Telerobotic Decontamination and Decommissioning with QRobot, a PC-Based Robot Control System

Markus Loffler, Nick Costescu, Erkan Zergeroglu, Darren Dawson
Department of Electrical and Computer Engineering
Clemson University, Clemson, SC 29634

Abstract

Robotic systems are well suited for decontamination and decommissioning (D&D) tasks in hazardous environments. Advanced semi-autonomous telerobotic solutions for D&D tasks go beyond simple video-based interaction and include virtual reality (VR) interfaces and flexible sensor integration. The capability of those systems range from real-time control tasks to graphical user interface (GUI) components utilizing video and VR. This paper describes QRobot, a PC-based system for telerobotic D&D operations. The system integrates hardware interfacing, real-time joint level control, sensors, tool control, networking and task level programming as well as video and VR based operator interfaces. The system demonstrates that the personal computer (PC), a cost effective and widely used computing platform, is well suited to the integration of real-time control tasks and advanced user interfaces. An experimental section demonstrates the system's functionality by using an example workpiece as the subject of a D&D operation.

1 Introduction

The U.S. Department of Energy (DOE) is facing the decontamination and decommissioning of a high number of surplus facilities. These facilities often contain radioactive or other hazardous material. Current technologies are often labor intensive, time consuming, expensive, or they unnecessarily expose workers to the hazardous material. The DOE is looking for new and innovative technologies that allow D&D operations to be faster, safer, and more cost-effective. Telerobotic systems provide a good solution to this problem. They allow robots to be remotely controlled from an operator console and provide visual feedback to the operator. In basic systems, an operator controls the robot directly (*e.g.* with a joystick) and receives video feedback [1]. Performing a remote disassembly is a complicated, often repetitive task, which requires skilled operators. Therefore, much of the ongoing research focuses increasingly on the development of semi-autonomous systems. These systems perform higher level tasks, such as removing a bolt, triggered by the operator. Furthermore, VR based operator interfaces are desired to simplify interaction with the system.

This paper describes how the QRobot joint level control [2] was extended to a complete semi-autonomous robot control system for D&D operations. QRobot is the robotic part of the research being conducted under DOE Grant DE-FG07-96ER14728, entitled "Advanced Sensing and Control Techniques to Facilitate Semi-Autonomous Decommissioning of Hazardous Sites" [3]. It is a purely PC based system that integrates the following components:

- A joint level control and a trajectory generator with a high level programming interface for Puma manipulators.
- A 3D OpenGL-based hardware-accelerated robot simulator.
- Both video based and VR based operator interfaces.
- Teleobservation programs.
- Interfacing of different sensors.
- Control of different robotic end-effectors.

The experimental section of this paper documents the capability of the QRobot system to perform a sample D&D operation.

2 Motivation for Developing a PC Based System

The different components of an advanced semi-autonomous telerobotic system have different hardware and software requirements:

- The joint level control task and the trajectory generator require hard real-time performance.
- The GUI needs to integrate VR and video techniques. Hardware accelerated 3D rendering is required for the VR interface.
- Networking capabilities are required in order to locate the robot control hardware remotely from the operator console. Networking is also required if a multiprocessor architecture is used to divide the work among multiple computers (*e.g.* video capture on one PC, robot control on another PC, *etc.*)

The above requirements usually lead to the integration of proprietary solutions and expensive hardware platforms. As an example, consider an RCCL based system. The software consists of a robot control library (RCCL [4]) running on a Sun workstation, Moper running on a LSI 11/2 processor, and firmware joint level control running on digital servo boards inside the Mark II [5]. Additional hardware required includes an SBUS to VME bus adapter and a VME card cage. The closed architecture of the Mark II controller prevents the implementation of new, state-of-the-art control algorithms [2], as well as the integration of sensors such as cameras into the control loop. The heterogeneous architecture of this type of system leads to a higher complexity of integration and higher costs.

QRobot is entirely PC-based. The entire computational functionality of the system, including the joint level control, is implemented exclusively as PC software. Neither a dual processor architecture (like PC/digital signal processing boards solutions) nor special controller hardware (such as the Mark II's digital servo boards) is necessary.

This system has the following advantages:

- The system is cost-effective, because PCs and their components are less expensive than proprietary controllers or traditional Unix workstations.
- The system has a simpler architecture, since the additional effort to integrate completely different hardware components (such as VME cards with an SBUS computer) is not required.
- The system is more flexible. To modify or extend the system, only a change to PC source code is necessary.

- The PC is a widely known and technically advancing technology. Many powerful software packages as well as a great variety of interface boards are available for the PC.

There are two developments that allow a PC based system to fulfill the different hardware and software requirements stated above. First, the advent of high-speed PC CPUs provides computing power similar to or exceeding Unix workstations or special purpose computers such as DSP boards [6]. Second, hard real-time PC operating systems (OSs) are able to execute real-time tasks (such as joint level control and trajectory generation) as well as non real-time tasks (such as GUIs) on one CPU [7].

3 Overview of the Disassembly System

Figures 1, 2 and 3 show the software and hardware components that are distributed across three PCs. The *VR Operator Interface PC* runs Windows NT, while the *Robot Control PC* and the *WebCam PC* run QNX, a real-time OS. The *VR Operator Interface* and the *Robot Simulator* are integrated into one Windows NT program. The *Video Operator Interface* contains the actual disassembly program and communicates with the VR Operator Interface over Internet Domain TCP/IP sockets.

The *Disassembly Program* issues high level commands to *ARCL*, a robot control library that serves as the programming interface and as the trajectory generator. ARCL generates a stream of setpoints that are fed into the *Joint Level Control*. *Observation Windows* provide visual feedback of the D&D operation. They show a continuously updated image from one of the video cameras. Multiple observation windows can be used with different video cameras. The *WebCam System* allows video feedback over the World Wide Web by using a standard web browser. The camera, the pan/tilt unit (PTU) and the zoom lens of the WebCam System are also accessible from the observation windows running on the Robot Control PC. Special programs, called *Hardware Servers*, are responsible for accessing the PC boards (*e.g.* PTU server, MultiQ server, *etc.*) *Control Servers* implement control algorithms (*e.g.* zoom lens control, robot joint level control.) Applications use *Clients* to communicate with servers.

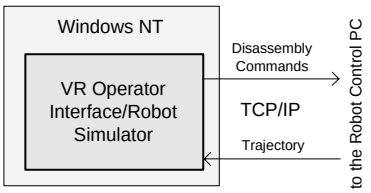

Figure 1. The VR Operator Interface PC

Different PC boards are used in the system: Quanser's MultiQ board and ServoToGo's S8 board for digital and analog I/O, the Imagenation PXC200 and Matrox Meteor frame grabbers for video capturing, and a custom board for interfacing the force/torque (F/T) sensor. The main hardware component is a *Puma 560 Manipulator*. A hardware retrofit interfaces the encoders and potentiometers of the Puma and the power amplifiers of the Mark II directly to an I/O board. The *F/T Sensor* and the *Toolchanger* are mounted on the Puma's wrist. A tool rack provides three tools for the disassembly: a *Gripper*, an *Air Motor* for bolt removal, and a *Laser Diode* to simulate torch cuts. The vision system consists of two cameras. The first, called the *Overhead Camera*, is mounted directly over the Puma's workspace and is connected to the Robot Control PC (Figure 2.)

The second camera, called the *WebCam*, is mounted on a PTU, equipped with a zoom lens, and is connected to the WebCam PC (Figure 3.)

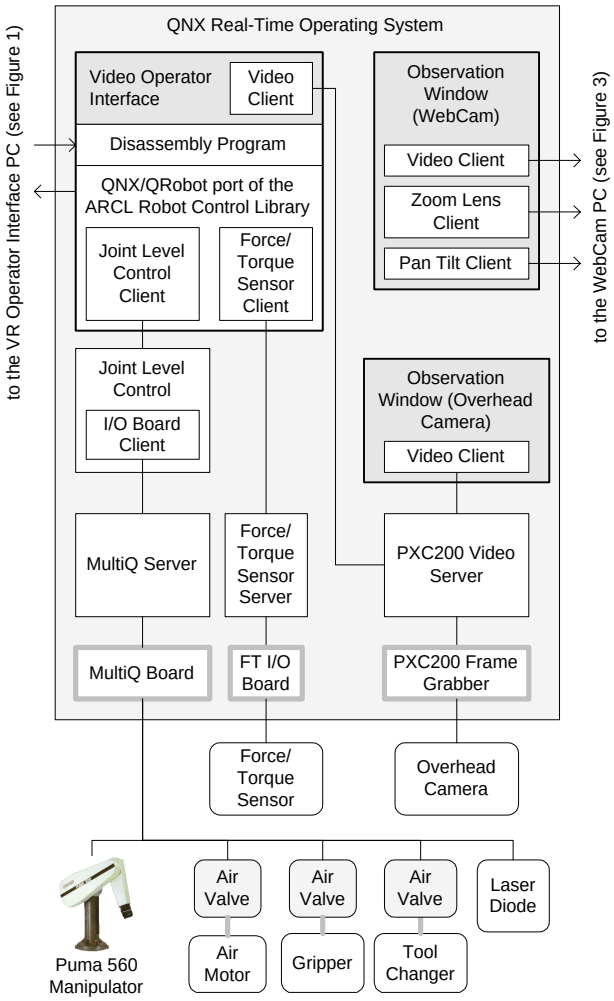

Figure 2. The Robot Control PC

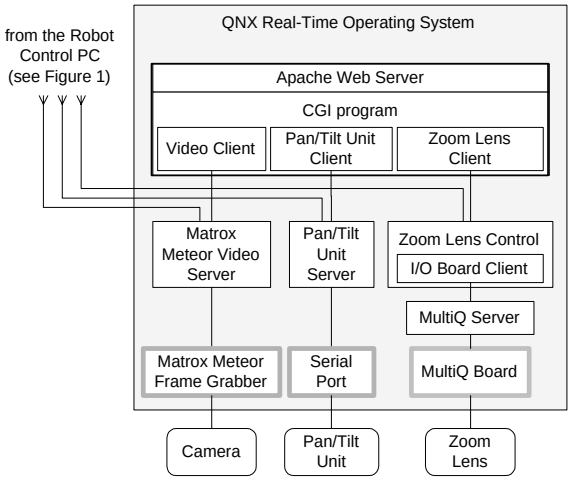

Figure 3. The WebCam PC

4 Hardware Components

4.1 The Puma 560 Retrofit

The standard controller for Puma manipulators is the VAL-II based Unimation Mark II. Six MC6503 based digital servo boards perform the joint level control [5, 8]. Implementing the control as a PC program allows the development of arbitrary user-defined joint level controls and the integration of sensor information into the control loop. To achieve this, a retrofit of the Mark II hardware is necessary.

The hardware retrofit directly interfaces the encoders and potentiometers of the Puma and the power amplifiers of the Mark II directly to an PC I/O board. The TRC boards (TRC004, TRC006 and the TRC041 card cable set), produced by Trident Robotics and Research Inc. [9], were initially used for this purpose [2]. The TRC boards provide a simple but proprietary solution. That is, the user is dependent on one source of hardware and software – Trident Robotics Research. In order to make the system more flexible and less dependant on one vendor's hardware, the next step was to replace the TRC boards with generic interface boards. Quanser Consulting's MultiQ board [10] was selected to replace the TRC004 and TRC006 boards. It was necessary to develop an additional simple interface board to connect the MultiQ board to the amplifier circuits of the Mark II. This interface board contains preamplifiers and filtering circuitry for the noisy potentiometer readings. The TRC041 cable card set was replaced by an in-house developed cable card set.

The MultiQ boards do not support latching of digital inputs, a feature that the TRC004 board provides for use with the Puma 560 encoder index pulses, required during the robot calibration procedure. To solve this problem, the MultiQ hardware server simulates the latching in software.

To demonstrate that the architecture is flexible enough to easily accommodate other motion control interface boards, another Mark II controller was retrofitted with a ServoToGo (STG) S8 interface board instead of a MultiQ board. A problem occurred when using the encoders: the encoder channels of the MultiQ board operate in the reverse direction than those of the STG board. To compensate, the encoder values are reversed in the STG hardware server.

4.2 End-Effector Hardware

The D&D tasks require various tools and sensors. To accommodate these tasks, an ATI Industrial Automation Gamma 30/100 F/T sensor is mounted at the end of the robot arm. The FT sensor is interfaced to the PC via an ISA bus controller board.

The toolchanger is mounted on the FT sensor. It is a Light 5 Robotic Tool Changer, also from ATI. It contains 10 electric and 6 pneumatic pass-through ports, for electrical and pneumatic connections on the end-effector. The custom-built tool rack provides space for three tools.

- An MMR-002 air motor, from Micro-Motor Inc., is used for bolt removal. A Gator Grip socket tool is mounted on the end of the air motor. The Gator Grip is a universal socket that automatically adjusts to varying bolt sizes, allowing bolt removal operations to proceed even with several millimeters of positioning error.
- A standard pneumatic gripper is used to remove stuck bolts and the motor end cap.
- A laser diode is used to simulate a cutting torch.

Electrically controlled air valves actuate the tool changer, the gripper and the air motor. Digital output lines of the MultiQ board control all tools.

4.3 Observation System

The observation system consists of two Pulnix TMC-7 cameras. One is connected to a Matrox Meteor PCI bus frame grabber, the other uses an Imagenation PXC200 PCI bus frame grabber. One camera is mounted in a fixed direction above the workspace. The other camera is mounted on a Directed Perception PTU, model PTU-46-17.5. The PTU is connected to the PTU controller (a micro controller based constant acceleration open loop control), which is in turn connected to the PC via an RS232 serial port. The PTU mounted camera also uses a zoom lens. The motors and potentiometers of the zoom lens are connected to a custom interface board (containing amplifiers and voltage dividers), which is then connected to a MultiQ for A/D and D/A.

5 Software Components

5.1 The Multitasking and Communication Architecture

The system's functionality is split into many cooperating tasks. For these tasks to work seamlessly together, the OS must fulfill certain requirements. It must provide priority based deterministic CPU scheduling to ensure that high priority real-time tasks (*e.g.* the joint level control) are not delayed by low priority non real-time tasks (*e.g.* GUIs). It also must provide robust interprocess communication (IPC) mechanisms so that the cooperating tasks can synchronize and communicate.

The real-time microkernel based OS QNX, developed by QSSL [11], meets all of these requirements. Unlike real-time extensions such as RT-Linux or Hyperkernel for Windows NT, QNX is a true microkernel real-time OS. One benefit of this is that the whole spectrum of OS functions, including file access and networking, can be used in real-time tasks.

Client/Server Architecture. The system utilizes two types of servers. Hardware servers are used to access hardware. Control servers implement a control algorithm. Both types of servers are separate programs that usually cycle at a fixed rate. To exchange data with a server (*e.g.* to send setpoints to a control server or to read analog inputs from a hardware server), a program is linked with the appropriate client library. The client library uses shared memory or message passing to communicate with the server.

Message passing is a QNX IPC mechanism that is very flexible because it is network transparent. This means that the same client code will work with a server whether it is located on the same PC or a remote PC. For example, the video client located on the Robot Control PC can connect to the video server of the overhead camera, also running on the Robot Control PC. Alternately, it can connect to the WebCam Video Server, which is running on the WebCam PC. This mechanism allows great flexibility in distributing the resources of the system.

Another advantage of the client/server concept is that multiple clients can use the same server. In this way, resources can be accessed from multiple tasks. For example, the video clients of the operator interface, the observation windows and the WebCam can all use the overhead camera video server simultaneously. Since QNX message passing provides

synchronization implicitly, concurrent requests are automatically serialized.

Finally, the interface between client and server adds a level of abstraction to the hardware interfacing. To use different I/O boards different servers are implemented, but the same generic client can be used. The communication protocol between client and server does not change. For a client to use a different I/O board, a different server must be started, but no client code needs to be changed or recompiled.

All clients and servers are written in C++. The MultiQ and the STG S8 server perform digital and analog I/O at a fixed frequency. A generic I/O board client is used to communicate with either of these servers. The Matrox Meteor Server and the Imagenation PXC200 Server capture frames on demand. A generic video client is used to request and receive frames from either server. The F/T sensor server is interfaced to the ISA F/T sensor controller board. It reads the force and torque values continuously and provides them to the F/T client in a shared memory space. The PTU server controls the PTU over the RS232 serial port. It receives messages from the PTU client that contain the desired angles and issues move commands to the controller. The zoom lens server receives the desired zoom factor from the zoom lens client and uses a proportional position control to set the focal length of the lens.

The timer server is a special server. It provides the clock for all other servers, which guarantees synchronous behavior of the different tasks.

5.2 Joint Level Control

The joint level control is implemented as a QNX program. This is a very flexible solution, since the control algorithm can be modified directly by changing and recompiling the control program. The ever-increasing computing power of PCs allows the implementation of more complex control algorithms. In addition, it is now possible to include arbitrary sensor information in the joint level control loop, which allows the implementation of advanced sensor-based control algorithms such as force-based control or direct visual servoing.

The joint level control used in the D&D system was developed using QMotor [6], which is an environment for PC based control program development and implementation. The control program implements a PD controller with static and coulomb friction compensation for all joints and gravity compensation for the second and third joint. Joint velocities are manufactured via a backward difference method and low pass filtered [2]. The joint level control program works as a server and receives the stream of setpoints via message passing from the joint level control client, which is part of the QRobot version of ARCL. The control can be switched to a zero gravity mode. In this mode, the control only compensates for the gravity on the robot links instead of servoing to desired setpoints [12]. The robot can be freely moved by hand in this mode, which is used to teach end-effector positions and orientations with the teachpendant program.

5.3 Trajectory Generation and Robot Programming Interface

To achieve the goal of an entirely PC-based system, a high-level robot control API and trajectory generator package for the PC is required. As there is no such package available for QNX, the quickest solution is a port from a different platform. One of the most sophisticated and well-known high level robot control libraries is RCCL. John Lloyd, one of the developers of RCCL, had unsuccessfully tried to port RCCL to QNX. For this reason, RCCL was not considered for use in this project.

The Advanced Robot Control Library (ARCL), developed by Peter Corke at CSIRO [13][14], provides similar functionality to RCCL, although in a much more limited fashion. ARCL seemed to be more suitable for a QNX port, because its modular architecture separates platform dependent and platform independent parts. ARCL was developed for Unix workstations. Porting ARCL to QNX and integrating it with the QRobot system required significant effort. This effort included making the C source code C++ syntax compliant, debugging the existing ARCL source code, developing the platform specific part of ARCL (called the ARCL machine interface, or AMI), and embedding the ARCL modules into the real-time environment of QNX and the client/server architecture.

The main challenge of porting ARCL was writing the AMI for QNX. The AMI is the platform dependent module that contains functions for multitasking, timing and hardware interfacing of the manipulator. Two problems occurred when developing the AMI for QNX. First, the architecture of the AMI requires that both the trajectory generator task and the user program task share variables. This can be implemented either by using shared memory between these tasks or by using threads. Since ARCL was not designed to take advantage of shared memory, the development of a special memory manager would have been the only solution. Threads, the other approach, are multiple instances of a process that share the same address space. QNX supports threads, but in a limited fashion, and not all library functions are “thread-safe”. Tests showed that the thread-based solution, which is much easier to implement than the shared memory manager, was adequate.

Semaphore management posed the other significant problem. QNX semaphores do not behave in accordance with the ARCL specification. This was rectified by modifying the way ARCL uses semaphores. In addition, ARCL expects the OS to destroy all semaphores at program termination, which is not the case with QNX. Consequently, over time, the system runs out of semaphores. To solve this problem, a semaphore manager was added to the AMI to keep track of the semaphores in use and to destroy them as the program terminates.

To integrate ARCL into the QRobot system, the QNX/QRobot AMI was constructed to contain the following functions:

- A joint level control client sends the trajectory to the joint level control server.
- Functionality was added to connect to the robot simulator running on the Windows NT PC. Sockets were used as the communication mechanism. A 100Mbps full-duplex point-to-point Fast Ethernet link proved to be fast enough to ensure that the robot simulator does not fall behind the trajectory generator. The protocol is simple: the data sent contains the stream of setpoints generated by the trajectory generator. There are two modes of operation, test mode and standard mode. Test mode allows running robot control programs with the simulator alone, without accessing hardware. This is useful for debugging robot programs without the risk of damaging the robot. In standard mode, the trajectory is sent to both the robot simulator and to the joint level control, so it is possible to compare the movement of the real robot with the simulator. A communication channel in the other direction, from the robot simulator to ARCL, was added to allow the virtual operator interface (which is part of the robot

simulator) to send disassembly commands to the task level program.

- Force and torque information is used to trigger an emergency stop to minimize damage from collisions between the end-effector and the workspace.

Rather than using ARCL's limited functionality to control tools (there is only a function to control the gripper), a C++ class was developed for each tool. The tool classes use I/O board clients to control the tools.

Although the port of ARCL achieved satisfactory results, it is not an ideal solution. The use of threads is not 100% safe, since not all QNX library functions are "thread-safe", although we never identified any crashes that could be attributed to threads. If a user chooses to use a non thread-safe function in one of his programs, the behavior of the system is undefined.

5.4 Robotic Utility Programs

To facilitate the use and calibration of the D&D system, a set of robot utility programs was developed.

Teachpendant: To learn the end-effector positions and orientations used in the D&D operation, a teachpendant program was developed, see Figure 4.

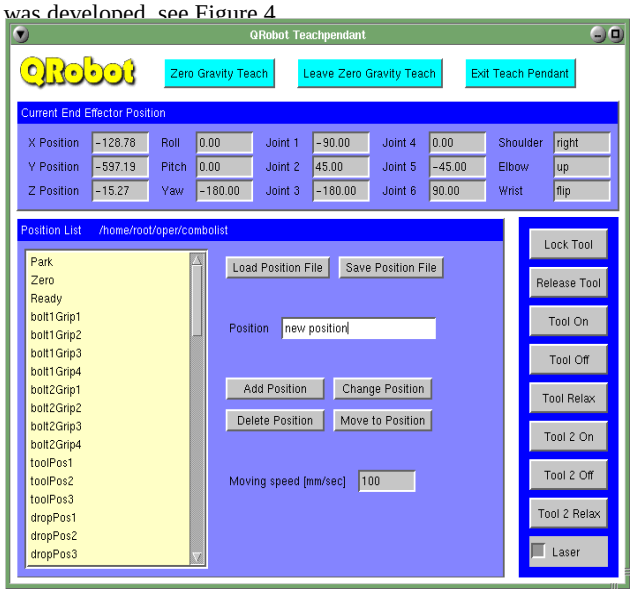

Figure 4. The QRobot Teachpendant

The teachpendant uses the zero-gravity mode of the joint level controller, which allows the user to easily push the robot around in the workspace. Once a position and orientation is found, it can be stored under a given name in a position list. It is possible to leave teach mode at any time and move the robot to previously taught positions. The position list can be stored in an ASCII file for later use in the teachpendant, or for use from an ARCL program.

PotVal: The PotVal utility performs the initial calibration procedure of the Puma 560 that relates joint potentiometer readings to encoder readings.

PumaCal: The PumaCal utility performs encoder calibration after power up of the manipulator. It determines the current position of the robot by using potentiometers and index pulses. This utility is similar to RCCL's pumacal utility [4], but performs the calibration in only a fraction of the time.

5.5 The Disassembly Program

A motor is used to demonstrate a simple disassembly. The objective is to remove the cap of the motor. Figure 5 shows the steps performed by the system:

1. Remove the first bolt. The gator grip is used to unscrew the bolt. Since the bolt usually stays in the housing, the operator has the additional option to remove the unscrewed bolt with the gripper and drop it into a container.
2. Remove the second bolt in the same fashion as described in step 1.
3. Perform a torch cut. In the experiment, the torch is simulated by a laser diode.
4. Remove the cap with the gripper and put it on the table.

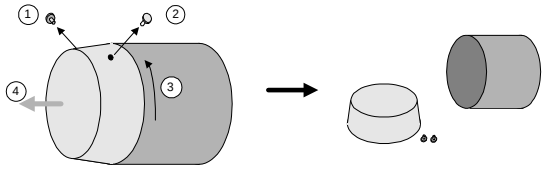

Figure 5. Steps to Disassemble the Motor

The disassembly program is written in C++. For each disassembly step, via points are determined with the teachpendant program and saved to a file. The disassembly program reads this file and creates transformations and position equations for each via point. The position equations are the input to the ARCL *move* function calls. Each disassembly step consists of picking up the right tool from the tool rack, performing the operation, and returning the tool to the rack. Some special functions are defined to allow the operator to intervene in case the system fails to complete a step. These functions include manually getting or returning a tool and manually locking or releasing a tool.

5.6 The Operator Programs

Four operator interface programs offer different control and feedback functions. Observation windows and the video operator interface run on the same PC as the robot control, but at a lower priority. They use Photon, the windowing system for QNX. Photon provides functionality similar to the X Window System and Xt. To accelerate GUI development under Photon, a C++ class library (CPhoton) was developed.

Observation Windows. The observation window (see Figure 6) provides live visual feedback from a video camera. When using the WebCam, the observation window offers additional functionality. Clicking in the image centers the PTU about that point. The buttons at the bottom of the window control the WebCam's zoom lens.

It is possible to start multiple observation windows and connect them to the same or different cameras. Since the observation windows use message-passing based client/server communication, the camera servers can be distributed over multiple PCs. The disadvantage of message passing is reduced speed in the image transfer. Depending on the PC's performance, image size, image color depth, network traffic, and the video display driver, the observation window displays 1-5 frames per second.

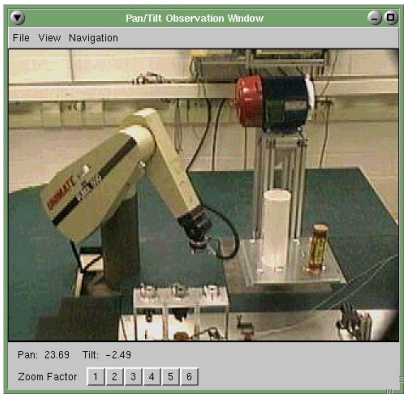

Figure 6. Observation Window

Web Camera. The WebCam is a World Wide Web based visual feedback, with similar functionality to the observation windows. The Apache web server starts a CGI program whenever the web page is accessed. The request for the web page contains the desired pan/tilt angles and the desired zoom factor as parameters. The CGI program moves the PTU to the desired position, sets the zoom factor, and captures an image. This image is converted to JPEG format, and a web page is dynamically created to show the image. The client/server architecture allows multiple observation windows and any number of web browsers to request images at the same time. The advantage of the WebCam is the accessibility from any Internet connected computer. The disadvantage is the lack of continuous and fast updates of the image.

Video Operator Interface. The video operator interface provides even unskilled operators an easy-to-use interface to control the disassembly tasks. The video image is used to trigger disassembly operations. The operator moves the mouse cursor over a certain part of the motor that he wants to disassemble. The operator interface then displays a pop-up menu with a list of disassembly options. For example, when the operator moves the mouse over the motor end cap, the end cap is highlighted, and a menu pops up with the menu item "Remove Cap" (see Figure 7). After the operator selects an operation, the program begins to perform the task and shows progress information in a dialog. As the disassembly is being performed, the operator is able to supervise the operation in the observation windows. In case the disassembly of a part is unsuccessful, the operation can be repeated.

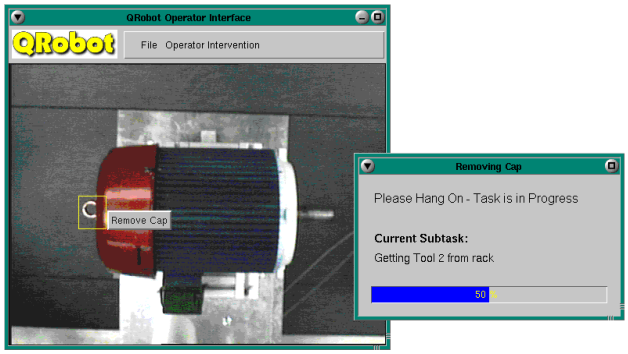

Figure 7. Video Operator Interface

The image-based selection of disassembly operations is convenient for the operator, but it also requires that the system know where the parts of the object are located in the image. The Image Processing group at Clemson University is investigating the use of advanced image processing and 3D-object virtualization techniques to automatically identify and locate these parts for the disassembly task [3]. This research is not addressed in this paper. To demonstrate the basic concept of the operator interface, the coordinates are manually determined in the current system.

Virtual Operator Interface/Robot Simulator. The video operator interface works fine with a workpiece such as the motor and the overhead camera. However, using different camera perspectives or more complex workpieces can result in hidden parts that can not be viewed and selected by the operator. For instance, the front perspective of the motor would not show the second bolt at the back of the motor. VR based operator interfaces overcome this problem. Using a VR interface the operator is able to navigate within the virtual scene and view parts from different angles. The QRobot virtual operator interface is integrated into the robot simulator.

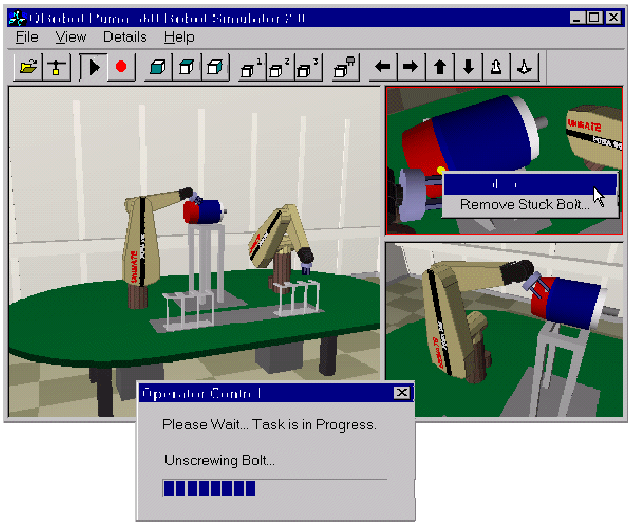

Figure 8. The Virtual Operator Interface/Robot Simulator

Figure 8 shows a screenshot of the robot simulator. Since there are no hardware accelerated 3D graphics libraries available for QNX, the program runs under Windows NT and uses OpenGL, a standard 3D graphics library [15].

The 3D scene consists of two Puma 560 robots, the toolrack and the workpiece. The main window is split into three parts that show the scene from different perspectives. In each sub-window, the operator can navigate by using the mouse, selecting and defining custom viewpoints or selecting the end-effector view. The latter option simulates the view of a camera mounted on the end-effector. The level of detail in the display can be reduced to accelerate the display.

While the disassembly program is running, ARCL forwards the trajectory information to the robot simulator. The VR operator interface requires that the video based operator interface be running, since the video based operator interface is linked to the actual disassembly program. The convenient side effect is that both operator interfaces work hand in hand (*i.e.* the disassembly task can be started from either.)

A special technique of 3D programming, called object picking, gives the operator functionality similar to the video based operator interface. Moving the mouse cursor over parts of the motor highlights these parts. Clicking on the parts displays menus with disassembly options. Once the operator selects a disassembly option, the software encodes this operation into a command word, and sends it to the video based operator interface, which initiates the operation. This data transport uses the same TCP/IP connection that is used to send the trajectory information. Progress reports are also sent back to update the progress dialog box as shown in Figure 8.

The disadvantage of the current version of the robot simulator/operator interface is that the scene is hard-coded in the simulator. Changing the scene is not trivial and requires extensive programming effort. However, this compromise establishes a working system at this stage of the project.

6 Experimental Results

There are three aspects of the experiment. The first aspect is the reliability of the control system and the mechanical parts. The system was successful in repeating the disassembly multiple times without any problems. The joint level control was capable of precisely moving different tools of different weights. Figure 9 shows the desired position trajectories for links 2 and 3 for the bolt removal task in the top graphs.

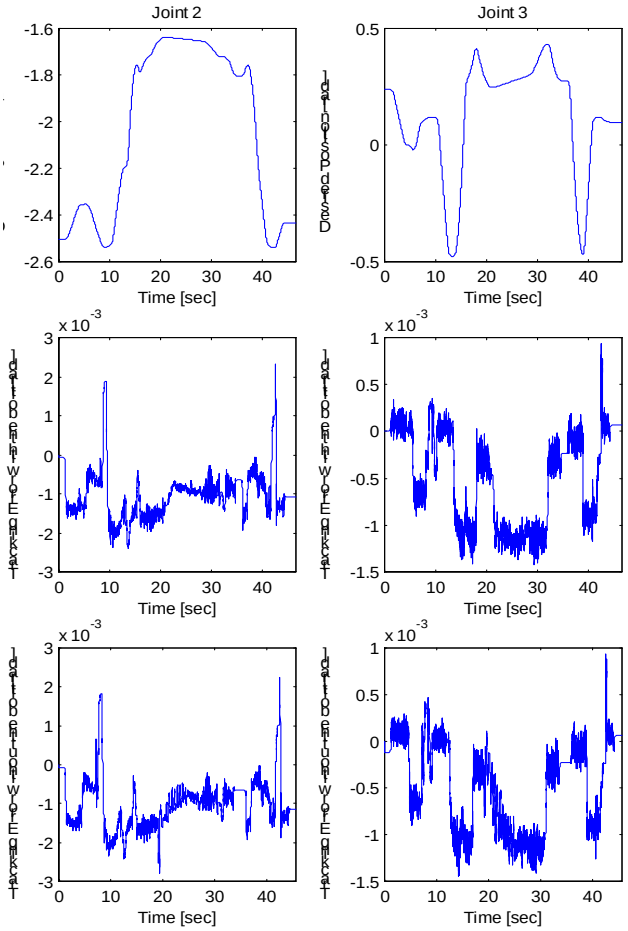

Figure 9. Trajectories and Tracking Errors for the Bolt Removal

The middle graphs are the tracking errors. The bottom graphs show the tracking errors for a test run without the bolt actually being present. The actual bolt removal happens from t = 20 sec to t = 30 sec. Comparing the tracking errors in the middle and bottom graphs shows that the effect of the force created by removing the bolt does not affect the control significantly.

The second aspect is the usability and reliability of the system used by an untrained operator. The experiment shows that both video and VR based operator interfaces provide an intuitive way to control the D&D operation. Only a few mouse clicks are necessary to guide the complete disassembly. Problems occur when the operator needs to recover from a system failure or a handling error. Generally, the capabilities of aborting operations and returning to the initial state are limited. Often, the robot, the tools, or workpieces have to be moved back manually to initial positions.

Finally, the most interesting part is the stability of the GUIs and the real-time control programs running together. The system shows high stability in this issue. For example, it is possible to open many observation windows while the D&D task is in progress. The observation windows slow the GUIs down, but they do not negatively impact the control or disassembly tasks.

7 Conclusion

The QRobot system described in this paper demonstrates the feasibility of using a PC for the various tasks required in telerobotic semi-autonomous D&D operations. It was shown how real-time tasks, such as the joint control, can be integrated with non real-time tasks, such as a GUI, on a single cost-effective hardware platform. The client/server concept, using a modern real-time OS as its platform, provides a flexible method of communication for these tasks. Priority based scheduling allows complex real-time control programs to coexist with low-priority GUIs. The operator interfaces and the teleobservation programs provide an easy-to-use telerobotic operator environment.

Future research is motivated by the limitations of the system, which originate in its software components and its architecture. ARCL as a high-level control library is not a satisfactory, robust, general-purpose solution. Although QRobot is entirely PC based, it still uses two OSs: QNX (for all functions except the 3D robot simulator) and Windows NT (for the 3D robot simulator). This introduces higher complexity and costs. It would be desirable to have all components running under the same OS. The system was developed by integrating different software components. All components have been specifically modified to work together, which leads to a static architecture. For example, extending the system to a different robot type would result in modifications of ARCL, the joint level control, the robot simulator and the way they communicate. The biggest problem is the limited flexibility in adapting the system to different applications. Basically, certain parts of QRobot (*e.g.* the robot simulator, the GUIs, the tool clients) are specific to the D&D example of the motor disassembly. Adapting the system to a different application requires some knowledge of its internal workings.

To overcome these problems, the current research targets the specification, design, and implementation of a new robot control environment. Rather than being a specific robot control library like RCCL or ARCL, it would be an open framework to manage different mechatronic components that work together in D&D operations and other robotic applications. The framework would be object oriented, using the language C++. Objects would

represent the different components of the system, for example the trajectory generator, manipulators, tools and sensors. Object oriented programming techniques like inheritance can be used to reuse code and easily extend the system without knowing its internals, thereby avoiding recompiling the whole system. The framework would provide functionality to create and delete objects, communication and relationships between objects, and real-time task management.

The proposed research would address the following challenges:

- Design of an open system for different applications and research areas.
- Design of a flexible system that allows the distribution of objects over multiple PCs and uses real-time communication over a network.
- Design of a system architecture that is highly platform independent.
- Implementation of the complete system under one OS to achieve a single PC, single OS solution.

This new robot control environment would use the concepts and advantages of a purely PC based system described in this paper, but improve the design to utilize state-of-the-art software development tools and techniques. The resultant system would allow researchers to focus on their actual application rather than the technical framework.

Acknowledgements

This work is supported in part by the U.S. NSF Grants DMI-9457967, CMS-9634796, ECS-9619785, DMI-9813213, EPS-9630167, DOE Grant DE-FG07-96ER14728, ONR Grant N00014-99-1-0589, and a DOC Grant.

References

[1] Vladimir Lumelsky, “On human performance in telerobotics”, IEEE Transactions on Systems, Man and Cybernetics, 21(5).

[2] N. Costescu, M. Loffler, E. Zergeroglu, and D. Dawson, "QRobot - A Multitasking PC Based Robot Control System", Microcomputer Applications Journal Special Issue on Robotics, Vol. 18 No. 1, pages 13-22.

[3] http://ece.clemson.edu/iaal/doeweb/doeweb.htm

[4] J. Lloyd, “Implementation of a Robot Control Development Environment”, Masters Thesis, McGill University, Dec.1985

[5] Unimation Inc., Danbury, Connecticut, “500 Series Equipment and Programming Manual”, 1983.

[6] N. Costescu, D. M. Dawson, and M. Loffler, "QMotor 2.0 - A PC Based Real-Time Multitasking Graphical Control Environment", June 1999 IEEE Control Systems Magazine, Vol. 19 Number 3 pages 68 - 76.

[7] D. Hildebrand, “An Architectural Overview of QNX”, Proceedings of the Usenix Workship on Micro-Kernels & Other Kernel Architectures, Seattle, April 1992.

[8] P. Corke, “The Unimation Puma Servo System”, CSIRO Division of Manufacturing Technology, Preston, Australia.

[9] Trident Robotics and Research, Inc., 2516 Matterhorn Drive, Wexford, PA 15090-7962, (412) 934-8348, http://www.cs.cmu.edu/~deadslug/puma.html

[10] Quanser Consulting, 102 George Street, Hamilton, Ontario, CANADA L8P 1E2, Tel: 1 905 527 5208, Fax: 1 905 570 1906, http://www.quanser.com.

[11] QSSL, Corporate Headquarters, 175 Terence Matthews Crescent, Kanata, Ontario K2M 1W8 Canada, Tel: +1 800-676-0566 or +1 613-591-0931, Fax: +1 613-591-3579, Email: info@qnx.com, http://www.qnx.com (QNX web site)

[12] B. Armstrong, O. Khatib, J. Burdick , “The Explicit Dynamic Model and Inertial Parameters of the PUMA 560 Arm”, Proc. IEEE int. conf. Robotics and Automation 1 (1986) pp. 510-518

[13] P. Corke, “The ARCL Robot Programming System”, CSIRO Division of Manufacturing Technology.

[14] P. Corke, P. Dunn, and R. Kirkham, “An ARCL Tutorial”, CSIRO Division of Manufacturing Technology, Preston, Australia, 1992.

[15] J. Neider, T. Davis, and M. Woo, “OpenGL Programming Guide”, Addison-Wesley Publishing Company, New York, 1993.