displays, referred to as Data Walls hereafter, for group use as well as desktops/laptops for individual use. In this project, we have developed a collaborative environment to make efficient use of these computer systems. The ability to interact with all the computers and to share information among these computers in a command and control setting are two very crucial issues. For interaction, we consider two kinds of popular pointing devices, mice and LED pens. An LED pen is used by the user standing in front of the Data Wall and a mouse is used by a user sitting before a desktop. However, a regular LED pen has a lot of shortcomings. For example, a regular LED pen has no ID associated with it. Therefore two regular LED pens look the same to a computer. Also a mouse is limited within the computer to which it is attached. In this project, we developed a collaborative environment that allows the LED pens and mice to interact with any computer and to carry information from one compute to another. Specifically, we developed the following hardware and software set.

- Customized LED pens that have their own processing and communication capability with their own unique identification.
- A camera system that is able to monitor multiple screens including hard and soft screens, and track multiple LED pens including regular and customized ones.
- Mice/cursors without borders to allow a mouse to interact with another computer.
- A Java package to take advantage of multiple LED pens and multiple mice in a single application and to develop a truly multi-pen/cursor application.
- Software that allows LED pens and mice to copy and paste between two computers.
- Desktop sharing using DVI2USB™ frame grabbers.

In this report, we will detail how the hardware and software set is implemented.

2. LED Pens for Interaction with Data Walls

In the past, we have used regular laser pointers and LED pens to interact with a Data Wall [1][2][3]. The advantage is their low cost and wide availability. However, regular LED pens have their shortcomings. For example, a regular LED pen has only one button. To simulate either the left or right mouse button with the only button, we need the help of a mouse resource window that uses screen real estate [1]. On a large screen the mouse resource window also needs to be moved periodically for accessibility. Most importantly, the dot from a regular LED pen does not have an identity. When multiple LED pens are used with a Data Wall, multiple dots can be seen, but for each dot, it is unknown from which LED pen the dot comes. To address these issues, we have built a customized LED pen. The customized LED pen has four buttons (referred to as H, L, W, and M) as shown in Figure 2.1. Their functions will be discussed later. With the customized LED pen, no mouse resource window is needed. We have also extended the Laser/LED software such that both regular and customized LED pens can be used to interact with Data Walls.

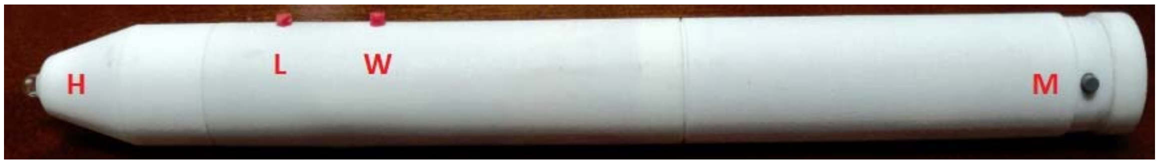
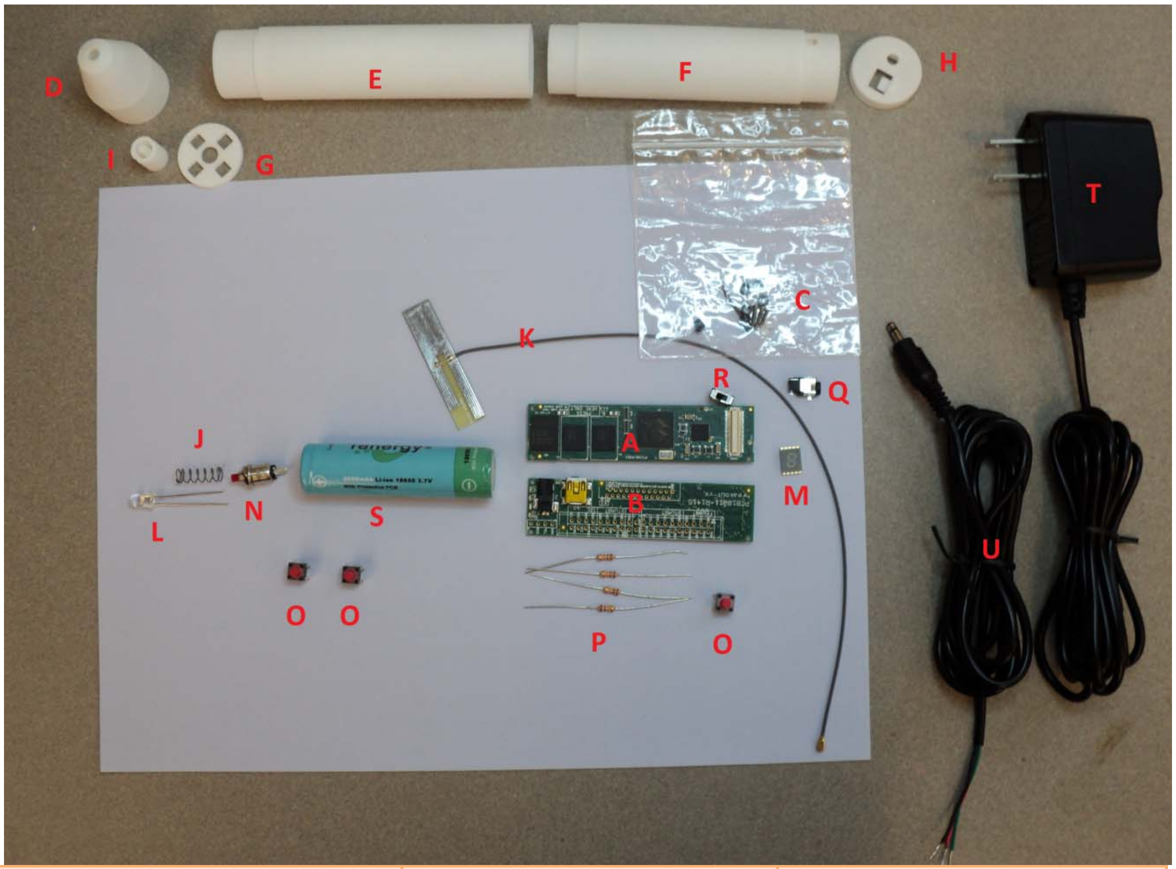


Figure 2.1: A customized LED pen.

2.1 Parts used in an LED Pen

Building a customized LED pen involves several aspects including mechanical, electrical, and software. The heart of the pen is a Gumstix Verdex Pro running Embedded Linux, and the pen is powered by one Li-ion 18650 rechargeable battery. Figure 2.2 shows the parts used in a customized LED pen.



A. Gumstix Verdex Pro	H. Case_Tailcap	O. Tactile Switches(3)
B. Gumstix Breakout-vx	I. Case_LEDseat	P. 10K Ohm Resistors(4)
C. Screws & Spacers Kit	J. Spring	Q. Slide Switch
D. Case_Head	K. Antenna	R. Power Jack
E. Case_Battery	L. LED	S. Li-ion 18650 Battery
F. Case_Chip	M. LCD Display	T. Li-ion Battery Charger
G. Case_Disk	N. Momentary Switch	U. Charger Cable

Figure 2.2: The parts used in a customized pen.

Figure 2.3 illustrates how to connect the electronic components within the LED pen. In particular, when the slide switch is on, the battery supplies power to the Gumstix computer, and when the slide switch is off, the battery can be recharged.

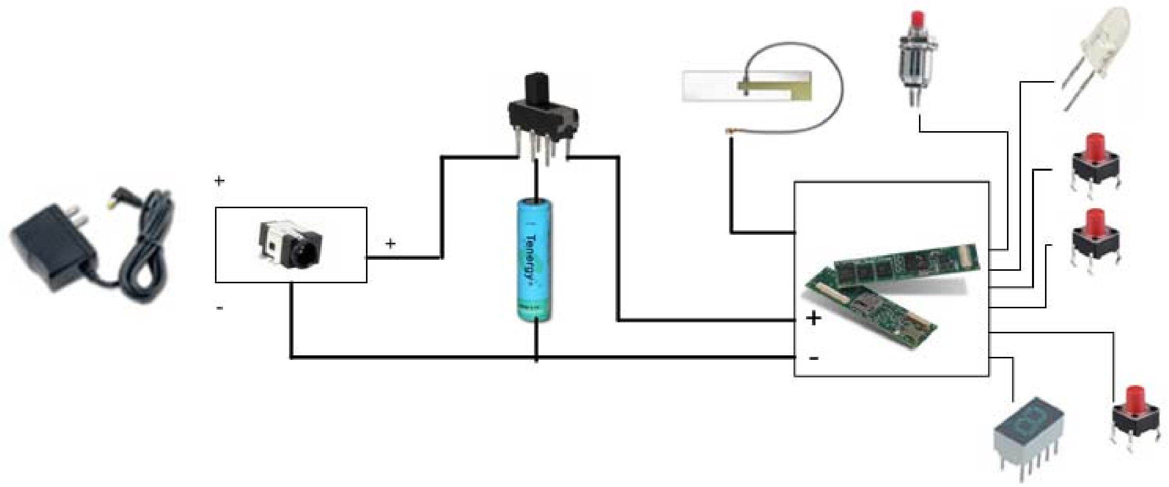
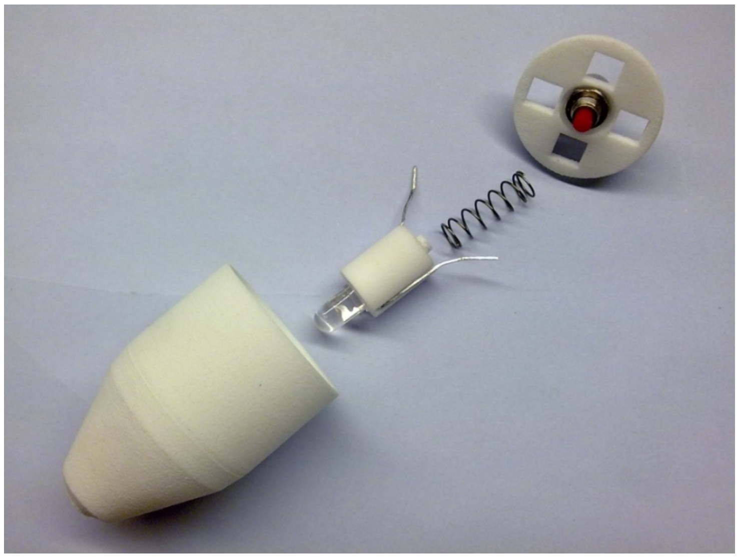
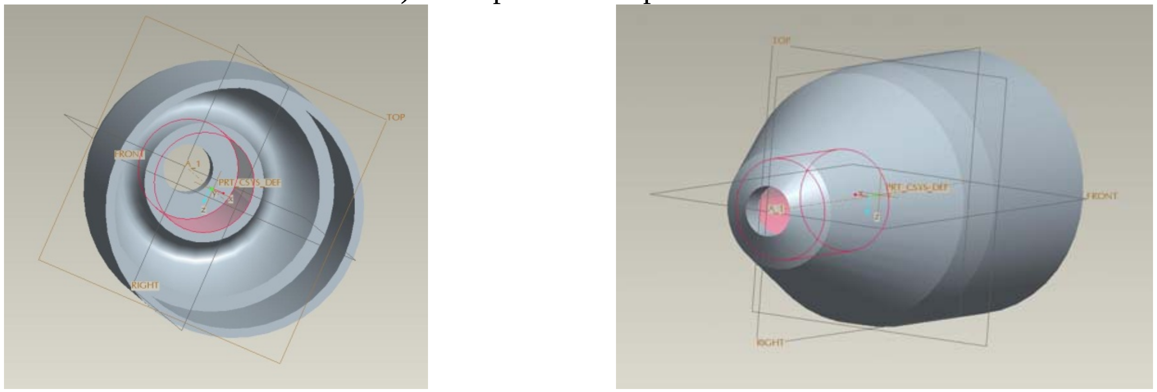


Figure 2.3: The diagram for connecting the electronic components.

To design the housing for the LED pen, we use a parametric feature-based solid-modeling software named Pro/Engineering. The parametric modeling approach enables us to create a complete 3D digital model and optimize the design quickly. Pro/Engineering allows a user to build a design and change part of the design while maintaining the constraint of geometric association at any state of building. The viewer tool presents the designed model with 360-degree perspective views even after regeneration of the design, which is critical to a successful design. The software can export the design into various file formats, including the manufacturing information, e.g. tolerance of accuracy, used as the input to manufacturing.

To realize our design, the best solution is to use a technique called “3D Printing”. 3D printing is a form of additive manufacturing technology where a three dimensional object is created by successive layers of material. It has been proved as a high-quality production technique for 3D prototyping. Advanced 3D printing technologies yield models that closely emulate the look, feel and functionality of product prototypes. After the pen housing was designed, we asked Shapeways.com to print it. Their united facilities turn the design of solid modeling to real objects quicker and cheaper than a standalone research lab does. Their online order system accepts Collada or X3D formatted files. Their 3D printers use liquid materials to “print” thermodynamics sets that are cured in certain temperature. The printers also offer product developers the ability to print parts and assemblies made of several materials with different mechanical and physical properties in a single build process.


a). The parts in the pen head.


b). The head shell design.

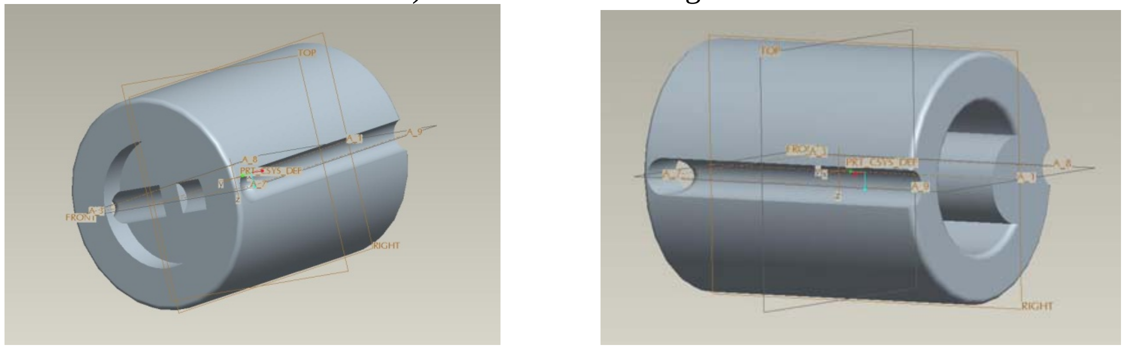

c). The LED seat design.

Figure 2.4: The head design for LED pens.

For easy assembly of the LED pen, we designed the whole LED pen into four parts: a head, a battery body, a chip body and a tailcap as shown in Figure 2.2. The head design is a bit complicated and consists of the following parts as shown in Figure 2.4(a).

- LED
- LED seat
- Head shell
- Spring
- Momentary switch
- Disk base

The LED is glued to the LED seat securely as if they were a single unit, and the momentary switch is secured on the disk base by its screw-nut fastener. The disk base can rest on the inside rim of the head shell, and a long cylindrical track is built within the head shell so the LED holder can move within the track with very little lateral movement. (Please see Figure 2.4(b) for the drawing of the head shell.) A compression spring is placed between the disk base and the LED seat. The spring is partially recessed into the LED holder. (Please see Figure 2.4.c for the drawing of the LED seat.) When the LED is pressed on the hard screen, the LED and its holder will compress the spring, and the seat will push on the momentary switch to turn on the LED. When the LED is away from the screen, the homing force of the spring pushes the LED and its seat away from the momentary switch, and the LED is switched off. The track within the head shell ensures the LED seat moves within the track in one direction, and the spring makes the LED holder move more responsively.

2.2 Gumstix Motherboard

The Gumstix motherboard used in the customized LED pen is Verdex Pro XM4-bt. Figure 2.5 shows its image with a microSD card. The motherboard size is 3.15 in. x 0.79 in. x 0.25 in. (80mm x 20mm x 6.3mm) and weights about ¼ oz (8g). It can run on a 5VDC power supply. The board uses 400 MHz Marvell® PXA270 with XScale™ as its processor, 64 MB RAM, and 16MB flash memory. Verdex Pro XM4-bt comes with a microSD adaptor and an Infineon PBA31308 RoHS-compliant (Class 2 Bluetooth) module. The motherboard has three connectors: 60-pin Hirose I/O connector, 80-pin Hirose I/O connector, and 24-pin flex ribbon connector. These connectors can be used to add an expansion board such as breakout-vx, console-vx, and tweener.

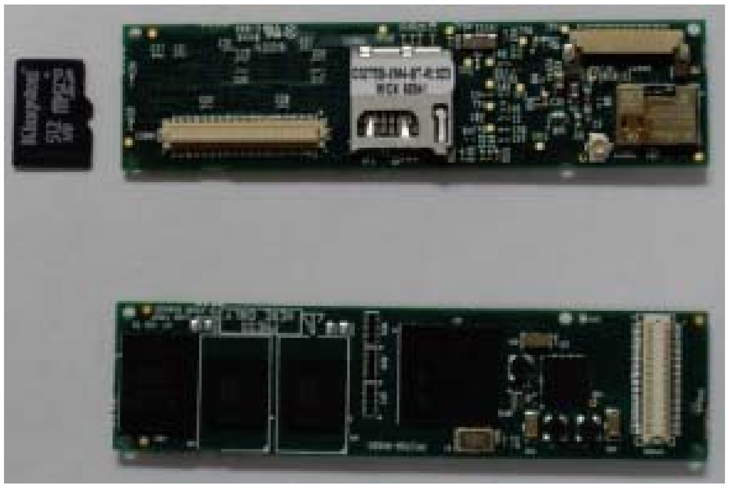


Figure 2.5: The Gumstix Verdex Pro XM4-bt motherboard.

There is no switch to turn on the Gumstix motherboard itself. Once the power supply is connected, the Gumstix motherboard will boot from the boot loader. The boot process can be shown within an X terminal of a Linux machine. Figure 2.6 shows how to use a Linux machine, as an X terminal to the Gumstix motherboard. In the figure, the console-vx expansion board was connected to the Verdex Pro XM4-bt motherboard with the 60-pin connector. Then a serial null modem cable is plugged in the middle serial port of the console-vx at one end and the serial port of the Linux machine at the other end. The mother board and the expansion board are powered with a 5.0 Volt wall adaptor.

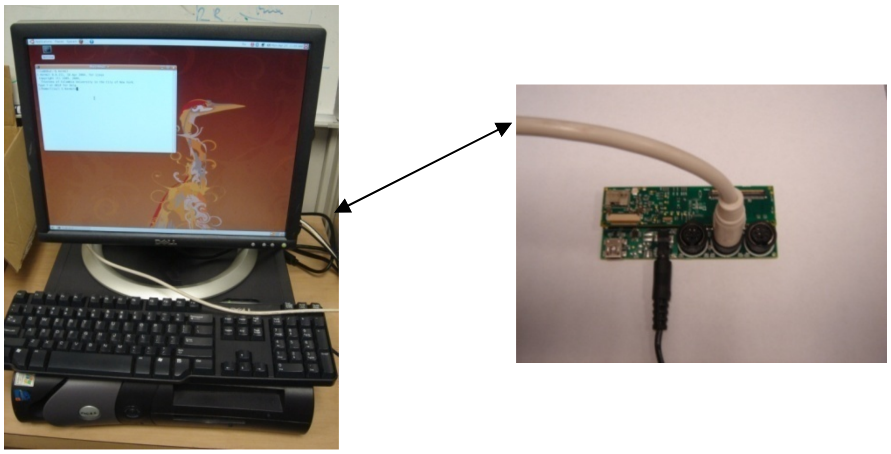


Figure 2.6: Use a Linux machine as a terminal to the Gumstix motherboard.

To use an X terminal as a terminal to the Gumstix, a terminal program such as Kermit needs to run in the X terminal. In this report, we assume that the Linux machine is running Ubuntu 8.04. The following commands can be used to download and install

Kermit in Ubuntu. From here on, the $ symbol is used to represent the Linux shell prompt.

```
$ sudo apt-get install ckermit
```

To start Kermit in an X terminal, type in the following.

```
$ kermit -l /dev/ttyS0
```

Here */dev/ttyS0* is the serial port used by the serial null modem cable on the Linux machine. At the Kermit prompt (i.e. *C-Kermit>*), type in the following command to set up the parameters for the serial communication. Here the *~/gumstix/gumstix-oe* directory is the directory to install the OpenEmbedded build system, to be described later.

```
C-Kermit> take  ~/gumstix/gumstix-oe/extras/kermit-setup
```

Then use the following command to use the X terminal as a terminal to the Gumstix.

```
C-Kermit> connect
```

After launching either of these two programs, plug in the power adaptor to the Console-VX expansion board to boot the Gumstix. A message from the U-Boot boot loader and the normal Gumstix boot sequence will show on the X terminal. Through the X terminal, the keyboard can be used to provide input to the Gumstix.

To return to the Kermit, type [CTRL-\], then press c. Type quit at the Kermit prompt to exit the Kermit.

2.3 Build and Install Software for Gumstix

The Gumstix runs Linux. There are three software subsystems: the boot loader named U-boot, the Linux kernel image, and the root file system, stored on the onboard flash memory. When the Gumstix motherboard is powered-on, the U-boot boot loader is started first to perform the first-pass initialization of the hardware on the motherboard. Then it loads and jumps to execute the Linux kernel. Normally, it is not necessary to update U-boot. If needed, there are pre-built U-boot binaries that can be downloaded and reflashed to the motherboard. The other two subsystems can be built on a Linux machine using a build system called OpenEmbedded (OE for short). In this Section, we will first describe how to build the image files for the Linux kernel and the root file system. We will then show how to install the Linux kernel and the root file system. Finally, we will present how to build and execute the client software for the Gumstix. Please note most contents in this section are adapted from http://gumstix.net/wiki.

2.3.1 Build the Linux kernel and the root file system

The build environment for Gumstix is the OpenEmbedded (OE) build system that needs to be installed on a Linux machine. The Linux machine used in this report runs Ubuntu 8.04. The OE will be installed under the ~/gumstix/gumstix-oe directory. The OE sets up a cross development build environment and creates standard images for the Linux kernel and a root file system for Gumstix.

Before installing OE, we need to first install a list of essential utility packages: svn, gcc, patch, help2man, diffstat, texi2html, texinfo, libncurses5-dev, cvs, gawk, python-dev, python-pysqlite2.

Each package can be installed using the following command.

```
$ sudo apt-get install package_name
```

According to the requirement of OE, /bin/sh shall be linked to /bin/bash for Ubuntu. This can be achieved using the following command, and selecting the “No” option.

```
$ sudo dpkg-reconfigure dash
```

The second step is to get the Gumstix OE. With the help of svn, the subversion source code control package, the source code of the Gumstix OE build system can be copied to the local machine under ~/gumstx/gumstix-oe using the following command after changing to the *~/gumstix* directory.

```
$ svn co https://gumstix.svn.sourceforge.net/svnroot/gumstix/trunk gumstix-oe
```

Once the build environment has been set up, the directory structure should look like the following.

```
gumstix-oe
|-bitbake
|-build
|-org.openembedded.snapshot
|-com.gumstix.collection
|-user.collection
|-extras
`-tmp
```

To use the OpenEmbedded build system, we need to add some settings to the .bashrc file using the following command.

```
$ cat ~/gumstix/gumstix-oe/extras/profile >> ~/.bashrc
```

To build a basic Linux kernel and root file system image with the build OE environment, we can run:

```
$ bitbake gumstix-basic-image
```

If the first step has not been done, some essential packages will be missing. The bitbake will prompt you both in command line output and build log file. If that occurs, install the missing package(s) and run the above command again.

The building process needs to download source code. So be sure the internet connection is available. The build process usually takes 60 to 180 minutes depending on the packages to be built, building machine and internet bandwidth. Once it is successful, the kernel image and root file system image including boa, cron and ntp components are in the ~/gumstix/gumstix-oe/tmp/deploy/glibc/images/gumstix-custom-verdex directory. The name of the root file system image is:

```
Angstrom-gumstix-basic-image-glibc-ipk-2007.9-test-20071101-gumstix-custom-
verdex.rootfs.jffs2
```

with a link in a shorter file name:

```
gumstix-basic-image-gumstix-custom-verdex.jffs2
```

and the name of the Linux kernel image for Gumstix is:

```
uImage-2.6.22-r1-gumstix-custom-verdex.bin
```

2.3.2 Install the system software for Gumstix

Recall there are three software subsystems, the U-boot boot loader, the Linux kernel image and the root file system. This subsection shows how to install them. An easy way to install or update the system software for Gumstix is to use the microSD adaptor on the motherboard. To install the Linux kernel and the root file system, copy their image files built in the previous subsection to a microSD card in the FAT format. Insert the microSd card into the microSD adaptor before powering on the Gumstix.

Once the Gumstix is powered on, you will see a line similar to the following line.

```
Hit any key to stop autoboot:  2
```

Press a key before the number reaches 0 to enter the boot loader interactive command mode, and it shows the following prompt.

```
GUM>
```

First initialize the microSD card by typing

```
GUM> mmcinit
```

Then erase the old Linux kernel and the root file system on the flash memory by protecting the boot loader followed by erasing all as follows.

```
GUM> protect on 1:0-1
GUM> erase all
```

Now, we can upload the new Linux kernel and the root file system. To upload the root file system image from the microSD card to the RAM starting at address a2000000, use the following command.

```
GUM> fatload mmc 1 a2000000 gumstix-basic-image-gumstix-custom-verdex.jffs2
```

Copy the root file system image from the RAM to the flash memory starting at the address 40000 by the following command.

```
GUM> cp.b a2000000 40000 ${filesize}
```

Here *filesize* is a variable that is used to catch exactly how many bytes have been copied. To upload the new Linux kernel image to the RAM, type in the following command.

```
GUM> fatload mmc 1 a2000000 uImage-2.6.22-r1-gumstix-custom-verdex.bin
```

To install the kernel from the RAM to the flash memory, call the following command in the *kat* toolkit.

```
GUM> katinstall 100000
```

The above command copies the kernel up to 0x100000 (1MB) bytes from the RAM starting at a2000000 to the flash memory starting at 1f00000. If the new kernel is copied successfully, the following command can be executed to restart the new Linux kernel and remount the new root file system.

```
GUM>bootm
```

2.3.3 Build and install the client software for Gumstix

The previous two subsections dealt with how to build and install the system software for the Gumstix. In this subsection, we will show how to build and install the client software for the Gumstix. The software for the Gumstix is organized into packages. Every software package is associated with a description file with the .bb extension.

For each software package, we need to create a directory under the *user.collection/packages* directory to put all the program files and the .bb file. The .bb file tells the build system how to build the package. Our client software is called *pointer*, and we create a directory called *pointer* under the *user.collection/packages* directory. Under the *pointer* directory, we create a subdirectory named *files* for putting all the source code files and a receipt file named *pointer.bb*. The following directory diagram illustrates the structure of our package.

```
user.collection
|
`-packages
  |
  `-pointer
    |
    |-files
    | |
    | `-pointer.c
    |
    `-pointer.bb
```

Then we can call bitbake to build our package simply by typing the following at a shell prompt. You don't need to change the directory into the package directory to do this. BitBake knows where to look for the .bb file and how to build the pointer package based on the .bb file.

```
$ bitbake pointer
```

After the build process completes, a package file named *pointer-r0_armv5te.ipk* can be found in the *gumstix-oe/tmp/deploy/glibc/ipk/armv5te* directory.

To install the package into the Gumstix, we copy the *pointer-r0_armv5te.ipk* file to a microSD card, and insert the card into the MMC slot on the expansion board. By default, the microSD card is mounted on the */mnt/card* directory in the Gumstix. On the Gumstix, we can change the directory to */mnt/card* and use the ipkg tool to install the package as follows.

```
$ ipkg install pointer-r0_armv5te.ipk
```

The command line will show whether the package has been successfully installed or not. If it has been successfully installed, the client program can be started by calling the program directly.

```
$ pointer
```

To start the client program automatically after the Gumstix boots, we create the *lpen.sh* file under the */etc/init.d* directory of the Gumstix file system and include the following two lines in the file.

```
Pointer&
echo "lpen started"
```

And change the mode of *lpen.sh* file to make sure it is executable. Then in the /etc/rc.d/rc5.d/ directory, create a symlink by using

```
ln –s ../init.d/lpen.sh S99lpen
```

The "S" means to run this script when the system is starting up, as opposed to shutting down. The “99” is a number which tells when to execute the script. We use 99 to start the program after regular Gumstix boot process is completed.

2.4 GPIO Programming on Gumstix

One reason the Gumstix was chosen is that its PXA270 processor provides 121 highly-multiplexed general-purpose I/O (GPIO) pins for use in generating and capturing application-specific input and output signals. Each pin can be programmed as an output, an input, or as bidirectional for certain alternate functions that override the value programmed in the GPIO direction registers. When programmed as an input, a GPIO pin can also serve as an interrupt source. All GPIO pins are configured as inputs during the assertion of all resets, and they remain inputs until configured otherwise. In addition, select special-function GPIO pins serve as bidirectional pins where the I/O direction is driven from the respective unit (overriding the GPIO direction register).

When a GPIO pin is programmed as output, if the load is less than 250mA running between 3.8V-5V, it could be supplied by the GPIO directly. The LED lamp we have selected has a forward voltage of approximately 3.6V and a forward current of 20mA. Therefore the LED can be directly soldered to the output GPIO pin. The logic high signal will power the LED and the logic low (about 1.87V) will turn it off. It is a bit tricky when a GPIO pin is programmed as input. Figure 2.7 shows the circuit diagram about how to compose a GPIO input module. We use a 10K Ohm resister to pull down the voltage on the switch. So the switch is able to shift between high voltage (more than 3.8V) and low voltage (0V).

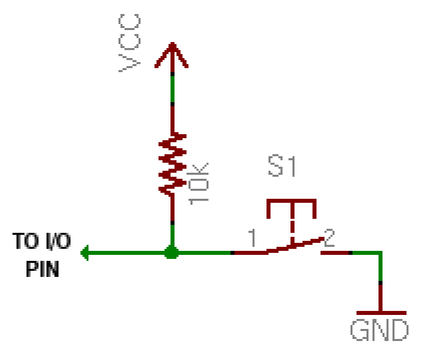


Figure 2.7: The circuit diagram of a GPIO input module.

Table 2.1 shows how eight LCD segments are connected to the GPIOs (all output signals) by soldering into the corresponding holes on the breakout-vx expansion board, and Table 2.2 shows how to connect the LED lamp and four buttons to the GPIOs by soldering into the corresponding holes on the breakout-vx expansion board.

Table 2.1: Pin Assignment for LCD segments

LCD Pin#	**LCD Segment#**	**GPIO#**	**Breakout-vx Pin#**
1	e	GPIO(69)	LDD_11
2	d	GPIO(70)	LDD_12
3	NA	NA	NA
4	c	GPIO(72)	LDD_14
5	.	GPIO(73)	LDD_15
6	b	GPIO(71)	LDD_13
7	a	GPIO(87)	LDD_17
8	NA	NA	Vcc
9	f	GPIO(67)	LDD_09
10	g	GPIO(68)	LDD_10

Table 2.2: Pin Assignment for LEDs and buttons

LEDs/Buttons	**GPIO#**	**Breakout-vx Pin#**
Head LED, signal output	GPIO(59)	LDD_01
H button, signal input	GPIO(58)	LDD_00
L button, signal input	GPIO(62)	LDD_04
R button, signal input	GPIO(64)	LDD_06
M button, Signal input	GPIO(65)	LDD_07

2.5 *Laser/LED Server Software*

We enhanced the Laser/LED Server software in the following four aspects. First, it can monitor both soft and hard screens. Second, it can monitor multiple Data Walls. Third, it can handle both regular and customized (also called Bluetooth) LED pens. Finally, LED pens can be used for annotation and text entry in addition to mouse interaction.

2.5.1 Laser/LED Server for both hard and soft screens

2.5.1.1 Avoiding specular spots

The Laser/LED Server software developed previously works with soft screens only [1]. In order to allow more intuitive interaction with the display using an LED pen that would shine when pressed to the screen required a hard screen surface. There are a number of issues associated with large hard screens that needed to be addressed. The first of which are surface reflections. The Data Wall screen installed at AFRL/RI is 15 feet x 4 ½ feet and necessitated it be made of glass to ensure rigidity. It is extremely rigid but the rear surface is very reflective. As a result the light emitted from the video projector lens is seen in the reflection on the rear of the screen and subsequently in view of the pen tracking cameras. Figure 2.8 shows two specular spots caught by two cameras on a hard screen. Specular spots are not welcome in the Laser/LED Server software because they interfere with laser/LED dots.

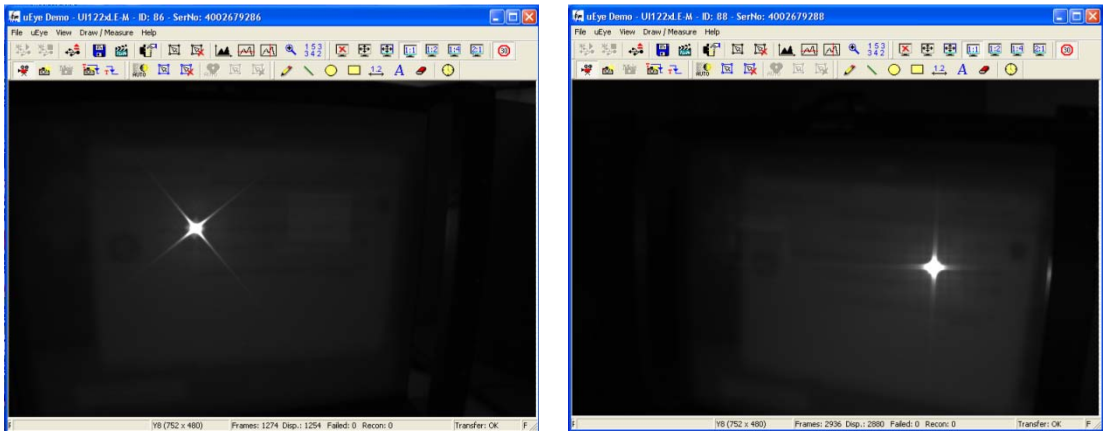
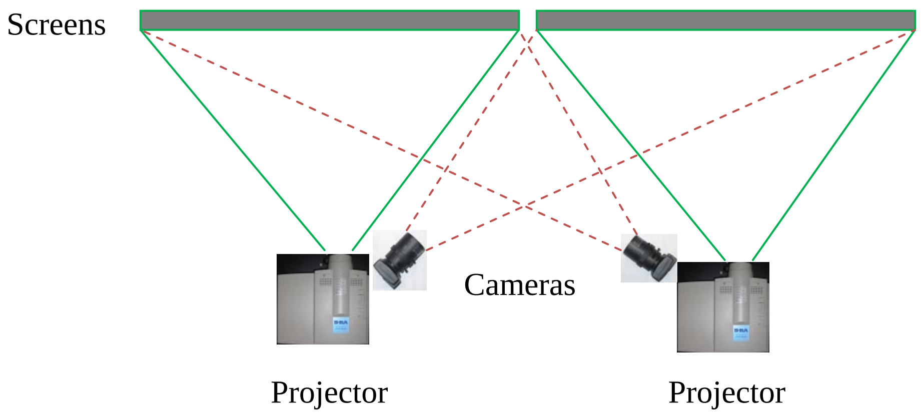

uEye Demo - UI122xLE-M - ID: 86 - SerNo: 4002679286
File uEye View Draw / Measure Help
Y8 (752 x 480) Frames: 1274 Disp.: 1254 Failed: 0 Recon: 0 Transfer: OK
uEye Demo - UI122xLE-M - ID: 88 - SerNo: 4002679288
File uEye View Draw / Measure Help
Y8 (752 x 480) Frames: 2936 Disp.: 2880 Failed: 0 Recon: 0 Transfer: OK

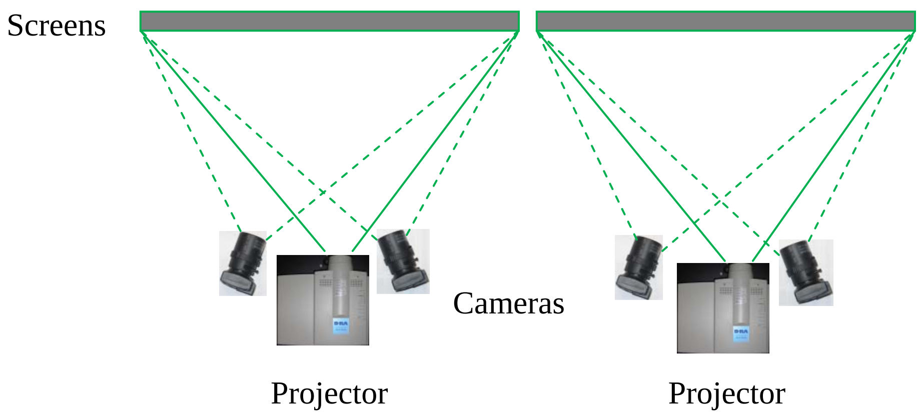
Figure 2.8: Specular spots on a hard screen.

There are two methods that can be used to solve the problem. One is to place the camera at an extremely oblique angle to avoid the specular spot as shown in Figure

2.9(a). Because the camera is put at an extremely oblique angle, the specular reflection from the screen is out of the camera's view. The other is to place two cameras at slightly different angles to watch the same screen as shown in Figure 2.9(b). In this method, both cameras can still see the specular spots. Figure 2.8, in fact, consists of two images of the same screen taken by two different cameras placed at slightly different angles. However, when the two cameras are placed properly, these two specular spots will show at different locations in different cameras. We can then combine these two images into one to eliminate specular spots. The first method uses one camera for each screen while the second one uses two cameras for each screen. However, the image obtained from the second method is less distorted than that from the second method because of the ways these cameras are placed. We decided on the first method because it requires less bandwidth, less equipment, lower cost, and the result is very satisfactory.



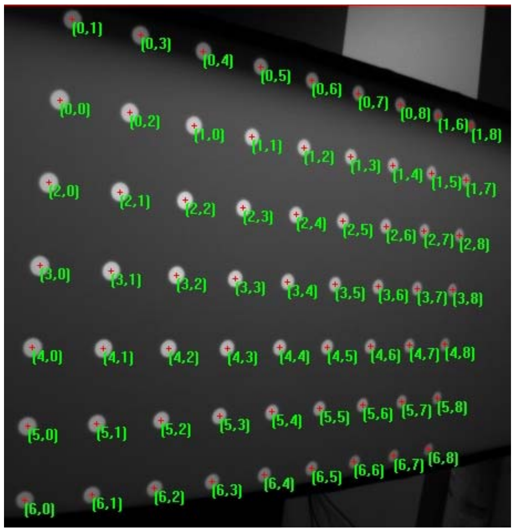
a). One camera each screen



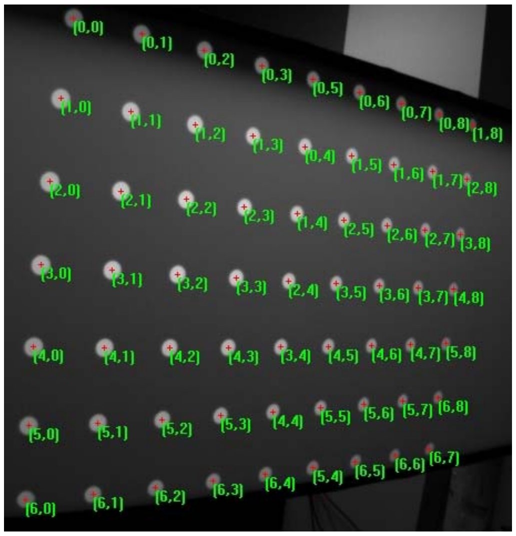
b). Two cameras each screen

Figure 2.9: Methods to avoid or eliminate specular spots.

We first made modifications to the camera calibration. When a camera is placed at an oblique angle, our previous camera calibration does not work. After the calibration dots have been detected, our previous camera calibration first orders them into rows by sorting their *y* coordinates and then sorts each row by their *x* coordinates. This works fine when a camera is placed almost perpendicularly to the screen, but it does not work when a camera is placed is at an oblique angle. We have also tried to first order them into columns by sorting their *x* coordinates and then sorts each column by their *y* coordinates. Although it works in situations in which a camera can be placed such that the top and bottom halves can be imaged almost equally, it does not work in all cases. Recall that the left and right halves can never be imaged equally due to the fact that the camera is put at an oblique angle. Figure 2.10 shows that both sorting methods cannot number these calibration dots in the right order.



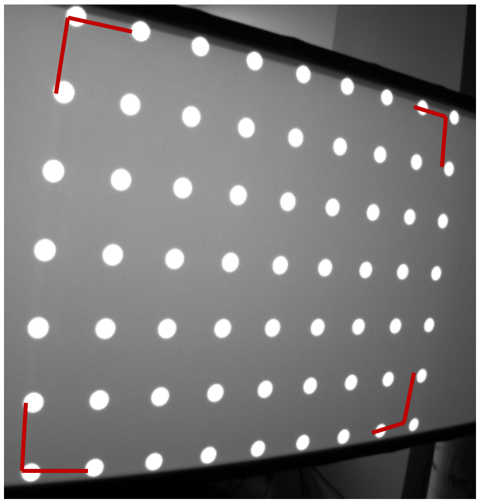
a). Sort by *y* first, then by *x*



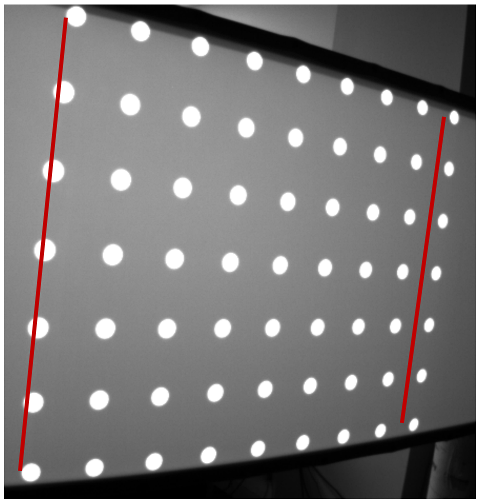
b). Sort by *x* first, then by *y*

Figure 2.10: Sorting does not work in camera calibration.

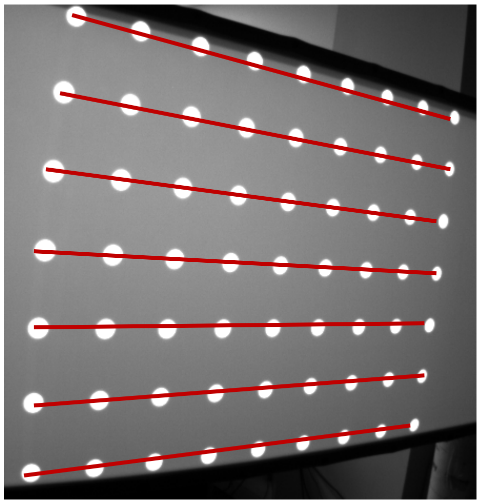
The new ordering procedure starts with finding the four corner dots at the top-left, bottom-left, top-right and bottom-right corners as shown in Figure 2.11(a). The top-left and bottom-left corner dots define a line, and all the dots close to the line will then be identified and sorted from top to bottom. Similarly the top-right and bottom-right corner dots define a line, and all the dots close to the line will be identified and sorted from top to bottom. See Figure 2.11.b for the dots on the first and last columns. Finally a dot in the first column and its corresponding dot in the last column define a line that can be used to identify all the dots on a row, and all the dots on a row can be sorted from left to right as shown in Figure 2.11.c. Figure 2.11.d shows all the dots are correctly ordered.



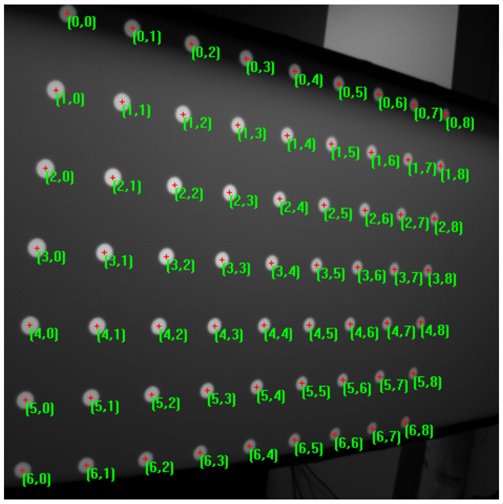
a). Find the four corner dots



b). Find the first and last columns



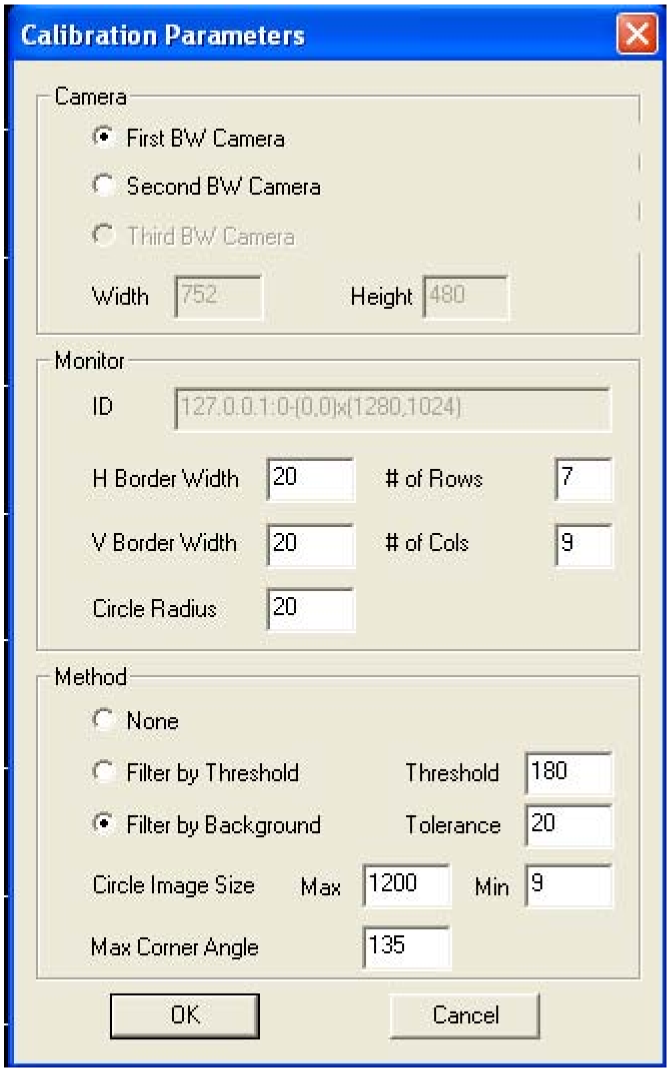
c). Find all the rows



d). All the dots are correctly ordered

Figure 2.11: Steps to order all the calibration dots.

To check whether a calibration dot is a corner dot, we introduce a new calibration parameter named "Max Corner Angle" as shown in Figure 2.12. For each dot, we can find the smallest angle such that all the other dots can be contained within the angle starting from said dot. For an interior dot, the angle is almost 360 degrees, and for an edge dot, the angle is approximately 180 degrees. For a corner dot, the angle is around 90 degrees. See Figure 2.11(a) for an illustration of the angles at the four corners. The "Max Corner Angle" is the angle that is used to separate the corner dots from the edge dots and the interior dots. The default value is 135 degrees. In almost all cases, it is sufficient to separate the corner dots from the non-corner dots. In case 135 is not appropriate, use manual calibration to check the calibration image, find a better angle for this parameter, and recalibrate manually using the new angle.



Calibration Parameters

Camera

[✓ First BW Camera] Second BW Camera Third BW Camera

Width 752 Height 480

Monitor

ID 127.0.0.1:0-(0,0)x(1280,1024)

H Border Width 20 # of Rows 7

V Border Width 20 # of Cols 9

Circle Radius 20

Method

None Filter by Threshold [✓ Filter by Background]

Threshold 180

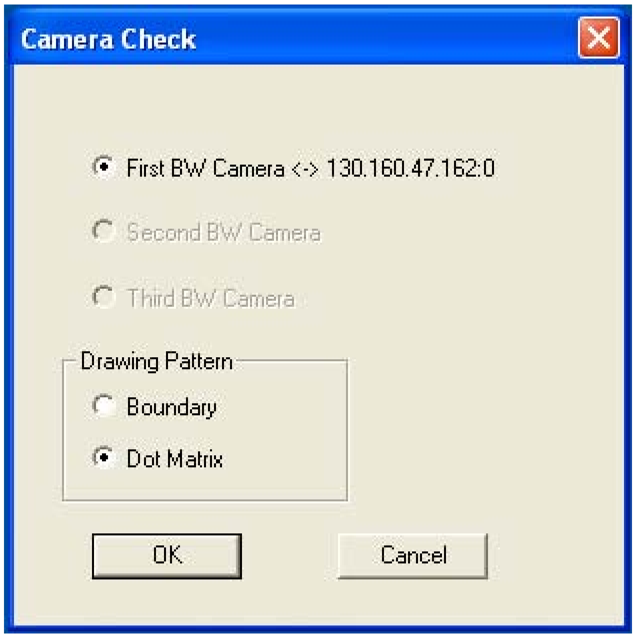
Tolerance 20

Circle Image Size Max 1200 Min 9

Max Corner Angle 135

Figure 2.12: A new calibration parameter named "Max Corner Angle".

When a camera is placed at an oblique angle, the calibration dots at the left column will have a different distance to the camera from the dots at the right column. The dots at the far end will look smaller than those at the near end. When a dot is small, adding or removing a pixel to or from the dot can cause a significant change in its centroid calculation. Therefore it is a good idea to have the dots at the far end well focused. In order to do that, we introduce the "Drawing Pattern" in the "Camera Check" dialog box as shown in Figure 2.13. The "Boundary" drawing pattern will show the boundary of a monitor/projector image that can be used to adjust the camera to keep the entire projector image in view. The "Dot Matrix" drawing pattern will show a matrix of dots that can be used to adjust the camera to keep the dots at the far end well focused.



[✓ First BW Camera <-> 130.160.47.162:0]

Second BW Camera

Third BW Camera

Drawing Pattern

Boundary

[✓ Dot Matrix]

Figure 2.13: Drawing pattern selection in Camera Check.

We also made some changes to the automatic camera configuration. When a camera is placed at an oblique angle, a camera can see more area of the display. During the automatic camera configuration, a big dot along with a small dot can sometimes be seen by more than one camera. In this case, we calculate the distance between the mid-point of two dots and the camera image center, and select the one with the minimum distance, assuming the center of the projector image will be near the center of the corresponding camera during camera adjustment.

2.5.1.2 Handling dimmer images

The camera is placed at an oblique angle to avoid the specular spot. Another issue associated with the hard screen is the projector image appears very dim on the rear of the screen. This is due to the thickness of the screen and the diffusion layer being near the front surface. Please see the image at the right side of Figure 2.14 for an illustration. To handle a dimmer image, we made changes to Auto Configuration and Camera Calibration.

Auto Configuration

During Auto Configuration, the software first displays a black background and asks every camera to take a picture. Then for each screen, it displays two dots with one dot being significantly larger than the other, and asks which cameras can see the two dots. In this way, a camera is paired with a screen automatically. At the same time, how the camera is placed in relation to the screen can be determined by the positions of two dots on the captured image.

In the past, we compared the background picture and the dot picture using a fixed value between 0 and 255 called *tolerance*. If the difference between two corresponding pixels from two pictures is within the toleration, they are considered as the same. If two corresponding pixels differ beyond the tolerance, the pixel is a part of the dots. For the soft screen, we have successfully used a fixed value of 20 for the tolerance. However, for the hard screen, even the pixel intensity for the dot can be very small. Figure 2.14 shows a background image and a dot image. It is hard to see the dots in the right figure. It is also hard to assign a predefined value for the tolerance. With two captured images, we wrote a function called SurveyDifference to count the number of pixels for each possible difference, and let the result be called the difference map. Then another function called EstimateTolerance works on the difference map to suggest a tolerance. Basically there are two peaks in the difference map; one peak corresponding to no change between the two images and the other to the change. A suggested tolerance represents the valley between the two peaks. This method works well with both hard and soft screens. We have not encountered a failure with the new Auto Configuration so far. Even if Auto Configuration does not work, Manual Configuration is always available.

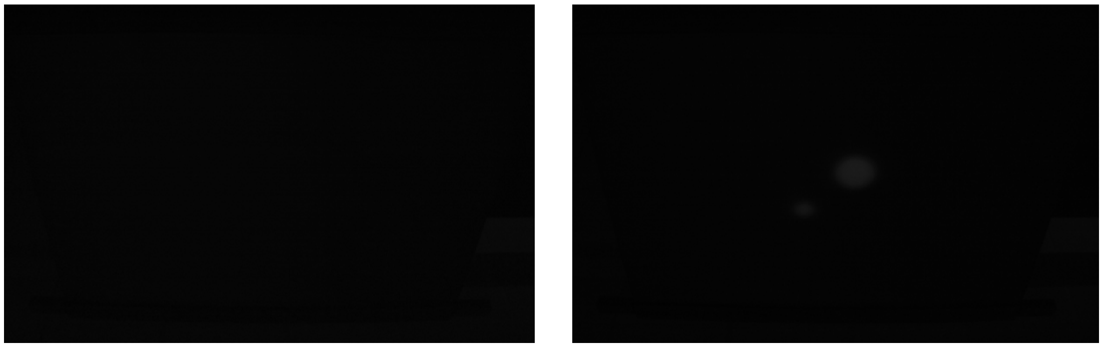
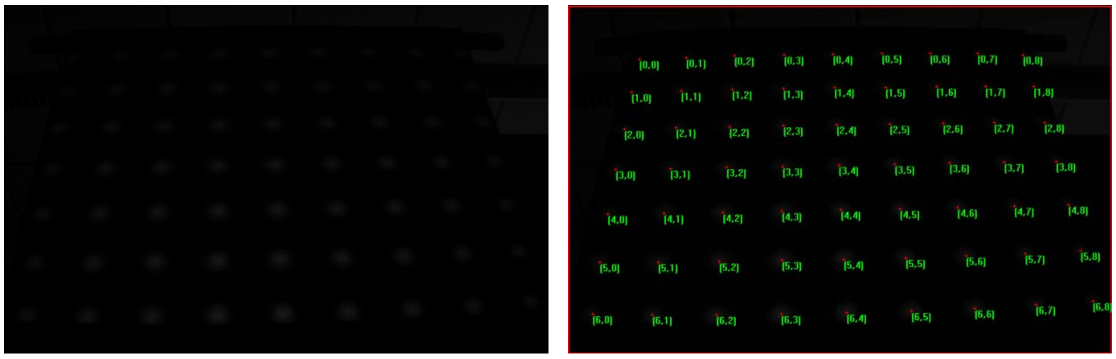


Figure 2.14: A background image (left) and a dot image (right).

Camera Calibration

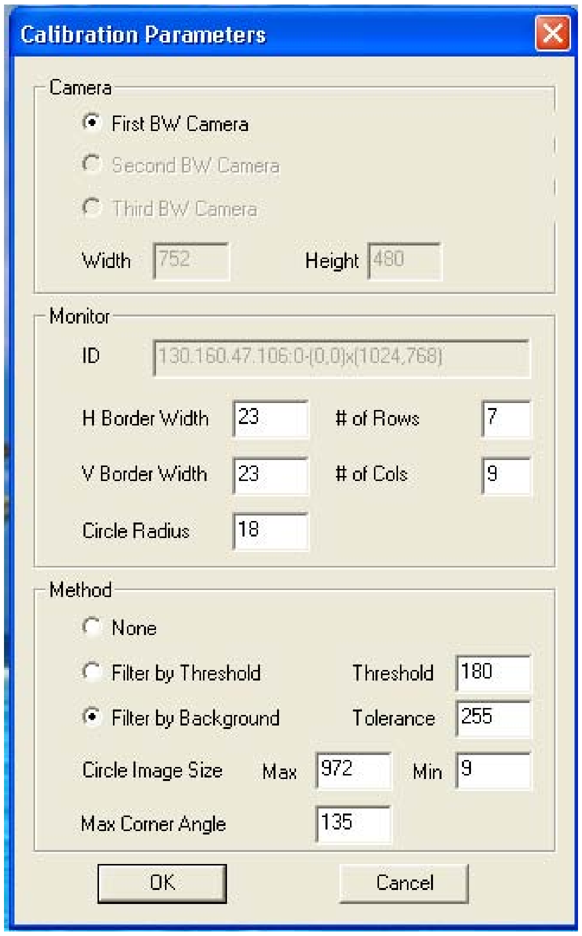
Camera calibration can be done using filtering by threshold or filtering by background. Filtering by background works in more situations than filtering by threshold. The main advantage of filtering by threshold is that it takes less CPU time, but the difference is hardly noticeable by human beings. Currently filtering by threshold is phasing out and filtering by background prevails. One difficulty using filtering by background is how to choose the value for tolerance. In the past, a value of 20 for tolerance worked almost flawlessly with a soft screen. However, for a hard screen, the image intensity of a dot in the calibration pattern can be very weak just as in the auto configuration. Figure 2.15(a) shows an image of a dot matrix captured for calibration. It is very hard to see the dots. We first tried to use the method used in the auto calibration, i.e. applying SurveyDifference and then EstimateTolerance, but it did not work. We could only identify one peak instead of two in the auto calibration case. This is mainly because the dots are small in the captured image. Additionally there exists a significant portion of the pixels comprising the dot which are of a lower intensity than at the center of the dot itself.



(a) (b)

Figure 2.15: An image of a calibration dot matrix and the calibration result.

To find an appropriate tolerance, we use the binary search method. A user can specify a value for tolerance in the Calibration Parameters dialog box shown in Figure 2.16. If the value is less than 32, it will be the tolerance to be used exactly. If the value is 32 or above, it is the largest possible value for the tolerance, and an appropriate value between 0 and the specified value will be determined by the computer using binary search. Figure 2.16 shows the default value 255 for the tolerance that means the computer will find an appropriate tolerance between 0 and 255. The binary search method works fine. A user really does not need to change the default value. Figure 2.15(b) show the successful calibration result of applying the algorithm to the image shown in Figure 2.15(a).


Calibration Parameters

Camera

(●) First BW Camera
( ) Second BW Camera
( ) Third BW Camera

Width 752 Height 480

Monitor

ID 130.160.47.106:0-(0,0)x(1024,768)

H Border Width 23 # of Rows 7
V Border Width 23 # of Cols 9
Circle Radius 18

Method

( ) None
( ) Filter by Threshold Threshold 180
(●) Filter by Background Tolerance 255

Circle Image Size Max 972 Min 9

Max Corner Angle 135

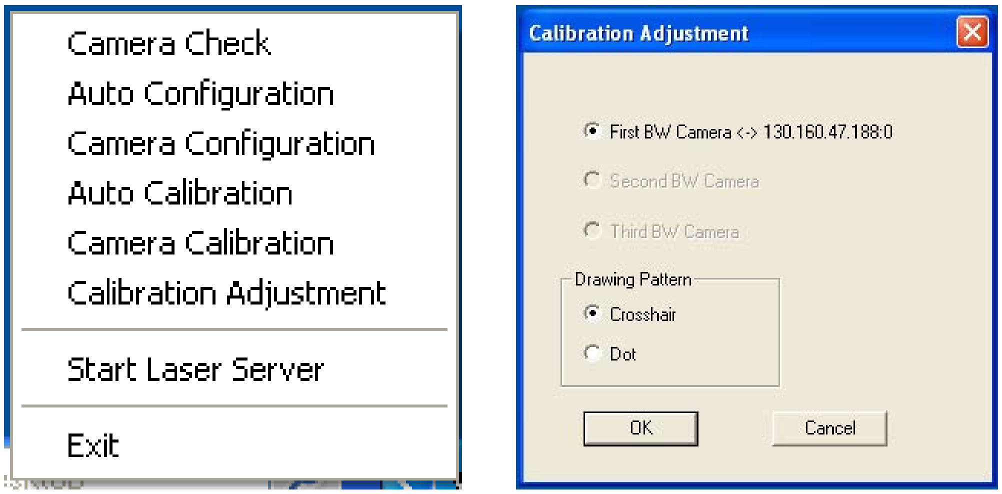
Figure 2.16: Parameters for camera calibration.

2.5.1.3 Handling hard screen refraction

Another issue with the hard screen thickness is the images seen from the front and rear will be different. This is especially true when the camera is placed either above or below the screen to avoid the specular reflection from the screen. In this setting, even if an LED pen aims exactly at a spot on the diffusion layer within the screen, the detected position will be either higher or lower depending on whether the camera is placed below or above the screen since the LED dots and the calibration dots are originating from opposite sides of the screen. To address this issue, we have introduced an optional step called Calibration Adjustment especially for hard screens.

Calibration Adjustment

Just like the calibration, calibration adjustment needs to be done for each screen section/camera, and it can be done only after the screen/camera is calibrated. In addition, it has to be done manually. Figure 2.17 shows the new Calibration Adjustment menu item at the left and the Calibration Adjustment dialog box at the right after the Calibration Adjustment menu item is selected. A user can ask the computer to draw either a crosshair or a dot at the center of the screen. The user can then use an LED pen to click the crosshair or the dot. After clicking, a pair of offsets in X and Y directions will be recorded. By using the offsets, the LED dot plane is shifted to match the calibration dot plane. Our test results shows with the offsets, the LED pen can be accurately tracked. Please note after the calibration, the existing offsets will be reset be zero. If the computer can't track the LED pen correctly after calibration, a new calibration adjustment step is needed.



Calibration Adjustment

[✓ First BW Camera <-> 130.160.47.188:0]
Second BW Camera
Third BW Camera

Drawing Pattern
[✓ Crosshair]
Dot

Figure 2.17: The menu item and dialog box for calibration adjustment.

2.5.2 Laser/LED Server for multiple Data Walls

The enhanced Laser/LED server software can deal with multiple Data Wall systems. Figure 2.18 illustrates the Laser/LED Server is connected by two Laser/LED clients. However, based on the use interface designed for the previous Laser/LED Server, the maximum number of the cameras that can be used by the Laser/LED Server is three. To overcome this limit, we redesigned the use interface in which we use a list to store all the available cameras. Therefore all the camera-related operations will have a new dialog box.

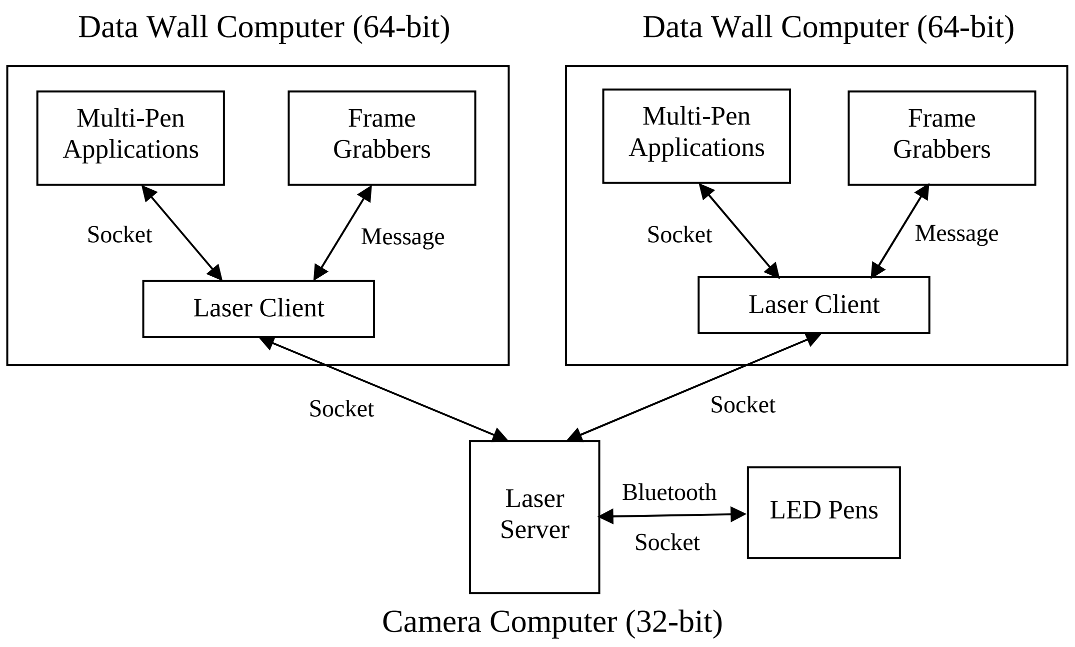
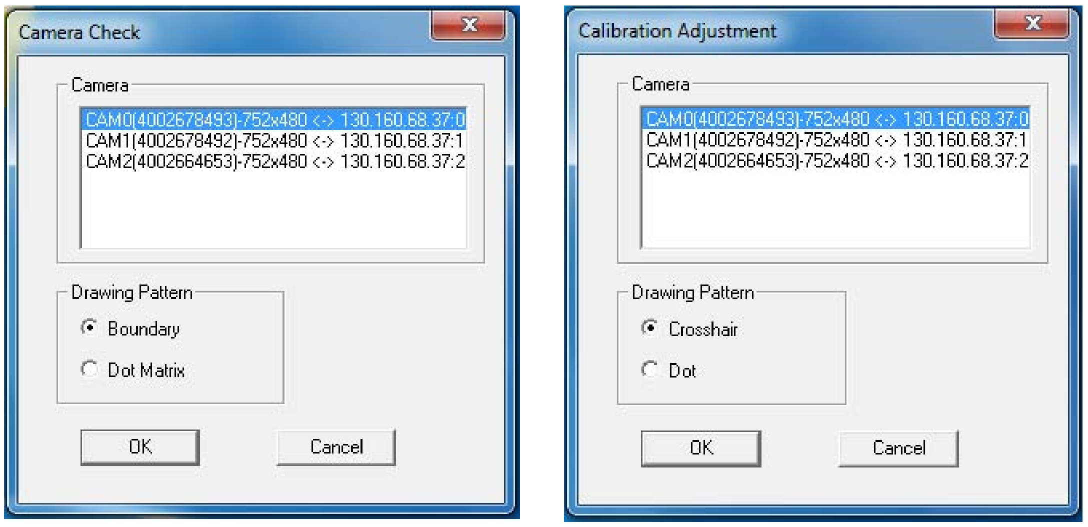


Figure 2.18: The communication among different software components.

The camera selection for “Camera Check” and “Calibration Adjustment” are similar. Figure 2.19 shows the dialog boxes for selecting a camera for these two operations.


Camera Check

Camera

```
CAM0(4002678493)-752x480 <-> 130.160.68.37:0
CAM1(4002678492)-752x480 <-> 130.160.68.37:1
CAM2(4002664653)-752x480 <-> 130.160.68.37:2
```

Drawing Pattern

[✓ Boundary] Dot Matrix

Calibration Adjustment

Camera

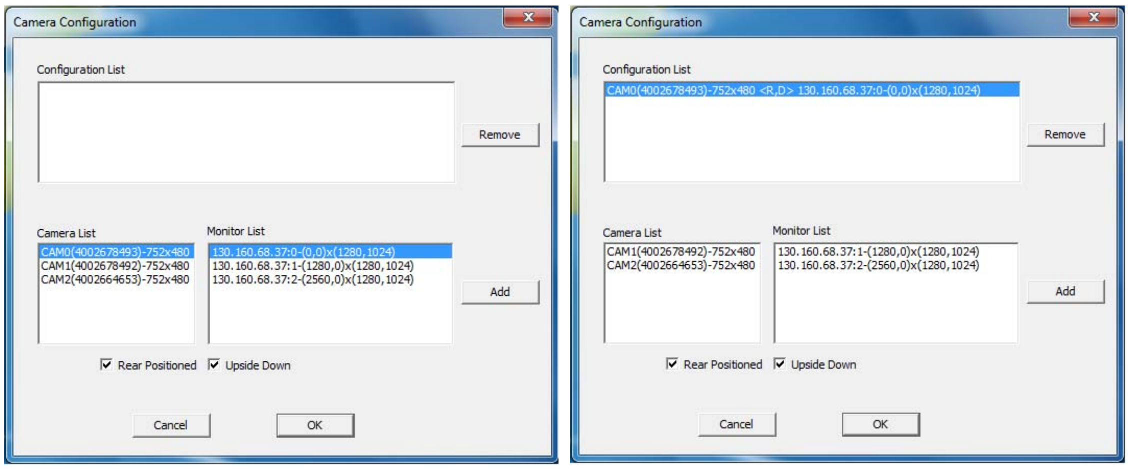
```
CAM0(4002678493)-752x480 <-> 130.160.68.37:0
CAM1(4002678492)-752x480 <-> 130.160.68.37:1
CAM2(4002664653)-752x480 <-> 130.160.68.37:2
```

Drawing Pattern

[✓ Crosshair] Dot

Figure 2.19: Dialog boxes for Camera Check and Calibration Adjustment.

The dialog box for “Camera Configuration” is a little complicated. It consists of three lists: the camera list, the monitor list and the configuration list as shown in Figure 2.20. The left side of Figure 2.20 shows nothing in the configuration list, three cameras in the camera list, and three monitors in the monitor list. It also shows one camera and one monitor are selected. If the “Add” button is hit at that time, the selected camera will be configured with the selected monitor. The new configuration will show up in the configuration list, and the corresponding camera and monitor will disappear from the respective lists as shown at the right side of Figure 2.20. At the right side of Figure 2.20, a configuration is selected. If the “Remove” button is hit, the selected configuration will disappear from the configuration list, and its corresponding camera and monitor will be returned to the respective lists.



Configuration List

Camera List
CAM0(4002678493)-752x480
CAM1(4002678492)-752x480
CAM2(4002664653)-752x480

Monitor List
130.160.68.37:0-(0,0)x(1280,1024)
130.160.68.37:1-(1280,0)x(1280,1024)
130.160.68.37:2-(2560,0)x(1280,1024)

☑ Rear Positioned ☑ Upside Down

Configuration List
CAM0(4002678493)-752x480 <R,D> 130.160.68.37:0-(0,0)x(1280,1024)

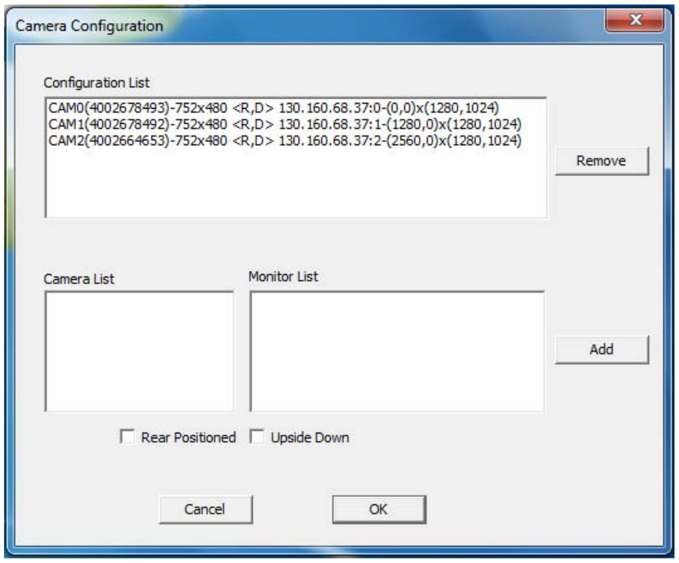
Camera List
CAM1(4002678492)-752x480
CAM2(4002664653)-752x480

Monitor List
130.160.68.37:1-(1280,0)x(1280,1024)
130.160.68.37:2-(2560,0)x(1280,1024)

☑ Rear Positioned ☑ Upside Down

Figure 2.20: Dialog boxes for Camera Configuration.

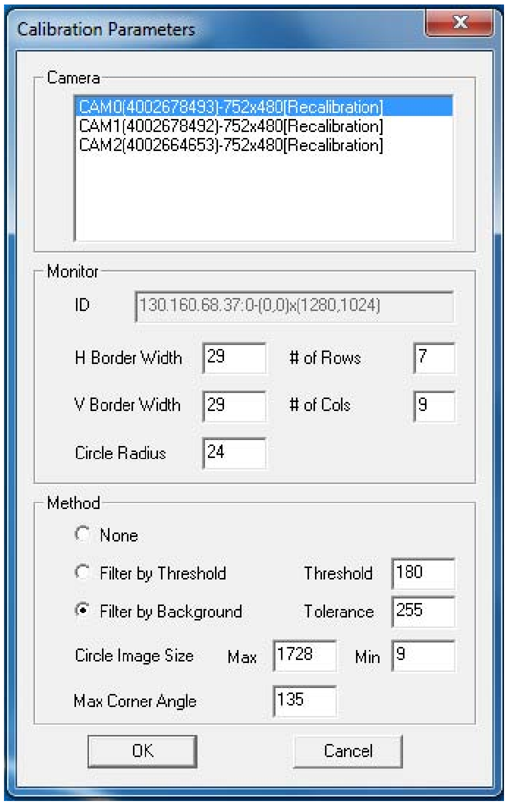
As before, the dialog box for Camera Configuration can be used to check the result of Auto Configuration. Figure 2.21 shows the result after a successful automatic configuration. For each configuration, there are two letters between the camera and the monitor. The first letter is either ‘R’ for a rear-positioned camera or ‘F’ a front-positioned camera. The second letter is either ‘U’ for an upside-up camera or ‘D’ an upside-down camera. Figure 2.22 shows the dialog box for selecting a camera for camera calibration.



CAM0(4002678493)-752x480 <R,D> 130.160.68.37:0-(0,0)x(1280,1024)
CAM1(4002678492)-752x480 <R,D> 130.160.68.37:1-(1280,0)x(1280,1024)
CAM2(4002664653)-752x480 <R,D> 130.160.68.37:2-(2560,0)x(1280,1024)

☐ Rear Positioned ☐ Upside Down

Figure 2.21: Check result of Auto Configuration using Camera Configuration.



CAM0(4002678493)-752x480[Recalibration]
CAM1(4002678492)-752x480[Recalibration]
CAM2(4002664653)-752x480[Recalibration]

ID 130.160.68.37:0-(0,0)x(1280,1024)

H Border Width	29	# of Rows	7
V Border Width	29	# of Cols	9
Circle Radius	24		

None

Filter by Threshold — Threshold 180

[✓ Filter by Background] — Tolerance 255

Circle Image Size Max 1728 Min 9

Max Corner Angle 135

Figure 2.22: Dialog box for Camera Calibration.

2.5.3 Communication with Bluetooth LED pens

The enhanced Laser/LED Server software can handle both regular and customized LED pens. Since the customized LED pens communicate with The Laser/LED Server using Bluetooth, they are also called Bluetooth LED pens.

2.5.3.1 Service sdvertising for Bluetooth LED pens

We added the service advertising for Bluetooth LED pens. Specifically we implemented the advertisement of the LED pen service within the Laser/LED Server, and the search for the LED pen service within a Bluetooth LED pen.

Advertising the LED Pen Service

The advertisement of the LED pen service is done by the Laser/LED Server. When the Laser/LED Server starts, it will try to find a Bluetooth adaptor. If the Laser/LED Server finds the adaptor, the Laser/LED Server will find an available channel to be used for the LED pen service. The service advertisement is done manually. For that purpose, we added the "Start Advertising" menu item to the Laser/LED Server menu as shown in the left side of Figure 2.23. Once the advertisement is started, the menu shows the "Stop Advertising" menu item as shown in the right side of Figure 2.23. In this way, the advertisement can be stopped manually. Therefore the time period for the service advertisement is totally controlled by a user.

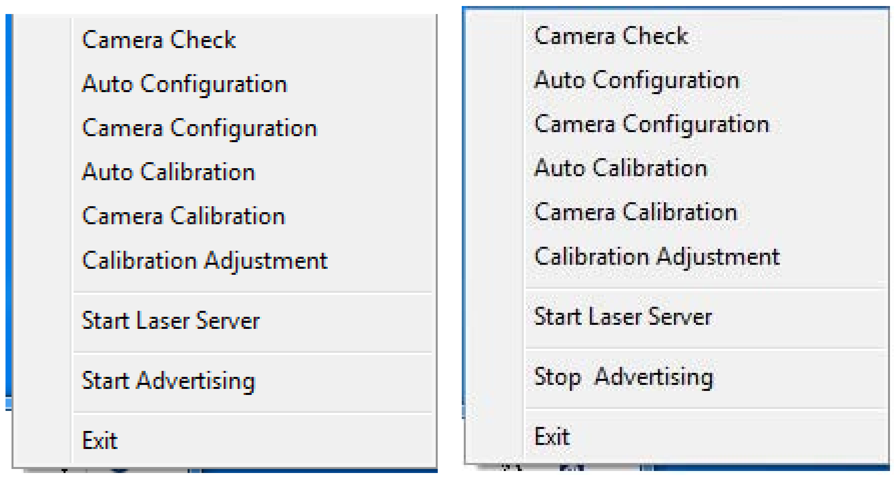


Figure 2.23: Start and stop the LED pen service advertising.

Each Bluetooth service is identified by a unique GUID. To advertise a Bluetooth service, two data structures have to be filled out first. The first is the `CSADDR_INFO` structure that contains all the information that a LED pen needs to connect to the server, including the Bluetooth adaptor address and the channel used for the service. The second is the `WSAQUERYSET` structure that includes the information for service discovery and device inquiry. The unique service GUID is set to the `lpServiceClassId` member of `WSAQUERYSET`. Finally `WSASetService` is called with the `RNRSERVICE_REGISTER` to advertise the service through the SDP (Service Discovery Protocol) server. To stop the advertisement, a similar procedure is to be followed, but `WSASetService` is called with the `RNRSERVICE_DELETE`.

Searching for the Service

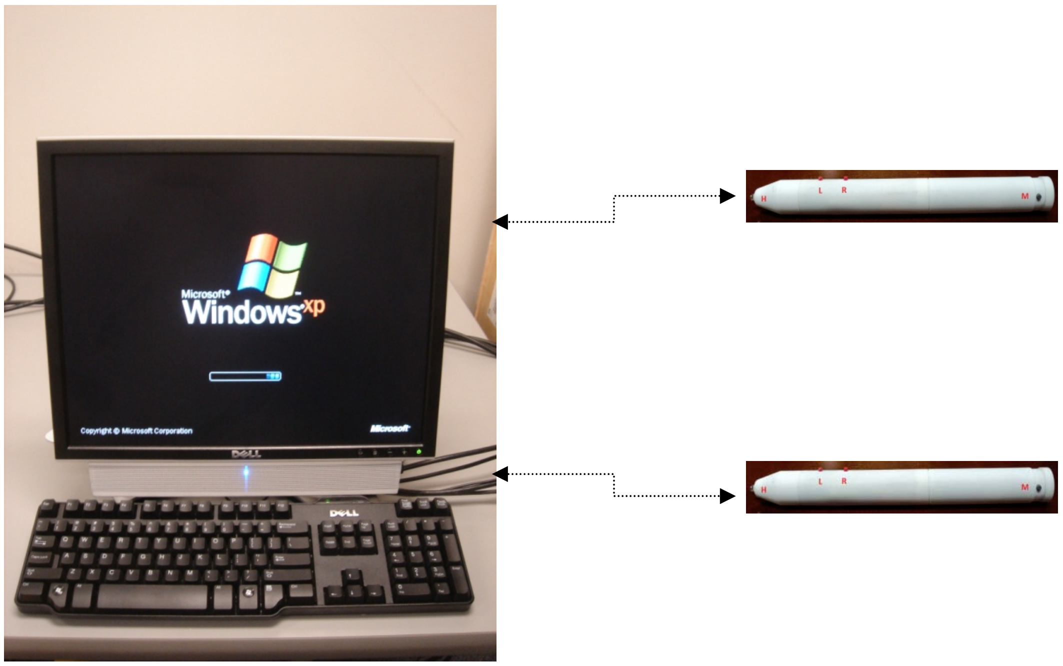
If the Laser/LED Server is advertising the LED pen service, a Bluetooth LED pen is able to discover the service by searching. Within a LED pen, the service discovery is written in C with BlueZ. Two header files `sdp.h` and `sdp_lib.h` in the BlueZ package include all necessary data structures and function prototypes for the SDP (Service Discovery Protocol).

The process of searching for the LED pen service involves two steps: detecting all nearby devices and connecting to each of those devices to search for the desired service. To make a list of all nearby devices, we use `hci_inquiry()` to run a HCI (Host/Controller Interface) inquiry to scan all nearby Bluetooth devices.

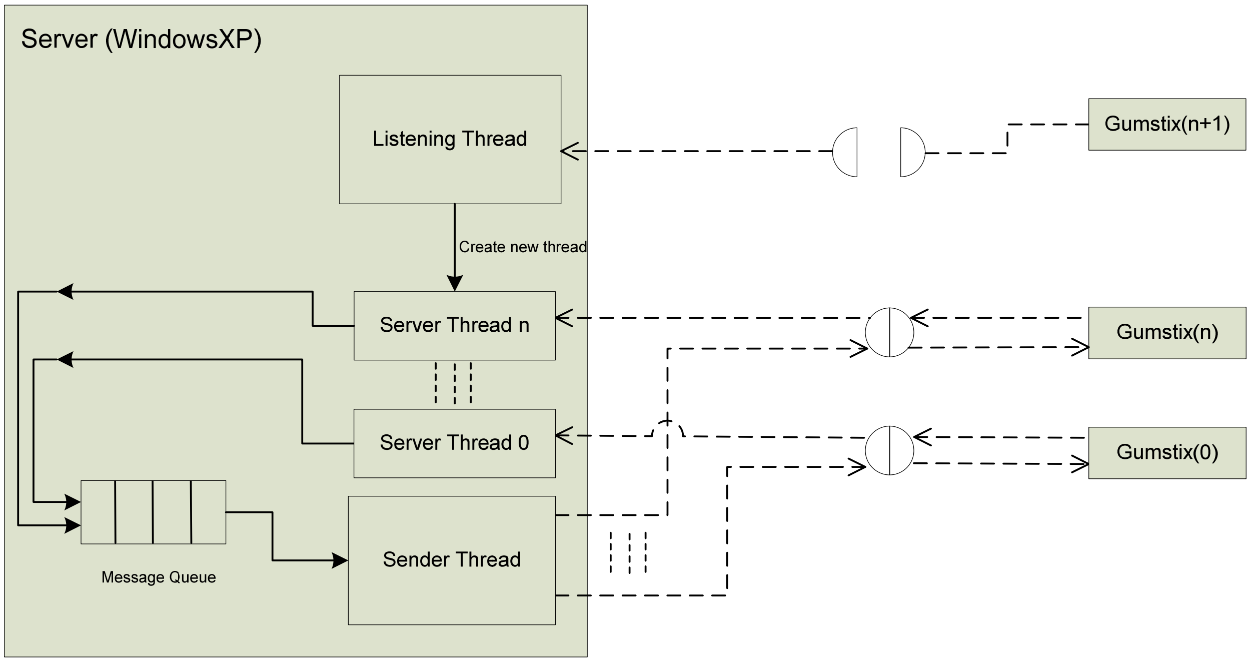
For each device in the list, we first make a connection to it using `sdp_connect(),` then search for a list of service records that have the specified UUID (the same as GUID in Windows) using `sdp_service_search_attr_req()`. If the search returns a list of service records, we have found the device. Then for each service record, get a list of protocol sequences using `sdp_get_access_protos()`. For each protocol sequence, get a list of protocols. Finally, for each protocol, get a list of attributes. To obtain the channel number of the LED pen service, we need to find the port number protocol attribute in the RFCOMM protocol entry. Once the LED pen has found the device and the channel number, the LED pen can make a connection to the server using the Bluetooth address of the device and the channel number.

2.5.3.2 Establishing the communication

Figure 2.24 illustrates the communication between the Laser/LED Server running Windows XP and the Gumstix running Linux via Bluetooth. Figure 2.24(a) shows multiple LED pens can simultaneously communicate with the Laser/LED Server. Figure 2.24(b) illustrates how the communication works.



(a)



(b)

Figure 2.24: Communication between the Laser/LED server and the Gumstix.

The laser/LED server runs Windows XP. The Microsoft® Bluetooth stack running on Windows XP extends its API from Windows Socket 2 API. Therefore Bluetooth programming at the server side is similar to socket programming.

As Figure 2.24(b) shows, when a new Bluetooth connection is established between the server and a Gumstix, the listening thread creates a new server thread to serve the Gumstix. For some requests, a reply is needed. In this case, the server thread will save the reply in the message queue. A sender thread is created to forward a reply to the corresponding client. The reason the replies are processed by only one sender thread is to synchronize all the replies to different clients. For example, the sender may stipulate that only one reply can be sent out during certain time period.

At the Gumstix, the switch state is constantly monitored. Once the switch is pressed, the LED lamp is not turned on immediately. Instead the “REQ_ON” request will be sent from the client to the server. The server thread will put the “REP_ON” reply into the message queue, and the sender thread will forward the “REP_ON” reply to the client that makes the corresponding “REQ_ON” request. Once the client receives “REP_ON” reply, it turns on the LED lamp. Therefore the LED lamp is turned on after a round trip. We measured the time for a round trip is around 50 milliseconds. At the server side, we also have the cameras to monitor the LED dots, at 60 frames per second for example. Assume that the sender thread sends out at most one REP_ON” reply each frame. When a new LED dot is detected within a frame, we can know which Gumstix lights the new LED dot.

The LED pen can be in either soft-screen mode or hard-screen mode, indicated by the state of the dot on the LCD display on the back of the pen. If the dot is lit, the pen is in the soft-screen mode, and if it isn’t, the pen is in the hard-screen mode. The tactile switch at the tail (referred to as the Mode switch/button, or M for short) can do the following three things.

- If the M button is pressed for less than one second, it will circle among 10 digit modes (‘0’->’1’->’2’->’3’->’4’->’5’->’6’->’7’->’8’->’9’->’0’), and the LCD shows the current digit mode. A digit mode can be used to represent something like pen colors and/or pen widths.

- If the M button is pressed for more than one second, but less than five seconds, it will change between the soft-screen and the hard-screen modes. The dot of the LCD shows the current screen mode.

- If the M button is pressed for more than five seconds, the pen will check whether the pen’s connection to the camera computer is still good. If it isn’t, the pen will try to reconnect to the camera computer via Bluetooth. (The pen first tries to establish a connection with the camera computer when the pen is switched on.)

From here on, we refer to the momentary switch inside the pen as the Head switch/button (H for short), the first tactile switch on the body as the Left switch/button

(L for short), and the second tactile switch on the body as the Write switch/button (W for short). When the pen is in the hard-screen mode, the two tactile switches on the body (L and W) will act as the state change switches, and the momentary switch inside the pen (H) will serve as the action switch. The default state is ‘L’ (for simulating the left mouse button). The state changes follow the following rules.

- A short press (for less than one second) of the L switch will set the state to ‘L’ if the current state is not ‘L’, or to ‘Z’ (for simulating mouse movement without any button pressed) if the current state is ‘L’.
- A long press (for more than one second) of the L switch will change the state to ‘R’ (for simulating the right mouse button).
- A short press of the W switch will set the state to ‘W’ (for simulating the writing action), and a long press of the W switch will change the state to ‘D’ (for simulating the double-click action).
- When the H switch is pressed on, the LED will be turned on, and the corresponding action will be simulated based on the current state. Among the five states, ‘R’ and ‘D’ are temporary, which means the state will be automatically changed back to ‘L’ after their corresponding action has been simulated once. ‘L’, ‘Z’, and ‘W’ are permanent states.

When the pen is in the soft-screen mode, the L and W switches will act as the action switches, and the H switch will serve as the state change switch. The default state is still ‘L’. The state changes follow the following rules.

- A short press of the H switch will set the state to ‘L’ if the current state is not ‘L’, or to ‘Z’ if the current state is ‘L’. A long press of the H switch will change the state to ‘R’.
- When the L switch is pressed, the LED will be turned on, and the corresponding action will be simulated based on the current state (‘L’, ‘Z’ or ‘R’). Note ‘R’ is a temporary state while ‘L’ and ‘Z’ are permanent states.
- The W switch always simulates the writing action regardless of the current state.

2.5.4 Annotation and text entry

The Bluetooth LED pen has a switch called the W (for writing) switch. In both soft-screen and hard-screen modes, the LED pen can be in the ‘W’ state and perform the ‘W’ action. A ‘W’ action (i.e. turning on the LED while in the ‘W’ state) can be interpreted as annotation, text entry, or both. If it is interpreted as annotation, the digit mode ‘0’ through ‘9’ can be used to define the color and/or width of the pen. The pen in the ‘W’ state can also be used for text entry. We used Microsoft’s Ink Analyzer to recognize hand-writing. The recognized text will be associated with the pen that produces the ink. Therefore, the system knows who enters what text. In addition, we have implemented four gestures that corresponds four special characters: <Enter>, <Space>, <Backspace> and <Tab>, to help enter the text.

2.6 Laser/LED Client Software

There are not a lot of changes to the Laser/LED Client software. The Laser/LED Client software will receive the events from Laser/LED Server software as before, except that the event format is richer now. The only major difference is that the Laser/LED client can act as a server to serve multi-pen/cursor applications on the same machine. A multi-pen/cursor application can subscribe event services from the Laser/LED Client. Whenever the Laser/LED Client receives the events from Laser/LED Server, it will distribute the events to a multi-pen/cursor application that subscribes to its service.

3. Cursors/Mice without Borders

A mouse is usually limited within the computer to which it is attached. If you have two computers on the desk, you need two sets of mice and keyboards to interact with these two computers. In this project, we define a virtual computer as consisting of multiple physical computers, and use the same set of mouse and keyboard to interact with all the physical computers in a virtual computer. To achieve this, we developed a mouse filter driver as well as a keyboard filter driver. A mouse filter driver can be installed with a mouse. Once the filter driver is installed, we can intercept all the mouse events generated from the mouse. We also represent the mouse with a software cursor, and use the intercepted mouse events to drive the software cursor. With the help of socket communication between computers, we are able to move the software cursor between computers. In this way, the software cursor can interact with multiple computers. Figure 3.1 illustrates how the mouse filter driver is used for interaction within a virtual computer consisting of two physical computers.

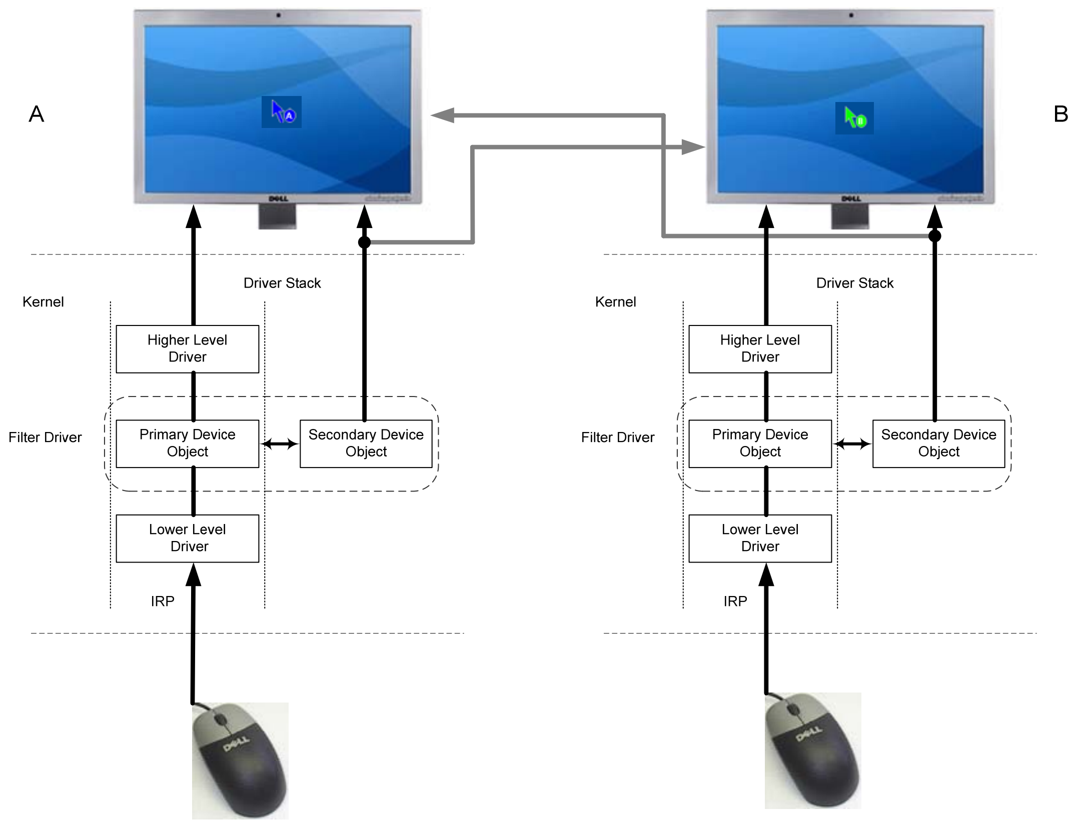


Figure 3.1: Mouse filter driver for multiple computer interaction.

3.1 Mouse/Keyboard Filter Drivers

A filter driver for an input device is the best way to capture input events and pass them to an application on a Windows platform. Usually an application sends custom device I/O control requests (IOCTLs) to an added filter driver to retrieve the input events. The filter driver is able to get the custom requests as long as it is the upper-most filter in the stack, because that is the one that handles the request first. If another filter driver is added on top of it, the custom requests could be rejected by the filter above it. To address this issue, we created a standalone named device object, called control device object, to handle communication with the application. For an application to access the named device object, we need to create a user-visible interface either by a symbolic link to the device object name or a unique GUID. The application can open the standalone device object via the symbolic link name or the unique GUID with the CreateFile function and use the DeviceIoControl function to issue custom IOCTLs that will be sent directly to the control device object. Figure 3.2 shows the filter driver with two device objects. Device object 1 (unnamed) in the figure is inserted in the driver stack. It filters IRPs from the top and captures input events from the bottom. Device object 2 (named) reads the captured input events captured by device object 1, and sends them out of kernel to an application.

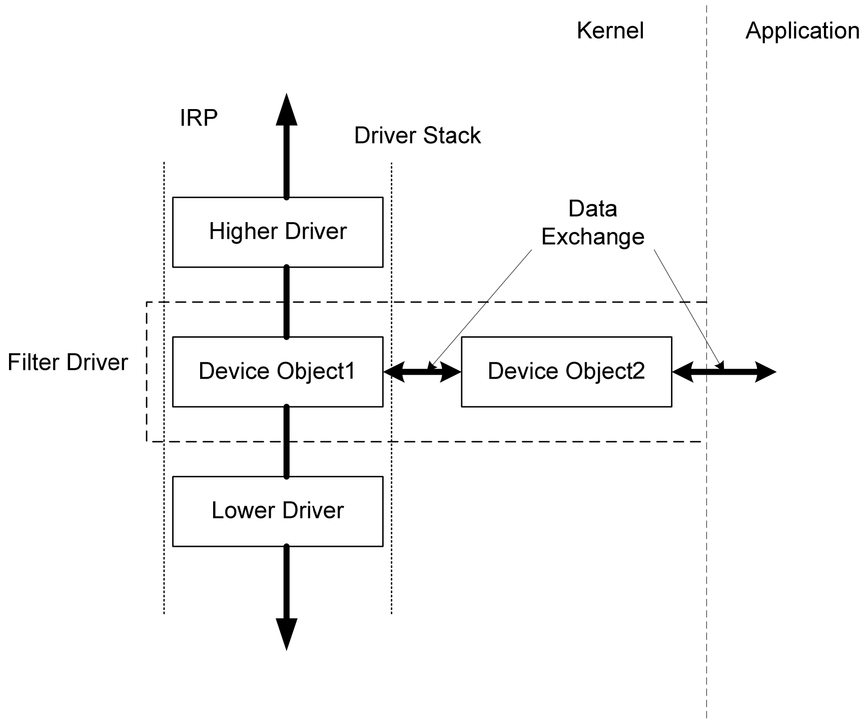


Figure 3.2: The filter driver with two device objects.

3.2 Mouse/Keyboard Filter Drivers Developed

The Cursor/Mice without Borders software consists of MultiCure.exe and mouse/keyboard filter drivers. The 32-bit MultiCure.exe can run on any type of Microsoft® Windows, but each Windows system requires different mouse/keyboard filter drivers. We have developed mouse/keyboard filter drivers for Windows XP, Vista, and 7. For Windows 7, we developed all the filter drivers. Table 3.1 lists all the filter mouse/keyboard drivers we have developed for Windows 7. We considered both PS/2 type (used mainly in laptops) and USB type. We also considered both 32-bit and 64-bit Windows. Each driver consists of three files: the driver itself (the .sys file), the coinstaller, and the setup .inf file. Our drivers are not digitally signed, which means "Disable Digital Signature Enforcement" must be selected during booting if you want to install or to use the 64-bit driver. Also the MultiCure.exe application must be run as the administrator.

Table 3.1: Mouse/keyboard filter drivers developed for Windows 7

Architecture	Port	Device	Installation Files
32-bit	USB	Mouse	moufiltr.sys, WdfCoInstaller01009.dll, Mouse_Win7_32_HID.inf
		Keyboard	kbfiltr.sys, WdfCoInstaller01009.dll, Keyboard_Win7_32_HID.inf
	PS/2	Mouse	moufiltr.sys, WdfCoInstaller01009.dll, Mouse_Win7_32_PS2.inf
		Keyboard	kbfiltr.sys, WdfCoInstaller01009.dll, Keyboard_Win7_32_PS2.inf
64-bit	USB	Mouse	moufiltr.sys, WdfCoInstaller01009.dll, Mouse_Win7_64_HID.inf
		Keyboard	kbfiltr.sys, WdfCoInstaller01009.dll, Keyboard_Win7_64_HID.inf
	PS/2	Mouse	moufiltr.sys, WdfCoInstaller01009.dll, Mouse_Win7_64_PS2.inf
		Keyboard	kbfiltr.sys, WdfCoInstaller01009.dll, Keyboard_Win7_64_PS2.inf

While our main focus was on Windows 7, we also developed mouse/keyboard filter drivers for both Windows Vista and Windows XP in the process. For 64-bit Windows Vista, we developed filter drivers for both USB mice and USB keyboards. For Windows XP, we developed all the filter drivers except for PS/2 mouse and keyboard filter drivers on 64-bit Windows XP.

3.3 Merging Events from LED Pens and Software Cursors

The developed environment allows two popular input devices: LED pens (including both Bluetooth and regular ones) and mice. These two popular devices can interact with any computer in the command and control room. For example, an LED pen can interact with any Data Wall screen in the room, and a mouse (with the help of the filter driver) can move out of its home computer to interact with any computer in the room. In this project, we have also unified the events generated by both pens and mice, and process these events in the same manner. As a result, these two types of input devices can interact with a computer in a uniform way. In particular, we have used a 32-bit integer called Channel and a 64-bit integer called PenId to identify an input device. Initially, we used these two integers to identify a Bluetooth LED pen. Each Bluetooth LED pen has a unique 48-bit Bluetooth address, and we use the 64-bit integer to represent the address. When the Bluetooth LED pen is connected to the Laser/LED Server, it has a nonnegative session ID, and we use the 32-bit integer to represent the session ID. When we add the regular LED pens, we assume the Channel of a regular LED pen is always -1, and its PenId is randomly assigned. A mouse with the filter driver installed is represented by a software cursor. We represent such a mouse with Channel being -2 and PenId being assigned in the MultiCur.ini configuration file. Currently the Laser/LED client can handle events from both LED pens and software cursors in the same way.

4. Java Package for Multi-pen/cursor Application Development

As shown in Figure 2.18, multi-pen/cursor applications can run within each Data Wall computer. The Laser/LED client can run a service to distribute the events received from the LED pens or software cursors to multi-pen/cursor applications through socket communication within the same computer. The multi-pen/cursor applications can be written in C++ or Java. To facilitate the development of multi-pen/cursor applications in Java, we developed a Java package called `LaserEventSubscriber`.

The `LaserEventSubscriber` package consists of three public classes: `LaserEvent`, `LaserFrame`, and `LaserEventThread`. The data members of the `LaserEvent` class are shown below.

```
public class LaserEvent {
      public static final int LASER_OFF   =0;
      public static final int LASER_ON    =1;
      public static final int LASER_MOVE  =2;

      int Channel;              // 4 bytes
      long LaserPenId;          // 8 bytes
      char Mode;                // 2 bytes, '0' through '9'
      char BtnMode;             // 2 bytes, 'Z', 'L', 'R', ‘D’ or 'W'
      int Flag;                 // LASER_OFF, LASER_ON, or LASER_MOVE
      int X;                    // screen coordinates X and Y
      int Y;
}
```

For a regular LED pen, `Channel` is -1, `LaserPenId` is a randomly assigned number, `Mode` is ‘0’ and `BtnMode` is ‘L’. For a Bluetooth LED pen, `Channel` is a non-negative integer that does not change during a connection between the Bluetooth LED pen and the Laser/LED Server. `LaserPenId` is the unique Bluetooth address of the LED pen, `Mode` can be ‘0’ through ‘9’ indicating the mode shown on the back of the Bluetooth LED pen and set by the M button. `BtnMode` is ‘L’ for simulating left mouse button, ‘R’ for simulating right mouse button, ‘Z’ for simulating no mouse button, ‘D’ for double-clicking, and ‘W’ for writing. For an event generated by a software cursor, `Channel` is -2 and `LaserPenId` is determined in the MultiCur.ini configuration file.

The data members of the `LaserFrame` class are defined below. It consists of the time in the frame number (`NumFrame`), the number of dots detected at that frame (`EventCount`), and an array of laser events (`LaserEvents`), with each laser event (`LaserEvent`) corresponding to a detected dot.

```
public class LaserFrame {
      int NumFrame;
      int EventCount;
      // LaserEvent[] LaserEvents;
      ArrayList<LaserEvent> LaserEvents;
}
```

Finally, the `LaserEventThread` class is an `abstract` class that extends the `Thread` class. There are two constructors. One uses the default port of 10006 to connect to the Laser client while the other uses the port number provided in the argument. Both throw an `IOExeption` if the connection fails. The `run` method enters a loop in which it first reads a laser frame, and then calls the `abstract` method of `ProcessFrame` to process the laser frame just read. The actual `ProcessMethod` method needs to be defined in the extended class of `LaserEventThread`, depending on how the programmer wants to process each laser frame.

```
public abstract class LaserEventThread extends Thread {
      final static int PORT_DIST_COM = 10006;
      final static int LASER_EVENTS = 7;

      private Socket socket = null;
      private InputStream input = null;
      // private OutputStream output = null;

      public LaserEventThread() throws IOException {/*code omitted*/}
      public LaserEventThread(int port) throws IOException
                                                {/*code omitted*/}
      public void run() {     /*code simplified*/
            boolean bContinue=true;
            while (bContinue) {
                  int type = ReadInt();
                  switch (type) {
                  case LASER_EVENTS:
                        LaserFrame frame = ReadLaserFrame();
                        bContinue=ProcessFrame(frame);
                        break;
                  }
            }
      }

      public abstract boolean ProcessFrame(LaserFrame frame);
}
```

To use the package, a programmer first needs to extend the `LaserEventThread` class, and in the extended class, define a concrete `ProcessFrame` method to process a laser frame. Then the programmer can create an instance of the extended class, which will make a connection to the Laser client, and call the `start` method to start the thread that will read and process each laser frame received from the Laser/LED client.

The following code shows how to use the `LaserEventSubscriber` package. The `MyLaserEventThread` class is first defined to extend the

LaserEventThread class of the LaserEventSubscriber package. An instance of MyLaserEventThread is then created. If the instance is created successfully, a connection is made to the Laser client successfully. Finally the start method of the instance is called to start a thread in which the ProcessFrame method of the MyLaserEventThread class will be called to process each laser frame received from the Laser/LED client.

```
class MyLaserEventThread extends LaserEventThread {
      MyLaserEventThread() throws IOException {
            super();
      }
      MyLaserEventThread(int port) throws IOException {
            super(port);
      }
      public boolean ProcessFrame(LaserFrame frame) {
            /*code omitted*/
            return true;
      }
}

MyLaserEventThread laser_thread=null;
try {
      laser_thread = new MyLaserEventThread();
}
catch (IOException ex) {
      logfile.println("Unable to connect to the Laser client.");
}
if (laser_thread!=null) laser_thread.start();
```

To illustrate how to use the LaserEventSubscriber Java package, we have developed a multi-pen/cursor application called WhiteBoard. Figure 4.1 shows a snapshot of WhiteBoard in action. In WhiteBoard, we subscribe to the events generated by both pens and software cursors. We also handle the events generated by the system cursor, and merge them with the events from subscription. To merge the events, we create a frame queue protected by a mutual exclusion mechanism. The concrete ProcessFrame method of the extended LaserEventThread class really just puts the frame into the queue. For each event generated by the system cursor, we construct a frame and deposit it into the queue. We also create a thread that retrieves the frame from the queue, and actually process the frame.

WhiteBoard is a truly multi-pen/cursor application in which each input device has its own representation. Each input device can save the pen color and pen width in its own representation. When the input device is dragged within the client area, the saved pen color and pen width will be used to trace the movement. If no conflict arises, two input devices can resize the window simultaneously. For example, if one input device is resizing the window using the top-left resizing handle, another device can resize the same window using the bottom-right handle.

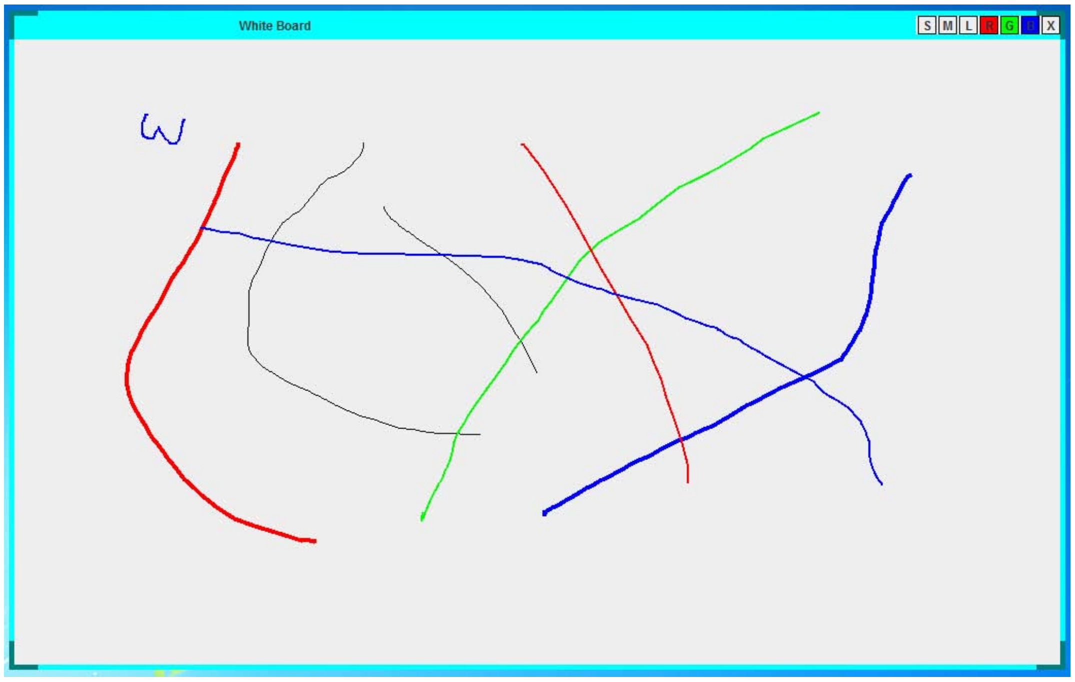


Figure 4.1: A multi-pen/cursor application in Java.

5. Copy and Paste between Two Computers

A software cursor corresponds to an actual mouse and is uniquely identifiable. A Bluetooth LED pen is also uniquely identified by its Bluetooth address. A user can copy an item from a computer and store it with the software cursor. Then the user can move the software cursor to another computer and paste the stored item to the second computer. With a Bluetooth LED pen, a user can user can copy an item form one Data Wall and save it with the pen. Then the user can move to another Data Wall, and paste the saved item to the other Data Wall. To be consistent with the software cursor, we create a hidden software cursor on each Data Wall for a Bluetooth LED pen. The copied item is really saved with the hidden software cursor instead of the actual pen. Therefore the handling of copy and paste is the same for both software cursors and Bluetooth LED pens.

5.1 Keeping Clipboard with Software Cursor

In today's operation system of conventional applications, the clipboard is associated with a computer so that multiple applications running on the same computer can exchange the information through the shared clipboard. In our implementation, a user can operate a software cursor and a clipboard can be associated with a software cursor. The clipboard associated with a computer is referred to as a *machine clipboard*, and the one with a cursor is called a *cursor clipboard* hereafter. Since we made no changes to the operating system or the applications running on the operating system, our cursor clipboard implementation is still based on the machine clipboard. Whenever the machine clipboard is changed, its new content may be copied to the cursor clipboard of the software cursor active on the machine. To paste the content of a cursor clipboard to an application, the cursor clipboard content has to be copied to the machines clipboard first. When a clipboard is kept with a software cursor, different software cursors can have different clipboards. Figure 5.1 illustrates three software cursors with three different clipboards.

To monitor whether there is any change to the machine clipboard, we added the software cursor as a clipboard viewer using the *SetClipboardViewer* function. Before the software cursor is destroyed, it needs to be removed from the chain by calling the *ChangeClipboardChain* function. When the software cursor is in the chain, it must process the clipboard messages *WM_CHANGECBCHAIN* and *WM_DRAWCLIPBOARD*, and call the *SendMessage* function to pass these two messages to the next window in the clipboard viewer chain.

Whenever there is a change to the machine clipboard, the *WM_DRAWCLIPBOARD* message will be sent to the clipboard viewer. The clipboard viewer can retrieve the content of the machine clipboard and try to copy it to the cursor clipboard if it determines that the machine clipboard was not changed by itself using the *GetClipboardOwner* function. Whenever a user clicks a software cursor on a computer, the software cursor will be actively associated with the corresponding machine if the association does not exist before. At the same time, the content of the cursor clipboard will be sent to the corresponding machine clipboard. After that, the paste operation will use the new machine clipboard content which is the same as the cursor clipboard of the active software cursor.

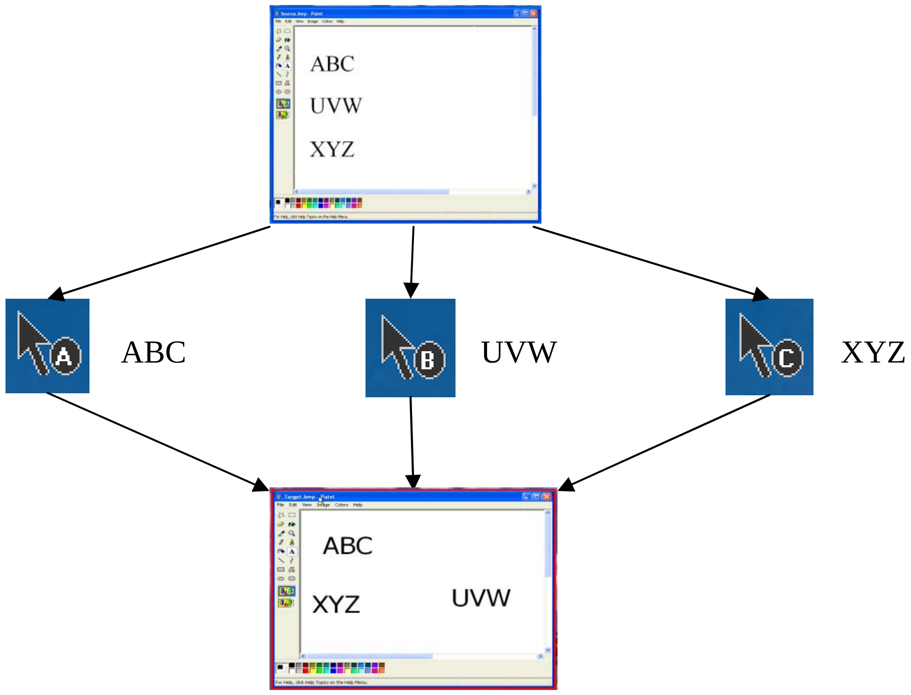


Figure 5.1: Three software cursors with three different clipboards.

5.2 Data Structure for Cursor Clipboards

To perform copy and paste operations between two computers, we define the data structure for cursor clipboards as shown in Figure 5.2. The *count* field indicates the number of formats for the clipboard. For each format, it consists of the format name, the length of the data, and the data. The currently supported formats include text (CF_TEXT, CF_UNICODETEXT, CF_OEM_TEXT, CF_DSPTEXT), files/directories (CF_HDROP), and an image (CF_DIB and CF_DIBV5). The *CF_HDROP* format is for file copy and paste, and its data field is a list of file and directory names. Each file or directory is NULL-terminated and the list is also NULL-terminated. Therefore, the *CF_HDROP* data is composed of multiple NULL-delimited strings that are ended with a double NULL terminator. The *IP_Addr* field indicates the clipboard source. Later, the target machine can use the *IP_Addr* and *Port* fields to retrieve the files from the source machine. For the other implemented data formats, the data field stores the actual data such as a text or an image.

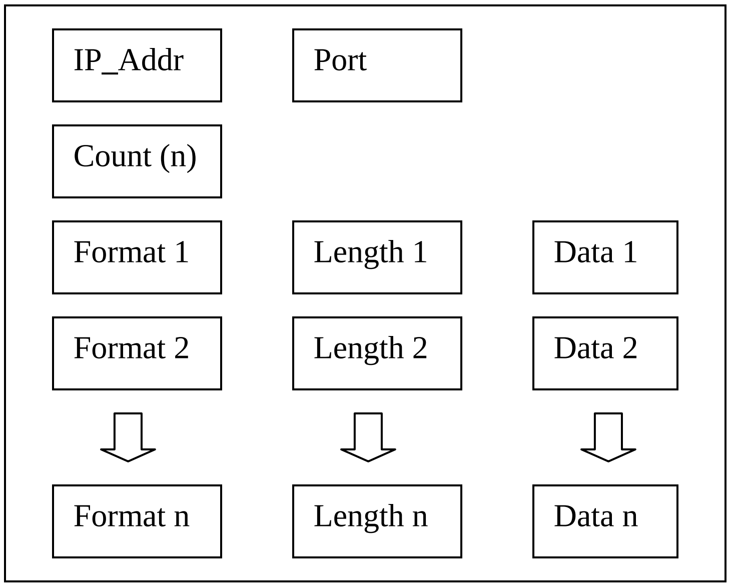


Figure 5.2: The data structure for cursor clipboards.

Since only a list of file and directory names is stored in the clipboard for the *CF_HDROP* format, the handling of the *CF_HDROP* format is more complicated. Whenever there is a change to the machine clipboard, the *WM_DRAWCLIPBOARD* message will be sent to all the clipboard viewers. Our clipboard viewer will enumerate all the formats using *EnumClipboardFormats*. If the format is of *CF_HDROP*, we use the *GetClipboardData* function to get the *HDROP* handle, and then use the *DragQueryFile* function to get all the path names from the *HDROP* handle. The path names along with the machine IP address and the port number of the file transfer service will be packed as shown in Figure 5.2 and stored with the software cursor. For example, when a user selects a list of files and directories and performs a copy operation in Windows Explorer, the path names for these files and directories along with the machine IP address and the service port number will be saved into the cursor clipboard under the *CF_HDROP* format.

Whenever the software cursor is associated with a new computer, the software cursor needs to set the cursor clipboard into the machine clipboard. However, each association does not always lead to a paste operation. If there is no paste operation, there is no need to transfer files. To be efficient, we adopted delayed rendering. Specifically, we use *SetClipboardData* to set the *CF_HDROP* data to NULL. Whenever there is a paste operation, the *WM_RENDERFORMAT* message with the *CF_HDROP* format will be sent to the owner. The owner then can make a connection to the server using the IP address and the service port number. Once the connection is established, the owner (i.e. the client) can send the pathnames one by one to request the files. The received files will be saved in a temporary directory. At the same time, all the pathnames are mapped to reflect their new locations under the temporary directory. The mapped path names prefixed by a *DROPFILES* structure will then be set into the *CF_HDROP* format of the machine clipboard using *SetClipboardData*. At this time, the corresponding files will be transferred from the temporary directory into the target directory. Therefore, if a user performs a paste operation in Windows Explorer, the files recorded in the corresponding software cursor will be transferred from the source to the target machine and directory. This is how the files are pasted to the target machine and directory using the software cursor.

Whenever there is an owner change to the machine clipboard, made by the new owner using the *EmptyClipboard* function, the *WM_DESTROYCLIPBOARD* message is sent to the previous owner. At that time, the previous owner can delete the files under the temporary directory.

6. Frame Grabbing

Desktop sharing is a useful operation in a command and control room. For example, if a person has found something interesting on his or her individual computer, that person may want to share the interesting thing with the people in the room by showing his/her desktop on a group-use computer. In the past, we have used the VNC (Virtual Network Computing) based and the mirror driver based approaches to implement desktop sharing. In this project, we use Epiphan's DVI2USB™ frame grabbers to achieve desktop sharing. There are several advantages of using DVI2USB™ frame grabbers to perform desktop sharing. First, the source computer can run any operating system as long as it generates DVI signals for display. Second, it is fast. It is done by hardware to capture the desktop and to transport the captured frames to the destination. Finally, there is no network connection required between source and destination computers.

6.1 Implementation

Our implementation allows multiple frame grabbers to grab and display the frames simultaneously, three threads have been created for each frame grabber, responsible for connection check, frame grabbing and frame displaying respectively. Figure 6.1 illustrates the relationship among these three threads.

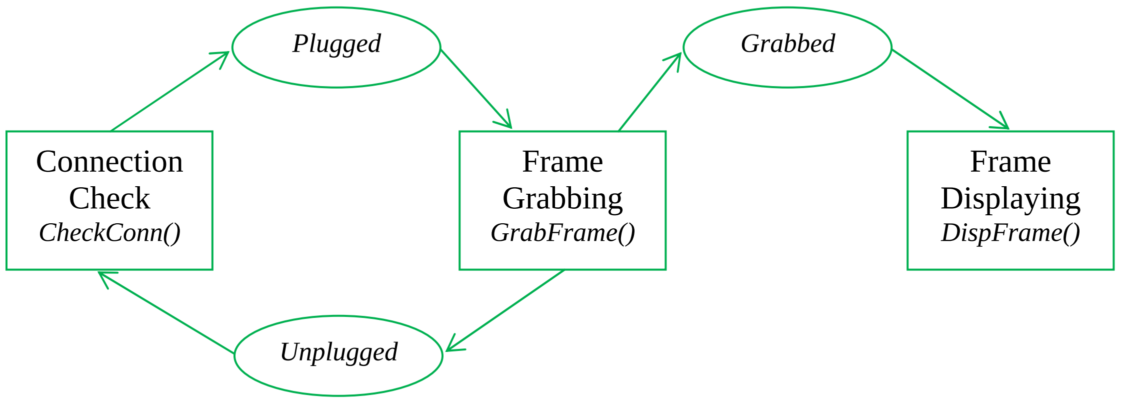


Figure 6.1: The relationship among the three threads created for each grabber.

The Connection Check thread executes the *CheckConn* function. It runs when there is no device connected to the DVI2USB™ grabber. Specifically it checks whether a device is connected to the DVI2USB™ grabber. If it detects such a connection, it wakes up the Frame Grabbing thread using the *Plugged* event, and then it sleeps on the *Unplugged* event. The Frame Grabbing thread executes the *GrabFrame* function. When there is a device connected to the DVI2USB™ grabber, the Frame Grabbing thread grabs the frames from the connected device one at a time. A frame can be grabbed with and without compression using the `V2U_GRABFRAME_FORMAT_CBGR24` and

`V2U_GRABFRAME_FORMAT_BGR24` formats respectively. If a frame is grabbed with compression, the grabbed frame needs to be decompressed using the *v2udec_decompress_frame* function. Once a frame is grabbed, it wakes up the Frame Displaying thread using the *Grabbed* event, and it starts to grab the next frame. The Frame Displaying thread executes the *DispFrame* function. It just displays the grabbed frame using *SetDIBitsToDevice* for the one-to-one scale and *StretchDIBits* for the reduced scale. It goes to sleep again on the *Grabbed* event after displaying. When the Frame Grabbing thread is not able to grab a frame, it is assumed that the device is disconnected from the DVI2USB™ grabber. At that time, it wakes up the Connection Check thread through the *Unplugged* event, and then sleeps on the *Plugged* event. When a device is connected to the DVI2USB™ grabber, it will be automatically detected by the Connection Check thread.

After the Frame Grabbing thread has grabbed a frame, it asks the Frame Displaying thread to display the grabbed frame, and it immediately begins grabbing the next frame. In order for the Frame Displaying thread and the Frame Grabbing thread to run simultaneously, double buffers are used. Figure 6.2 illustrates how double buffers are used for simultaneous grabbing and displaying. When the *IsZeroAcitve* flag is *TRUE*, the Frame Grabbing thread grabs the frame into the *ImageBits0* buffer while the Frame Displaying thread displays the previous grabbed frame stored at the *ImageBits1* buffer. After the Frame Grabbing thread has grabbed the frame to *ImageBits0*, it changes the *IsZeroAcitve* flag to *FALSE*. At that time, the Frame Grabbing thread grabs the frame into the *ImageBits1* buffer while the Frame Displaying thread displays the frame at the *ImageBits0* buffer.

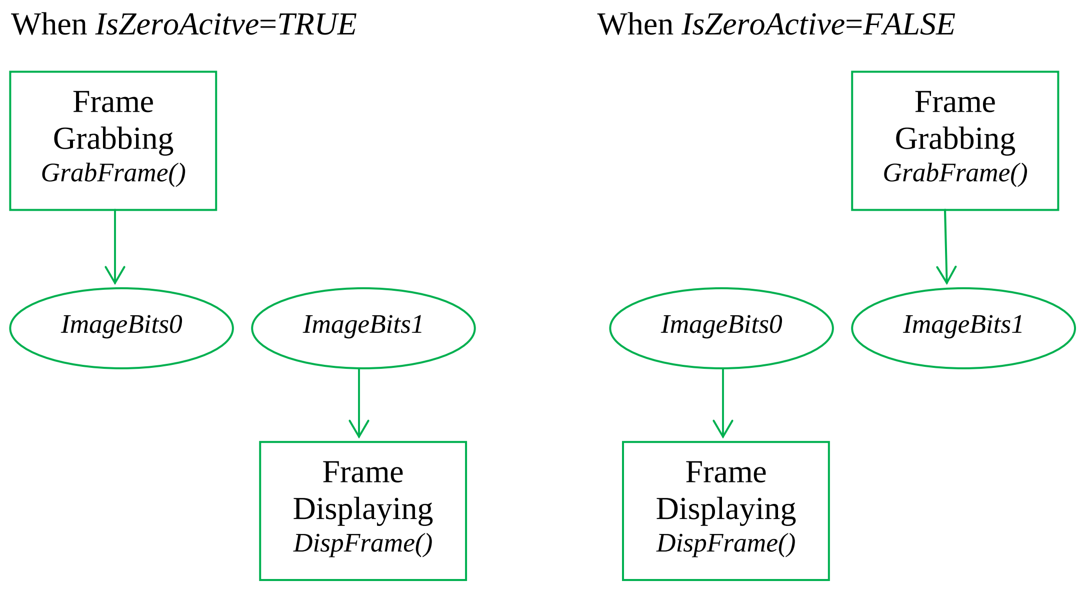
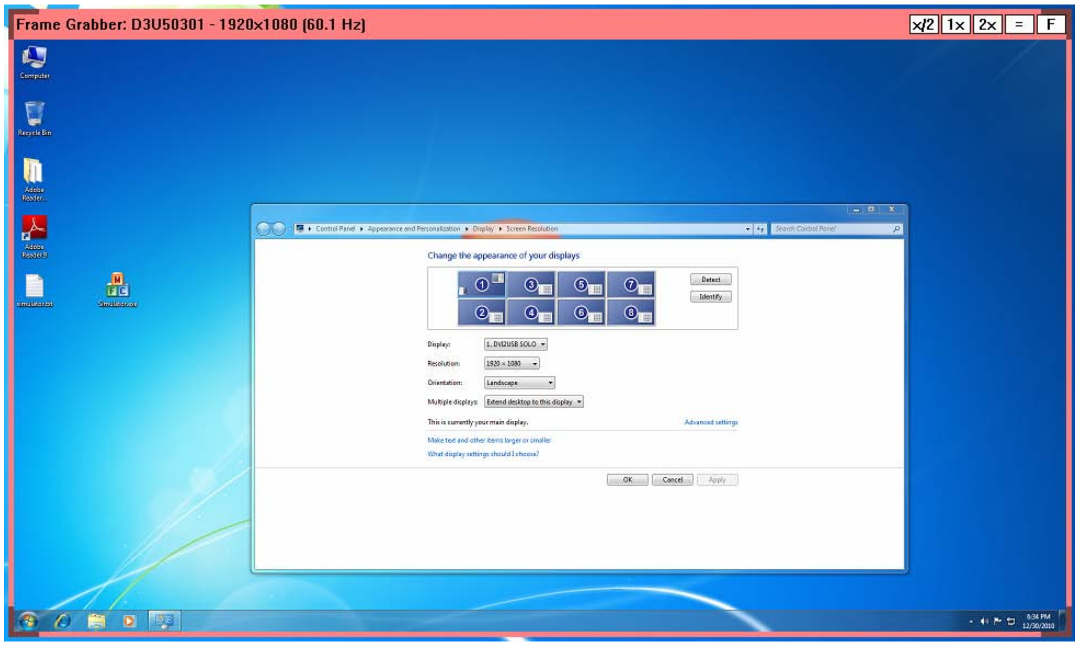

When *IsZeroAcitve=TRUE*
Frame Grabbing *GrabFrame()*
ImageBits0
ImageBits1
Frame Displaying *DispFrame()*
When *IsZeroActive=FALSE*
Frame Grabbing *GrabFrame()*
ImageBits0
ImageBits1
Frame Displaying *DispFrame()*

Figure 6.2: Double buffers for simultaneous grabbing and displaying.

In the Frame Grabber software, we wrote a simple window manager. Figure 6.3 shows a typical window created and managed by the Frame Grabber software. The window has eight resizing handles (two horizontal, two vertical and four at the corners) to be used to resize the window. The window has a title bar that can also be used to move the window. The main area of the window is the client area that is used to display the grabbed image. On the title bar, there are five action buttons (x/2, 1x, 2x, =, and F), and their action are explained below.

- x/2, the grabbed image will be shown at the half size in both X and Y directions.
- 1x, the grabbed image will be shown at the original size
- 2x, the grabbed image will be shown at the double size.
- =, the grabbed image will be shown at the same aspect ratio as the original. Recall that the aspect ratio will be distorted by using the resizing handles. The = button can be used to restore the original aspect ratio.
- F (for Full Screen), the grabbed image will be shown within the full screen. The full screen can be restored to the window mode using the <Esc> key.



x/2
1x
2x
=
F
Computer
Recycle Bin
Control Panel ▸ Appearance and Personalization ▸ Display ▸ Screen Resolution
Search Control Panel
Detect
Identify
Advanced settings
Make text and other items larger or smaller
What display settings should I choose?
OK
Cancel
Apply

Frame Grabber: D3U50301 - 1920x1080 (60.1 Hz)

Change the appearance of your displays

Display: 1. DVI2USB SOLO

Resolution: 1920 × 1080

Orientation: Landscape

Multiple displays: Extend desktop to this display

This is currently your main display.

Figure 6.3: A typical window created by the Frame Grabber software.

6.2 Mirror Server - an Application

As an application of the Frame Grabber software, we implemented a mirror server that can capture up to eight DVI channels, each of 1920x1080 pixels. We assembled a machine to house eight frame grabbers. The assembled machine uses ASUS P6T7 WS Supercomputer motherboard. As shown in Figure 6.4, the motherboard has seven PCI Express slots. Four blue slots can run at X16 speed and three black ones can run at X8. We installed one graphics card on a blue slot and six USB 2.0 PCI Express cards on the remaining slots. Each USB 2.0 PCI Express card provides a USB 2.0 host controller. With two more USB2.0 host controllers from the motherboard, a total of eight USB2.0 host controllers can be provided to host eight DVI2USB™ solo frame grabbers.

\

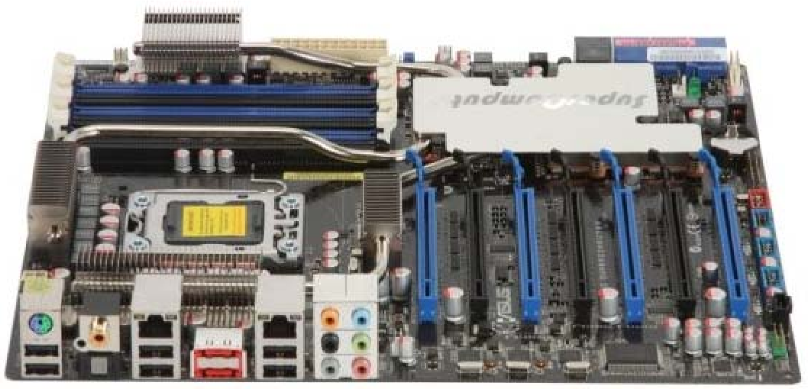


Figure 6.4: ASUS P6T7 WS Supercomputer Motherboard.

Figure 6.5 shows eight DVI2USB™ solo frame grabbers in action. The computer at the right, referred to as the source computer, provides eight DVI signal sources, each of 1080p. The computer at the left, referred to as the grabber computer, has been installed with eight DVI2USB™ solo frame grabbers, each of which is to capture one DVI signal source. The source computer does not have any monitors connected to it, and the grabber computer is connected to two monitors, each of a resolution of 1920x1200 pixels. Eight input sources generated from the source computer are simultaneously captured by the grabber computer. We display the captured input sources at half size in eight windows as shown in Figure 6.6. The arrangement of these eight windows is exactly the same as the arrangement of eight monitors on the source computer. In this way, the two monitors connected to the grabber computer provide eight virtual monitors to the source computer although the source computer does not have any physical monitors connected.

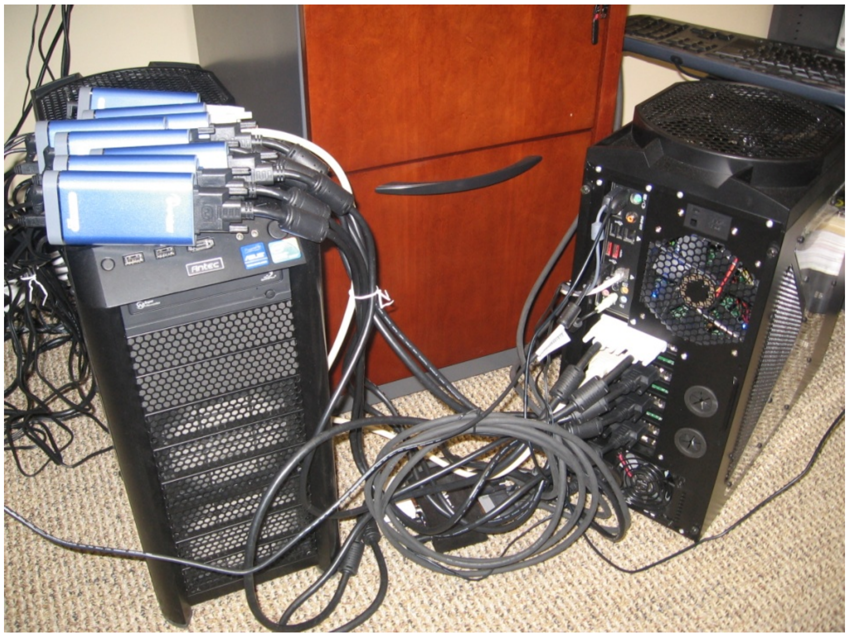


Figure 6.5: Eight DVI2USB™ solo frame grabbers in action.

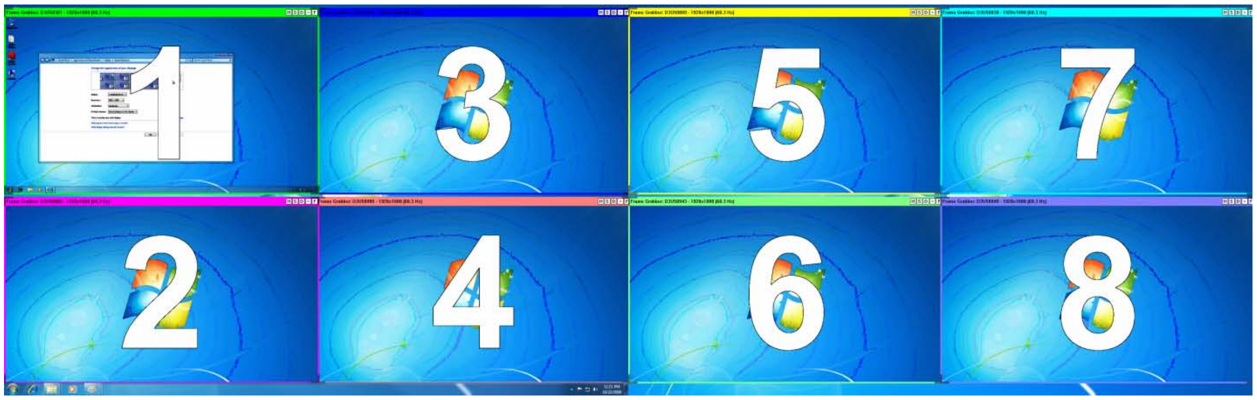
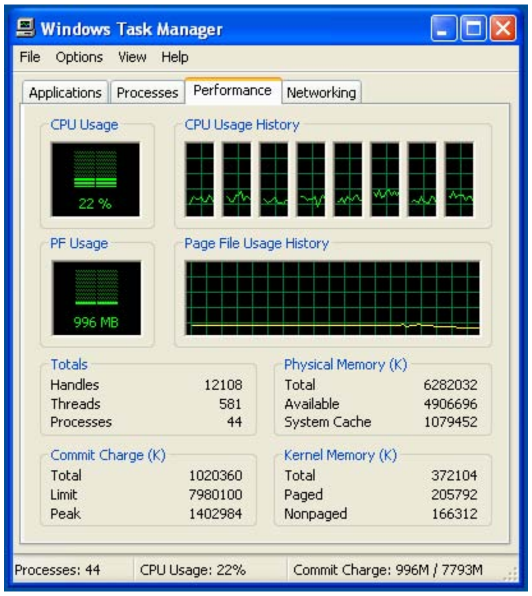


Figure 6.6: Eight monitors captured on the grabber computer.

The experiment results show when a 64-bit XP is running on the grabber computer, the frame rate for capturing is between 13 and 15 frames per second. The other applications are not very responsive to interaction. When the grabber computer runs a 64-bit Windows 7, the grabbing rate is increased to between 25 and 30 frames per second, and a user can still interact with other applications smoothly. Figure 6.7 shows the performance difference between Windows XP and Windows 7 when eight frame grabbers are working simultaneously. The XP task manager shows a CPU usage of about 25% while the counterpart of Windows 7 shows an 80% CPU usage. It shows that the XP system is not able to handle events fast enough, which causes CPU idle. We also verified this point using other methods such as with printf and without pritntf in a C program. Therefore, a 64-bit Windows 7 is recommended to be installed on the grabber computer.

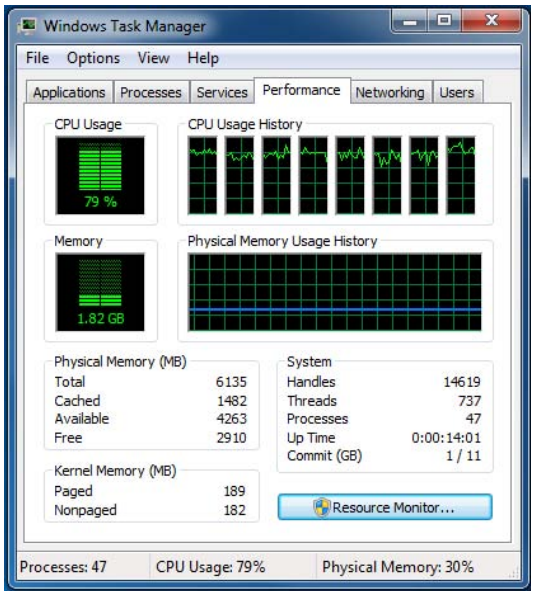


```
Totals                      Physical Memory (K)
Handles        12108        Total          6282032
Threads          581        Available      4906696
Processes         44        System Cache   1079452

Commit Charge (K)           Kernel Memory (K)
Total        1020360        Total           372104
Limit        7980100        Paged           205792
Peak         1402984        Nonpaged        166312

Processes: 44    CPU Usage: 22%    Commit Charge: 996M / 7793M
```

(a) Windows XP



```
Physical Memory (MB)        System
Total          6135         Handles          14619
Cached         1482         Threads            737
Available      4263         Processes           47
Free           2910         Up Time     0:00:14:01
                            Commit (GB)     1 / 11
Kernel Memory (MB)
Paged           189
Nonpaged        182

Processes: 47    CPU Usage: 79%    Physical Memory: 30%
```

(b) Windows 7

Figure 6.7: Performance difference between Windows XP and 7.

7. Results and Discussion

In this project, we designed and built a customized LED pen that can be used to interact with Data Walls. The customized LED pen has its own processing and communication power. The customized LED pen has a unique identification. Because of its unique identification, we can copy an item from a computer to the pen and then paste the item from the pen to another computer. We can also use the customized LED pen for annotation and simply as a color pen.

We enhanced the Laser/LED software in the following aspects. First, the software can monitor multiple Data Walls. Second, the Data Wall can have a soft or hard screen. We addressed all the issues associated with a hard screen. Third, The LED pen interacting with a Data Wall can be a regular one or a customized one.

In this project, we developed software called mice/cursors without borders. With mice/cursors without borders, a mouse can move out of its home computer. It can move into and interact with any computer. With the software, the mouse can copy an item from a computer to itself. It can then move to and paste the item to the other computer.

Multiple LED pens and/or mice can interact with a single computer or a single application. We have developed the software to collect the events generated by these LED pens and/or mice, and to provide a service to distribute the events. An application can obtain and process these events by subscribing to the service. To facilitate the development of such a multi-pen/cursor application in Java, a Java package has been developed to simplify the service subscription.

In this project, we have used Epiphan's DVI2USB™ frame grabbers to capture a desktop, which allow us to perform desktop sharing. Desktop sharing by DVI2USB™ frame grabbers has advantages over other approaches in varieties, speed, and security. The DVI2USB™ frame grabbers have also been used to implement a mirror server.

8. Concluding Remarks

A hardware and software set has been developed to facilitate the collaboration among computers in a command and control room, including wall-sized computers and desktops/laptops. In particular, we have improved two kinds of popular pointing devices, mice and LED pens. An LED pen is preferred by the user standing in front of the Data Wall and a mouse is favored by a user sitting before a desktop. The hardware and software set works well with improved mice and LED pens, and moves command and control automation one step forward.

In this project, both regular and customized LED pens are monitored by a camera system. The advantage is that both regular and customized LED pens can be used with the system. It is certainly understandable that a regular LED pen needs a camera-tracking system to make it an interaction device. How about a customized LED pen? Recall that a customized LED pen has its own processing and communication power. The power is under-used. Imagine putting a small camera at the head of the pen, and let the pen simulate a human eye. With its unique identification, such a pen will be much better than any identification card we carry today.

REFERENCES

1. J. Zhang, *Development of Enhanced Interactive Datawalls for Data Fusion and Collaboration*, AFRL-IF-RS-TR-2008-205, Jul 2008.

2. P. Jedrysik and R. Alvarez, *Advanced Displays and Intelligent Interfaces (ADII)*, AFRL-IF-RS-TR-2006-230, Jul 2006.

3. R. Alvarez, P. Jedrysik, and J. Zhang, "Enhanced Interactive DataWall: Display Architecture for Data Fusion and Collaboration in C2 environments", Proceedings of the SPIE International Symposium on Defense and Security, Apr 2006.

ACRONYMS

LED	Light-Emitting Diode
LCD	Liquid Crystal Display
GPIO	General Purpose Input/Output
GUID	Globally Unique Identifier