MODULAR INTEROPERABILITY IN SURGICAL ROBOTICS SOFTWARE

BY PETER KAZANZIDES
RESEARCH PROFESSOR

ANTON DEGUET
ASSOCIATE RESEARCH SCIENTIST

BALAZS VAGVOLGYI
ASSOCIATE RESEARCH SCIENTIST

ZIHAN CHEN
PHD CANDIDATE

RUSSELL H. TAYLOR
PROFESSOR

DEPARTMENT OF
COMPUTER SCIENCE
JOHNS HOPKINS UNIVERSITY

Computers have been used to assist medical diagnosis and treatment for decades; early examples include computer-assisted tomography (CAT or CT) and stereotactic neurosurgery. While computers can only provide information to guide a surgeon, the introduction of robotics enables computers to physically act on the patient, either directly or by providing mechanical assistance to the surgeon. For example, in stereotactic neurosurgery, the location of a suspected tumor is identified in a three dimensional (3D) CT scan of a patient's brain. Initially, passive stereotactic frames were used to position a guide for a biopsy needle based on the 3D coordinates of the suspected tumor. In 1985, a robot was used to position the needle guide [1], representing the first reported clinical use of a robot. In the early 1990s, robots were introduced for invasive surgical procedures, such as transurethral resection of the prostate (TURP) [2] and total hip replacement (THR) surgery [3]; in both of these cases, the robot autonomously performed part of the surgical procedure.

Today, the da Vinci Surgical System (Intuitive Surgical, Sunnyvale, CA) is the most widely used surgical robot, with 3,266 installations according to the company's 2014 Annual Report. The da Vinci is used for minimally-invasive surgery, especially in urologic and gynecologic applications, but is currently limited to teleoperated control, where the surgeon sits at a master console and controls instruments inserted into the patient's body through small incisions, called ports. Stereo visualization is provided by a stereo endoscope (also robotically controlled) inserted through one of the ports.

Recently, some common research platforms have emerged. The da Vinci Research Kit (dVRK) [4] is a research system based on the mechanical components of the first-generation da Vinci Surgical System. Another common platform is provided by the Raven II surgical robot (Applied Dexterity, Inc., Seattle, WA) [5], which is functionally similar to the da Vinci Patient Side Manipulator (PSM).

The focus of this article is on modular interoperability of the software that is used for these types of systems. This is important for several reasons. First, surgical robots are not just robots, but rather integrated systems that typically incorporate other sources of information. This includes static information, such as preoperative images or models, and real-time information such as mono or stereo computer vision, ultrasound, optical coherence tomography (OCT), fluoroscopy (x-ray), tissue properties, external forces, and user (surgeon) input. There is no single software package that can provide all these capabilities and few, if any, researchers have expertise in all of them. Thus, it is necessary to have modular interfaces to enable interoperability between the different software packages that incorporate the state-of-the-art knowledge and capabilities in each area. Second, even within robotics, there is a need for interoperability between systems. For example, the da Vinci Console (stereo display and master manipulators) could be used to teleoperate other robots, including the Raven II.

SYSTEM AND SOFTWARE ARCHITECTURES

Surgical robots typically adopt the hierarchical multi-rate control architecture that is found in general robotics. This architecture is depicted in **Figure 2** and each layer is further discussed in Section 3. The Hardware and Low-Level Control (LLC) layers are similar to those in general robotics, possibly with additional safety mechanisms. The unique characteristics of surgical robots become more evident at the High Level Control (HLC) and Application layers. The HLC may interface to external sensing, such as force sensing or real-time imaging, to implement closed-loop behaviors based on these sensors. The Application layer implements the surgical workflow and user interface and may include interfaces to other sub-systems, such as a database of patient information and possibly other medical devices.

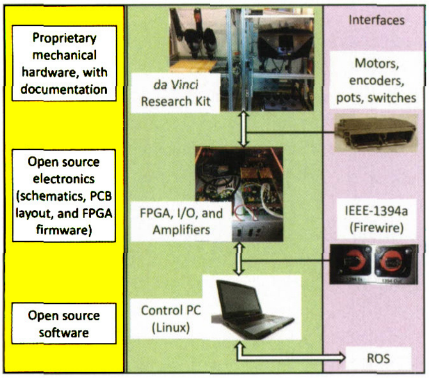
Proprietary mechanical hardware, with documentation
Open source electronics (schematics, PCB layout, and FPGA firmware)
Open source software
da Vinci Research Kit
FPGA, I/O, and Amplifiers
Control PC (Linux)
Interfaces
Motors, encoders, pots, switches
IEEE-1394a (Firewire)
ROS

FIGURE 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.**

From a wider perspective, the surgical robot is often just one component in a larger medical system. **Figure 3** depicts one representative system design that incorporates a telesurgical robot and an ultrasound scanner. In this figure, the Left Master Robot controls Slave Robot 2, which holds an ultrasound (US) probe, and the Right Master Robot controls Slave Robot 1, which holds the surgical instrument (not shown). In a conventional telesurgical setup, the Cartesian position of each Master Robot provides the desired Cartesian position of the corresponding Slave Robot (after transformations from the master to slave coordinate systems). The conventional setup also includes a stereo Camera that provides video images that are displayed on the stereo Diplay Hardware. The new capabilities are due to the addition of the US probe. The system acquires the US images and a Feature Detection module looks for a specified target inside the organ. If the target is found, it is presented as an augmented reality overlay (e.g., a cross-hair marker) in the stereo display and is used to provide haptic guidance to the surgeon via the Right Master Robot. For example, the High-Level Controller can apply a small force on the Right Master Robot to guide the instrument held by Slave Robot 1 to the target. Alternatively, if the target is a critical structure that the surgeon should avoid, the system can impose a safety barrier to prevent accidental damage. In either case, it is necessary for the target to be transformed to the camera and robot coordinate systems. The transformation to camera coordinates is enabled by the Tool Tracking module, which detects the US probe in the stereo images. This module uses the measured position of Slave Robot 2, which is subject to kinematic and non-kinematic errors, but improves the accuracy by directly detecting the tool in the stereo images. Finally, the target position is transformed from camera coordinates to robot coordinates using a known (calibrated) transformation matrix.

One key point in **Figure 3** is that while hierarchical multi-rate control may be suitable for the master and slave robots, there is also a requirement to handle the video and ultrasound images. These image channels have their own timing requirements; for example, the video will typically run at about 30 frames per second, whereas the US is likely to run at a different rate. Furthermore, execution of these channels is distinct from the periodic execution of the robot's low-level and high-level controllers. But, it is also necessary to share data between the channels and the robot controller, as illustrated in **Figure 3**.

The above example motivates the discussion of a software architecture to enable its implementation. In robotics (as in other domains), the original functional programming model gave way to object-oriented programming (OOP), and has more recently transitioned to component-based software engineering (CBSE). In OOP, each module in the system is an object that is an instance of a class. The class methods define the capabilities of that module. For example, the low-level controller (LLC) could contain methods to query the current joint position and to move the robot to a new joint position. The high-level controller would directly invoke the LLC methods; for example, the current Cartesian position is obtained by querying the joint position and then applying forward kinematics. The OOP approach represents a tight coupling, where objects directly invoke methods of other objects. It is more challenging to implement when multiple computations occur in parallel at different rates, as in **Figure 3**, because data transfer between parallel execution threads requires proper use of synchronization primitives such as mutexes and semaphores.

In CBSE, the various modules in **Figure 3** become separate components and interact via message passing. This results in a loose coupling between the components. Essentially, CBSE is similar to the electrical engineering domain, in that software components are the equivalent of integrated circuits, and systems are built by "wiring" software components much like integrated circuits are wired together on circuit boards. Some CBSE implementations require each component to be in a separate process, whereas others enable multiple components to exist in a single, multi-threaded process. The latter is advantageous for hard real-time systems because communication between components can be done more efficiently, especially when the framework provides efficient, thread-safe mechanisms, as in Orocos [6] and *cisst* [7]. In

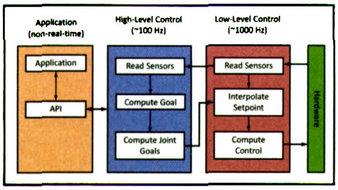

FIGURE 2
Layers of Canonical Robot Control Architecture.

contrast, the Robot Operating System (ROS) [8] requires each component (node) to be a separate process, although there is support for multi-threading via nodelets.

SYSTEM LAYERS AND INTERFACES

The following sections describe the layers shown in **Figures 2** and **3**, focusing on the interfaces to each layer. Although the real-time data channel could be considered part of the high-level control, it is sufficiently distinct from traditional high-level robot control to warrant its own subsection. For interfaces, the Robot Operating System (ROS) [8] provides a common middleware and standardized message types for robots and other devices and has been widely adopted by robotics researchers. The current version of ROS is not designed for real-time processing, however, and thus it is more suitable for the higher-level layers. For the lower-level layers, it is common to use a separate framework, such as OROCOS [6] or cisst [7], often with bridges to ROS. Because ROS is best supported on Ubuntu Linux, it is also common to use other standard protocols, such as OpenIGTLink [9], to interface to software on other platforms.

Physical (Hardware) Layer

The physical layer consists of mechatronics hardware, such as motors, encoders, potentiometers, and associated electronics. Traditionally, the electronics has consisted of input/output (I/O) devices, such as analog-to-digital (A/D) or digital-to-analog (D/A) converters, and power amplifiers to drive the motors. Recently, there has been a trend toward intelligent drive electronics, which combine the functions of the physical and low-level control layers.

Many systems employ custom interfaces to the physical layer, though some standard interfaces have emerged. One common standard is CANOpen (www.can-cia.org), originally developed for the Controller Area Network (CAN) bus, but now available for other physical network layers, including Ethernet. Another option is EtherCAT (www.ethercat.org), which uses a standard Ethernet port on the master device (e.g., PC) and custom hardware on the slave devices. Slave devices can be daisy-chained to form a common bus topology, and all slave nodes can receive data and respond via a single Ethernet frame.

For the dVRK shown in **Figure 1,** the physical layer consists of the mechanical components of the da Vinci and the custom electronics provided by the FPGA and QLA boards. The interface to the physical layer is via IEEE-1394a (FireWire), which is well suited for real-time control due to its high bandwidth, low latency, and support for daisy-chaining, broadcast, and peer-to-peer transfers. This interface was selected to achieve a *centralized computation and distributed I/O* architecture [10], where all control computations are performed on a familiar development environment (Linux PC). The FPGA implements the FireWire link layer so that packet data can be sent to, and received from, the I/O hardware with minimal latency. An Ethernet-to-FireWire bridge has recently been prototyped for the dVRK [11] to take advantage of the wider availability of Ethernet.

Low-Level Control (LLC) Layer

The low-level control layer is often referred to as the servo control layer. Typically, it consists of a simple control algorithm, such as proportional-integral-derivative (PID) control, periodically executing at a high rate (e.g., 1 kHz), to control the individual axes of the robot. The typical low-level control flowchart is to read the robot internal sensor feedback, such as joint encoder positions, compute the error between the desired and measured positions and/or velocities, apply the control law, and then output the desired motor voltage or current. Thus, this layer requires a reliable operating environment, preferably with real-time performance. For this reason, it is often implemented on special-purpose hardware, such as an off-the-shelf (commercial) controller board, or on a PC using a framework that supports real-time processing.

The LLC interface is an obvious candidate for standardization because most robots contain similar low-level controllers. In particular, a standard low-level control inter-

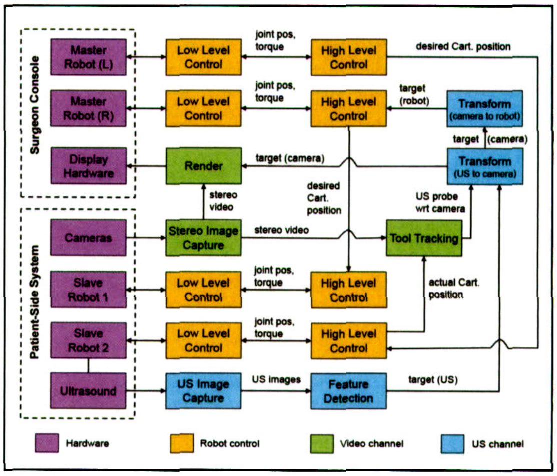

FIGURE 3 **Illustrative system architecture with telesurgical robot (two masters and two slaves), stereo video cameras and display hardware, and ultrasound (US) scanner. US probe is held by Slave Robot 2 and Feature Detection module looks for target. Tool Tracking module estimates position of US probe from stereo camera images, using Slave Robot 2 position to initialize image localization. Result of Tool Tracking is used to transform detected target to camera coordinates, which enables augmented reality overlay within Render module. Target is also transformed to Right Master Robot coordinates (using previously measured transformation) and provides haptic guidance to surgeon to position instrument held by Slave Robot 1 at target.**

face would include commands to enable/disable motor power, home (initialize) the robot, get the current joint positions, and move the joints to a specified position. The other advantage to standardizing at this interface is that it typically forms the bridge between the hard real-time and soft real-time parts of the system. This enables plug-and-play interoperability between systems with very different low-level control and physical layers.

High-Level Control (HLC) Layer

While the LLC provides joint-level control, the HLC provides more sophisticated motion capabilities. One example is Cartesian-level control, where the pose (position and orientation) of the robot end-effector can be measured and controlled in Cartesian coordinates. This requires knowledge of the robot's kinematics. This layer often also integrates feedback from other sensors external to the robot system, such as vision and force, that can be used for visual servoing or force control, respectively. Many surgical robot systems require a human (surgeon) in the loop, and so the high-level control may include assistive control behaviors, such as virtual fixtures.

For the HLC interface, it is straightforward to standardize some basic capabilities, such as Cartesian position control, and to allow system-specific extensions for other capabilities such as sensor-based control modes. This is also the level where ROS interfaces are most common, since the major advantage offered by ROS is the ability to integrate with other high-level software modules.

Data Channel Layer

Data channels are common in surgical robot systems due to the integration with real-time imaging such as video and ultrasound. A data channel consists of a source (e.g., a camera), several filters that process the data, and one or more sinks (e.g., a rendering device). There are two common implementation strategies: (1) a pipeline, where a separate thread or thread pool is used for each filter, and (2) a stream, where a single thread or thread pool is used to sequentially execute each filter. The advantage of the pipeline is that it can provide higher throughput, since processing of new data can begin immediately. The advantage of the stream is that it provides lower latency because there is no need for synchronization primitives between the execution of different filters.

The choice of a pipeline or stream depends on the application requirements and leads to the choice of implementation framework. For human-in-the-loop systems, which are common in surgery, minimizing latency (delay) may be critical, since added delay can affect surgical performance. For this reason, the *cisstStereoVision* (SVL) library (part of the cisst package) supports the stream processing paradigm. In SVL, each filter is a separate component, but the components can exist in a single executable and share memory buffers to reduce overhead. Synchronization primitives are not required because SVL sequentially executes each filter.

If low latency is not required, the pipeline is an attractive option because it enables the use of ROS nodes as filters (ROS provides a large collection of useful image processing components). In ROS, each node is a separate executable, so by default it contains its own thread (or thread pool) and a network of these nodes forms a pipeline.

Application Layer

The application layer primarily consists of the application logic (e.g., surgical workflow) and the user interface. There are many different packages that can be used to implement the application layer. If the application is primarily a graphical user interface, one could adopt a framework such as Qt (www.qt.io). Alternatively, if the application requires the display and manipulation of preoperative and/or intraoperative medical images, the application layer could be implemented in an extensible, open source framework such as 3D Slicer (www.slicer.org). In this case, it would be convenient to use Slicer's built-in OpenIGTLink interfaces. The rviz package provided by ROS is also an attractive option. It is based on the OGRE graphics rendering engine and has plugins to support Qt widgets, images, and other data types. Finally, some researchers choose Matlab/Simulink (The MathWorks, Inc., Natick, MA) as their development platform.

CONCLUSIONS

This article presented an overview of surgical robot systems, with the recognition that these systems are not just robots, but integrated systems that include robots, databases, and real-time sensors such as video and other medical imaging devices. Common research platforms, such as the da Vinci Research Kit (dVRK) and Raven II, have recently become available. This has underscored the need for modular software interoperability, so that researchers can share software modules and more easily integrate other robots and devices. Standardization and interoperability are most applicable at the higher software layers, and can benefit from the availability of widely-adopted middleware such as ROS. Other interface protocols, such as OpenIGTLink, can be useful due to their wide support within the medical imaging and image-guided intervention domains. ■

REFERENCES

1 Kwoh, Y., Hou, J., Jonckheere, E., and Hayati, S., 1988. "A robot with improved absolute positioning accuracy for CT guided stereotactic brain surgery". *IEEE Trans. on Biomedical Engineering*, 35(2), Feb, pp. 153–160.

2 Ng, W., Davies, B., Hibberd, R., and Timoney, A., 1993. "Robotic surgery–a first-hand experience in transurethral resection of the prostate". *IEEE Engineering in Medicine and Biology Magazine*, 12(1), Mar, pp. 120–125.

3 Mittelstadt, B., Paul, H., Kazanzides, P., Zuhars, J., Williamson, B., Pettitt, R., Cain, P., Kloth, D., Rose, L., and Musits, B., 1993. "Development of a surgical robot for cementless total hip replacement". *Robotica*, 11, pp. 553–560.

4 Kazanzides, P., Chen, Z., Deguet, A., Fischer, G., Taylor, R., and DiMaio, S., 2014. "An Open-Source Research Kit for the da Vinci ® Surgical System". In IEEE Intl. Conf. on Robotics and *Automation (ICRA)*.

5 Hannaford, B., Rosen, J., Friedman, D. W., King, H., Roan, P., Cheng, L., Glozman, D., Ma, J., Kosari, S. N., and White, L., 2013. "Raven-II: an open platform for surgical robotics research". *IEEE Trans. on Biomedical Engineering*, 60(4), pp. 954–959.

6 Bruyninckx, H., Soetens, P., and Koninckx, B., 2003. "The real-time motion control core of the Orocos project". In IEEE Intl. Conf. on Robotics and Automation (ICRA), Vol. 2, pp. 2766–2771.

7 Kapoor, A., Deguet, A., and Kazanzides, P., 2006. "Software components and frameworks for medical robot control". In IEEE Intl. Conf. on Robotics and Automation (ICRA), pp. 3813–3818.

8 Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T. B., Leibs, J., Wheeler, R., and Ng, A. Y., 2009. "ROS: an open-source robot operating system". In ICRA Workshop on Open Source Software.

9 Tokuda, J., Fischer, G. S., Papademetris, X., Yaniv, Z., Ibanez, L., Cheng, P., Liu, H., Blevins, J., Arata, J., Golby, A. J., Kapur, T., Pieper, S., Burdette, E. C., Fichtinger, G., Tempany, C. M., and Hata, N., *2009. "OpenIGTLink: an open network protocol for image-guided therapy environment".* The International Journal of Medical Robotics and Computer Assisted Surgery, 5(4), pp. 423–434.

10 Kazanzides, P., and Thienphrapa, P., 2008. "Centralized processing and distributed I/O for robot control". In Technologies for Practical Robot Applications (TePRA), pp. 84–88.

11 Qian, L., Chen, Z., and Kazanzides, P., 2015. "An Ethernet to FireWire bridge for real-time control of the da Vinci Research Kit (dVRK)". In IEEE Intl. Conf. on Emerging Technologies and Factory Automation (ETFA).