eBox 2300 Surveillance Video System

Computer Network Programming

Author: Carlos Daboin

CNT 4104

Instructor: Janusz Zalewski

Florida Gulf Coast University

December 09, 2008
1. Introduction

The eBox 2300 is a thin client “sometimes called a lean client, is a low-cost, centrally-managed computer devoid of CD-ROM players, diskette drives, and expansion slots. Also, a thin client accesses programs and data from a server instead of storing them locally, and the software that performs the majority of its operations reside on a server rather than the local computer”[1]. The eBox 2300 is the device to be used for an application that makes the eBox 2300 accessible to clients/user in a LAN or WAN. The application in this project is a client/server application based on C#, and its function is to transmit and receive video using sockets.

The Server application’s main use is to stream live video capture from an USB camera connected to eBox 2300. The server application runs under the Windows CE 6.0 operating system. This OS is designed and customized based on the needs of the server streaming application. The OS and server application are developed on more powerful and faster computer, called the host. Then, a Windows 6.0 CE image and the C# server application are deployed on the eBox 2300, called the target.

The Client application main use is to receive the streaming video from eBox2300 and display on the computer that runs this application. The client application will be deployed on the host Windows PC, because the .NET framework is needed to run the C# client. After client application is deployed the user will be able to connect to the server application running on the eBox 2300 and see the live video streamed from the remote location.
The eBox 2300 makes use of a video surveillance web camera that streams video over a network, and then remote client running a client application is able to receive the media transmission. This project can be expanded by adding phidgets to increase eBox 2300 interaction with the real environment events, for example, adding a sensor that detects movement, and servo motor the makes the camera point to the source of movements. The picture below, figure 1, shows the main configuration that this project will use to stream video.

Figure 1. Main Project System Architecture
2. Problem Definition

The problem is to develop video streaming application over a network using a thin client. There are three major issues, related to this.

The first and most important issue is to develop an operating system capable to run software applications, to communicate with external devices, and to transmit data over a network. The operating system has to be designed and tested on a more powerful computer called the host (Figure 2). It has to be customized for the video stream application, so when deployed it will make the eBox 2300 a capable device to run a video server application, to send stream video over a network, and the most important communicate with an USB device, camera.

The second issue is to develop an application that runs and executes under the deployed eBox 2300 Operating System. The application has to interact and communicate with the USB camera, then stream the USB camera video feed over a network to users/clients in the same LAN or WAN (Figure 3).
Third, when developing an application that will receive the video stream over the network, the application has to identify that the streamed video comes from the eBox 2300, and has to display the video on a friendly graphic interface for the connected to the same LAN or WAN (Figure 4).

Figure 3 Server Application

Figure 4 Client Application
3. Solution

The problem or challenge to transforms the eBox 2300 into a video streaming device is to first, deploy an operating system to the eBox 2300 to allow the thin client to run applications and drivers. Second, deploy or install a driver that allows communication between the eBox 2300 and the video capture device (USB camera). Third, develop a client/server application that allows communication between remote PC’s and the eBox 2300. Finally, configure eBox 2300 server application to stream video being capture from the USB camera, and set the client application on the remote PC to display steamed video.

3.1 Deploy an Operating System

Windows Embedded CE 6.0 and Linux OS (Puppy Linux) are two operating systems that can be installed or deployed on eBox 2300. Windows Embedded CE 6.0 is the operating system that will handle the drivers and client/server applications running on the eBox 2300. The installation of Puppy Linux is also covered in this project to offer an alternative approach of OS deployment.

Windows Embedded CE 6.0

The steps needed to install Windows Embedded CE 6.0 package on Windows PCs:

2. Download and run the VS 2005 executable installation file.


4. VS2005 must already be installed onto the develop station.

5. Also, you can Insert the Jumpstart CD to install the platform builder, then run the auto executable, if the installer does not run automatically.

6. Make sure X86 CPU support is selected during installation (Figure 5).

7. Install the Board-Support-Package, To install this BSP, simply double click on the ICOP_Vortex86_60B_BSP.MSI file located in the \BSP directory on the jumpstart CD

8. Install VS2005 CoreCon Component, CoreCon is used to establish link between CE 6.0 device and VS2005 development station. The file is included in the Jump
Start CD, in the “\CoreCon” folder. To install this component, double click on CoreCon_x86_VS2005.MSI file in the Jump Start CD’s “\CoreCon” directory.

9. Install Windows Embedded CE 6.0 SDK, From VS2005 IDE, select Project | Add New SDK to bring up the SDK Property Pages screen.

![Figure. 6 SDK Property Page](image)

10. Enter eBox2300_WinCE600_SDK as the name for the SDK (Fig.6).

11. Fill in the company name and company website information

12. On the left side of SDK Property Page, click on Install and enter MSI folder path and MSI file name on the right side. Use the default path, and enter eBox2300_WinCE600_SDK.msi as the file name (Figure 7).
13. On the left side of SDK Property Page, click on Development Languages and select both Native and Managed development support.

14. Click on the Apply and then OK button to complete the Add New SDK process (Figure. 8).

After deployment, Windows Embedded CE 6.0 image should look very close to the picture below (Figure 9).

![Windows Embedded CE 6.0 Image](image)

Figure 9 Windows Embedded CE 6.0 Image

Using a very small operating system that can be completely boot from an USB flash drive or Compact Flash card can work as a base for software applications and communicate with external devices. Linux has a very small version of Linux (Puppy Linux) that can be booted from a 250MB USB key or a 250MB compact flash card.
Installing Linux OS, Puppy Linux, on the eBox 2300.

“Puppy Linux is a Live CD Linux distribution that is very small and focuses on ease of use. If the computer has at least 256 MB of RAM, the entire operating system and all the applications will run from RAM, allowing the boot medium to be removed after the operating system starts”[5]. In this project Puppy Linux will be booted form an USB Key drive.

Hardware need to create an USB Puppy Linux includes:

- 128MB or larger USB Stick
- PC with a CD reader/burner

Puppy Linux contains its own Universal Installer so installation is fairly straightforward.

The process for running Puppy Linux is as follows.

2. Burn the ISO to a CD.
3. Start your computer via Puppy Linux CD.
4. Once Puppy has loaded, from the Taskbar, Click: Start-> Setup-> Puppy universal installer (Figure 10).
5. Follow the onscreen instructions which will walk you through the Puppy USB installation process

6. When finished with the Puppy USB install, reboot your PC and go into system BIOS to change your boot order to boot from the USB device.

7. Save your settings and reboot your PC.

   After boot up, Puppy Linux should look very close to the picture below (Figure 11).
3.2 **USB CameraDriver to communicate the Operating System and Camera Device**

The driver was developed on C++ by Doug Boling senior Windows Embedded CE developer, and Microsoft. The USB Webcam driver and released it under a Microsoft Shared Source license. This is the first shared source community project involving Microsoft's Mobile and Embedded Devices group, according to Microsoft.

The driver will create the necessary dynamic libraries to allow applications to the read the streams coming from the USB port where the USB camera, Logitech 5000 PRO, is connected. The driver will be added to the operating system image design as a subproject, and has to be build to integrate to OS image before deployment. This driver only work for a selected number of Logitech cameras and still under continuous development. Doug boiling tested the driver with a Logitech 5000 PRO USB camera, the camera used in this project is the same, so the driver is guarantee to work.
3.3 **Software client/server application**

The software application is strictly related to the operating system, so the programming language used for development will be determinate based on the operating system selection. Java is a better choice to work under a Linux environment, and C# or VB is the best choice to work Windows CE 6.0, because of the .Net framework. Therefore, the programming language used for this project is C#, and a client/server application under C# is the responsible to allow communication between client computers connected to the network and the eBox 2300. Then, the C# server application connects to the USB camera and streams the video capture from the camera over a computer network. Finally, the C# client application running on remote PC has a GUI, graphic user interface, which display the video streamed over the network, and will interact with the remote computer’s user. Refer to figure 29, 30, and 31 in the project’s Case Study section to see actual C# application’s GUI.
4. Case Study

The project development starts by connecting the eBox 2300 and the Windows PC that deploys the embedded CE 6.0 image and the C# application, on the same network. Then, Ethernet cables will be the physical medium that connects the eBox 2300 and PC to the network creating an open path between the eBox 2300 and PC for data communication.

Windows Embedded CE 6.0 is the operating system that the eBox 2300 runs, and the OS is designed and customized using Visual Studio 2005. The OS will have all media drivers, and networking utilities to be able to communicate with the network and the USB camera, also the .NET framework to help the C# application and the USB drivers to work together. As soon as the operating system’s image is deployed over the network; the eBox 2300 hardware transforms into thin computer that is able to talk to a network and USB devices.

4.1. OS Configuration

The steps to configure the system software are described below.

1. Open Visual Studio 2005; select new project.

2. Select Platform Builder for CE 6.0, then OS Design, and then name it (eBoxOS). See Figure 12.
3. In the BSP (Board Support Packages), select ICOP_Vortex86_60B: x86 (Figure 13).

4. In the design templates selection step, select Industrial Device (Figure 14).
5. Design template, select Windows Internet Appliance (Figure 15).

6. In the applications & media selection step, select .NET, Windows Media Audio/MP3, and Windows Media Video/MPEG-4 Video (Figure 16).
7. In the networking & communications step, select Local Area Network, and Wide Area Network. The OS design wizard will finish after this step (Figure 17).

Catalog Items View Tap (Enhancing the OS Design)
8. Select Microsoft USB camera driver for future use with server application from Third Party directory (Figure 18).

9. Select ConMan_86 files components for communication and application deployment from Third Party directory (Figure 18).

10. Select MJPEG Decompression Filter for compressed media from Third Party directory (Figure 18).

11. Expand Third Party catalog folder, and select Vortex86_Audio from BSP/ICOP_Vortex86_60b:X86/Device Drivers/Audio; select Vortex86_Display
from BSP/ICOP_Vortex86_60b:X86/Device Drivers/Display, to enable eBox 2300 media interfaces (Figure 19).

![Figure 19 Windows Embedded CE 6.0 Catalog Items View](image)

12. Expand Core OS => CEBASE folder, find and include the following components to the OS Design.

a) **Applications-End User | CAB File Installer/Uninstaller**


b) **Core OS Services | USB Host Support | USB Storage Class Driver**
This component provides support for hot pluggable USB storage device supporting most USB flash drive and external USB mass storage device.

13. Expand Core OS => CEBASE => File Systems and Data Store folder, select and include the following components to the OS Design.

   a) File System – Internal | RAM and ROM File System

   b) Registry Storage | Hive-based Registry

   Both of the above two components are needed to support Hive-based registry implementation. Hive-based registry is used to store and save registry entries changed between power reset.
4.2. OS Deployment

The steps to deploy the OS to the eBox 2300 are described below.

1. From VS2005 IDE, select Build | Configuration Manager. From the Active solution configuration selection options, select ICOP_Vortex86_60B x86 Release (Figure 20).

![VS 2005 Configuration Manager](image)

Figure 20  VS 2005 Configuration Manager

2. From VS2005 IDE, select Project | OSeBoxCamServer (Your OS Name) Properties

3. Click to expand Configuration Properties tree on the left side of the screen (Figure 21).
4. Click to high-light Build Options, a list of Build Options with check boxes will be shown on the right side of the screen (Figure 21).

5. Select Enable eboot space in memory (IMGEBOOT=1). This option adds support for Ethernet debugging by bundling the Ethernet boot loader in the image (Figure 21).

6. Select Enable ship build (WINCESHIP=1). This option will enable a retail build and suppress debug messages (Figure 21).

7. Disable KITL – Make sure the Enable KITL (no IMGNOKITL=1) check box is NOT selected (Figure 21).

Figure 21 VS 2005 Property Page
8. Click to highlight Environment on the left side of screen (Figure 22).

![Figure 22 VS 2005 Property Page](image)

9. Enter new variable name and set associated value for the following environment variables (Figure 23). Click New; enter variable name, and variable value. Refer to table below:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Variable Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMGRAM128</td>
<td>1</td>
<td>Supports hardware with 128MB of RAM</td>
</tr>
<tr>
<td>PRJ_BOOTDEVICE_ATAPI</td>
<td>1</td>
<td>Designates booting from ATAPI device</td>
</tr>
<tr>
<td>PRJ_ENABLE_FSREGHIVE</td>
<td>1</td>
<td>Enables Hive-based registry function</td>
</tr>
<tr>
<td>PRJ_ENABLE_FSMOUNTASROOT</td>
<td>1</td>
<td>Enables external file to mount as root</td>
</tr>
</tbody>
</table>
10. From the VS2005 IDE, select Build | Build Solution to build OS image from the OS Design project.

11. From VS2005 IDE, select Target | Connectivity Options, Target Device set to eBox2300_WinCE600_SDKx86 Device. Download and Transport set to Ethernet. Debugger set to KdStub (Figure 24).
12. Turn on power and boot up eBox-2300 with the provided Windows CE SDK boot image. It will boot to DOS and provide the menu selections. Select 3, or 4 image will load from Ethernet connection).

13. After eBox-2300 boots-up and sends a boot-me request, the Ethernet Download Settings dialog screen, select device ID listed in the Active target devices list box (Figure 25).
14. From VS2005 IDE select Target | Attach Device.

15. After the image download process is completed, eBox-2300 will load the Windows CE image.
4.3. USB Camera Driver Development

“Windows CE only supports the host side of the USB specification. This means that any USB-capable device can be connected to a CE platform, but a CE device cannot act as a USB device to another computer. The full USB host-side specification is supported on CE, including the various data transfer methods-control, isochronous, interrupt-driven, and bulk.

USB drivers can be implemented using three approaches. First, a USB driver can be implemented as a standard stream-interface driver, allowing applications to use the file API to access the device. Second, a USB driver may interact with an existing Windows API, if such an API already exists for the type of the device. Microsoft's USB mouse sample driver uses this approach-the USB driver directly interacts with the mouse API. Third, a USB driver may provide its own API, which may best fit the device's capabilities” [6].

The driver used in the project to allow communication between operating system applications and USB camera was develop by Doug Boling, author of Programming Windows CE, a group of developers, including some from Microsoft [7]. The USB Webcam driver has been released under the Microsoft Shared Source license. The driver supports cameras in compliance to the USB Video Standard. The driver has been tested against the Logitech QuickCam Pro 5000 and video streaming has been tested by WebCam2 application also developed by Doug Boling under C++.

Driver and test application subproject on Windows Embedded CE 6.0 are used as follow
1) In the OS design project, right click on the Subprojects folder locator in the Solution Explorer tab. (Figure 26)

![Figure 26](image)

2) Select Add New Subproject, then in the wizard window select WCE Application, and for Subproject name use WebCam. (Figure 27)

![Figure 27](image)
3) Click Finish

The WebCam subproject is only needed to have a location or directory to place the driver and test application files developed by Doug Boling. Then, the driver and test application are placed in WebCam subproject directory that is normally located at C:\WINCE600\OSDesigns\eBoxOS\eBoxOS\WebCam. To get the driver and the test application files go to my website: http://satnet.fgcu.edu/~cedaboin/WebCam.zip and click on the eBox 2300 link, then copy all the files is under the WebCam zip file. Replace the files created by the IDE in the WebCam subproject directory with the ones you copy from my web site. Finally, right click on the subproject, and select build to add this component to the OS image.

After deployment, the eBox 2300 OS image has the compiled driver and the test application to check compatibility between USB camera and the OS. CamTes2 executable file is under OS Windows’s directory, and is the test application to check video stream display from USB camera. (Figure 28)
Figure 28 CamTest2 video capture application
4.4 Application Development

4.4.1 Server Application Development

The server application was developed in C# as a Windows device application. The application is a server application that uses .NET framework's Socket class and its main function is to send and receive data to test remote communication. The server has to allow a client application to make connections by listening at some port; the server does not need to know client I.P. addresses. Server's responsibility is to manage client connections.

1) The server application listens to the connections and accepts the connection

   public Socket m_socListener;
   public void StartListening()

2) The server creates the listening socket

   m_socListener = new
   Socket(AddressFamily.InterNetwork,SocketType.Stream,ProtocolType.Tcp);

3) The server binds to eBox 2300 IP Address

   m_socListener.Bind( ipLocal );
   m_socListener.Listen (4);

4) The server also sends data to the client; the application uses m_socWorker socket for that purpose.

   Object objData = txtDataTx.Text;
   byte[] byData = System.Text.Encoding.ASCII.GetBytes(objData.ToString ());
   m_socWorker.Send (byData);
The source code of the server application is located in the project Appendix A, and the code is a customized version of a server socket application example developed by Ashish Dhar [8].

4.4.2 Remote Client Application

The client application was developed in C# as a windows application. The application is a client application that uses .NET framework's Socket class and its main function is to send and receive data to server application running on the eBox 2300. The client application uses a GUI to enter eBox 2300 IP address and the server application port to start data transmission.
1) Client application creates the socket

```csharp
m_socListener = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.IP);
```

2) Client application makes a connection to the server using eBox 2300 IP address and serves application port number.

```csharp
IPAddress ip = IPAddress.Parse(txtIPAddr.Text);
int iPortNo = System.Convert.ToInt16(txtPort.Text);
IPEndPoint ipEnd = new IPEndPoint(ip.Address, iPortNo);
```

3) Client proceeds to connect to server application on the eBox 2300

```csharp
m_socClient.Connect(ipEnd);
```

4) Client application listen to eBox 2300 incoming data

```csharp
m_asynResult = m_socClient.BeginReceive(theSocPkt.dataBuffer, 0, theSocPkt.dataBuffer.Length, SocketFlags.None, pfnCallBack, theSocPkt);
```

---

**Figure 30. C# Client Application**
The source code of the server application is located in the project Appendix B, and the code is a customized version of a server socket application example developed by Ashish Dhar [8].

4.4.3 Video Streaming application

The video streaming application is the enhancement of the C# server application, the improvement of this application will allow the communication with the USB webcam driver and the server application. Then, the application will use the socket to send video to a remote client. The remote client will be modified to create a bitmap or picture holder to display the changing stream of bites, the video, coming from the USB camera connected to the eBox 2300.

The video server application is this project has taken from ServoCAM project developed by Keith Jarvis, David Kimn, and Justin Belisle, students of Georgia Institute of Technology. “The ServoCam project’s major design goal was to implement a camera that was capable of being viewed remotely over a network connection” [9]. This application has been use to prove and test remote stream video capabilities, however an improved and personalized version will be develop in the future to enhance project’s capabilities.

The video server application was develop on C#:

1) It has socket bind it to a port, 1888, that listening from any client’s IP request.

```csharp
private const int LISTEN_PORT = 18888;

tl = new TcpListener(IPAddress.Any, LISTEN_PORT);
```
2) Then, connect to camera driver to get stream of bites coming from the USB port

```csharp
WebCamSDK.FORMATPROPS fmtData;

int rc = camera.GetFormatInformation((ushort)m_nFormat,
(ushort)m_nFrame, out fmtData);
```

3) Finally, sends stream of video

```csharp
rc = camera.GetFirstStreamFrame((ushort)m_nFormat, (ushort)m_nFrame,
(uint)nFrameInterval, out address, out nSize, nTimeout);
```

The source code of the server application is located in the in the project appendix, see Appendix C.

### 4.4.4 PC Remote Viewer Application

The client application was developed on C# as a windows application. The application uses asynchronous sockets to connect to remote video server application running on the eBox 2300. The application uses a GUI to allow user to enter remote IP and port numbers.

1) Client connect to remote application hosted by eBox 2300

```csharp
AsyncCallback ac = new AsyncCallback(Go);
client.BeginConnect(ip, port, ac, client);
```

2) Then, put bytes received by server into a bitmap

```csharp
byte[] bt = br.ReadBytes(len);
MemoryStream ms = new MemoryStream(bt);
```
Bitmap b = new Bitmap(ms);

3) Display bitmap into picture box

this.pictureBox1.Image = b;

The source code of the server application is located in the in the project appendix, see Appendix D. Figure 3 shows client application during in class demo presentation.

Figure 31. PC Remote Viewer Application
4.4.5 Application Deployment Procedure for Text, and Video C# Client/Server Socket Apps

To create CE 6.0 applications using Visual Studio 2005 and establish a connection to download applications to the eBox-2300 for testing and debug, VS2005 needs a SDK for eBox-2300.

1. From VS2005 IDE, select Project => Add New SDK.

2. Enter eBox2300_WinCE600_SDK as the name for the SDK.

3. Fill in the company name and company website information.

4. On the left side of SDK Property Page, click on Install and enter MSI folder path and MSI file name on the right side.

5. Use the default path, and enter eBox2300_WinCE600_SDK.msi as the file name.

6. On the left side of SDK Property Page, click on Development Languages and select both Native and Managed development support.

7. Click on the Apply and then OK button.

8. From VS2005 IDE, select Build => Build All SDKs to build and generate SDK installation file.

9. A SDK with the file name “eBox2300_WinCE600_SDK.msi” is generated in the following directory: \WINCE600\OS Designs\eBoxOS\eBoxOS\SDKs\SDK1\MSI\.
10. Install this SDK to the VS2005.

4.4.6 From eBox 2300 (with CE 6.0 image created in the earlier steps running):

1. Click on Start => Run from CE 6.0 desktop with the cmd command to open a console command window.

2. Type `IpConfig` to view the eBox’s assigned IP-address

3. Double click on My Device and open the Windows folder.

4. Double click on ConmanClient2.exe follow by double click on cMaccept.exe to launch Corecon connection service.

4.4.7 From VS2005 IDE

5. Set the target device to “eBox2300_WinCE600_SDK x86 Device” (Figure 31).

![ChatServer - Microsoft Visual Studio](image)

Figure 32 VS 2005 Target Device

6. Select Tools => Options

7. Click to expand the “Device Tools” folder and select the “Devices” sub-folder.
8. Select eBox2300_WinCE600_SDK from the list of available platform in the Show devices for platform combo text box (Figure 32).

![Figure 33 VS 2005 Option Window](image)

9. Click on the Properties button to bring up eBox2300_WinCE600 x86 Device Properties setting screen (Figure 33).

![eBox2300_WinCE600_SDK x86 Device Properties](image)
10. Click on the Configure button to bring up Configure TCP/IP Transport (Figure 34).

11. Select Use specific IP address and enter eBox-2300’s IP address (Figure 34).

12. Select Tools => Connect to device

13. Select eBox2300_WinCE600_SDK from the list of available devices, and click on the Connect button (Figure 36).
14. Select Debug => Start Debugging to bring up the Deploy eBox2300_Demo screen.

15. Select eBox2300_WinCE600_SDK and click Deploy (Figure 36).

4.4.8 Video Surveillance Hardware Components

The video server system will be implemented using the following hardware:
• An eBox 2300, thin client.

• Windows XP PC that deploys the Operating System, and the client/server software application.

• Logitech PRO 5000 USB camera.

• Ethernet cables to connections to the network.

The video server system will be implemented using the following software:

• Visual Studio 2005 to design operating system, and the development of client/server software application.

• The Operating system that will deploy on eBox 2300 is Window Embedded CE 6.0.

• The programming language to use C# for client and server applications

• Framework dot Net.

• Microsoft USB driver to connect OS/C# to USB camera.
5. Summary

- The project will make the eBox 2300 a thin client to be accessible from remote users. The communication between the eBox 2300 and the remote user is over TCP/IP networks infrastructures.

- The project will demonstrate the use of a network to communicate with remote devices, in this case an USB camera attach to eBox 2300.

- The project demonstrates the use of top level applications, like a C# client/server application, to send and receive data over a network, to communicate with an USB device, and to create user interfaces to interact with remote device.

- The project might be the starting point of many remote applications that can be combined with remote video display for future network systems developments.

A student who would like to evolve this project further would need to go through the following steps to set this system up:

**Hardware required**

1) eBox 2300 thin client.

2) Windows PC (XP/VISTA).

3) Logitech QuickCam Pro 5000.
4) 2 Ethernet cables (connect eBox 2300 to network, connect Windows PC to network), Monitor.

5) Keyboard and a mouse (single USB cable connection is recommended).

6) Network infrastructure (LAN).

Software Installation

1) Install visual studio 2005. Refer to project’s section 3, Solution, page 5.

2) Install Windows Embedded CE 6.0. Refer to project’s section 3, Solution, pages 5 – 9.

3) Create SDK for application deployment. Refer to project’s section 4.4, Case Study, pages 37 - 38.

Application Development

Windows Embedded CE Image

1) Develop Windows Embedded CE image. Refer project’s section 4.1, Case Study. Pages 14 – 20. Also, install USB Camera driver on OS, refer to section 4.3, Case Study, pages 28-29.

C# Video Server application

1) Copy the VideoHostCE directory (file folder) from zip folder located at my website: http://satnet.fgcu.edu/~cedaboin/VideoHostCE.zip. Make sure to download and unzip the folder before copying the directory.

2) Open Visual Studio 2005 projects’ directory located at: C:\Documents and Settings\Administrator\My Documents\Visual Studio 2005\Projects. Paste the directory
you copied from step 1 to: C:\Documents and Settings\Administrator\My Documents\Visual Studio 2005\Projects

3) Open visual studio 2005, select File => Open => Project/Solution => VideoHostCE => VideoHostCE (solution file, not the directory).

4) Click Open.

C# Video Client Application

1) Copy the VideoClientWin directory (file folder) from zip folder located at my website: http://satnet.fgcu.edu/~cedaboin/ClientVideoXP_PC.zip. Make sure to download and unzip the folder before copying the directory.

2) Open Visual Studio 2005 projects' directory located at: C:\Documents and Settings\Administrator\My Documents\Visual Studio 2005\Projects. Paste the directory you copied from step 1 to: C:\Documents and Settings\Administrator\My Documents\Visual Studio 2005\Projects

3) Open visual studio 2005, select File => Open => Project/Solution => VideoClientWin => VideoClientWin (solution file, not the directory).

4) Click Open.

Connect eBox 2300 to the network

1) Connect power cord to eBox power port, monitor cable to eBox VGA port, and connect network Ethernet cable to eBox Ethernet port.

2) Connect mouse, keyboard to eBox 2300 to the USB, or PS2 back ports. See below picture for port locations.
Connect Windows (XP/VISTA) to the network

1) Connect Windows PC to the network, wirelessly or wired, but make sure the PC and eBox 2300 are in the same network. A wired connection to the network was used in this project. See connection diagram, below, for entire project’s hardware configuration.
3) Leave the two USB front ports to connect external devices, in this project a Logitech QuickCam Pro 5000 have been used to connect to one of the USB front ports.

Application Deployment


2) C# application deployment to eBox 2300. Refer to project’s section 4.4.7, Case Study, pages 38 – 41.

3) C# client application deployment on Windows PC, open visual studio, select File => Open => Project Solution => browse to the directory => Open. Then, Select Debug => Start Debugging
Testing

1) Run C# windows client application, described in step 3 from Application Deployment. Enter eBox 2300 IP address in client’s application GUI, Windows form, the port number entry is already set by the client’s application.

2) Click connect.
6. References


using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.Collections;

namespace eBoxServerApp
{
    public partial class Form1 : Form
    {
        public AsyncCallback pfnWorkerCallBack;
        public Socket m_socListener;
        public Socket m_socWorker;
        delegate void AppendTextCallback(string text);

        public Form1()
        {
            InitializeComponent();

            try
            {
                //create the listening socket...
                m_socListener = new Socket(AddressFamily.InterNetwork,
                                              SocketType.Stream, ProtocolType.Tcp);
                IPEndPoint ipLocal = new IPEndPoint(IPAddress.Any, 8221);
                //bind to local IP Address...
                m_socListener.Bind(ipLocal);
                //start listening...
                m_socListener.Listen(4);
                // create the call back for any client connections...
                m_socListener.BeginAccept(new AsyncCallback(OnClientConnect),
                                            null);

                //btnConnect.Enabled = false;
            }
            catch (SocketException se)
            {
                MessageBox.Show(se.Message);
            }
        }

        public void OnClientConnect(IAsyncResult asyn)
        {

        }
    }
}
try {
    m_socWorker = m_socListener.EndAccept(asyn);
    WaitForData(m_socWorker);
} catch (SocketException se) {
    MessageBox.Show(se.Message);
}
send();

public void WaitForData(System.Net.Sockets.Socket soc)
{
    try {
        if (pfnWorkerCallBack == null)
        {
            pfnWorkerCallBack = new AsyncCallback(OnDataReceived);
        }
        CSocketPacket theSocPkt = new CSocketPacket();
        theSocPkt.thisSocket = soc;
        // now start to listen for any data...
        soc.BeginReceive(theSocPkt.dataBuffer, 0, theSocPkt.dataBuffer.Length, SocketFlags.None, pfnWorkerCallBack, theSocPkt);
    } catch (SocketException se) {
        MessageBox.Show(se.Message);
    }
}

public void OnDataReceived(IAsyncResult asyn)
{
    try {
        CSocketPacket theSockId = (CSocketPacket)asyn.AsyncState;
        //end receive...
        int iRx = 0;
        iRx = theSockId.thisSocket.EndReceive(asyn);
        char[] chars = new char[iRx + 1];
        int charLen = d.GetChars(theSockId.dataBuffer, 0, iRx, chars, 0);
        System.String szData = new System.String(chars);
        AppendRxText(szData);
        WaitForData(m_socWorker);
    } catch (SocketException se) {
        MessageBox.Show(se.Message);
    }
}
private void AppendRxText(string text)
{
    // appends every element of the array to the textbox: txtRecv

    if (this.txtRecv.InvokeRequired)
    {
        AppendTextCallback d = new AppendTextCallback(AppendRxText);
        this.Invoke(d, new object[] { text });
    }
    else
    {
        txtSwitch.Visible = true;
        txtRecv.Text = txtRecv.Text + text;
    }
}

private void btnSend_Click(object sender, EventArgs e)
{
    try
    {
        Object objData = txtDatasnd.Text;
        byte[] byData = System.Text.Encoding.ASCII.GetBytes(objData.ToString());
        m_socWorker.Send(byData);
    }
    catch (SocketException se)
    {
        MessageBox.Show(se.Message);
    }
}

class CSocketPacket
{
    public System.Net.Sockets.Socket thisSocket;
    public byte[] dataBuffer = new byte[1];
}

private void send()
{
    try
    {
        Object objData = "Connected to eBox 2300";
        byte[] byData = System.Text.Encoding.ASCII.GetBytes(objData.ToString());
        m_socWorker.Send(byData);
    }
    catch (SocketException se)
    {
        MessageBox.Show(se.Message);
    }
}
Appendix B

C# Windows Client Application Source Code

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Net;
using System.Net.Sockets;

namespace AsyncClient
{
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        byte[] m_DataBuffer = new byte [10];
        IAsyncResult m_asynResult;
        public AsyncCallback pfnCallBack ;
        public Socket m_socClient;
        delegate void AppendTextCallback(string text);

        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.TextBox txtPort;
        private System.Windows.Forms.Button cmdSend;
        private System.Windows.Forms.TextBox txtDataRx;
        private System.Windows.Forms.GroupBox groupBox2;
        private System.Windows.Forms.GroupBox groupBox3;
        private System.Windows.Forms.TextBox txtDataTx;
        private System.Windows.Forms.TextBox txtIPAddr;
        /// <summary>
        /// Required designer variable.
        /// </summary>
        /// </summary>
        private System.ComponentModel.Container components = null;

        public Form1()
        {
            // Required for Windows Form Designer support
            InitializeComponent();
        }
    }
}
// TODO: Add any constructor code after InitializeComponent

call

//

/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if (components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.groupBox1 = new System.Windows.Forms.GroupBox();
    this.cmdClose = new System.Windows.Forms.Button();
    this.cmdConnect = new System.Windows.Forms.Button();
    this.txtPort = new System.Windows.Forms.TextBox();
    this.label2 = new System.Windows.Forms.Label();
    this.txtIPAddr = new System.Windows.Forms.TextBox();
    this.label1 = new System.Windows.Forms.Label();
    this.txtDataTx = new System.Windows.Forms.TextBox();
    this.cmdSend = new System.Windows.Forms.Button();
    this.txtDataRx = new System.Windows.Forms.TextBox();
    this.groupBox2 = new System.Windows.Forms.GroupBox();
    this.groupBox3 = new System.Windows.Forms.GroupBox();
    this.groupBox1.SuspendLayout();
    this.groupBox2.SuspendLayout();
    this.groupBox3.SuspendLayout();
    this.SuspendLayout();
    //
    // groupBox1
    //
    this.groupBox1.Controls.Add(this.cmdClose);
    this.groupBox1.Controls.Add(this.cmdConnect);
    this.groupBox1.Controls.Add(this.txtPort);
    this.groupBox1.Controls.Add(this.label2);
    this.groupBox1.Controls.Add(this.txtIPAddr);
    this.groupBox1.Controls.Add(this.label1);
    this.groupBox1.Location = new System.Drawing.Point(8, 8);
    this.groupBox1.Name = "groupBox1";
    this.groupBox1.Size = new System.Drawing.Size(214, 209);
    this.groupBox1.TabIndex = 0;
    this.groupBox1.TabStop = false;
    this.groupBox1.Text = "groupBox1";
    //
    // groupBox2
    //
    this.groupBox2.Controls.Add(this.txtDataTx);
    this.groupBox2.Location = new System.Drawing.Point(128, 132);
    this.groupBox2.Name = "groupBox2";
    this.groupBox2.Size = new System.Drawing.Size(88, 101);
    this.groupBox2.TabIndex = 1;
    this.groupBox2.TabStop = false;
    this.groupBox2.Text = "groupBox2";
    //
    // groupBox3
    //
    this.groupBox3.Controls.Add(this.txtDataRx);
    this.groupBox3.Location = new System.Drawing.Point(128, 236);
    this.groupBox3.Name = "groupBox3";
    this.groupBox3.Size = new System.Drawing.Size(88, 101);
    this.groupBox3.TabIndex = 2;
    this.groupBox3.TabStop = false;
    this.groupBox3.Text = "groupBox3";
    //
    // pnlMain
    //
    this.pnlMain.Controls.Add(thiscomboBox1);
    this.pnlMain.Location = new System.Drawing.Point(128, 236);
    this.pnlMain.Name = "pnlMain";
    this.pnlMain.Size = new System.Drawing.Size(88, 101);
    this.pnlMain.TabIndex = 3;
    this.pnlMain.TabStop = false;
    this.pnlMain.Text = "pnlMain";
    //
    // formMain
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
    this.ClientSize = new System.Drawing.Size(312, 340);
    this.ControlBox = false;
    this.Name = "formMain";
    this.Text = "Main Form";
    this.ResumeLayout(false);
}
this.groupBox1.Size = new System.Drawing.Size(280, 80);
this.groupBox1.TabIndex = 0;
this.groupBox1.TabStop = false;
this.groupBox1.Text = "Settings";

//
// cmdClose
//
this.cmdClose.Enabled = false;
this.cmdClose.Location = new System.Drawing.Point(160, 48);
this.cmdClose.Name = "cmdClose";
this.cmdClose.Size = new System.Drawing.Size(96, 24);
this.cmdClose.TabIndex = 5;
this.cmdClose.Text = "Close";
this.cmdClose.Click += new System.EventHandler(this.cmdClose_Click);

//
// cmdConnect
//
this.cmdConnect.Location = new System.Drawing.Point(160, 16);
this.cmdConnect.Name = "cmdConnect";
this.cmdConnect.Size = new System.Drawing.Size(96, 24);
this.cmdConnect.TabIndex = 4;
this.cmdConnect.Text = "Connect to eBox";
this.cmdConnect.Click += new System.EventHandler(this.cmdConnect_Click);

//
// txtPort
//
this.txtPort.Location = new System.Drawing.Point(72, 48);
this.txtPort.Name = "txtPort";
this.txtPort.Size = new System.Drawing.Size(40, 20);
this.txtPort.TabIndex = 3;
this.txtPort.Text = "8221";

//
// label2
//
this.label2.Location = new System.Drawing.Point(16, 48);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(32, 16);
this.label2.TabIndex = 2;
this.label2.Text = "Port:";

//
// txtIPAddr
//
this.txtIPAddr.Location = new System.Drawing.Point(72, 24);
this.txtIPAddr.Name = "txtIPAddr";
this.txtIPAddr.Size = new System.Drawing.Size(80, 20);
this.txtIPAddr.TabIndex = 1;
this.txtIPAddr.Text = "172.28.74.168";

//
// label1
//
this.label1.Location = new System.Drawing.Point(16, 24);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(48, 16);
this.label1.TabIndex = 0;
this.label1.Text = "eBox IP:";
//
// txtDataTx
//
this.txtDataTx.Location = new System.Drawing.Point(16, 104);
this.txtDataTx.Multiline = true;
this.txtDataTx.Name = "txtDataTx"
this.txtDataTx.ScrollBars = System.Windows.Forms.ScrollBars.Vertical;
this.txtDataTx.Size = new System.Drawing.Size(224, 56);
this.txtDataTx.TabIndex = 1;
//
// cmdSend
//
this.cmdSend.Location = new System.Drawing.Point(59, 78);
this.cmdSend.Name = "cmdSend"
this.cmdSend.Size = new System.Drawing.Size(130, 25);
this.cmdSend.TabIndex = 2;
this.cmdSend.Text = "Send to eBox"
this.cmdSend.Click += new System.EventHandler(this.cmdSend_Click);
//
// txtDataRx
//
this.txtDataRx.Location = new System.Drawing.Point(8, 18);
this.txtDataRx.Multiline = true;
this.txtDataRx.Name = "txtDataRx"
this.txtDataRx.ScrollBars = System.Windows.Forms.ScrollBars.Vertical;
this.txtDataRx.Size = new System.Drawing.Size(256, 72);
this.txtDataRx.TabIndex = 3;
//
// groupBox2
//
this.groupBox2.Controls.Add(this.txtDataRx);
this.groupBox2.Location = new System.Drawing.Point(8, 203);
this.groupBox2.Name = "groupBox2"
this.groupBox2.Size = new System.Drawing.Size(280, 96);
this.groupBox2.TabIndex = 4;
this.groupBox2.TabStop = false;
this.groupBox2.Text = "Data Arrived";
//
// groupBox3
//
this.groupBox3.Controls.Add(this.cmdSend);
this.groupBox3.Location = new System.Drawing.Point(8, 88);
this.groupBox3.Name = "groupBox3"
this.groupBox3.Size = new System.Drawing.Size(280, 109);
this.groupBox3.TabIndex = 5;
this.groupBox3.TabStop = false;
this.groupBox3.Text = "Send Data";
// Form1
//
this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
this.ClientSize = new System.Drawing.Size(292, 311);
this.Controls.Add(this.txtDataTx);
this.Controls.Add(this.groupBox1);
this.Controls.Add(this.groupBox2);
this.Controls.Add(this.groupBox3);
this.Name = "Form1";
this.Text = "Remote Client ";
this.groupBox1.ResumeLayout(false);
this.groupBox1.PerformLayout();
this.groupBox2.ResumeLayout(false);
this.groupBox2.PerformLayout();
this.groupBox3.ResumeLayout(false);
this.ResumeLayout(false);
this.PerformLayout();

#endregion

/// <summary>
/// The main entry point for the application.
/// </summary>
/// <STAThread>
static void Main()
{
    Application.Run(new Form1());
}

private void EnableCommands( bool abEnableConnect )
{
    cmdClose.Enabled = !abEnableConnect;
    cmdConnect.Enabled = abEnableConnect;
}

private void cmdConnect_Click(object sender, System.EventArgs e)
{
    try
    {
        EnableCommands(true);
        //create the socket instance...
        m_socClient = new Socket
        (AddressFamily.InterNetwork,SocketType.Stream ,ProtocolType.Tcp );

        // get the remote IP address...
        IPAddress ip = IPAddress.Parse (txtIPAddr.Text);
        int iPortNo = System.Convert.ToInt16 ( txtPort.Text);
        //create the end point
        IPEndPoint ipEnd = new IPEndPoint
        (ip.Address,iPortNo);

        //connect to the remote host...
        m_socClient.Connect ( ipEnd );
        EnableCommands(false);
        //watch for data ( asynchronously )...
    }
    catch
    {...
public void WaitForData()
{
    try
    {
        if  ( pfnCallBack == null )
        {
            pfnCallBack = new AsyncCallback (OnDataReceived);
        }
        CSocketPacket theSocPkt = new CSocketPacket ();
        theSocPkt.thisSocket = m_socClient;
        // now start to listen for any data...
        m_asynResult = m_socClient.BeginReceive (theSocPkt.dataBuffer ,0, theSocPkt.dataBuffer.Length,
        ,SocketFlags.None,pfnCallBack, theSocPkt);
    } catch(SocketException se)
    {
        MessageBox.Show (se.Message );
    }
}

public void OnDataReceived(IAsyncResult asyn)
{
    try
    {
        CSocketPacket theSockId = (CSocketPacket)asyn.AsyncState;
        //end receive...
        int iRx = 0;
        iRx = theSockId.thisSocket.EndReceive(asyn);
        char[] chars = new char[iRx + 1];
        int charLen = d.GetChars(theSockId.dataBuffer, 0, iRx, chars, 0);
        System.String szData = new System.String(chars);
        AppendRxText(szData);
        WaitForData();
    } catch (SocketException se)
    {
        MessageBox.Show(se.Message);
    }
}

private void AppendRxText(string text)
if (this.txtDataRx.InvokeRequired)
{
    AppendTextCallback d = new AppendTextCallback(AppendRxText);
    this.Invoke(d, new object[] { text });
}
else
{
    txtDataRx.Text = txtDataRx.Text + text;
}

private void cmdSend_Click(object sender, System.EventArgs e)
{
    try
    {
        Object objData = txtDataTx.Text;
        byte[] byData = System.Text.Encoding.ASCII.GetBytes(objData.ToString());
        m_socClient.Send(byData);
    }
    catch(SocketException se)
    {
        MessageBox.Show(se.Message);
    }
}

private void cmdClose_Click(object sender, System.EventArgs e)
{
    if (m_socClient != null)
    {
        m_socClient.Close();
        m_socClient = null;
        EnableCommands(true);
    }
}

public class CSocketPacket
{
    public System.Net.Sockets.Socket thisSocket;
    public byte[] dataBuffer = new byte[1];
}
}
Appendix C

C# Windows CE Video Server Application Source Code

using System;
using System.Collections.Generic;
using System.Text;
using System.Net.Sockets;
using System.Net;
using System.IO;
using System.Drawing;
using System.Diagnostics;
using System.Drawing.Imaging;
using WebCam;
using System.Threading;

namespace VideoHostCE
{
    class BitmapStreamer
    {
        private const int LISTEN_PORT = 18888;

        TcpListener tl;
        bool runBool;

        public bool RunBool
        {
            get { return runBool; }
            set { runBool = value; }
        }

        public const int LOW_LIMIT = 24; // -23
        public const int HI_LIMIT = 198; // 232.99

        private ManualResetEvent goTime;

        Camera camera = new Camera();

        // Format of 1, 1, 3 is 160x120 at 15 fps
        int m_nFormat = 1;
        int m_nFrame = 3;
        int m_nInterval = 3;

        WebCamSDK.FORMATPROPS[] m_formats;

        #region Init Camera
        /// <summary>
        /// Initialize a camera driver
        /// </summary>
        /// <returns></returns>
        #endregion
bool InitCamera()
{
    string err;

    // See if we can talk to the camera
    if (!camera.InitCamera("CAM1:", out err))
    {
        return false;
    }

    m_formats = camera.Formats.ToArray();

    GetFormatInformation();
    return true;
}
#endif

void GetFormatInformation()
{
    // Get information about the format
    WebCamSDK.FORMATPROPS fmtData;
    int rc = camera.GetFormatInformation((ushort)m_nFormat,
                                          (ushort)m_nFrame, out fmtData);
}

static int m_nSize;                // Size of the frame data
static bool m_fRunning = false;     // Flag for controlling the read

thread
static bool m_fShowFrame = true;    // Flag for drawing
static int m_nFrameCnt = 0;
static int m_nLastCnt = 0;
static int m_nErr = 0;
static int m_nMissed = 0;

private delegate Bitmap ShowFrameData(IntPtr address);

// Refresh frame data in the picture box
private Bitmap ShowFrameDataMethod(IntPtr address)
{
    if (address != IntPtr.Zero &amp;&amp; m_fShowFrame)
    {
        Stream stream = camera.MJPEG2JPEG(address, m_nSize);
        try
        {
            return new Bitmap(stream);
        }
        catch
        {
            finally
            {
                stream.Close();
            }
        }
    }
    return new Bitmap(1,1);
public BitmapStreamer()
{
    tl = new TcpListener(IPAddress.Any, LISTEN_PORT);
    goTime = new ManualResetEvent(false);

    Console.WriteLine("q"); // ****
    // s = new Servo();
    Console.WriteLine("2"); // ***
    // ik = new InterfaceKit();
    Console.WriteLine("3"); // ***

    /// s.PositionChange += new Phidgets.Events.ServoPositionChangeEvent(s_PositionChange);
    /// // ik.SensorChange += new Phidgets.Events.SensorChangeEvent(ik_SensorChange);

    try
    {
        Console.WriteLine("4"); // **
        // s.open();
        Console.WriteLine("5"); // **
        // ik.open();
        Console.WriteLine("6"); // **
    }
    catch (Exception e)
    {
        Console.Out.WriteLine(e.Message);
        Console.Out.WriteLine(e.StackTrace);
    }
}

unsafe internal void Listen()
{
    Console.WriteLine("Listen");
    if (!InitCamera())
    {
        return;
    }

    tl.Start();

    #region CameraOneTime
    ShowFrameData show = new ShowFrameData(ShowFrameDataMethod);

    int rc = 0;
    IntPtr address;
    uint nSize;
    int nTimeout = 10000;
uint nMissed;

m_nFrameCnt = m_nLastCnt = m_nMissed = m_nErr = 0;

// Find the frames per second value
int nFrameInterval = -1;  // Assume slowest interval

if (m_nInterval < m_formats[m_nFrame - 1].nNumInterval)
    fixed (int* p = m_formats[m_nFrame - 1].nInterval)
    {
        nFrameInterval = p[m_nInterval];
    }

rc = camera.GetFirstStreamFrame((ushort)m_nFormat,
(ushort)m_nFrame, (uint)nFrameInterval,   out address, out nSize,
out nTimeout);

m_nSize = (int)nSize;

m_fRunning = true;

#endregion

rc = 0;
Console.WriteLine("Ouside loop");
while (runBool)
{
    TcpClient client = tl.AcceptTcpClient();
    NetworkStream ns = client.GetStream();
    BinaryWriter bw = new BinaryWriter(ns);
    BinaryReader br = new BinaryReader(ns);

    Font f = new Font(FontFamily.GenericSansSerif, 20,
FontStyle.Regular);
    SolidBrush brush = new SolidBrush(Color.RoyalBlue);

    try
    {
        Console.WriteLine(rc);
        Console.WriteLine(m_fRunning);
        while (rc == 0 && m_fRunning && runBool)
        {
            Bitmap b = new Bitmap(1, 1);

            m_nFrameCnt++;

            int nFlags = WebCamSDK.GETFRAMEFLAG_GET_LATESTFRAME;
            nFlags |= WebCamSDK.GETFRAMEFLAG_FREEBUFF_VALID;
            nFlags |= WebCamSDK.GETFRAMEFLAG_TIMEOUT_VALID;

            rc = camera.GetNextStreamFrame(ref address, out
nSize, out nMissed, nFlags, nTimeout);
            if (rc == 0)
            {
                m_nSize = (int)nSize;
                m_nMissed += (int)nMissed - 1;
b = ShowFrameDataMethod(address);
//Thread.Sleep(0);

} else {
    m_nErr++;
    Console.Out.WriteLine("rc not 0"); //**
}

if (ns.DataAvailable)
{
    Console.WriteLine("Data Avail");
    string input = br.ReadString();
    ProcessStartInfo psi = new ProcessStartInfo();
    psi.UseShellExecute = false;
    psi.FileName = @"\ServoHost\Servo\ServoControllerCE.exe";
    psi.Arguments = input;
    Process.Start(psi);
    Console.WriteLine(input);
    Console.WriteLine(int.Parse(input));
    //ServoPosition = int.Parse(input);
}

Graphics g = Graphics.FromImage(b);
//compass = ik.sensors[2].Value;
//string gString = compass.ToString() + "\xb0";
// g.DrawString(gString, f, brush, 0, 0);

MemoryStream ms = new MemoryStream();
b.Save(ms, ImageFormat.Bmp);
byte[] toSend = ms.ToArray();
ms.Close();

//Console.Out.WriteLine(toSend.Length);
bw.Write(toSend.Length);
bw.Write(toSend);

Thread.Sleep(100);
}
}

internal void Close()
{
    tl.Stop();
    camera.CloseStreamFrame();
    Console.WriteLine("EXIT"); //**
}
Appendix D

C# Windows Remote Video Viewer Client Application Source Code

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net.Sockets;
using System.IO;
using System.Threading;

namespace VideoClientWin
{
    public partial class Form1 : Form
    {
        TcpClient client;
        delegate void PicHandler(Bitmap b);

        public Form1()
        {
            InitializeComponent();

            client = new TcpClient();
            client.ReceiveBufferSize = 200000;
        }

        private void Go(IAsyncResult ia)
        {
            TcpClient tc = ia.AsyncState as TcpClient;

            while (!tc.Connected)
            {
                continue;
            }
            NetworkStream ns = tc.GetStream();
            BinaryReader br = new BinaryReader(ns);

            try
            {
                while (true)
                {
                    if (client.Available > 0)
                    {
                        int len = br.ReadInt32();
                        if (len > 100)
                        {
                            byte[] bt = br.ReadBytes(len);
                        }
                    }
                }
            }
            catch
            {
                Console.WriteLine("Network failure.");
                throw;
            }
        }
    }
}
MemoryStream ms = new MemoryStream(bt);
Bitmap b = new Bitmap(ms);
ms.Close();
this.Invoke(new PicHandler(UpdateImage), new object[] { b });
}

Thread.Sleep(0);