An open-source research kit for the da Vinci® Surgical System

Conference Paper *in* Proceedings - IEEE International Conference on Robotics and Automation · May 2014

DOI: 10.1109/ICRA.2014.6907809

6 authors, including:

Anton Deguet
Johns Hopkins University
105 PUBLICATIONS **1,694** CITATIONS

SEE PROFILE

Gregory S Fischer
Worcester Polytechnic Institute
163 PUBLICATIONS **4,074** CITATIONS

SEE PROFILE

Russell Taylor
Johns Hopkins University
407 PUBLICATIONS **11,188** CITATIONS

SEE PROFILE

2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

An Open-Source Research Kit for the da Vinci® Surgical System

Peter Kazanzides† , Zihan Chen, Anton Deguet, Gregory S. Fischer, Russell H. Taylor, and Simon P. DiMaio

***Abstract*— We present a telerobotics research platform that provides complete access to all levels of control via open-source electronics and software. The electronics employs an FPGA to enable a *centralized computation and distributed I/O* architecture in which all control computations are implemented in a familiar development environment (Linux PC) and low-latency I/O is performed over an IEEE-1394a (FireWire) bus at speeds up to 400 Mbits/sec. The mechanical components are obtained from retired first-generation *da Vinci* ® Surgical Systems. This system is currently installed at 11 research institutions, with additional installations underway, thereby creating a research community around a common open-source hardware and software platform.**

I. Introduction

While open-source robot software, such as the Robot Operating System (ROS) [1], has seen widespread adoption, there are relatively few open hardware/software platforms in widespread use within the robotics research community. We consider a platform to be "open" if it allows researchers to modify all levels of the control software. We specifically focus on telesurgical systems, which require *master* input devices, preferably with haptic feedback, and *slave* (or patient-side) robots with the ability to actuate surgical instruments. Currently, there are several haptic input devices with open interfaces, ranging from low-cost systems such as the Phantom Omni (now Geomagic Touch) and Novint Falcon, to more costly alternatives. On the slave side, the Raven II research robot [2] was recently disseminated to several institutions via support from the National Science Foundation (NSF) and is available for purchase from Applied Dexterity, Inc. (Seattle, WA). The Raven II enables researchers to modify the real-time servo control code, which runs on a Linux PC and communicates with the hardware (e.g., motors and encoders) via a USB interface. For research purposes, it is also possible to employ a non-medical robot with open interfaces, such as the Whole Arm Manipulator (WAM, Barrett Technology, Inc., Cambridge, MA). An open telesurgical platform can be created from these components, but would likely require significant system integration effort and would not present a unified control framework.

One alternative is to create a research platform from a complete telesurgical system, such as the da Vinci Surgical System ® (Intuitive Surgical, Inc., Sunnyvale, CA). The da Vinci System, however, is a proprietary product and therefore provides limited access to researchers. It can be configured (by the manufacturer) to provide a read-only research interface to both the master and slave manipulators [3]. While useful for some research projects (e.g., skill assessment), this interface does not enable modification of the control algorithms and therefore cannot support research in new control methods, including autonomous or semi-autonomous control.

†P. Kazanzides, Z. Chen, A. Deguet, and R. Taylor are with the Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA. P. Kazanzides can be reached at pkaz@jhu.edu. G.S. Fischer is with the Dept. of Mechanical Engineering, Worcester Polytechnic Institute, Worcester, MA, USA. S. DiMaio is with Intuitive Surgical, Inc., Sunnyvale, CA USA.

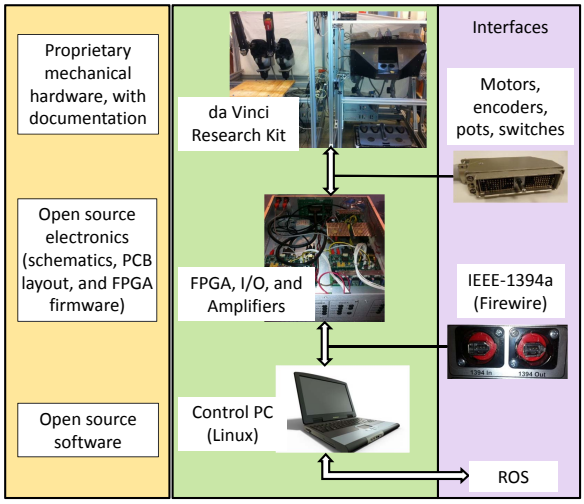

Fig. 1. Overview of telerobotic research platform: Mechanical hardware provided by *da Vinci* Surgical System, electronics by open-source IEEE-1394 FPGA board coupled with Quad Linear Amplifier (QLA), and software by open-source *cisst*/SAW package with ROS interfaces.

This paper presents the development of an "open-source mechatronics" system, consisting of electronics, firmware, and software (see Fig. 1), that is being used to control research systems based on the first-generation da Vinci system. This robot hardware is becoming increasingly available to researchers via the reuse of retired clinical systems. The following sections describe the overall design approach, followed by the da Vinci mechanical components, open-source control electronics, firmware, and software. The software is organized into several layers to enable researchers to integrate with their desired robot framework. We provide a fully functional component-based system using the open-source *cisst* libraries [4], [5], [6]. These libraries support both real-time device (robot) control and real-time computer vision, which are necessary components of telesurgical robot systems. The Surgical Assistant Workstation (SAW) package, which is built on *cisst*, includes components that implement interfaces to many devices, including haptic input devices and robots [7], [8]. For the research da Vinci System, our implementation includes SAW components for low-level I/O, joint-level control, high-level control, and teleoperation. It also includes components that provide ROS interfaces.

II. System Design

The primary design goal is to provide a system that enables researchers to easily implement new algorithms at any level of control. We therefore did not use an off-the-shelf motor controller because it would not allow modification of the low-level servo control algorithm. We assume that researchers will be familiar with a Linux development environment, preferably with either the RT-Preempt patch or a real-time extension such as Xenomai or RTAI, and therefore focused on a system architecture that enables all software to be implemented in this environment.

We considered several design approaches, which can be categorized based on whether the computation and I/O are centralized or distributed. Historically, processing and network limitations favored either *centralized computation and I/O*, where all robot cables are connected to I/O boards inside a central computer, or *distributed computation and I/O*, where high-level control is performed on a single computer, with low-level control performed on embedded microprocessors connected via a serial network such as Controller Area Network (CAN), Ethernet, or RS-485. The latter approach did not require a high-performance (i.e., low latency and high bandwidth) network because the high-level control typically executes at hundreds of Hertz and provides setpoints to the low-level control at this rate. The availability of high-speed serial networks with real-time performance, such as Ethernet for Control Automation Technology (EtherCAT), SERCOS III, and IEEE-1394 (FireWire), has enabled an approach that can be called *centralized computation and distributed I/O* [9]. In this approach, the real-time communication network allows all control computations to be implemented on a high-performance computer that contains a familiar software development environment (e.g., Linux, with or without real-time extensions), while preserving the advantages of reduced cabling by distributing the I/O. This allows a researcher to develop both high-level supervisory control and low-level joint control in the same development environment, thus enabling high flexibility in control algorithms while maintaining precise real-time hardware control. This is particularly useful for developing haptic interactions and virtual fixtures. We implemented this architecture by designing custom electronics that uses a field-programmable gate array (FPGA) to provide direct, low-latency, interfaces between the high-speed serial network (IEEE-1394a, in our case) and the I/O hardware. We chose IEEE-1394a because it is widely available, has high performance (up to 400 Mbits/sec), supports daisy-chaining at the physical layer, and there is ample documentation [10] to enable implementation of the link layer protocol on an FPGA, as described in Section IV-D. The potential disadvantages of IEEE-1394a are the lack of high-flex cables and the length limits that make it difficult to route cables inside a robot arm.

The *centralized computation and distributed I/O* architecture has been used in other systems. Pratt reported a system that uses IEEE-1394a to communicate between a control PC and distributed FPGA boards in a 12-axis biped robot system[11]. The MIRO surgical robot developed by the German Aerospace Center (DLR) uses SpaceWire, a 1 GB/s full duplex serial link with latency less than 20 μs, to connect distributed FPGA-based I/O boards to a centralized control PC, running the QNX real-time operating system [12]. Among the Ethernet-based real-time protocols, EtherCAT appears to be gaining the widest deployment. As an example, Willow Garage uses EtherCAT to close a 1 kHz loop between a control PC (with real-time operating system) and the encoders and motors in its two-armed mobile robot system (PR2) [13].

III. Mechanical Hardware

The mechanical hardware is obtained from retired first-generation *da Vinci* Surgical Robot Systems (often called *da Vinci* Classic). There are two paths for researchers to obtain this hardware: (1) by directly acquiring a retired clinical system (e.g., from a local hospital), or (2) by obtaining the Research Kit for the *da Vinci* System from Intuitive Surgical. The Research Kit consists of the following components: two Master Tool Manipulators (MTMs), two Patient Side Manipulators (PSMs), a High Resolution Stereo Viewer (HRSV), a footpedal tray, and documentation (e.g., wiring diagrams, connector pinouts, kinematic parameters). It does not include the passive Setup Joints that support the PSMs, the Endoscopic Camera Manipulator (ECM), the stereo endoscope, control electronics, and software. Because the electronics and software are either proprietary (closed) or not included, it motivates the development of a common, open-source electronics and software platform for the research community, which is described in the following sections.

IV. Electronics

The control electronics is based on two custom boards (Fig. 2): (1) an IEEE-1394 FPGA board, and (2) a Quad Linear Amplifier (QLA). The schematics, firmware, low-level software interface, and documentation are available via a public git repository. These boards were designed for general mechatronics use, but are well suited for controlling the da Vinci Surgical System. Although we expect that most research will be implemented via software on the PC (as encouraged by the *centralized computation and distributed I/O* architecture), the availability of the electronic designs (Altium Designer format) and FPGA firmware (Verilog source code) enables researchers to modify any aspect of the system. Some possibilities include: (1) an alternate FPGA board with a different interface, such as EtherCAT, that could be used with the QLA, (2) an I/O board to mate with the IEEE-1394 FPGA board and interface to different hardware, such as the da Vinci passive setup joints, and (3) estimation or closed-loop control in the FPGA firmware.

A. IEEE-1394 FPGA Board

This board contains a Xilinx Spartan-6 XC6SLX45-2 FPGA, configuration PROM, IEEE-1394a physical layer (PHY), two IEEE-1394a 6-pin connectors, a low-speed USB interface (virtual COM port), and required power supplies. It contains two 44-pin connectors that provide power and FPGA I/O to a companion board, such as the QLA. It also contains a 16-position rotary switch for board identification.

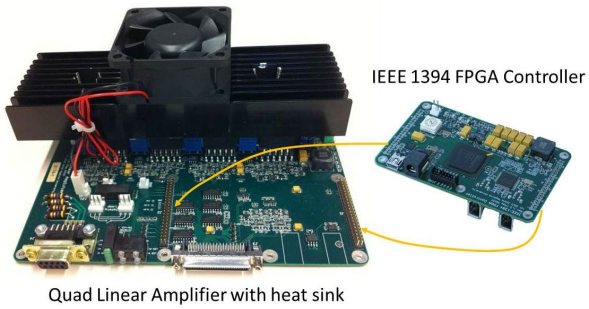

Fig. 2. IEEE-1394 FPGA board and Quad Linear Amplifier (QLA)

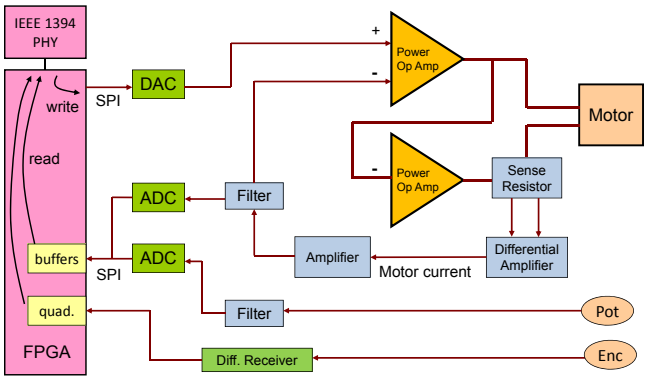

Fig. 3. Block diagram of I/O devices (digital I/O, safety relay, and temperature sensors not shown)

B. Quad Linear Amplifier (QLA)

The Quad Linear Amplifier attaches to the IEEE-1394 FPGA board and provides all hardware required for current (torque) control of four DC brush motors, using a bridge linear amplifier design (Fig. 3). Each of the four channels contains the following components:

- One 16-bit digital-to-analog converter (DAC) to enable the FPGA to set the desired motor current.
- Two 16-bit analog-to-digital converters (ADCs) to digitize the measured motor current and an external analog sensor (e.g., potentiometer).
- Differential receivers for one quadrature encoder with A, B, and Z (index) channels; these signals are routed to the FPGA board for quadrature decoding.
- Two OPA-549 power operational amplifiers (op amps) to provide bi-directional control of a motor from a single power supply (up to 6.25 Amps at up to 48 Volts).
- Digital inputs for one home and two limit switches; these can also be used as general-purpose inputs.
- One open-collector digital output with high current drive (up to 1 Amp).

The board also contains a software-controlled, normally-open safety relay, which allows the software to disable the motor power supply, and two heat sink temperature sensors.

C. Controller Packaging

The electronics have been packaged into rackmount enclosures (see Fig. 4) by the group at Worcester Polytechnic Institute. Each enclosure contains two sets of FPGA and

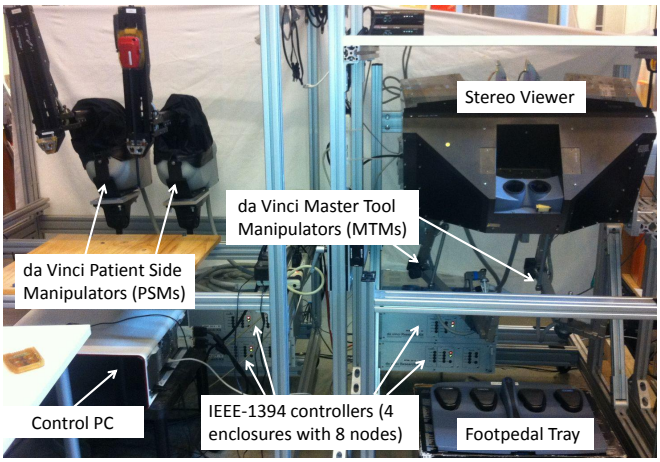

Fig. 4. da Vinci Research Kit with four enclosures (two MTMs, two PSMs)

QLA boards, which are sufficient to drive a single da Vinci manipulator (MTM or PSM). The enclosure also contains all necessary power supplies and a manipulator interface board, provided by Intuitive Surgical, Inc., that connects the DL-156 Zero Insertion Force (ZIF) connector used by the da Vinci manipulators to the DB9 and VHDCI68 connectors used by the QLA for motor power and signals, respectively. This simplifies the hardware setup, since a researcher must only connect the da Vinci manipulator cable, one or two FireWire cables, and a power cord (see accompanying video).

D. FPGA Firmware

The FPGA has three major responsibilities: (1) exchanging data with the PC via the IEEE-1394 bus, (2) interfacing to I/O devices, and (3) hardware-level safety checking.

The IEEE-1394 protocol supports two types of services: isochronous and asynchronous transfers. We selected asynchronous transfers for our application because it was relatively easy to implement in the FPGA and was sufficient to perform servo control at a 1 kHz rate. To conserve FPGA resources and simplify implementation, we implement only a subset of the IEEE-1394 link-layer protocol. Specifically, our FPGA nodes are not capable of serving as bus master (we instead rely on the PC to fulfill this role) and all transfers much be asynchronous quadlet (32-bit) or block (multiple quadlets) read or write transactions.

When the FPGA receives a write packet over the IEEE-1394 bus, it does the following: (1) checks the incoming packet's Cyclic Redundancy Check (CRC) and silently drops the packet if the CRC is invalid, (2) generates and sends an acknowledgement packet, (3) decodes the destination device address and data, and (4) writes data to internal registers and I/O devices. For example, the desired motor current is shifted out via the Serial Peripheral Interface (SPI) to the DAC. Similarly, to respond to a read request from the PC, the FPGA latches various I/O device data and sends all requested data in a single block transfer. To avoid latency, the FPGA ensures that all feedback data is available in local registers. For example, because one ADC conversion cycle requires 0.7 μs, the FPGA firmware continuously requests data conversions and stores the results in registers.

This communication protocol is sufficient for implementing a 1 kHz control loop on a system with two MTMs and two PSMs (8 FPGA nodes). As we have previously measured, an IEEE-1394 asynchronous read or write transaction (for a small number of quadlets) requires approximately 35μs [14]. Thus, a read and write to each of the 8 boards requires approximately 560μs, which is a little more than half of the available cycle time. Fortunately, the IEEE-1394 protocol supports broadcast communication and peer-to-peer transfers, so it is possible to achieve faster control rates or scale up to more axes by having the PC broadcast a single control packet to all FPGA nodes and read a single feedback packet that has been assembled via peer-to-peer transfers between the FPGA nodes. Our preliminary testing has indicated that the broadcast write has a lower overhead (because there is no acknowledgment packet) and that a peer-to-peer transfer between two FPGA nodes requires less than 5μs. Experimentally, we have found that with this protocol, the I/O time on an 8-node system can be reduced to approximately 100μs.

In addition to the read and write requests to the devices involved in motor control, the FPGA firmware also supports reading and writing to the configuration PROM that initializes the FPGA. It is therefore possible to update the firmware via the IEEE-1394 interface, which provides several advantages: (1) no special JTAG programming cable is required, (2) no special programming software is required, and (3) it is much faster than the conventional JTAG programming method (about 20 seconds versus several minutes). There is no protection, however, against programming failures that could prevent future firmware updates via the IEEE-1394 interface; in this unlikely case, the JTAG programming cable would be required.

The firmware currently includes two safety features: a watchdog timer and a motor current safety check. The watchdog timer provides a range of timeout periods from 1 to 340 ms (setting the period to 0 disables it). If the watchdog is not refreshed during this period (by writing to the FPGA), it trips and disables all power amplifiers. This is especially useful when the PC control software exits or communication is lost. The motor current safety module is designed to catch cases where the absolute value of the measured motor current is significantly greater than the commanded motor current, which would indicate a hardware defect.

V. Software

In general, telerobotic software can be arranged into the following *functional* layers (see Fig. 5): hardware interface (I/O), low-level control (e.g., PID), high-level control, teleoperation, and application. Within each of these functional layers, we provide one or more of the following *development* layers: primitive, object-oriented, and component-based. The primitive layer is so named because it uses only primitive C/C++ data types and avoids dependencies on external packages. The object-oriented layer consists of C++ classes that implement most of the functionality. This layer is provided for researchers who either do not want to use a component-based architecture, or prefer to use a different real-time framework, such as Orocos [15]. The component-based layer uses the classes from the object-oriented layer to implement the components, using the *cisst* component-based framework. Some components, such as the PID controller, are reused from the Surgical Assistant Workstation (SAW). Both packages are available via a public SVN/Trac repository, https://trac.lcsr.jhu.edu/cisst (soon to be on GitHub). The following sections describe the functional layers and illustrate the three development layers that exist within the Hardware Interface layer.

A. Hardware Interface Layer

The **primitive development layer** is provided by a C++ library that enables direct access to the raw I/O data via the IEEE-1394 bus. This library has no external software dependencies, other than `libraw1394`, which is a standard Linux library for communication over IEEE-1394. Other drivers, such as RT-FireWire [16], could be used to obtain hard real-time performance. There is also a Microsoft Windows implemention of `libraw1394`[17].

The API consists of two main classes: a `FirewirePort` class to represent an IEEE-1394 port, and an `AmpIO` class to represent one FPGA node on the bus. For a typical system, one FireWire port will connect to multiple FPGA nodes; thus the `FirewirePort` object maintains a list of `AmpIO` objects. The `FirewirePort` class contains two methods, `ReadAllBoards` and `WriteAllBoards`, which read all feedback data into local buffers and transmit all output data from local buffers, respectively. This allows the class to implement more efficient communication mechanisms, such as the broadcast write and consolidated read described in Section IV-D. The `AmpIO` API provides a set of functions to extract feedback data, such as encoder positions, from the read buffer, and to write data, such as desired motor currents, into the write buffer. All data types are unsigned integers because they are stored as counts (or bits) in FPGA registers.

The **objected-oriented development layer** provides a more convenient API because it relies on a vector package to represent robot data as vectors of meaningful units, such as radians and millimeters. We use conditional compilation to select either *cisstVector* (the vector library within *cisst*) or Eigen. This layer also defines data structures for configuration parameters, such as sensor scale factors and the mapping of hardware to robot joints. Configuration file parsers can be created to populate these data structures from any defined format. Currently, the software includes an XML file parser.

The **component-based development layer** consists of the `mtsRobotIO1394` component, which is a "wrapper" around the software libraries provided by the primitive and object-oriented development layers described above. This component is specific to the FPGA-1394/QLA board set, but is not specific to the da Vinci robot. It contains several provided interfaces: one for each configured robot (4 in the case of Fig. 5) and one for each configured digital input (e.g., for the footpedal and PSM buttons, not shown in Fig. 5). Finally, this layer includes an optional Qt Widget component (Fig. 6) that provides a convenient interface to `mtsRobotIO1394`.

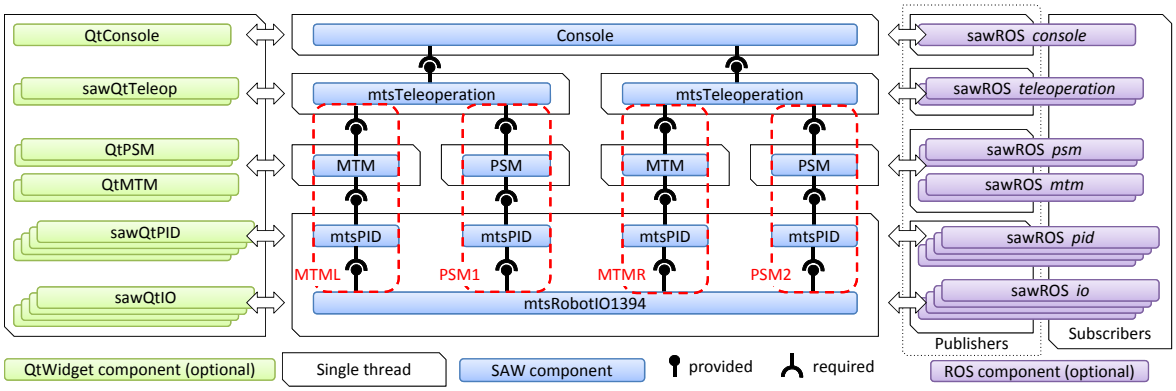

Fig. 5. Robot tele-operation control architecture with two MTMs and two PSMs, arranged by functional layers and showing thread boundaries

B. Control and Application Layers

The low-level control layer consists of the PID joint controllers (one for each manipulator), which are general-purpose SAW components that are configured via an XML file. The high-level control is provided by two components that are specific for the da Vinci MTM and PSM. These provide the forward and inverse kinematics, trajectory generation, and gripper control. They also manage the state transitions for the da Vinci manipulators, such as homing (MTM and PSM), engaging the sterile adapter plate (PSM), and engaging the instrument (PSM). The teleoperation layer is provided by two instances of a general-purpose SAW component that each connect one MTM to one PSM. Finally, the application layer is provided by a console application that emulates the master console environment of a da Vinci system. Each layer also includes an optional Qt Widget that can be used to visualize and interact with the corresponding SAW component.

One challenge for such a component-based approach is data synchronization; this is especially true for servo loop control running at a high frequency of 1 kHz or greater. If a separate thread is created for each servo control loop and the I/O component, it is likely that the feedback data used in the servo loop control could be out of synchronization and potentially affect controller performance. As illustrated in Fig. 5, our solution puts the I/O component and all servo control (PID) components in one single thread, while keeping the advantage of a component-based approach.

C. ROS Interfaces

ROS (Robot Operating System) provides a set of libraries and utility tools and enables communication between different robot control processes in one computer or across multiple computers [1]. We developed components that publish the robot state in ROS messages and accept commands by subscribing to ROS messages (topics). This is embodied in the *sawROS* library, which contains: (1) a set of global data type conversion functions (e.g., *cisst* matrix to ROS `geometry_msgs::Transform` and vice versa), (2) a *cisst* publisher that fetches, converts, and then publishes the data, (3) a *cisst* subscriber with a ROS subscriber callback function that converts data and triggers the corresponding *cisst* write

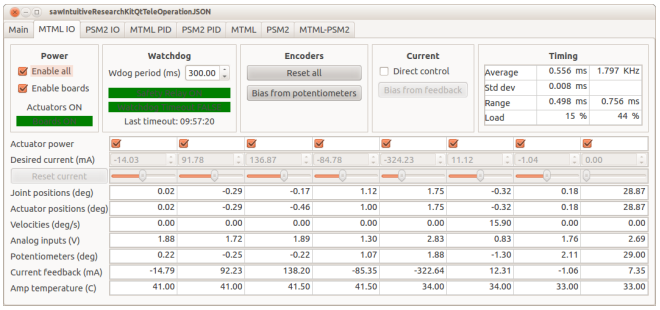

Fig. 6. Qt Widget interface to Robot I/O component

function, and (4) a *cisst*-to-ROS bridge component that serves as a container for *cisst* publishers and subscribers. Each bridge runs periodically at the publishers' desired frequency. A separate thread is used for the ROS event loop (`ros::spin`) that handles the subscribers.

In addition, MTM and PSM models have been generated in Unified Robot Description Format (URDF) and can be used for visualization and simulation. Some use cases that take advantage of the above mentioned ROS interface and simulation are to use a real MTM and foot pedal as input devices to tele-operate a simulated PSM or alternate slave robot, such as the Raven-II[2]. We expect that most ROS users will interface to the high-level MTM and PSM control components and implement their own teleoperation control and applications within ROS.

VI. Research Community

A research community is forming around this common hardware and software platform. Intuitive Surgical has created a public wiki page at research.intusurg.com/dvrk. This page lists all groups that currently have research da Vinci systems, with links to group pages on that wiki. The group wiki pages require a login and password, and are used to share information between the researchers. For example, one group has uploaded the design files for an aluminum frame that can be used to mount the MTMs, PSMs, and Stereo Viewer. There is also a google group, *research-kit-for-davinci*, that provides a mailing list for discussion among researchers.

The open source mechatronics and software are not specific to the da Vinci Research Kit and are hosted in public

Fig. 7. Research da Vinci System at UBC, with 6 controller boxes (and 1 spare) to power 2 MTMs, 3 PSMs, and 1 ECM (functionally compatible with a PSM). Photo courtesy of Omid Mohareri, UBC.

repositories. The mechatronics is on GitHub, consisting of a overall project page, jhu-cisst.github.io/mechatronics, with separate git repositories for the board designs, FPGA firmware, and low-level software. The *cisst*/SAW software, including the ROS bridge, is available via an SVN/Trac server at JHU (soon to be moved to GitHub).

The mechatronics hardware is built in production batches to reduce overall cost. The first batch provided hardware for JHU, WPI, Stanford University, and the University of British Columbia (UBC). These systems have been installed and are actively used for research. The system at JHU has been used to teleoperate other slave robots in ground-based simulations of satellite servicing [18]. Figure 7 shows the system at UBC, which uses a retired clinical da Vinci. This system is currently being used for research in the applicability of an asymmetric force feedback control framework for bimanual robot-assisted surgery. The da Vinci Research Kit at Stanford has been used to study how users modulate their grip force when interacting with an environment with elastic forces [19]. The second batch provided systems that were installed at 7 additional sites, and the third batch is currently underway.

VII. Conclusions

This paper presented a telerobotics research platform that is based on the *da Vinci* Surgical System, with open-source electronics and software. The software is implemented in a component-based C++ framework, with ROS interfaces to facilitate integration with other systems and software packages. Low-level (primitive) interfaces and object-oriented interfaces are available for researchers who do not wish to adopt a component-based architecture or who wish to integrate with a different component-based framework. The platform has been replicated at several research institutions – currently 11 sites have acquired the platform, with additional sites in process. Collaboration tools include wikis and mailing lists, with the open source hardware and software available via public git or svn repositories.

Acknowledgments

This work was supported by National Science Foundation grants EEC 9731748, EEC 0646678, MRI 0722943, NRI 1208540, and by NASA NNX10AD17A. Paul Thienphrapa, Simon Leonard, Kwang Young (Eddie) Lee, Jonathan Bohren, Ravi Gaddipati, Lawton Verner, Ankur Kapoor, and Tian Xia provided technical assistance at JHU. Gang Li, Nirav Patel, and Zhixian Zhang contributed at WPI, as did Alex Camilo from Neuron Robotics. Arpit Mittal, Kollin Tierling, and Dale Bergman provided assistance at ISI.

References

[1] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[2] B. Hannaford, J. Rosen, D. Friedman, H. King, P. Roan, L. Cheng, D. Glozman, J. Ma, S. N. Kosari, and L. White, "Raven-II: An open platform for surgical robotics research," *IEEE Trans. on Biomedical Engin.*, vol. 60, no. 4, pp. 954–959, Apr 2013.

[3] S. DiMaio and C. Hasser, "The da Vinci research interface," in *MICCAI Workshop on Systems and Arch. for Computer Assisted Interventions*, Midas Journal: http://hdl.handle.net/10380/1464, July 2008.

[4] A. Kapoor, A. Deguet, and P. Kazanzides, "Software components and frameworks for medical robot control," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2006, pp. 3813–3818.

[5] A. Deguet, R. Kumar, R. Taylor, and P. Kazanzides, "The *cisst* libraries for computer assisted intervention systems," in *MICCAI Workshop on Systems and Arch. for Computer Assisted Interventions*, Midas Journal: http://hdl.handle.net/10380/1465, Sep 2008.

[6] M. Y. Jung, A. Deguet, and P. Kazanzides, "A component-based architecture for flexible integration of robotic systems," in *IEEE/RSJ Intl. Conf. on Intell. Robots and Systems (IROS)*, 2010, pp. 6107–6112.

[7] B. Vagvolgyi, S. DiMaio, A. Deguet, P. Kazanzides, R. Kumar, C. Hasser, and R. Taylor, "The Surgical Assistant Workstation: a software framework for telesurgical robotics research," in *MICCAI Workshop on Systems and Arch. for Computer Assisted Interventions*, Midas Journal: http://hdl.handle.net/10380/1466, Sep 2008.

[8] P. Kazanzides, S. DiMaio, A. Deguet, B. Vagvolgyi, M. Balicki, C. Schneider, R. Kumar, A. Jog, B. Itkowitz, C. Hasser, and R. Taylor, "The Surgical Assistant Workstation (SAW) in minimally-invasive surgery and microsurgery," in *MICCAI Workshop on Systems and Arch. for Computer Assisted Interventions*, Midas Journal: http://hdl.handle.net/10380/3179, Jun 2010.

[9] P. Kazanzides and P. Thienphrapa, "Centralized processing and distributed I/O for robot control," in *Technologies for Practical Robot Applications (TePRA)*, Woburn, MA, Nov 2008, pp. 84–88.

[10] D. Anderson, *FireWire System Architecture, 2nd Edition*. MindShare, Inc., Addison-Wesley, 1999.

[11] G. Pratt, P. Willisson, C. Bolton, and A. Hofman, "Late motor processing in low-impedance robots: impedance control of series-elastic actuators," in *American Control Conference*, Jun 2004, pp. 3245–3251.

[12] U. Hagn, M. Nickl, S. Jörg, G. Passig, T. Bahls, A. Nothhelfer, F. Hacker, L. Le-Tien, A. Albu-Schäffer, 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 Intl. Journal*, vol. 35, no. 4, pp. 324–336, 2008.

[13] R. Rusu, I. Sucan, B. Gerkey, S. Chitta, M. Beetz, and L. Kavraki, "Real-time perception-guided motion planning for a personal robot," in *IEEE/RSJ Intl. Conf. on Intell. Robots and Systems (IROS)*, 2009, pp. 4245–4252.

[14] P. Thienphrapa and P. Kazanzides, "A scalable system for real-time control of dexterous surgical robots," in *Technologies for Practical Robot Applications (TePRA)*, Nov 2009, pp. 16–22.

[15] H. Bruyninckx, P. Soetens, and B. Koninckx, "The real-time motion control core of the Orocos project," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, vol. 2, Sep 2003, pp. 2766–2771.

[16] Y. Zhang, B. Orlic, P. Visser, and J. Broenink, "Hard real-time networking on FireWire," in *RT Linux Workshop*, Nov 2005.

[17] M. A. Tsegaye, "A comparative study of the Linux and Windows device driver architectures with a focus on IEEE1394 (high speed serial bus) drivers," Master's thesis, Dept. of Computer Science, Rhodes University, Dec 2002.

[18] T. Xia, S. Leonard, I. Kandaswamy, A. Blank, L. Whitcomb, and P. Kazanzides, "Model-based telerobotic control with virtual fixtures for satellite servicing tasks," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 2013.

[19] T. Gibo, D. Deo, Z. Quek, and A. Okamura, "Effect of load force feedback on grip force control during teleoperation: A preliminary study," in *IEEE Haptics Symposium*, Feb 2014.