A Scalable System for Real-Time Control of Dexterous Surgical Robots

Paul Thienphrapa and Peter Kazanzides
Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218

Abstract—*The requisite cabling and control processing for surgical robots can become unwieldy as dexterity is increased, due to the additional degrees of freedom. Motivated by dexterous snake-like robots for minimally invasive surgery, this paper details the development of a low-level control system that uses IEEE 1394 (FireWire), linking the computer to low-latency field-programmable gate arrays, to distribute I/O while centralizing all control computations. A standard programming interface is defined as part of the control system to enhance its scalability. These features increase the viability of complex surgical robots and ease their development by reducing cables and enabling the processing of many axes of control on a single computer.*

1. INTRODUCTION

Minimally invasive surgery (MIS) is often beneficial due to reduction of trauma, leading to fewer complications and shorter hospital stays. However, it poses a number of challenges for surgeons, including constrained workspaces, limited field of view, and lack of dexterity at the distal end. Existing MIS tools are rigid, difficult to manipulate through narrow insertion tubes, and they lack adequate suturing and tissue reconstruction capability. In such situations, the efficacy of a surgical robot is strongly tied to its dexterity.

Research on the Snake Robot [1] seeks to improve MIS of the throat and upper airways by providing surgeons with highly dexterous robotically-controlled tools. This dexterity is achieved by incorporating more degrees of freedom (dof). More sophisticated surgical tasks can be accomplished by increasing dof, but the corresponding hardware increase imposes a practical limit on the exploration of these ideas. Similarly, research on different types of multi-axis surgical robots is often mired in the hardware construction effort. In response to these difficulties, this paper presents the development of a system that is well suited for real-time control of robots with many axes of control.

Figure 1 - Snake Robot prototype [3]

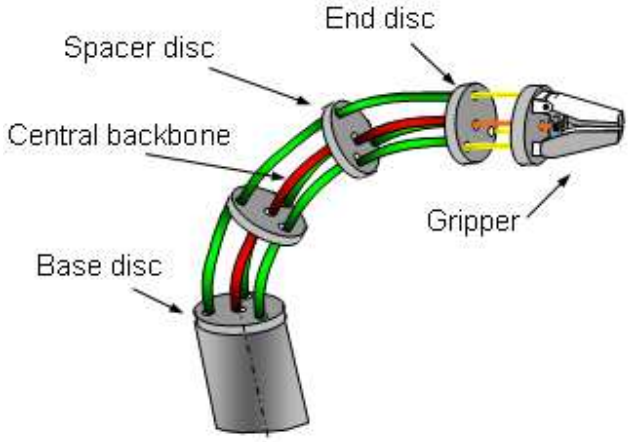

Figure 2 - Anatomy of a snake-like unit [2]

1.1. Background

A unique design targeted for MIS of the upper airways, the Snake Robot features small, dexterous snake-like units (SLUs). To avoid cluttering the work area, these SLUs are attached to a hollow, narrow, meter-long shaft containing its wires, and appear at the distal end of a laryngoscope; this shaft-SLU assembly is teleoperable (see Figure 1).

The Snake Robot is an eight-dof (11-actuator) manipulator. Multiple instances may be used for surgical tasks, depending on the application. The distal end of each manipulator consists of two SLUs connected in series; each SLU is constructed using four super-elastic NiTi tubes. The anatomy of an SLU is shown in Figure 2. The center tube, the *primary backbone*, is connected to all discs, including the base disc, end disc, and intervening spacer discs.

Three tubes surround the primary backbone at equally-spaced distances, forming the *secondary backbones*. These are fixed to only the end disc and are free to glide through holes in the intermediate spacer discs. Two dof result from pushing and pulling the secondary backbones using three actuators located at the proximal end. The push-pull actuation modes help prevent the backbones from buckling while satisfying structural statics [2]. A second 2-dof SLU is appended to the end of the first. The secondary backbones of this second SLU pass through the hollow secondary backbones of the first. Attached to the end of the second SLU is a gripper that is actuated via a wire passed through the hollow central backbone. This mechanical composition allows for a high payload capacity with a small size [3]. The existing SLUs are 4 mm in diameter.

The aforementioned actuators (seven actuators for four dof and a gripper) are encased atop the shaft in a compact cylindrical *actuation unit* (shown in Figure 1). The shaft-actuation unit assembly itself can be guided with four dof due to four actuators, leading to eight dof total. The Z-Θ stage allows for translation along and rotation about the shaft respectively. A passive universal joint mounted on a five bar mechanism gives the shaft XY mobility. This stage also stabilizes the robot against lateral perturbations.

Each Snake Robot is actuated by 11 dc motors, accounted for by two three-axis SLUs, a gripper, a two-axis Z-Θ stage, and a two-axis five bar mechanism. Two seven-axis da Vinci masters are used to command two Snake slaves for bimanual control. The pair of da Vinci masters we are using is an engineering version that did not originally include a controller.

The control system for the Snake Robot must be scalable in order to handle 36 axes (i.e. two 11-axis Snakes and two seven-axis da Vinci masters). A significant reduction in dimensionality and hardware complexity was achieved via the push-pull actuation of flexible wires combined with derived kinematics [1], as opposed to actuation of several precision joints. Nevertheless, the number of axes for the Snake Robot remains considerable and can only increase as robots are devised for more sophisticated surgical tasks.

1.2. Motivations and Objectives

The efforts presented here address important issues revealed by our previous work [18], which itself originated from the need to improve the old multi-axis controller of the Snake Robot [2] and replicate the controller for other research projects. The old controller utilizes a centralized I/O arrangement, whereby command and feedback signals are transmitted in raw analog form over long cables running between the robot and the computer. Though the design is conceptually straightforward, the cumbersome wiring associated with it introduces complications such as noise, cable drag, reduced reliability, and greater construction effort. The debug space is vast as there are many candidates for connectivity problems, so this approach limits the ability to develop increasingly dexterous surgical robots.

The long term benefits of developing a control system using IEEE 1394 are multifold. The high speed serial bus encourages the distributed I/O and centralized processing architecture we presented in [19]. One advantage of this approach is that the I/O processing logic is simple and requires little maintenance. Signal integrity is improved because digitization occurs near the actuators, reducing the potential for noise corruption.

Cable complexity is greatly reduced because distributed I/O hardware is accessed through a serial link. This has several benefits in a research environment. Less effort is required in creating cables and breakout boards when new robots are developed. Since I/O hardware is replicated, standard wiring conventions are enforced. Robustness is improved overall because there is less potential for wiring problems and less cable drag affecting mobility.

Parallel buses limit the number of I/O channels that can be connected to one computer. For example, an industrial-grade computer can reliably drive four ISA cards, and the number of channels per card is constrained to a modest number by physical size. As a result, a teleoperated dexterous robot system may need multiple computers to run.

In contrast, IEEE 1394 allows for centralized processing of a large number of channels on a centralized computer, so low-latency local data exchange can be used instead of network communication. The integration allows for a familiar software development environment and a standard API; this alleviates researchers and programmers from learning the idiosyncrasies of individual embedded micro-controllers, so they can instead focus on higher-level tasks. Furthermore, this architecture can more readily harness the power of high performance computing.

This work is also intended to facilitate further research by providing a generic interface and scalable mechanism for fine-grain real-time control. Such a custom solution was necessary for servoing the low power dc motors of the Snake Robot [2] because an adequate commercial solution was not

available. The solution allows for flexible customization of parameters, investigation of complex control laws, and high-density distributed I/O. It is designed to ease the development of dexterous surgical robots from both hardware and software perspectives.

1.3. Organization

The control system presented in this paper uses a high speed serial bus, IEEE 1394 (FireWire), to enable a large quantity of I/O signals to be processed on a single computer and cleanly distributed to the actuators. Its purpose is to provide a convenient interface for fine-grain, real-time control of highly dexterous surgical robots and to simplify software development, while mitigating the issues associated with unwieldy cabling.

This paper is organized as follows. Section 2 reviews some of the related work. Section 3 describes the control system in detail, while Section 4 demonstrates the performance of the system. Future work is outlined in the concluding remarks of Section 5.

2. Related Work

2.1. IEEE 1394 for Real-Time Control

IEEE 1394 was selected because the protocol supports real-time communication with guaranteed 8 kHz (125 µs) bus cycles in isochronous mode (though asynchronous mode is used instead as it allows for even faster access rates), and because it allows for daisy-chaining of nodes. It is an effective solution for real-time control, as shown in [6, 12], and by its use in fly-by-wire systems [7], but it is not necessarily the single best choice. One potential drawback is the lack of high-flexibility cables for installation within the moving structure of a robot arm, though for our applications this is not a serious limitation.

The works of [6] and [12] focus on real-time control bandwidth, but not on the physical benefits of distributed I/O and centralized processing. The differences manifest in their use of IEEE 1394 as a link to an onboard computer, contrasting with our use of compact custom electronics. Our work is most similar to that of [16], where custom FPGA-based I/O boards communicate with the computer over IEEE 1394. The bandwidth was sufficient for at least six (possibly 12) dof to be updated at 1 kHz, with unit delay latency. On the other hand, this study emphasizes the physical benefits of the architecture, performance, and scalability. Ref [17] notes that using IEEE 1394 for high bandwidth PET scan data acquisition is viable due to the availability of powerful commodity computers. We agree in principle, though our respective applications are fundamentally different.

2.2. Ethernet-Based Alternatives to IEEE 1394

Fair bus access is incorporated into IEEE 1394 hardware; bus arbitration in Ethernet is nondeterministic, but kilohertz-range motor control is achievable on isolated networks with software modifications [14, 15]. Several Ethernet variations have been developed that make the medium very promising. Powerlink (ethernet-powerlink.org) uses a bus manager that schedules 200-µs cycles of isochronous and asynchronous phases. SERCOS approached a communication bottleneck in [9] with increasing axes and cycle rates, but its recent combination with Ethernet (SERCOS-III) has endowed it with the ability to update 70 axes every 250 µs.

A relative newcomer, EtherCAT (ethercat.org) is an attractive protocol in which the nodes forward and append packets on-the-fly using dedicated hardware and software, resulting in the ability to communicate with 100 axes in 100 µs; [8] is an example showing its potential.

2.3. Other Alternatives to IEEE 1394

Many of the themes highlighted in this paper, including distributed I/O, centralized computing, scalability, and form factor, echo those of [10], which documents the MIRO surgical robot developed by the German Aerospace Center (DLR). Scalability in the MIRO robot is aided by the use of SpaceWire, a 1 GB/s full duplex serial link with latency less than 20 µs. Whereas SpaceWire has been developed by major international space agencies for space-borne systems, we prefer IEEE 1394 as it is a more accessible protocol for research, and its performance is more than adequate for demonstrating our claims. We are particularly more interested in the software-induced latency and overcoming this latency to enhance scalability.

PCI Express is a new serial interface designed to replace computer expansion buses; a cable-based standard was not fully established at the time of the designs presented in this paper. PCI Express supports real-time applications such as the industrial control example in [11].

High data rates are readily available with USB, but its reliance on the host processor for bus level tasks compromises its scalability in real-time control. Conversely, IEEE 1394 self-manages the bus at the physical layer.

The Controller Area Network (CAN, can-cia.org) bus is well-suited for real-time control and has been widely used, but its bandwidth is limited to 1 Mbps.

3. Scalable Platform for Real-Time Control

A key motivation for building a novel control system is to ease the process of developing multi-axis robots in terms of both hardware and software construction. The hardware provides fine-grain real-time control over a large number of motors, with I/O conversion tasks delegated to the actuator sites. Using IEEE 1394, the hardware confines raw analog signals to those sites and multiplexes the digital data for all channels over a high speed serial connection to a single computer. Meanwhile, the API provides a convenient and reusable software interface to the hardware resources.

3.1. Hardware

Figure 3 provides an overview of the control system, with I/O conversion distributed away from the computer to the actuator sites. Each node on the bus contains multiple channels (i.e. axes of control). Nodes can be added to the system by daisy chaining or by direct connection to the computer. The bus is attached to a real-time computer that reads feedback signals from the channels, generates actuation commands, and writes them to their respective channels. The completed controller hardware for the Snake Robot is shown in the photo (Figure 3). Note that this particular hardware has been designed to physically integrate on the top of the Snake Robot actuation units; we will create other form factors for general use.

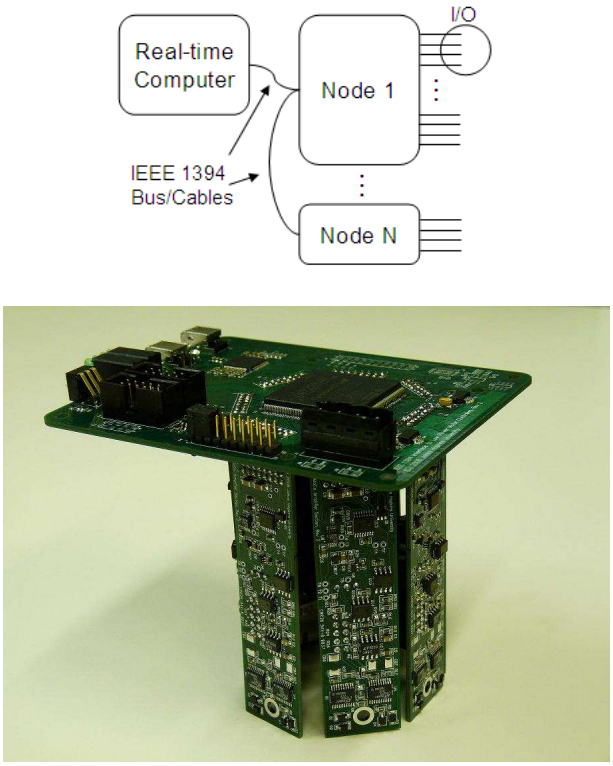

Figure 3 - Conceptual overview of the control system (top) and a photo of the completed controller hardware (bottom)

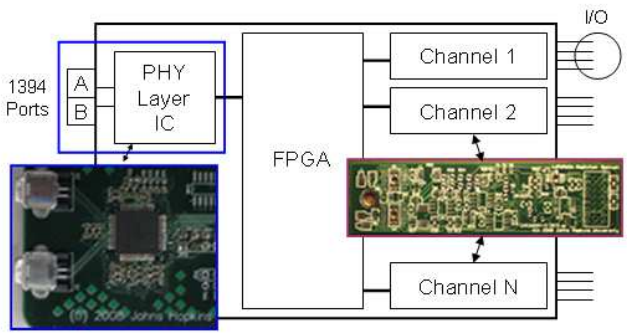

Figure 4 - Block diagram of a node

Nodes—A node (Figure 4) contains circuitry for accessing the I/O channels and handling bus transactions. IEEE 1394 allows up to 63 nodes per bus; each Snake Robot requires two nodes, one for the actuation unit and the other for the Z-Θ and five bar stages combined – a tally of four nodes for two Snake Robots. Multiple buses can be used for yet larger numbers of axes or for heterogeneous control environments.

Channels—One channel contains the I/O components (e.g., DAC, ADC, encoder counters) and power amplification required to control one dc motor. The channel module developed for the Snake Robot is shown in Figures 3 and 4; it also includes a digital potentiometer that allows software configuration for different motors, and a digital switch to select between speed and torque control. These features allow the node-channel set to be used with different robots.

The number of channels per node depends on the physical distribution of joints and is limited by the memory and I/O capacity of the resident FPGA. The low-end Altera Cyclone II FPGA on the Snake Robot controller can comfortably accommodate seven channels.

Field-Programmable Gate Array (FPGA)—Most of the functionality of each node is implemented as firmware on the FPGA, which serves as a low-latency interface between the channel I/O ports and the bus. The FPGA receives packets from the bus, responds to them, and communicates with the I/O devices. The computer can access the channels through control and data registers.

Figure 5 depicts the FPGA operation. A control cycle is initiated when the computer requests a data transaction (read sensors or write actuators). After the FPGA on the addressed node receives the request, it responds immediately with an acknowledgement (required by the IEEE 1394 protocol). For read requests, the FPGA then fetches data from an intermediate buffer and sends them to the computer with a timestamp. The contents of this buffer are refreshed continuously to preserve real-time performance. For writes, the FPGA loads the appropriate buffers and triggers the corresponding channel I/O devices.

Other Components—The Texas Instruments TSB41AB2 is an IEEE 1394 physical layer IC that can handle standard bus speeds up to 400 Mbps. In addition to serving as the interface to the physical bus, it generates the 49.152 MHz clock signal used to synchronize data and clock the FPGA.

For noise isolation, and to facilitate emergency shutdowns, the motor and digital voltages are drawn from separate regulated supplies. Power from the bus is not used for these reasons, as well as to simplify the development effort.

A conventional Linux PC is being used for development, with the intent of migrating to a real-time version of Linux (e.g. RTAI) to run the robot control software. Programs use the libraw1394 library for bus transactions; RT-FireWire [12] is being considered as an alternative.

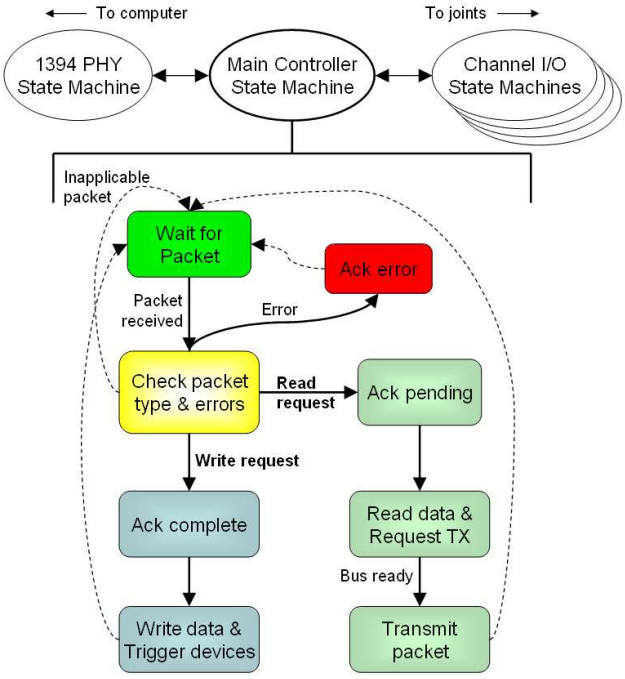

Figure 5 - FPGA structure and operation

3.2. Software

We developed a generic, easy-to-use API for the control hardware described above. The interface has the three-layer hierarchy shown in Figure 6. The top layer consists of abstract I/O operations that can be used for different robots. It includes commands to latch all sensors and to apply all outputs, which are often supported by the hardware (e.g., simultaneously sampled ADCs and double-buffered DACs). All read/write operations are performed for a single axis at a time, requiring just primitive C data types (e.g., int, long).

Abstract I/O Interface (Top)

```
bool LatchAllSensors();
bool GetEncoderPosition(int axis, long &value);
bool GetEncoderVelocity(int axis, long &value);
bool GetMotorCurrent(int axis, long &value);
…
bool SetMotorVoltage(int axis, long value);
bool SetMotorCurrent(int axis, long value);
bool ApplyAllOutputs();
```

Snake Robot I/O Implementation

`LatchAllSensors`: calls `1394_ReadAllData`, stores data (for all axes) in local read buffer
`Get` methods: return data (for single axis) from local read buffer
`Set` methods: write data (for a single axis) to local write buffer
`ApplyAllOutputs`: calls `1394_WriteAllData` to transmit local write buffer (all axes) to nodes

IEEE 1394-Specific Interface

```
bool 1394_WriteAllData(int len, long *data);
bool 1394_ReadAllData(int len, long *data);
```

Figure 6 - General robot control API

The second layer, which implements the abstract interface for the snake robot, is customized to work with data blocks, since for efficiency reasons the data for the multiple axes of a node are bundled into a single bus transaction. The second layer provides access to axis-specific data via local buffers that are filled by LatchAllSensors and emptied by ApplyAllOutputs. So that a fixed block size can be used to simplify the FPGA implementation, the second layer maintains a valid bit for each axis that indicates to the FPGA whether or not to update the corresponding axis during a write transaction. The bottom layer contains function calls to the IEEE 1394 API library (libraw1394). By carefully designing a general robot control API, the developed software can be easily maintained, and the system can be used in other surgical robots.

4. Results and Discussion

The FPGA response to read requests is implemented such that there is no protocol delay (i.e. no busy wait) between receiving a read request and generating a response. Similarly, there is no delay between the receipt of a write request and the start of the write. The I/O device access times are negligible (~2.5 µs, deterministic) relative to the system bandwidth, so as a result loopback tests of the DAC-to-ADC pair and digital I/O consistently return the appropriate values. Bus contention is not expected because the computer is implemented as the bus master and the nodes as slaves.

A read transaction entails a request from the computer, an acknowledgment from the node, and a data response from the node (a *concatenated read* in IEEE 1394—there is also a final acknowledgement from the computer to the node). A write transaction is the same, except for the response (a *unified write*). These transaction types were chosen for their ease of implementation and suitability for the application.

We previously found that the latencies for per-axis bus transactions were significant [18]. Using quadlet (32 data bits) transactions, the average transaction time was 34.5 µs for a read and 30.2 µs for a write. In a straightforward implementation (read-control-write) for a seven-axis robot, a combined read/write time of about 453 µs leaves only 547 µs for control computations at 1 kHz, and is not even feasible at 8 kHz. Given the bus speed of 400 Mbps, we concluded that software overheads were a predominant factor in the latency, as in [13]. As we concluded in [19], it became necessary to bundle the data for multiple axes into blocks in order to overcome these limitations.

Figure 7 shows a raw sampling of read/write times over the full range of block sizes (in quadlets, the smallest unit in IEEE 1394), up to the maximum of 512 quadlets for the 400 Mbps mode. The tests were run with one node connected to a 2 GHz Pentium 4 PC by a 6' cable. Trend lines added to the plots indicate that the base latency is about 33.2 µs for a read and 30.7 µs for a write, which closely matches our previous findings.

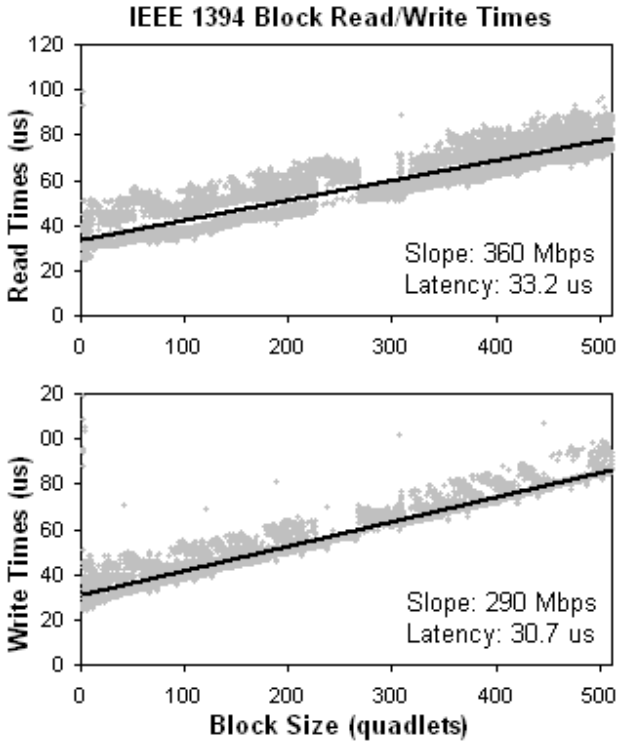

Figure 7 - IEEE 1394 transaction times vs. block size

From the slopes we compute the average speed to be roughly 360 and 290 Mbps for reads and writes respectively. Neither value reaches the nominal 400 Mbps rate due to transmission overhead, but it defies intuition that reads are faster than writes since read transactions are slightly more complicated and incur greater initial latency. A possible explanation may involve differences between how the computer and FPGA request bus access. We intend to resolve this anomaly in future work. At any rate, the results suggest that the number of axes can be scaled significantly with negligible incremental time delay.

High-valued outliers and variability in the transaction times (~20 µs, judging from the noisiness of the plots) may be due to the operating system, as we are using conventional Linux for development purposes. The latter observation may also be explained by variability in obtaining bus access.

5. Conclusions and Future Work

Though parallel buses such as ISA, Q-Bus, Multibus, and VME have become tried-and-true interfaces for robot control, they are increasingly deprecated with the emergence of IEEE 1394, PCI Express, and Ethernet-based protocols, which feature greatly simplified cabling. These high speed serial networks provide higher performance than traditional field buses, such as CAN, SERCOS, and RS-485, which have also been used for real-time control.

The IEEE 1394 bus helps reduce wiring complexity, making systems more robust and scalable to many axes of control. The consolidation of processing tasks eases intra-robot communication (e.g. master-slave) and allows systems to utilize ever-advancing computing power.

This paper describes a scalable controller design based on IEEE 1394 for communication between the computer and actuated joints. The advantages of distributing I/O to less obtrusive sites and centralizing processing are discussed. The concept was demonstrated by a custom controller for a small snake robot for laryngeal surgery. The results confirm real-time performance; the use of large packets carrying data for all nodes may further mitigate the effect of transaction latencies due to software overhead.

Though the described control system is not necessarily a novel design given existing technologies, we contend that it will ease the development of dexterous robots and allow researchers to experiment with more robot-assisted surgical tasks. For example, an additional arm for the Snake Robot can be more conveniently integrated and used for tasks such as camera manipulation. New applications being considered include dexterous ultrasound imaging and ablation. The API will be compatible with a standard medical robotics framework, the Surgical Assistant Workstation [5].

Acknowledgment

We thank Hamid Wasti of Regan Designs, Inc. (Coeur d'Alene, Idaho) for his help with the design and layout of the controller boards, and Renqiu Huang for his assistance with the development of software libraries. This work was funded by the National Science Foundation (NSF) under Engineering Research Center grant #EEC9731748, NSF grant #MRI0722943, and by the Johns Hopkins University internal funds.

References

[1] Simaan, N., R. Taylor, and P. Flint, "A dexterous system for laryngeal surgery," IEEE Robotics & Automation, vol. 1, pp. 351-357, 2004.

[2] Kapoor, A., N. Simaan, and P. Kazanzides, "A system for speed and torque control of DC motors with application to small snake robots," IEEE Mechatronics and Robotics, Aachen, Germany, Sep 2004.

[3] Kapoor, A., "Motion constrained control of robots for dexterous surgical tasks," Ph.D. dissertation, Johns Hopkins Univ., Sep 2007.

[4] Simaan, N., R. Taylor, and P. Flint, "High dexterity snake-like robotic slaves for minimally invasive telesurgery of the upper airway," MICCAI, Rennes-Saint-Malo, France, pp. 17-24, Sep 2004.

[5] Vagvolgyi, B., S. DiMaio, A. Deguet, P. Kazanzides, R. Kumar, C. Hasser, and R. Taylor, "The Surgical Assistant Workstation," MICCAI Workshop on Systems and Arch. for Computer Assisted Interventions (online at http://hdl.handle.net/10380/1466), Sep 2008.

[6] Sarker, M., C. Kim, S. Baek, and B. You, “An IEEE-1394 based real-time robot control system for efficient controlling of humanoids,” IEEE Intelligent Robots and Systems, pp. 1416-1421, Beijing, China, Oct 2006.

[7] Baltazar, G. and G. Chapelle, “Firewire in modern integrated military avionics,” IEEE Aerospace and Electronic Systems Magazine, vol. 16, no. 11, pp.12-16, Nov 2001.

[8] Robertz, S., K. Nilsson, R. Henriksson, and A. Blomdell, “Industrial robot motion control with real-time Java and EtherCAT," IEEE Emerging Technologies & Factory Automation, pp. 1453-1456, Sep 2007.

[9] Lin, S., C. Ho, and Y. Tzou, “Distributed motion control using real-time network communication techniques,” International Power Electronics and Motion Control, vol. 2, pp. 843-847, Aug 2000.

[10] Hagn, U., M. Nickl, S. Jorg, G. Passig, T. Bahls, A. Nothhelfer, F. Hacker, L. Le-Tien, A. Albu-Schaffer, R. Konietschke, M. Grebenstein, R. Warpup, R. Haslinger, M. Frommberger, and G. Hirzinger, “The DLR MIRO: a versatile lightweight robot for surgical applications,” Industrial Robot: An Int’l Journal, vol. 35, no. 4, pp. 324-336, 2008.

[11] Szydlowski, C., “Implementing PCI Express for industrial control,” RTC Magazine, vol. 13, Sep 2004.

[12] Zhang, Y., B. Orlic, P. Visser, and J. Broenink, “Hard real-time networking on FireWire,” RT Linux Workshop, Lille, FR, Nov 2005.

[13] Sarker, M., C. Kim, J. Cho, and B. You, “Development of a network-based real-time robot control system over IEEE 1394: using open source software platform,” IEEE Mechatronics, pp. 563-568, Jul 2006.

[14] Schneider, S., “Making Ethernet work in real time,” Sensors Magazine, vol. 17, no. 11, Nov 2000.

[15] Kerkes, J., “Real-time Ethernet,” Embedded Systems Design, vol. 14, no. 1, Jan 2001.

[16] Pratt, G., P. Willisson, C. Bolton, and A. Hoffman, “Late motor processing in low-impedance robots: impedance control of series-elastic actuators,” American Control Conference, vol. 4, pp. 3245-3251, Jun-Jul 2004.

[17] Lewellen, T., C. Laymon, R. Miyaoka, K. Lee, and P. Kinahan, “Design of a Firewire based data acquisition system for use in animal PET scanners,” IEEE Nuclear Science Symposium Conference Record, vol. 4, pp. 1974-1978, Nov 2001.

[18] Thienphrapa, P. and P. Kazanzides, “A distributed I/O low-level controller for highly-dexterous snake robots,” IEEE BioCAS, pp. 9-12, Nov 2008.

[19] Kazanzides, P. and P. Thienphrapa, “Centralized processing and distributed I/O for robot control,” IEEE TePRA, pp. 84-88, Nov 2008.