JAVA PROGRAMMING FOR ONLINE QUALITY CONTROL LABORATORY INTEGRATED WITH REMOTE ROBOT

Richard Chiou[1], Yongjin (James) Kwon[2], and Tzu-Liang (Bill) Tseng[3]

[1]Applied Engineering Technology
Goodwin College of Professional Studies
Drexel University, Philadelphia, PA

[2]Industrial and Information Systems Engineering
College of Engineering
Ajou University, South Korea

[3]Mechanical and Industrial Engineering
College of Engineering
University of Texas, El Paso, TX

Abstract

This paper presents computer Java programming for laboratory development on real-time quality control integrated with web-controllable robots. The remote quality monitoring system (referred to as E-quality) allows users to monitor the part status including data, image, and video through the Internet. The use of Java allows for interconnection and data transfer between the different components of the Web-based quality system and the robots. The ability to access and control quality control devices via the Web benefits the current manufacturing environment with access to the production floor, remote control/programming /monitoring capability, and integration of production equipment into information networks for improved process efficiency and product quality. The networked hardware and software components are integrated with quality methodologies to achieve maximum effective-ness in teaching E-quality concepts. The remote quality control laboratory has been developed for course delivery for engineering and technology education. The environment is also suitable for engineering undergraduate and graduate research work.

Introduction

In tomorrow's factory, design, manufacturing, quality, and business are integrated into the Internet[1-4]. It is a trend that Web based gauging, measurement, inspection, diagnostic system, and quality control have become critical issues in the integration with e-manufacturing systems and management. The constituents include highly advanced form of production equipment and sensor networks connected to the Internet, all of which identified by the unique Internet Protocol (IP) addresses. The remote accessibility and the ability to control the equipment over the Internet present unprecedented benefits to the current manufacturing environment. Designers located remotely from the production facility can carry out the inspection and quality inspection as their design processes evolve. The quality control issues and the positioning tolerance analysis of equipment can be tested according to the assembly and manufacturing specifications. Any changes in the product specifications and associated quality control routines can be instantly updated and verified, which will enhance the overall production efficiency. In the event of defective products, the operators are warned immediately through telecom-munications networks, such as text messages, email, and phone calls[5-7].

The objective of the project is to allow engineering and technology students to experience with such a system in Web-based applied quality control[8-11]. The system currently allows a student to remotely operate and monitor the inspection process through the Web and collect data about the ongoing process for future analysis. This system gives a vivid picture on the integration of robotics, machine vision systems, and sensors via the Internet, leading to a rich online laboratory learning experience[12-15]. It is the most popular choice for Web-based applications and is a proven technology for client–server communications deployed through embedded network systems. The firmware and associated scripting tools in the hardware are closely based on Java and this allows seamless operation with other Java applications. For a Web-based system, Java provides the best range of options to implement an efficient client – server system. Java Applets allows clients to build extremely user friendly and intuitive interfaces that can be deployed over a Web browser. Java software technology has been the most extensively supported by the Web browsers that are currently available in the market. Subsequently using a Java based application server to communicate with this Applet based client is the best choice for efficient and quick system development. Java makes it easy to move the code to another platform which is a common requirement in an academic environment.

Web-Based Quality Control System

The underlying idea in a Web based e-quality control systems is to have an application server that can communicate and control all the entities involved in the manufacturing and monitoring process. In our system, this server communicates with the machine vision system that is capable of making real-time inspections on the products and the robot that performs the required corrective action. It is also the point to which any remote user would connect to through their Web browsers to monitor and obtain information on the ongoing process. Figure 1 shows the system setup for remote quality control integrated with robot and machine vision[16-18]. It is composed of a Yamaha SCARA robot, RCX40 robot controller with on-board Ethernet card, Cognex DVT 540 machine vision sensor, DLink DCS-5300 Web cameras and a PC for hosting the application server and Web server. All these devices are Internet enabled – i.e. these devices can all be connected to a TCP/IP network through an Ethernet port and can be addressed through an IP address. Once hooked up to the network through a UTP Cat5 cable, these devices become unique nodes in a TCP/IP network and are able to communicate with other entities on the network. The Drexel University network is an extensive TCP/IP network running on Ethernet and wireless Ethernet. All IP addresses are of version 4.

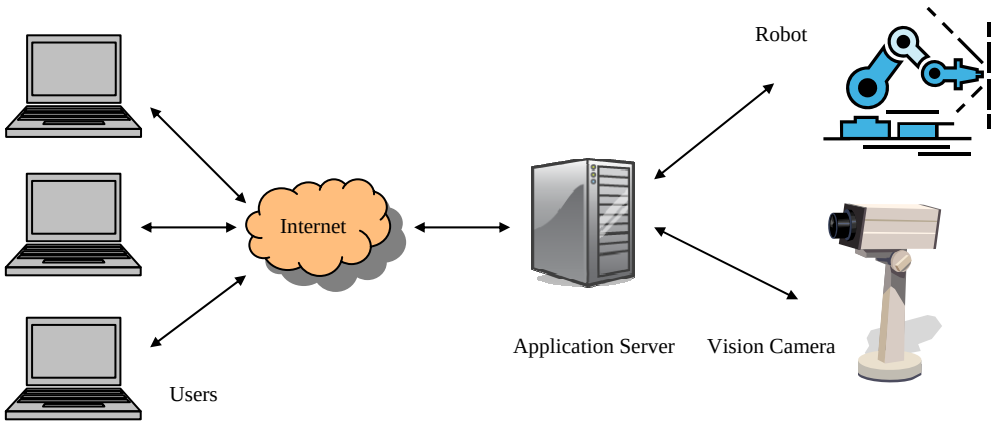

Figure 1. System Setup.

The RCX40 robot controller runs a Telnet daemon on TCP port 23, which is a program that allows Telnet clients to connect to it and communicate with it. Telnet is one of the oldest text based protocol for remote communications. The daemon allows a Telnet client to connect to it and type in textual commands which in turn are interpreted by the internal firmware and executed as manual actions by the robot. The machine vision sensor also has extensive TCP/IP communication facilities. Though it does not provide a standard protocol interface, it has an in-built scripting facility that allows communication programs to be written and executed in it. The communication program we implemented is explained in later sections.

The application server runs on a dedicated PC. It waits for connections to be made from clients. Once a connection is made, it initiates connections with the vision sensor and the robot and establishes a complete communication topology for the purposes of our system. We also run a Web server for serving the Applet which forms our client interface. Our choice is the Apache Web server version 2.0.59 running on a Windows XP platform. Apache is by far the most common and efficient Web server. It is also distributed for free by the Apache foundation. The server listens on TCP port 80. The Applet is stored under the Web server and when a client requests for the Web page containing this Applet through his browser, it responds by sending this page and applet to the client. Once received, the applet is loaded in the browser and starts executing. After all the connections are established, information flows from each device and client to the application server, in turn, redistributes it. All the devices are password protected for enhanced security.

At the core of this system is the Java-based software that integrates all the hardware components and provides the user with a unified view of the system in terms of quality control operation and information exchange. The software is implemented as three separate entities: scripts on the vision sensor, the application server, and the Applet based client interface. Figure 2 shows the overall software structure implemented in the Web-based quality control system. The measurements made by the machine vision sensor is formatted and transferred to the application server through two scripts. Inter-script communication between these two scripts happens by setting and clearing status flags in the common memory registers. The first script, called the inspection script, is executed after each snapshot. The second script, called the main script, runs constantly in the background. It is responsible for establishing and maintaining communications with the application server. It uses the standard Java style function *Socket* to open a TCP connection to the application server on a dedicated port. Once the connection is established, it exchanges information on this connection with the application server through the functions *send* and *receive*.

System Programming Structure

The application server is a communications program written in Java. The operations of the program are divided and implemented through three main Java classes and a main program in accordance with the object-oriented programming model. The main program is the part of the code that is executed first when the application server is started. The main program is shown in Figure 2.

The main program starts by waiting for a TCP connection on a predetermined port from a browser through the *ServerSocket* class object. Since the end user has to activate the inspection process, this is deemed as the appropriate entry point into the program. This connection is initiated by the browser through a Java Applet. Once a connection is accepted through the *accept* method, an object of *BrowserSession* class, which is a class designed to handle all the browser communication aspects is initialized with this new connection. This newly initialized object is then used as a parameter to initialize an object of the *Operations* class. The *Operations* class is designed to operate the communications with the machine vision camera and the robot.

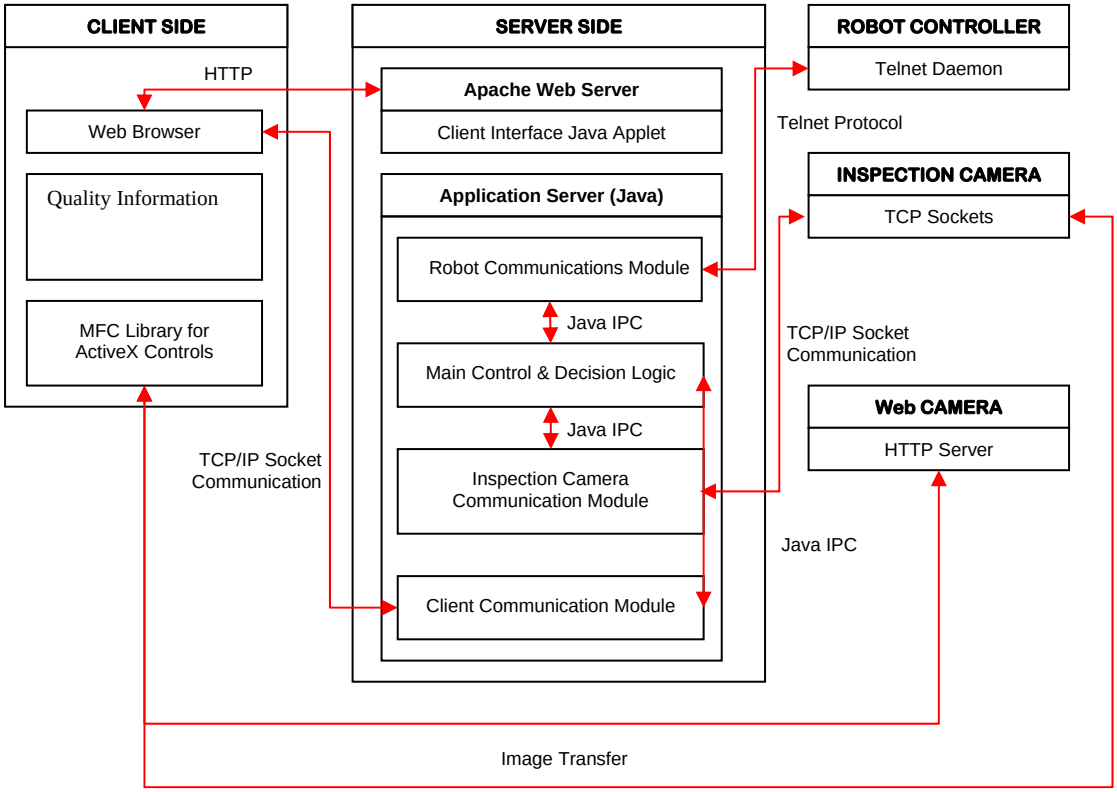

Figure 2. Java Programming Structure.

The *Operations* object implements the Java system class—therefore being able to run as a thread. A thread is a mini-program within a program that can execute in parallel with other threads. The reason behind assigning the browser communications to a separate class and the robot-camera operations to another class should be evident from the way the system is organized. The browser forms the client part of the operations while the robot and camera constitute the server side of the system and hence this division. Once the initialization of the objects is completed, a new thread is started with the newly initialized *Operations* object and the thread is started. This effectively establishes the required connections between the entities in the system and starts operation.

```
public class Main  {
   public static void main(String[ ] args)throws Exception {
            ……..
            ServerSocket BrowserListenSocket = new ServerSocket(BrowswerListenPort);
            BrowserSession ThisBrowserSession =  BrowserSession(BrowserListenSocket.accept());
            Operations ThisOperation = new Operations  (This BrowserSession);
           Thread CurrentThread = new Thread (ThisOperation);
            CurrentThread.start();
            …….
  }
 }
```

Communication with Web browser

The *BrowserSession* encapsulates all operations pertaining to the communication with the browser. Its constructor initializes an *Socket* object with the new connection passed on to it by the main program.

```
private Socket BrowserConnectionSocket;
public BrowserSession(Socket NewSocket) {
BrowserConnectionSocket = NewSocket;
}
```

The class provides a *WriteMessage()* method through which information is sent to the browser. The functions of the *isAlive()* and *Close()* methods are self-evident.

```
private BrowserSession ThisSession;
public Operations(BrowserSession bs) {
ThisSession = bs;
}
```

It accepts an instant of the *BrowserSession* class as an input which effectively gives it access to the connection with the browser. It then utilizes the *TelnetWrapper* class for communicating with the telnet daemon on the robot. An instance of the *Operations* class, once run as a thread, starts with opening a connection and initializing the robot.

```
   public boolean WriteMessage(String Message) {
      try {
         if (BrowserStream == null)
            BrowserStream = new PrintWriter(BrowserConnectionSocket.getOutputStream());
         BrowserStream.println(Message);
         BrowserStream.flush();
         return true;
      } catch (IOException Error) {
         return false;
      }
   }
   public boolean isAlive() {
……..
   }
   public void Close() {
……...
   }
}
```

Web-Based Robotic Operations

The main workhorse of this system is the *Operations* class. The constructor for this class is as follows:

```
RobotConnection = new TelnetWrapper(RobotIP);
RobotConnection.setPrompt("");
RobotConnection.login("USER", "PASSWORD");
RobotConnection.wait("OK",10000);
ThisSession.WriteMessage(
        RobotConnection.sendCommand("@DO(21)=1") + // Start conveyor belt
        RobotConnection.sendCommand("@DO(23)=0") + // Stop suction pump
        RobotConnection.sendCommand("@SERVO ON") +        // Turn on servo
        RobotConnection.sendCommand("@MOVE L,P126")       // Move to rest posn.
        )
```

The initialization sequence composes of four steps. It enables the digital output 21 connected to the robot's I/O controller. This output is essentially the conveyor belt turn on/off signal and therefore starts it. Digital output 23 is connected to the vacuum device which is turned off at the beginning. It is turned on when the robot is required to pick up an object. The "SERVO ON" statement turns on the robot's servo motor. The next command moves the robot to a rest position which is pre-defined by the point P126 in the robot controller's memory.

Integration with Remote Quality Inspection

The next step is to establish connection with the camera, again through a TCP/IP socket on a pre-assigned port.

```
ServerSocket Camera = new ServerSocket
   (CameraPort);
ThisConnection = Camera.accept();
```

Once connected, it continuously waits for messages from the camera that include measurements, object position and other status messages. It processes this incoming measurement information to decide on the quality of the product. Once a decision is made it instructs the robot to perform the necessary action on the inspected object.

```
BufferedReader FromCamera = new BufferedReader(
                new InputStreamReader(ThisConnection.getInputStream()));
while (true)
{
MessagefromCamera = FromCamera.readLine();
if (Request.equals("MessagefromCamera "))
{
        ThisSession.WriteMessage ("Object Detected \n" +
                RobotConnection.sendCommand("@DO(21)=0"));      \\ Stop belt
}
else
{
        .           // Parse Dimensions
       ThisSession.WriteMessage("Dimensions: " + Dimensions +
                RobotConnection.sendCommand("@MOVE L, " + X + " " + Y + " " + Z + " " +
R) +
                RobotConnection.sendCommand("@DO(23)=1"));    \\ Move to object & pick up
       if (Dimensions_Are_Not_Correct)
```

```
    {
            ThisSession.WriteMessage("Non Compliant Object\n
                RobotConnection.sendCommand("@MOVE L, P126, P125"));
            \\ Place object in non-compliant stack
     }
    else
    {
             ThisSession.WriteMessage("Compliant Object\n"
                RobotConnection.sendCommand("@MOVE L, P127, P128"));
             \\ Place object in compliant stack
     }
     .       //Reset  Operations
}
}
```

The 'Dimesions_Are_Not_Correct' phrase is a set of conditions to check the dimensions. A simple example would be (Length = 50 AND width = 50), though in the actual code this is a complex conditional statement incorporating all 7 parameters being measured. Depending on the object classification, it is picked and placed on to a stack defined by the pre-defined positions P125 and P128.

Communication with Machine Vision

The DVT vision sensor supports a Java style scripting language through which the communications program was implemented on it to communicate and transfer data with the application server. Part of the code to establish communications with the application sever is shown below:

The similarity with Java is readily seen from the code. This part of the code keeps trying to connect to the application server until it successfully makes connection. When measurements are available they reported back to the application server using the Send() method. The text message is converted into a Byte array as required by the Send() method using the toByteArray() method.

```
ToServer.Send(Message.toByteArray())
```

Web Interface

The end user interface is implemented on a Web browser through a Java Applet. Information is also exchanged between the applet and the application server through a

```
class RobotControl
{
        public static void main ()
        {
        Socket ToServer = new Socket();
        while (true)
        {
                ConnectResult = ToServer.Connect("144.188.xx.xx", 1500);
                If (ConnectResult<0)
                        continue;
        }
        DebugPrint("Connected to Server");
……
}
```

TCP/IP connection. The graphical user interface is implemented through the use of Java swing components. The Applet also follows a similar structure to that of a normal Java program, but it is not required to implement a main() method.

```
public class WebInterface extends java.applet.Applet implements Runnable {
   Thread thisThread;
   Socket toApplicationServer;
   PrintWriter  outApplicationServer;
   BufferedReader inApplicationServer;
}
```

The Applet does not execute any task upon loading. The user initiates the inspection procedure by clicking on the 'Start Inspection' button. Once clicked, it attempts to establish a connection with the application server. Successful establishment of connection also starts the inspection process at the other end as explained in earlier sections and the inspection process begins.

```
 private void iStartActionPerformed(java.awt.event.ActionEvent evt) {
       {
          AddText("Connecting to Application Sever at " + sIP + " on Port " + sPort, 0);
          toApplicationServer = new Socket(sIP, sPort);
          AddText("Connected to Application Server", 0);
          iStart.setEnabled(false);
          iStop.setEnabled(true);
       } catch (IOException ex) {
          AddText("Could not Connect to Application Server", 0);
          AddText("Try Again", 0);
          return;
       }
       doRun = true;
       thisThread = new Thread(this);
       thisThread.start();
    }
```

Once the connection is established the applet executes the run() method in a separate thread. The run() method actively exchanges information with the application server. The reason to dedicate a separate thread for the information exchange is that, the main thread is responsible for displaying and making any changes to the visual display of the applet and if the I/O operations are dedicated to this thread, it fails to refresh the display when any changes occur.

```
public void run() {
……..
      try {
         inApplicationServer = new BufferedReader(new
                        InputStreamReader(toApplicationServer.getInputStream()));
         while(doRun) {
            Text = inApplicationServer.readLine();
            if (Text.startsWith("Object")) {
               Temp = Integer.parseInt(iTotal.getText());
               Temp++;
               iTotal.setText(Temp.toString());
            }
            if (Text.startsWith("Non Compliant")) {
               Temp = Integer.parseInt(iNo.getText());
               Temp++;
               iNo.setText(Temp.toString());
            }
            if (Text.startsWith("Compliant")) {
               Temp = Integer.parseInt(iYes.getText());
               Temp++;
               iYes.setText(Temp.toString());
            }
            if (Text.startsWith("Dimensions")) {
               Dimensions = Text.substring(12).split(",");
               iLength.setText(Dimensions[0]);
               iWidth.setText(Dimensions[1]);
               iRad1.setText(Dimensions[2]);
               iRad2.setText(Dimensions[3]);
               iCenCen.setText(Dimensions[4]);
               iAngle.setText(Dimensions[5]);
            }
         }
      } catch (Exception ex) {
         AddText("Could not read from Application Server\n" + ex.getMessage() + "\n",  0);
         return;
      }
```

The run() method essentially waits for any message from the application server. Once it reads a message, it interprets and performs the appropriate update. The message 'Object' conveys the message that an object has been inspected. Therefore it updates the iTotal field which keeps track of the total number of objects that are inspected. A 'Non Compliant' means that the inspected object failed to comply with the required specifications, which leads to incrementing the iNo field which keeps track of the total number of non-compliant objects. Similar explanation applies to the 'Compliant' message. The 'Dimensions' message reports the measurement values of the object. The program parses the data and displays the information under the appropriate fields.

Experimental Verification

Web-based quality control systems can provide remote sensing, monitoring, and online quality diagnosis for robotics and automation in manufacturing processes. The quality control

issues and the tolerance analysis of an object can be tested according to the assembly and manufacturing specifications. Any changes in the product specifications and associated quality control routines can be instantly updated and verified, which will enhance the overall production efficiency. Figure 3 shows the setup used for the Web-based quality control. The experimental setup includes the following items: Yamaha SCARA robot YK-250X, RCX40 robot controller with optional on-board Ethernet card, Cognex DVT 540 machine vision, and HP m1050e PCs. The RCX40 controller is connected to the Ethernet and controlled using a PC/Server. Two DLink DCS-5300 Web cameras are used for constantly viewing the robot movement.

All devices, such as the robot, Web cameras, and machine visions, are connected to the Ethernet. This reduces the wire maze needed to link every device and enables users to operate and control the equipment remotely. The word network refers to the connection between the devices in the work area as well as with the users. Network used here is local area network (LAN), which is connected to a server. The Web server is connected to this LAN, and the LAN can also have access to the devices on the network from the outside. Every device connected to the network has a unique IP address, which is used to connect to them and also recognize them on the network. The IP address is separated into network address and host address sections. The network address section is extracted from the IP address by *AND* processing with the subnet mask. Devices belonging to the same network must be set to have the same network address.

The browser interface also includes two Web camera views that show the robotic operation (see Figure 4). The Web cameras are equipped with an embedded Web server, which captures stream of images over the Internet using the HTTP protocol. Our Web interface incorporates a Microsoft ActiveX object provided by the manufacturer, and this allows us to directly embed the camera view in our Webpage with

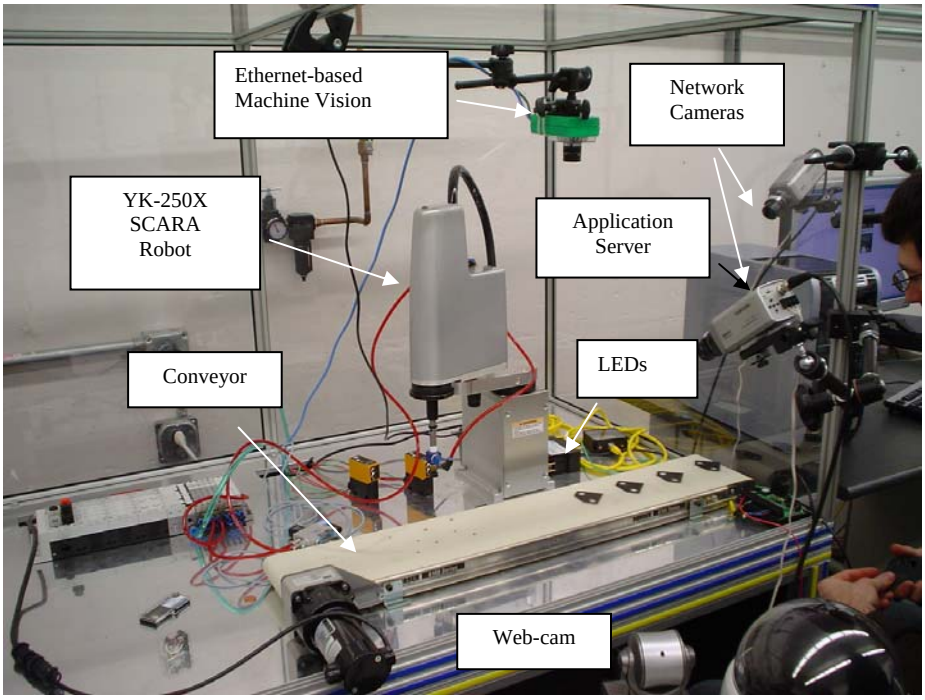

Figure 3. A System Setup for Web-based Quality Control.

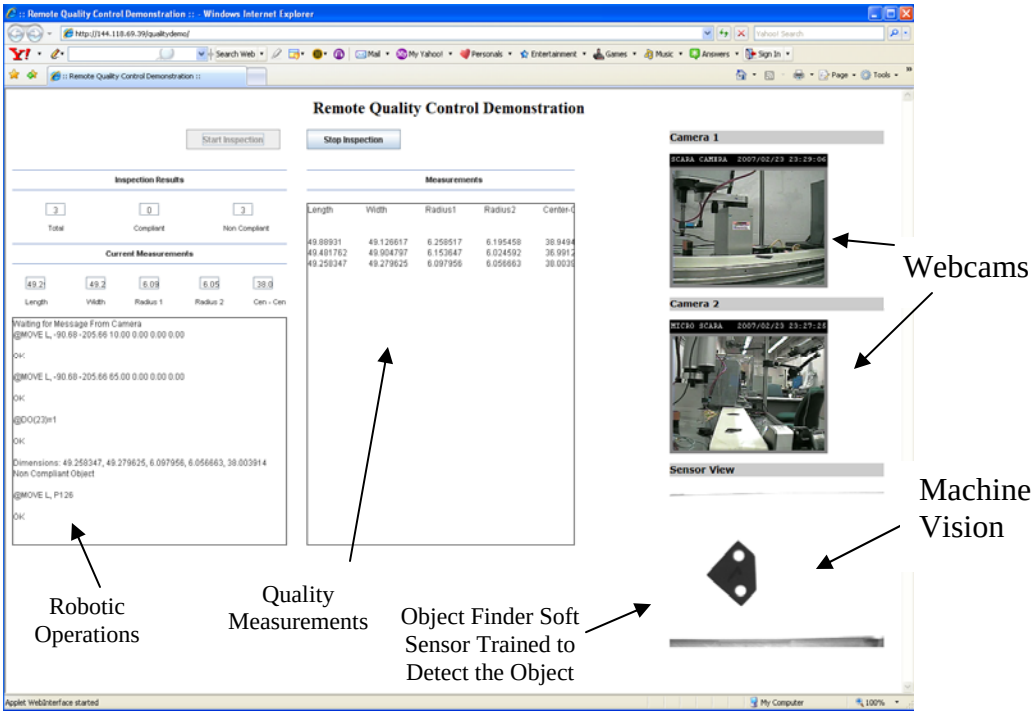

Figure 4. Web Interface for the Quality Measurement and Robotic Operations.

minimal configuration. Information is exchanged between the Applet and the application server through a TCP/IP connection. Figure 4 shows a screen shot of the Web-based end-user interface. The Applet does not execute any task upon loading. The user initiates the inspection procedure by clicking on the "Start Inspection" button. Once clicked, it attempts to establish a connection with the application server. Successful establishment of the connection also starts the inspection process at the other end, as explained in earlier sections.

There are several fields in the Applet that gives a live analysis of the ongoing inspection process such as the number of objects inspected, number of compliant objects, current dimensions, and cumulative measurement history. Since the programs, data, and access information are stored centrally, changes are made available to users immediately. The current Web technology also provides for extensive interactivity allowing for real-time interaction with the system.

Conclusions

The development of the Web-based laboratory positively impacts upon education, research, and service at the Drexel's Applied Engineering Technology curriculum. This work successfully demonstrates a concept of E-quality through the implementation of a Web-based quality control system using Java programming software. A computer vision system measures the dimensions of products on the conveyor belt and sends the information to an application server, which activates an appropriate action for the robot. Various image processing and analysis algorithms are integrated for remote quality inspection. Operators can remotely adjust the inspection routine in the case of process changes, such as lighting conditions, part variations, and quality criteria. This result provides a great impact in production since engineers can access and control the equipment and quality anytime, from anywhere. From educational point of view, such setting provides a convenience to not only place-bound students, but also to many students who work during the

day time, and consequently having not enough time to complete their lab assignments during the normal class hours. Due to such benefits, it is expected that online laboratory will continue to grow in terms of available numbers as well as the level of sophistications in the coming years.

Acknowledgement

This work was supported by the US National Science Foundation (CCLI Phase II DUE-0618665), the US Dept. of Education (Award # P116B060122) and Yamaha Robotics Company. This work was also supported by the 2007 Ajou University Faculty Start-up Funding for Research and Development. The authors wish to express sincere gratitude for their financial support.

References

1. Jose, Joao and Ferreira, Pinto, "E-Manufacturing: Business Paradigms and Supporting Technologies," Kluwer Academic Publishers, Dordrecht, The Netherlands. Pp. 280, 2004.

2. Michel, Roberto, "E-manufacturing Essentials," Manufacturing Systems (MSI), Vol. 18, Issue 5, pp. 36, May 2000.

3. Ming, X. G., Yan, J. Q., Lu, W. F., and Ma, D. Z., "Technology Solutions for Collaborative Product Lifecycle Management – Status Review and Future Trend," Concurrent Engineering, Vol. 13, pp. 311 – 319, Dec. 2005.

4. Lacroix, Eric and St-Denis, Richard, "Web Technologies in Support of Virtual Manufacturing Environments," Proc. Conf. on Emerging Technologies and Factory Automation, Lisbon, Portugal, Vol. 2, pp. 43- 49, Sept. 2003.

5. Lee, Jay, "E-manufacturing—fundamental, tools, and transformation", Robotics & Computer-Integrated Manufacturing, Vol. 19, No. 6, pp. 501 – 508, Dec. 2003.

6. Wang, Lihui, Orban, Peter, Cunningham, Andrew, and Lang, Sherman, "Remote Real-time CNC machining for Web-based Manufacturing," Robotics & Computer-Integrated Manufacting, Vol. 20, No. 6, pp. 563-571, Dec. 2004.

7. Zhou, Guanghui, Jiang, Pingyu, and Fukuda, Shuichi, "Using Mobile Agents to Schedule a Manufacturing Chain on the Internet," Concurrent Engineering, Vol. 10, pp. 311-323, Dec. 2002.

8. Shen, Yantao, Xi, Ning, Lai, King W.C. and Li, Wen J., "Internet-based Remote Assembly of Micro-electro-mechanical systems (MEMS)," Assembly Automation, Vol. 24, No. 3, pp. 289-296, 2004.

9. Chiou, Richard, Kwon, Yongjin, Rauniar, Shreepud, and Sosa, Horacio, "Visual Basic Programming for Internet-based Robotic Control", Computers in Education Journal, p. 81, vol. XVII, April – June of 2007.

10. Kwon, Y., Rauniar, S., Chiou, R. & Sosa, H., "Remote Control of Quality Using Ethernet Vision and Web-enabled Robotic System" Journal of Concurrent Engineering: Research and Applications, Vol. 14, No. 1, pp. 35-42, 2006.

11. Kwon, Y., Chiou, R., Rauniar, S. & Sosa, H., 2006, "Positioning Accuracy Characterization of Precision Micro Robot over the Internet," Journal of Advanced Manufacturing Systems, Vol. 5, No.1, pp. 45-57, 2006.

12. Jiang, Pingyu and Zhang, Yingfeng, "Visualized Part Manufacturing via an Online e-Service Platform on Web," Concurrent Engineering, Vol. 10: pp. 267-277, Dec. 2002.

13. Wang, Lihui, Xi, Fengfeng and Zhang, Dan, "A Parallel Robotic Attachment and Its Remote Manipulation," Robotics and Computer-Integrated Manufacturing," Vol. 22, Issues 5-6, pp. 515-525, Oct-Dec 2006.

14. Wang, Lihui, Sams, Ryan, Verner, Marcel and Xi, Fengfeng, "Integrating Java 3D Model and Sensor Data for Remote Monitoring and Control," Robotics and Computer-Integrated Manufacturing, Vol. 19, Issues 1-2, pp. 13-19, Feb-Apr 2003.

15. Yang, Yu, Zhang, Xiaodong, Liu, Fei and Xie, Qiu, "An Internet-based Product Customization System for CIM," Robotics and Computer-Integrated Manufacturing, Vol. 21, Issue 2, pp. 109-118, April 2005.

16. Cognex Smart Image Sensor DVT 545 Series Manual.

17. Yamaha Robot Operations Manual RCX40.

18. Cognex Smart Image Sensor FrameWork Manual.

Biographical Information

Dr. Richard Chiou's background is in mechanical engineering with an emphasis on manufacturing. Dr. Chiou is currently an associate professor in the Goodwin College of Professional Studies at Drexel University. His areas of research include machining, mechatronics, and internet based robotics and automation. He has secured many research and education grants from the NSF, the SME Education Foundation, and industries.

Dr. Yongjin (James) Kwon is a professor in the Division of Industrial and Information Systems Engineering at Ajou University in South Korea. His research interests include web-enabled robotic systems, computer vision, 3 dimensional representation of remote systems, and e-quality for manufacture. Dr. Kwon has many years of academic experience and his research articles have been published in many international journals.