Published in final edited form as:
Int J Med Robot. 2009 December ; 5(4): 423–434. doi:10.1002/rcs.274.

OpenIGTLink: an open network protocol for image-guided therapy environment

Junichi Tokuda[1,*], **Gregory S. Fischer**[2], **Xenophon Papademetris**[3], **Ziv Yaniv**[4], **Luis Ibanez**[5], **Patrick Cheng**[4], **Haiying Liu**[1], **Jack Blevins**[6], **Jumpei Arata**[7], **Alexandra J. Golby**[1,8], **Tina Kapur**[1], **Steve Pieper**[9], **Everette C. Burdette**[6], **Gabor Fichtinger**[10], **Clare M. Tempany**[1], and **Nobuhiko Hata**[1]

[1]Department of Radiology, Brigham and Women's Hospital and Harvard Medical School, 75 Francis Street, Boston, MA 02115, USA

[2]Department of Mechanical Engineering, Worcester Polytechnic Institute, 100 Institute Road, HL 130, Worcester, MA 01609, USA

[3]Diagnostic Radiology and Biomedical Engineering, Yale University School of Medicine, 300 Cedar Street, TAC N119, New Haven, CT 06511, USA

[4]Department of Radiology, Georgetown University Medical Centre, 2115 Wisconsin Avenue, Washington DC, USA

[5]Kitware Inc., 28 Corporate Drive, Clifton Park, NY 12065, USA

[6]Acoustic MedSystems, Inc., Suite 301, 206 N.Randolph St., Champaign, IL 61820, USA

[7]Department of Computer Science and Engineering, Nagoya Institute of Technology, Gokiso-cho, Showa-ku, Nagoya, Aichi 466-8555, Japan

[8]Department of Neurosurgery, Brigham and Women's Hospital and Harvard Medical School, 75 Francis Street, Boston, MA 02115, USA

[9]Isomics Inc., 55 Kirkland Street, Cambridge, MA 02138, USA

[10]School of Computing, Queen's University, 725 Goodwin Hall, 25 Union Street, Kingston, ON K7L 3N6, Canada

Abstract

Background—With increasing research on system integration for image-guided therapy (IGT), there has been a strong demand for standardized communication among devices and software to share data such as target positions, images and device status.

Method—We propose a new, open, simple and extensible network communication protocol for IGT, named OpenIGTLink, to transfer transform, image and status messages. We conducted performance tests and use-case evaluations in five clinical and engineering scenarios.

Results—The protocol was able to transfer position data with submillisecond latency up to 1024 fps and images with latency of <10 ms at 32 fps. The use-case tests demonstrated that the protocol is feasible for integrating devices and software.

*Correspondence to: Junichi Tokuda, Surgical Planning Laboratory, Department of Radiology, Brigham and Women's Hospital, 75 Francis Street, ASB-I, Room L1-050, Boston, MA 02115, USA. tokuda@bwh.harvard.edu.

Conclusion—The protocol proved capable of handling data required in the IGT setting with sufficient time resolution and latency. The protocol not only improves the interoperability of devices and software but also promotes transitions of research prototypes to clinical applications..

Keywords

image-guided therapy; surgical navigation; network communication protocol

Introduction

Standardization of communication among devices and software in the operating room (OR) environment is a common issue in image-guided therapy (IGT) (1). Today it is quite common to locate surgical tools relative to the patient's body by using position and orientation tracking systems with optical (2), electromagnetic (3) or ultrasonic (4,5) sensors, or to acquire images using real-time ultrasound, computed tomography or magnetic resonance imaging (MRI). This localization and image information is transferred from acquisition devices to navigation software for visualization and guidance. In addition, with increasing research on robotic devices that support image-guided interventions, there has been a strong demand for communication standards among these devices and navigation software to allow sharing of information such as target positions, images and device status.

There have been sporadic efforts to standardize the interconnections between medical devices and computers. The ISO 11 073/IEEE 1073 Standard for Medical Device Communication (6) defines transportation profile (IEEE 1073.3.1) and physical layer (IEEE 1073.4.1) to allow plug-and-play operation for bedside device communication. CANOpen (EN 50 325-4) (7), which is an application layer for Controller-Area Network (CAN) (ISO 118 988) (8), has also been used. In addition to those standards in physical and transportation layers of device connections, Massachusetts General Hospital and the CIMIT are leading an initiative called Medical Device Plug-and-Play (MD PnP) to facilitate interoperability among a number of medical devices in the OR (9).

With the increasing number of IGT applications and the availability of Ethernet in the IGT environment, standardization of information and communication technology is more important than ever (1). In particular, Ethernet is becoming a frequent choice for communication between devices and computers in the clinical research setting because of its availability, flexibility and bandwidth. Most modern personal computers have Ethernet interfaces and run operating systems supporting the TCP/IP model, which is the foundation of most network applications. DICOM (10) is a well-known standard for the transfer of image data through TCP/IP networks, as well as archiving the image data in storage or a database, and has been playing an important role in IGT. Recently, the DICOM Working Group 24 has been compiling surgical workflow models to determine the standard for integrating information about patient equipment and procedure (11). Despite the availability of DICOM in hospital networks, it has not been practical in IGT applications involving real-time imaging applications, e.g. ultrasound-guided interventions. This is due in part to the fact that DICOM contains large amounts of redundant information that make data packets unfeasibly large for sequential image transfer at sub-second frame rates, and most DICOM implementations are not tuned for real-time applications. Furthermore, there is no standard protocol for transferring synchronized image and tracking data through a single connection.

To address this gap in the standards, IGT system vendors have created proprietary research interfaces for their own products. For example, BrainLAB's interface, VectorVision Link (VVLink) (12–14), allows for output of multiple images, tools, labelled points and streaming bitmaps into the host system. Medtronic's StealthStation product and Intuitive Surgical's

DaVinci® robot contain network interfaces similar to those used in research projects (15,16). A perennial problem of proprietary interfaces is that those protocols or libraries are designed only for specific hardware/software, forcing academic researchers to negotiate access privileges and then customize their software to particular imaging and tracking devices. This prevents modularity and flexibility, and replacing a piece of the system requires massive overhaul of the communication software and often renegotiating access rights to proprietary interfaces. Making matters worse, vendor-provided protocols often cause issues of license incompatibility between the commercial and research software. Thus, several groups have experimented with standardized network communication between IGT devices and software. Schorr *et al.* proposed an application of the common object request broker architecture (CORBA) framework to IGT (17). Defined by the Object Management Group consortium, CORBA is a remote procedure call standard to achieve independence of programming languages and operating systems (OSs). CORBA lends itself to IGT, where multiple devices and software exchange various types of data over local networks. Von Spiczak *et al.* proposed middle-ware for device connectivity, based on the OpenTracker library (18,19). Due to its modular design, this library allows users to add a module for a specific device, after which they can connect the device to any software to transfer coordinate and/or image data. The connection structure can be configured simply by editing a configuration file described in extended markup language (XML), and modules are available for multiple commercial tracking systems and imaging scanners. While both CORBA and OpenTracker were extremely important initiatives, they failed to achieve the status of *de facto* standard in the IGT field, primarily because of overgeneralization, overabstraction and limited portability. Complicated specifications and libraries tend to force developers to spend considerable effort on understanding communication mechanisms that are not essential for their operation. No less important, they are decidedly inconvenient in multiplatform development, which is inherent in IGT. A wide variety of operating systems and CPU architectures, from embedded systems to high-performance computers, are used in IGT, and it is impractical to support all of them with a single library, due to differences in their application programming interfaces (APIs) and processing capabilities. As a consequence, we must deal with an ever-growing maze of vendor-specific or application-specific communication libraries and protocols, resulting in a complete lack of interoperability.

In this paper, we propose an extensible yet simple and easy-to-implement network protocol that allows for exchanging tracking data, images and device control/monitoring information among tracking devices, imaging scanners and systems software. We also describe several use-case scenarios, including an ultrasound navigation system, integration of tracking devices and navigation software, integration of research software and a commercial navigation system, an MRI-compatible robot system for prostate intervention, and a neurosurgical manipulator system.

Methods

OpenIGTLink Protocol

We defined an open, simple and extensible peer-to-peer network protocol for IGT called OpenIGTLink. This protocol emerged through a collaboration of academic, clinical and industrial partners in developing an integrated robotic system for MRI-guided prostate interventions (20). The OpenIGTLink protocol was designed for use in the Application Layer on the TCP/IP stack, while allowing developers to implement it for other network models, such as the User Datagram Protocol.

The OpenIGTLink protocol itself does not include mechanisms to establish and manage a session. A message, the minimum data unit of this protocol, contains all information necessary for interpretation by the receiver. The message begins with a 58-byte header section, which is

common to all types of data, followed by a body section. The format of the body section varies by data type, specified in the header section. Since any compatible receiver can interpret the header section, which contains the size and data type of the body, every receiver can gracefully handle any message, even those with unknown data type. Therefore, this two-section structure allows developers to define their own data type while maintaining compatibility with other software that cannot interpret their user-defined data types. This simple message mechanism eases the development of OpenIGTLink interfaces and improves compatibility. Details of standard data types are described in the following sections. Further information is available in the web page provided by the National Alliance for Medical Image Computing (21).

General header

The header contains generic information about the message, including the type of the data, a field called 'Device Name', a time stamp and the size of the data section. The 'Device Name' identifies the source of the data and is useful in sending multi-channel data through a single connection. For example, optical or electromagnetic tracking devices may track multiple objects and report the position of each object in a separate message. All numerical values (integers and floating and fixed points) are stored in big-endian (or network) byte order:

- *Version number (2 bytes)*: The version number of the OpenIGTLink protocol used (all data structures described in this paper were defined as Version 1).
- *Data type name (12 bytes)*: The OpenIGTLink protocol defines five default types, which are frequently used in most IGT use-case scenarios: 'IMAGE', 'POSITION', 'TRANSFORM', 'STATUS' and 'CAPABILITY', the details of which are described in the next section. The application developers may define their own data types with names in addition to the default types, as demonstrated in the 'Use-cases' section. It is recommended to use names starting with '*' to differentiate the name space for user-defined types from the default name space. In the OpenIGTLink protocol, usually the sender pushes data to the receiver, but the sender can also request the receiver to send data back. These requests are issued as messages with special data types with null-body section: 'GET_IMAGE', 'GET_POSITION', 'GET_TRANS', 'GET_STATUS', and 'GET_CAPABIL'. Developers can also define an application-specific format for their data type and associate it with a type name specified here in the message header. This practice, however, is not encouraged because it acts against portability.
- *Device name (20 bytes)*: Name of the source device.
- *Time stamp (8 bytes)*: This field informs the receiver of the time when the data were generated. The data are represented in seconds as a 64-bit fixed point number relative to 00 : 00 : 00 1 January 1970 UTC. The integer part is in the first 32 bits (Unix-style time-stamp) and the fraction part in the last 32 bits. In the fraction part, the non-significant low order can be set to 0 (developers may opt out of using this field by filling it with 0).
- *Body size (8 bytes)*: The size of the message body attached to this header in 64-bit unsigned integer.
- *CRC (8 bytes)*: The 64-bit cycle redundancy check for the body section. This is used to verify the integrity of data and detect system faults. This feature may be helpful when the protocol is integrated in devices requiring Food and Drug administration (FDA) or other regulatory approval for clinical use. Note that the CRC does not include the header section, relying on the CRC in the lower layers of the network protocol stack, e.g. TCP/IP.

Data body

The body structure varies by the data type being sent in the message as follows. All numerical values are stored in big-endian (network) byte order except image pixel values. Floating-point values are encoded in IEEE 754.

IMAGE—The IMAGE format in the OpenIGTLink protocol supports 2D or 3D images with metric information, including image matrix size, voxel size, coordinate system type, position and orientation. The body section of the IMAGE data consists of two parts: image header, to transfer the metric information; and image body, to transfer the array of pixel or voxel values. The data type of pixel or voxel can be either scalar or vector, and numerical values can be 8-, 16- or 32-bit integers or 32- or 64-bit floating points. The pixel values can be either big-endian or little-endian, since the sender software can specify the byte order in the image header. The format also supports 'partial image transfer', in which a region of the image is transferred instead of the whole image. This mechanism is suitable for real-time applications, in which images are updated region by region. The subvolume must be box-shaped and defined by six parameters, consisting of the indices for the corner voxel of the subvolume and the matrix size of the subvolume (Figure 1). The indices begin at 0, similar to the conventional zero-based array indices in the C/C++ programming languages.

The position, orientation and pixel size are represented in the top three rows of a 4 × 4 matrix defined by:

$$R=\begin{pmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{32} & p_y \\ r_{311} & r_{32} & r_{33} & p_y \\ 0 & 0 & 0 & 1 \end{pmatrix}=\begin{pmatrix} \mathbf{t} & \mathbf{s} & \mathbf{n} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (1)$$

where the upper left 3 × 3 matrix represents rotation and the upper right 3 × 1 column vector represents transformation (centre position of the image). The matrix is equivalent to the conjunction of normal column vectors for directions of voxel arrays in *i*, *j* and *k* indices, which are denoted as **t**, **s**, and **n** in equation (1). The position and orientation vectors can be described in either a left posterior superior (LPS) or right anterior superior (RAS) coordinate system, depending on the coordinate system specified, also in this data format. The LPS is a right-handed coordinate system used in the DICOM standard, while the RAS is more common in neuroimaging research and is used in open-source surgical navigation software, such as the 3D Slicer (22,23).

POSITION—The POSITION data type is used to transfer position and orientation information. The data are a combination of three-dimensional (3D) vector for the position and quaternion for the orientation. Although equivalent position and orientation can be described with the TRANSFORM data type, the POSITION data type has the advantage of smaller data size (19%). It is therefore more suitable for pushing high frame-rate data from tracking devices.

TRANSFORM—The TRANSFORM data type is used to transfer a homogeneous linear transformation in 4 × 4 matrix form. One such matrix is shown in equation (1). Note that if a device is sending only translation and rotation, then TRANSFORM is equivalent to POSITION. But TRANSFORM can also be used to transfer affine transformations or simple scaling. Like IMAGE and POSITION, TRANSFORM carries information about the coordinate system used.

STATUS—The STATUS data type is used to notify the receiver about the current status of the sender. The data consist of status code in a 16-bit unsigned integer, subcode in a 64-bit

integer, error name in a 20 byte-length character string, and a status message. The length of the status message is determined by the size information in the general header. The status code is defined as a part of the OpenIGTLink protocol specification listed in Table 1. The subcode is device-specific and is defined by developers. In addition, developers can build their own error name/code into the status message and additional optional description in the following data field.

CAPABILITY—The CAPABILITY data type lists the names of message types that the receiver can interpret. Although the OpenIGTLink protocol guarantees that any receiver can at least skip messages with unknown type and continue to interpret the following messages, it is a good idea to get the capability information at system start-up to ensure application-level compatibility of the various devices. In a CAPABILITY message type, each message type name comes with a format version number. If the receiver can interpret multiple versions for a certain message type, they should be listed as independent types.

User-defined data types

The OpenIGTLink protcol allows developers to define their own message types. As long as the general header has correct information about the size of the data body, it retains compatibility with any software compliant with the OpenIGTLink protocol because the receiver can skip data that it cannot interpret. The OpenIGTLink protocol specification recommends that the developer adds an asterisk (*) at the beginning of the type name in the general header, to avoid future conflicts with standard types and also to make it easier to distinguish between standard and user-defined message types.

The OpenIGTLink library

To provide a reference implementation of OpenIGTLink interface, we developed the OpenIGTLink Library (21). The library is a free open-source software (FOSS) distributed under a BSD-style open-source license, placing no restrictions on use. The library consists of three components, as seen in Figure 2:

1. A C-based library defining structures and utility functions to serialize data into OpenIGTLink Message. This library is useful in developing embedded systems or software for platforms where a modern C++ compiler is not available.
2. A set of high-level C++ classes wrapping the C-based library that provide a safer and more convenient way to implement OpenIGTLink messaging function into software. Developers can define their own message types by inheriting the base message class defined in the library.
3. A set of multi-platform C++ classes to handle sockets and threads. The multi-platform socket and thread classes are currently compatible with 32-bit Windows, Linux/UNIX and Mac OS X platforms.

Performance Evaluations

Experimental methods

We conducted experiments to evaluate the performance of the OpenIGTLink protocol and the library implementation. The objective of the study was to determine the capability of the protocol in terms of frame rate and latency of data transfer, to anticipate its possible applications. In the experiments, two Linux-based workstations were used:

- *Host 1*, Dell Precision 470; CPU, Intel Xeon (dual core/64 bit) 2.8 GHz; memory, 4 GB; OS, Fedora Core 6 (Dell Inc., Round Rock, TX, USA).

- *Host 2*, SunJava Worksation W2100z; CPU, Dual AMD Opteron 246; memory, 2 GB; OS, Fedora Core 2 (Sun Microsystems Inc, CA, USA).

Both Linux kernels were compiled with timer clock resolution of 1 KHz. The hosts were connected to a gigabit switch (Linksys SD2008, Cisco Systems Inc, Irvine, CA, USA) with category 5e cables. To measure the latency of data transfers between the two hosts, the system clocks of those hosts were synchronized using PTPd (24), the software-only implementation of the precision time protocol (PTP). The PTP is a time-transfer protocol defined in IEEE Standard 1588 – 2002 for precise time synchronization over networks (25) and is used as a key technology of LAN eXtensions for Instrumentation (LXI), which is a standard for connecting test and measurement instruments. Compared with other time protocols, such as network time protocol (NTP), PTP is more specialized for local systems that require high-accuracy time synchronization. According to the literature (24), the synchronization accuracy of PTPd under ideal conditions is on the order of 10 μs, which is sufficient for our evaluation. In the PTPd protocol, the most stable and accurate clock is selected as a master clock among the connected devices, based on the best master clock algorithm (BMCA) and used as the reference time. Clock servoing is performed to adjust the tick rate of the other slave clocks, so that the master-to-slave delay is minimized without resetting the clocks. During our experiments, the clock synchronization was performed every 500 ms as a background process, and clock difference of the hosts was monitored. It was reported that the PTPd's CPU utilization is below 1% on a 66 MHz m68k processor (24); thus, resource consumption by clock synchronization was negligible. In all experiments we examined data transfer from host 1 to host 2 and vice versa, to account for differences in performance between the hosts. We evaluated the performance of data transfer using the OpenIGTLink library under typical conditions. Three sets of experiments were performed:

Tracking data transfer—First we evaluated the latency and CPU load during tracking data transfer, varying the frame rate. The latency was defined as the time between the start of generating dummy tracking data at the sender host and the end of deserialization of OpenIGTLink message at the receiver host. The time point to start serialization of the message was implemented in the time-stamp field of the OpenIGTLink messages at the sender host and transferred to the receiver host, where the time-stamp was compared with the time point to finish deserialization. The frame rate of the tracking data was 2^n frames/s (fps), where n varied in the range 1 – 10, based on the fact that the frame rate of tracking devices is in the range 40 – 375 Hz and that of the sensor feedback of real-time robot controlling is usually on the order of KHz. The number of channels varied in the range 1 – 16.

Image transfer—Second, we evaluated the latency and CPU load in image data transfer, varying the data size of images. The image sizes were defined by 2^n bytes, where n was varied in the range 12 – 20 with a single channel. The frame rate was fixed at 32 fps (2^5 fps), which was near the typical frame rate of real-time ultrasound imaging and standard video frame rates. In this experiment, a series of dummy images were generated prior to the experiment and stored in a memory pool. The latency was defined as the time between the start of copying the dummy image into the OpenIGTLink message from the memory pool and the end of deserialization of the message at the receiver host.

Simultaneous tracking and image data transfer—The last experiment was to evaluate simultaneous transfer of tracking and image data. The objective was to demonstrate how image data transfer interferes with tracking data transfer. It is critical to keep latency of the tracking data transfer below a certain threshold in the clinical situation, where different types of data are transferred simultaneously through a single connection. In this experiment, each image was split into multiple messages and transferred part by part. The frame rate and the size of frames

were varied by 2^n fps and $4096/2^n$ KB, respectively, so that the bandwidth was fixed at 4096 KB/s. The frame rate of tracking data transfer was fixed at 100 fps.

All time measurements were performed using the gettimeofday() function defined in 4.3 BSD UNIX and a part of Standard POSIX 1003.1 – 2001. Although the function has a resolution of 1 μs, the precision of time measurement depended on the accuracy of clock synchronization between the sender and receiver hosts.

Results

All measurement started after the synchronization error was converged (Figure 3). The mean and standard deviation (SD) of time synchronization error between two hosts was 12.3 ± 11.8 μs throughout the experiments. Table 2 and Figure 4A show the latency and CPU loads, respectively, during tracking data transfer. For tracking data, the latencies were evaluated from 500 samples for each condition. To obtain the CPU loads, the user and system times are measured on both the sender and the receiver while performing the data transfer for 1000 s for each condition. The level of CPU load strongly depends on the number of channels and the role of the host, as shown in Figure 4A. The result of image data transfer is shown in Table 3 and Figure 4B.

For imaging data, the latencies were evaluated based on 100 samples for each condition. In addition, the latencies of the simultaneous data transfer are shown in Table 4 and Table 5. The latency of the image data was approximately proportional to the size of the message.

Use-cases

To demonstrate the extensibility and feasibility of the proposed method in integration of an image-guided therapy system, we have evaluated the OpenIGTLink protocol in several clinical and engineering use-case scenarios. The hypothesis here is that the open and simple protocol allows the developers to perform multivendor and multiplatform integration of image-guided therapy systems. We describe integration of: (a) an ultrasound navigation system; (b) tracking devices and navigation software; (c) a research software and commercial navigation system; and (d) an MRI-compatible robot system for prostate intervention.

Ultrasound navigation system

We utilized the OpenIGTLink protocol to incorporate intraoperative ultrasound (US) imaging into an existing CT-based navigation system for needle biopsies (26). The challenge for this application was integration of the imaging device and navigation software, which run on different computer platforms. This multiplatform integration was required by constraints of this particular application; the US imaging device must have a small physical footprint to keep the navigation system's physical obtrusiveness to a minimum in the cramped clinical environment. Thus, we chose the Terason T2000 portable US system (Teratech Corp., Burlington, MA, USA), in which APIs are available only for the Microsoft Windows platform; our navigation software, which was developed using Image-Guided Surgery Toolkit (IGSTK) (27), runs on a Linux-based system. Instead of incorporating the US image acquisition into our navigation software, we extended it using a client–server architecture, where the navigation software (server) receives US images from the client that runs on a Windows-based system. To minimize the lag between image acquisition and display, the two computers were connected via a dedicated local network. Figure 5 shows the hardware components of the extended navigation system. The architecture that decouples US acquisition from the main navigation system allows switching US systems to another system without modifying the navigation system.

Tracking devices and navigation software integration

To utilize various types of tracking devices, including encoded mechanical arms, optical tracking systems and electromagnetic tracking (28) from surgical navigation software in image-guided therapy, we implemented a client–server architecture for tracking devices and navigation software integration, using free open-source image-processing and visualization software, 3D Slicer (22) and IGSTK, which provides a high-level interface to commonly used optical and electromagnetic tracking devices. The rationale underlying this integration is that the OpenIGTLink protocol enabled us to develop a navigation system that does not require modification of software to support various tracking devices. Since there is no common API that can be used to communicate with tracking devices, it has been the application developer's responsibility to establish communication between navigation software and tracking devices, which is a tedious task. In our architecture, 3D Slicer works as a server to receive tracking data from an IGSTK-based client program, which acquires-measurement values from a tracking device through a network using the OpenIGTLink protocol. The server and client can run on the same computer as separate processes or on different computer systems connected via a network. Therefore the OpenIGTLink protocol provides a method to decouple the device-dependent software from the navigation software. This decoupling approach enables the navigation system to support tracking devices that will be added to the toolkit in the future without requiring any modification of its original implementation.

Research software integrated with commercial navigation system

We connected 3D Slicer to a commercial neurosurgical navigation system (VectorVision Cranial Navigation, BrainLAB AG, Feldkirchen, Germany), using the OpenIGTLink protocol to transfer image and tracking data from the commercial system to 3D Slicer during a clinical case. The idea behind this work is to take advantage of advanced image processing and visualization, which are not commercially available, from external research navigation software (e.g. 3D Slicer) and reliable surgical navigation features from an approved commercial navigation system (e.g. BrainLAB), in order to investigate new technologies without interfering with the existing clinical procedure. A key challenge in this work is that the BrainLAB system does not contain an OpenIGTLink layer. Instead it provides access using its internal VectorVision Link (VVLink) interface (29). In order to establish communication between the two systems, we developed bridge software that receives images and tracking data from the BrainLAB system using VVLink, converts them into OpenIGTLink messages, and sends them to 3D Slicer over the network. The bridge/proxy module is implemented within Yale University's BioImage Suite image analysis software suit (30), which already incorporates a VVLink interface. The integrated system has started to be tested in clinical cases to investigate new visualization techniques, which is not available in commercial systems (Figure 6). Despite the need for a 'double-hop' network connection, we have been able to maintain real-time tool tracking performance across the combined systems, demonstrating the feasibility of using a proxy system to translate from the proprietary protocol to OpenIGTLink, hence allowing for the interfacing of two unmodified systems (3D Slicer, BrainLAB VVCranial). This approach should be generalized to other commercial navigation systems, where a proxy server could be implemented to translate its internal research interface to OpenIGTLink, to simplify the task on the research end enormously.

MRI-compatible robot system for prostate intervention

We have integrated an MRI-compatible needle placement manipulator (20) with navigation software and an MRI scanner using the OpenIGTLink protocol. The goal of this work is to provide a 'closed-loop' therapy, where the robot's action is immediately captured in semi-real-time MRI, and instantaneous feedback is provided to a physician who decides about the next action. The software system consists of three major components: (a) a control unit for the needle

placement robot; (b) a closed-bore whole-body 3T MRI scanner (GE Excite HD 3T, GE Healthcare, Chalfont St. Giles, UK); and (c) commercial navigation software for prostate intervention (RadVision, Acoustic MedSystems, Champaign, IL, USA) or free open-source software, 3D Slicer, as a user interface for the entire system (Figure 7). Both navigation software packages display preoperative 3D images for planning and intraoperative semi-real-time MR images for guiding the procedure, so that the physician can interactively specify the target points on the preoperative image set. We used 3D Slicer for prototyping the system, while we developed RadVision for future commercialization. The advantage of RadVision is that the software is integrated with intraoperative dosimetry calculation, which have been used in clinical cases of MRI-guided prostate brachytherapy. The current position of the needle is indicated on the 3D view of 3D Slicer. All components were connected to one another via 100Base-TEthernet.TheOpenIGTLink protocol is used to exchange various types of data, including commands to the robot and scanner, semi-real-time images and positional data. The positions of the target lesion are specified on the navigation software and transferred to the robot control unit. While the robot control unit is driving the needle towards the target, the needle position is calculated from the optical encoders and sent back to the navigation software every 100 ms. The navigation software calculates the imaging plane that intersects the needle's axis and transfers it to the scanner, which in turn acquires semi-real-time images in that plane.

Neurosurgical manipulator system

We developed an open software platform for a neurosurgical manipulator system using OpenIGTLink protocol (Figure 8) (31). The manipulator system has a master–slave configuration, under which the slave manipulator follows the motion of the master manipulator, which is operated by the surgeon. The slave's end-effector position is measured in two ways for safety: by encoders implemented in the slave manipulator, and by an optical tracking device (Optotrak, Northern Digital Inc., Canada). Therefore, the system consists of four software components: (a) a master manipulator controller; (b) a slave manipulator controller; (c) an optical tracking interface; and (d) navigation software (3D Slicer). We used the OpenIGTLink protocol to transfer the position data from (a) to (b) to notify the slave of the current position of the master manipulator, and from (c) to (d) to visualize the 3D model of the surgical tool overlaid with a 3D brain model created from MR images. Our initial trial demonstrated that the tracking data were successfully obtained every 100 ms from the optical tracking, and the 3D model of the surgical tool was displayed at the current position of the virtual space overlaid with the brain model, on the 3D Slicer. The tool model moves as the user operates the master manipulator, allowing interactive operation of the slave manipulator using the 3D Slicer as a console.

Discussion

We have designed an open, simple, and extensible network communication protocol that can be used to transfer data required in the IGT setting, e.g. tracking and image data, device control/monitoring information, among the devices and software. The simple protocol allows developers to implement interfaces easily for their software and hardware, while allowing definition of application-specific data types. We have also developed a multi-platform open library as a reference implementation of the protocol, with the associated interface for the 3D Slicer, thereby aiding developers in the implementation and testing of OpenIGTLink with navigation software.

Our study showed that the protocol is capable of handling 1 KHz multi-channel tracking data, which is generally required for sensor feedback for real-time robot control. In realistic conditions, however, the latency may be prolonged by other processes, such as navigation software running on the same host. At the same time, the stability of interval and latency of

data transfer are also affected by other network activity and an OS's timer resolution, which is typically 10 ms and can be configured up to 1 ms in recent Linux and Microsoft Windows OSs (32). Therefore, it seems prudent to use real-time communication techniques and real-time operating systems in a high-performance critical clinical application that requires accurate interval and latency for position data feedback. This is desirable, for example, in controlling robots compensating for organ motion (33) or delivering conformal radiotherapy to mobile targets. In some real-time applications, it is also important to monitor the time-stamp in tracking data messages. Since time-stamp is usually checked by a computer different from the one that issued the time-stamp, clock synchronization between those two computers is crucial. In this study PTP was used for this purpose, where the synchronization error was kept within 12.3 ± 11.8 μs throughout the experiment. Although the time for convergence in the experiment (10 – 20 min) was too long for the clinical setting, the method is still feasible in most clinical applications that require clock synchronization with errors on the order of milliseconds, which can be achieved within 10 s from the start of synchronization.

We found that the latency of the tracking data transfer was in the order of sub-milliseconds, with frame rate of <1024 fps. The CPU load depended on the number of channels and frame rates, and the 16-channel tracking data transfer with frame rate of 1024 consumed less than 30% of the CPU load. Such performance is adequate for most IGT applications, for which the frame rate of optical tracking data is typically <100 fps. In the imaging data transfer, larger latencies were recorded, due to the larger size of the data (1 MB/message) compared with the tracking data (106 bytes/message). In fact, the latency of image data transfer increased with the size of the image data. The latency of image data transfer becomes more critical in transferring multiple types of data through a single connection simultaneously, because image transfer may block transfers of other types of data. Our study of simultaneous data transfer suggests that large bodies of data should be split into multiple messages, so as not to block other data that need to be transferred more quickly. Our experiments also revealed that in most cases receiving a message consumes more CPU time than sending it. This is partly because the OpenIGTLink message contains character strings to specify data type and device name, which requires the receiver to perform string comparison, a generally CPU-intensive function. This is the price we pay for having eliminated 'sessions', whereas every message self-contains the information necessary for parsing it.

Our experience with several clinical and engineering use-case scenarios revealed that a standardized communication protocol has other important roles besides providing improved interoperability for IGT. First, such protocols enable smooth transitions from prototyping to product phases. In contemporary IGT systems, where numerous hardware and software components are integrated, each component is developed independently. With a standardized communication protocol, developers can easily replace each of components from one developmental phase of prototype to another. The proposed protocol was initially designed through discussion of the development of an MRI-compatible robot for prostate interventions under a research partnership among academic and industrial sites (20), where the plan is to utilize academic research to transition to a commercial product. Second, as we investigated in this paper, standardizing this protocol is critical for linking clinical cases and scientific research. There has been the issue of sharing clinical data between proprietary software for clinical routines and research software for scientific data analysis. In terms of patient safety, proprietary software approved by the FDA or other regulatory agencies is preferable, but this usually limits access to image and other types of data from clinical cases to research purposes. Adding an open-protocol interface to the proprietary software allows researchers to access clinical data from their research software, promoting clinical research. It is our hope that research groups that have substantial experience with one particular commercial system that does not implement OpenIGTLink directly (or even the vendors themselves) will write and make available such bridge/proxy software for a given system, and that the rest of the research

community can simply interface to such systems using OpenIGTLink. Third, it has been proposed to record vital signs, images and device-monitoring information during clinical cases for later review (34). The standardized communication protocol allows the introduction of a comprehensive recording system that can store every message passed through the network and replay them after a clinical case.

In conclusion, we have proposed an open, simple and extensible network communication protocol specialized for IGT. We developed the library as a reference implementation for this protocol and the interface module for surgical navigation software as a research platform. Image and tracking data transfers were evaluated, and the protocol proved capable of handling data required in the IGT setting with sufficient time resolution and latency. The protocol helps not only to improve the interoperability of IGT devices and software but also to promote rapid and safe transition of research results to clinical use and commercial utilization.

Acknowledgments

The authors thank Mr Csaba Csoma from Johns Hopkins University and Mr Hiroaki Kozuka from Nagoya Institute of Technology for their help in designing and implementing the OpenIGTLink protocol. This work was supported by NIH Grants Nos 1R01CA111288, 5U41RR019703, 5P01CA067165, 1R01CA124377, 5P41RR013218, 5U54EB005149, 5R01CA 109246, R01EB006494 and R21EB007770, and also in part by NSF 9731748, CIMIT and the Intelligent Surgical Instruments Project of METI (Japan).

References

1. Dimaio S, Kapur T, Cleary K, et al. Challenges in image-guided therapy system design. NeuroImage 2007;37:S144–S151. [PubMed: 17644360]
2. Bucholz RD, Smith KR, Henderson J. Intraoperative localization using a three-dimensional optical digitizer. SPIE 1993:312–322.
3. Birkfellner W, Watzinger F, Wanschitz F, et al. Calibration of tracking systems in a surgical environment. IEEE Trans Med Imag 1998;17(5):737–742.
4. Barnett GH, Kormos DW, Steiner CP, et al. Intraoperative localization using an armless, frameless stereotactic wand. Technical note. J Neurosurg 1993;78(3):510–514. [PubMed: 8433160]
5. Roberts DW, Strohbehn JW, Hatch JF, et al. A frameless stereotaxic integration of computerized tomographic imaging and the operating microscope. J Neurosurg 1986;65(4):545–549. [PubMed: 3531430]
6. IEEE Standard for medical device communications – overview and framework. IEEE Standard No. 1073–1996. 1996
7. EN 50325–4. 2002 Industrial communications subsystem based on ISO 11898 (CAN) for controller–device interfaces, Part 4: CANopen; 2002. p. 1995EN 50325–4
8. ISO 11898–1. 2003. Road vehicles – interchange of digital information – controller area network (CAN) for high-speed communication; 2003. p. 2003ISO 11898–1
9. Schrenker RA. Software engineering for future healthcare and clinical systems. Computer 2006;39(4): 26–32.
10. The National Electrical Manufacturers Association (NEMA). Digital Imaging and Communication in Medicine (DICOM). Rosslyn, VA, USA: NEMA Publications PS; 2007. p. 31-318.
11. Lemke H, Vannier M. The operating room and the need for an IT infrastructure and standards. Int J Comput Assist Radiol Surg 2006;1(3):117–121.
12. Papademetris, X.; Vives, KP.; DiStasio, M., et al. Development of a research interface for image-guided intervention: initial application to epilepsy neurosurgery. International Symposium on Biomedical Imaging (ISBI); 2006. p. 490-493.
13. Fischer J, Neff M, Freudenstein D. Medical augmented reality based on commercial image guided surgery. Eurographics Symposium on Virtual Environments (EGVE) 2004:83–86.
14. Joshi A, Scheinost D, Vives KP, et al. Novel interaction techniques for neurosurgical planning and stereotactic navigation. IEEE Trans Visualiz Comput Graphics 2008;14(6):1587–1594.

15. Matinfar M, Baird C, Batouli A, et al. Robot assisted skull base surgery. IEEE International Conference on Intelligent Robots and Systems (IROS). 2007
16. Xia T, Baird C, Jallo G, et al. An integrated system for planning, navigation and robotic assistance for skull base surgery. Int J Med Robot 2008;4(4):321–330. [PubMed: 18803337]
17. Schorr O, Hata N, Bzostek A, et al. Distributed modular computer-integrated surgical robotic systems: architecture for intelligent object distribution. Medical Image Computing and Computer Assisted Intervention (MICCAI) International Conference 2000;1935:979–987.
18. Reitmayr, G.; Schmalstieg, D. Virtual Reality December 2005. London: Springer; 2006. A flexible software design for three-dimensional interaction; p. 79-92.
19. von Spiczak J, Samset E, Dimaio S, et al. Device connectivity for image-guided medical applications. Stud Health Technol Inform 2007;125:482–484. [PubMed: 17377332]
20. Fischer GS, Iordachita I, Csoma C, et al. MRI-compatible pneumatic robot for transperineal prostate needle placement. IEEE/ASME Trans Mechatron 2008;13(3):295–305.
21. National Alliance for Medical Image Computing (NA-MIC). OpenIGTLink. 2008. http://www.na-mic.org/Wiki/index.php/OpenIGTLink
22. Gering DT, Nabavi A, Kikinis R, et al. An integrated visualization system for surgical planning and guidance using image fusion and an open MR. J Magn Reson Imag 2001;13(6):967–975.
23. Pieper, S.; Halle, M.; Kikinis, R. 3D Slicer; IEEE International Symposium on Biomedical Imaging: Nano to Macro; Arlington, VA, USA. 2004.
24. Correll, K.; Barendt, N.; Branicky, M. Design considerations for software only implementations of the IEEE 1588 precision time protocol. Conference on IEEE 1588–2002; Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control System; 10–12; Winterthur, Switzerland. 2005 Oct. p. 1-6.
25. IEEE Standard No. 1588–2002. IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. IEEE Standard No. 1588–2002 2002:i–144.
26. Ordas, S.; Yaniv, Z.; Cheng, P., et al. SPIE 2009. Lake Buena Vista, FL, USA: 2009 Feb 11. Interfacing proprietary hardware with the image-guided surgery toolkit (IGSTK): a case for the OpenIGTLink protocol.
27. Enquobahrie A, Cheng P, Gary K, et al. The image-guided surgery toolkit IGSTK: an open source C ++ software toolkit. J Digit Imag 2007;20:21–33.
28. Birkfellner, W.; Hummel, J.; Wilson, E. Tracking devices. In: Peters, T.; Cleary, K., editors. In Image-Guided Intervention: Technology and Applications. London: Springer; 2008. p. 23-44.
29. Papademetris X, Delorenzo C, Flossmann S, et al. From medical image computing to computer-aided intervention: development of a research interface for image-guided navigation. Int J Med Robot. 2009 (in press).
30. BioImage Suite. Yale University; http://www.bioimagesuite.org/
31. Arata, J.; Kozuka, H.; Kim, W., et al. 23th International Congress and Exhibition, Computer Assisted Radiology and Surgery. Berlin, Germany: 2009 Jun. An open source control software using virtual fixture for surgical robots; p. 23-27.
32. Microsoft. MSDN Library. 2009. http://msdn.microsoft.com/enus/library/default.aspx
33. Lesniak J, Tokuda J, Kikinis R, et al. A device guidance method for organ motion compensation in MRI-guided therapy. Phys Med Biol 2007;52(21):6427–6438. [PubMed: 17951853]
34. Ikuta, K.; Kato, T.; Ando, S., et al. World Congress on Medical Physics and Biomedical Engineering. Korea: Seoul; 2006 27 August–1 September. Development of surgery recorder system for minimally invasive surgery.

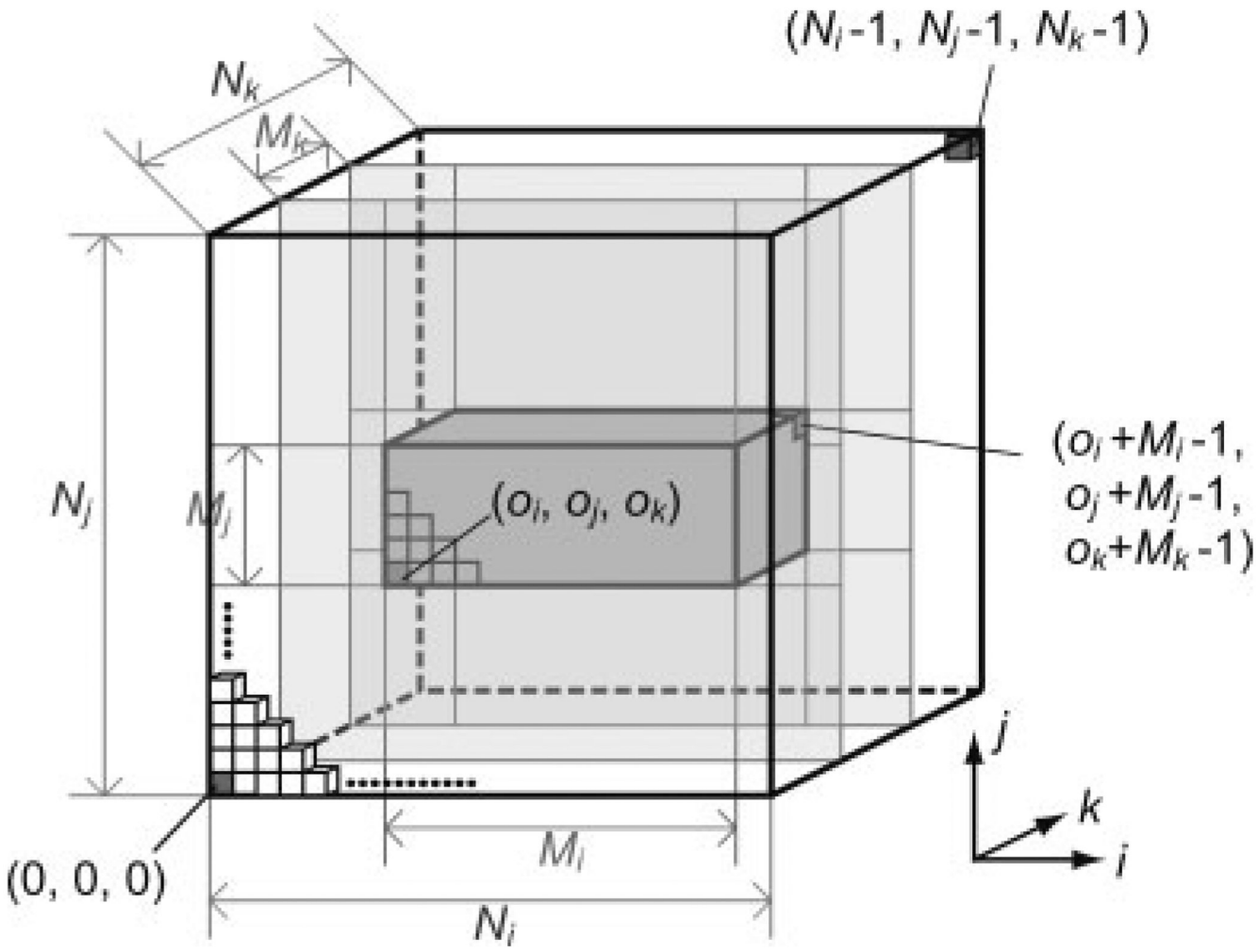
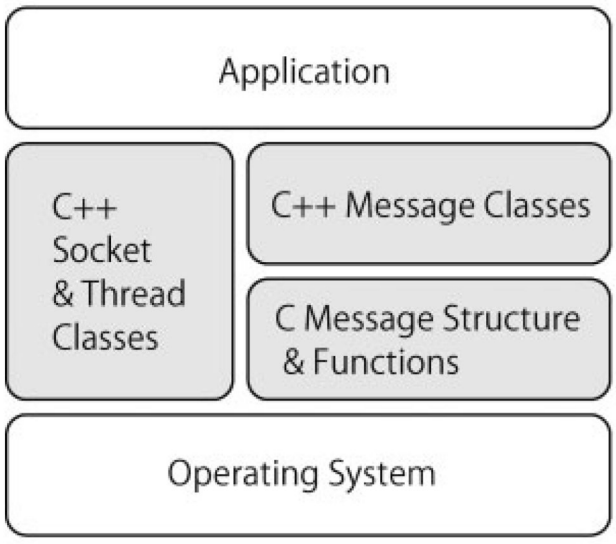


Figure 1.
Parameters to support partial 3D volume update. The outer cube denotes the entire range of the 3D image, and the inner cube denotes the partial 3D image that is contained in the message as a byte array




```
// Create and open a socket
igtl::ClientSocket::Pointer socket;
socket = igtl::ClientSocket::New();
socket->ConnectToServer(hostname, port);

// Create a TRANSFORM meesage
igtl::TransformMessage::Pointer transMsg;
transMsg = igtl::TransformMessage::New();
transMsg->SetDeviceName("Tracker");
transMsg->SetMatrix(matrix);

// Serialize the message
transMsg->Pack();

// Send the message
socket->Send(transMsg->GetPackPointer(),
             transMsg->GetPackSize());

// Close the socket
socket->CloseSocket();
```

Figure 2.

The structure of the OpenIGTLink Library (left) and an example code to send TRANSFORM data using the C++ message class and socket class (right). At the lowest level, the messages are defined as C structures with several supporting functions for message serialization. On top of the C structures and function, C++ message classes are built to provide an easier, safer and more extensible way to access to OpenIGTLink messages. As shown in the example code, the message classes have several access functions to set parameters for both message header and body, and a function to serialize the message and body. Besides message classes, multi-platform C++ socket and thread classes are provided to support writing platform-independent application codes

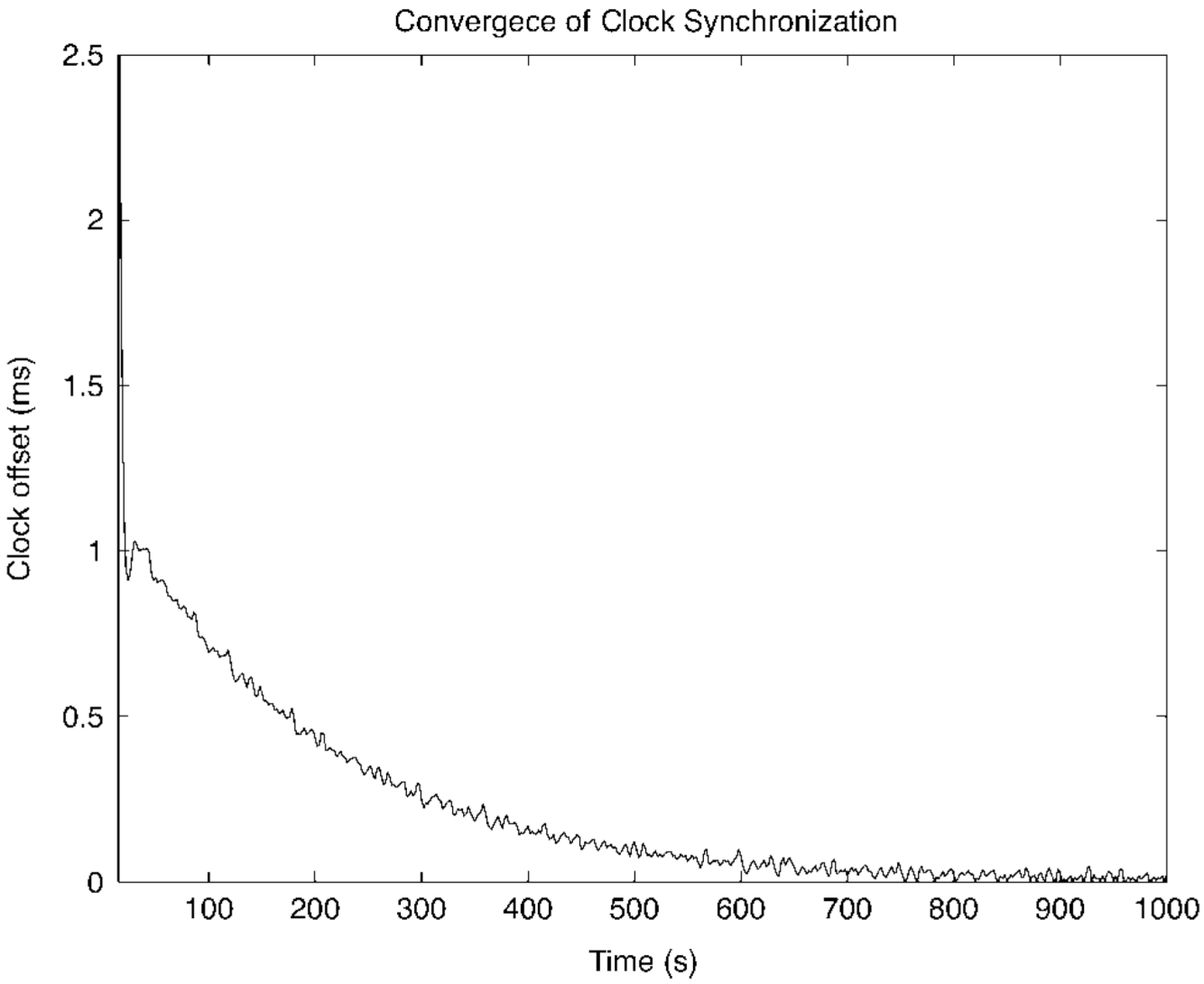


Figure 3.
The convergence of time offset between two hosts during time synchronization by PTPd

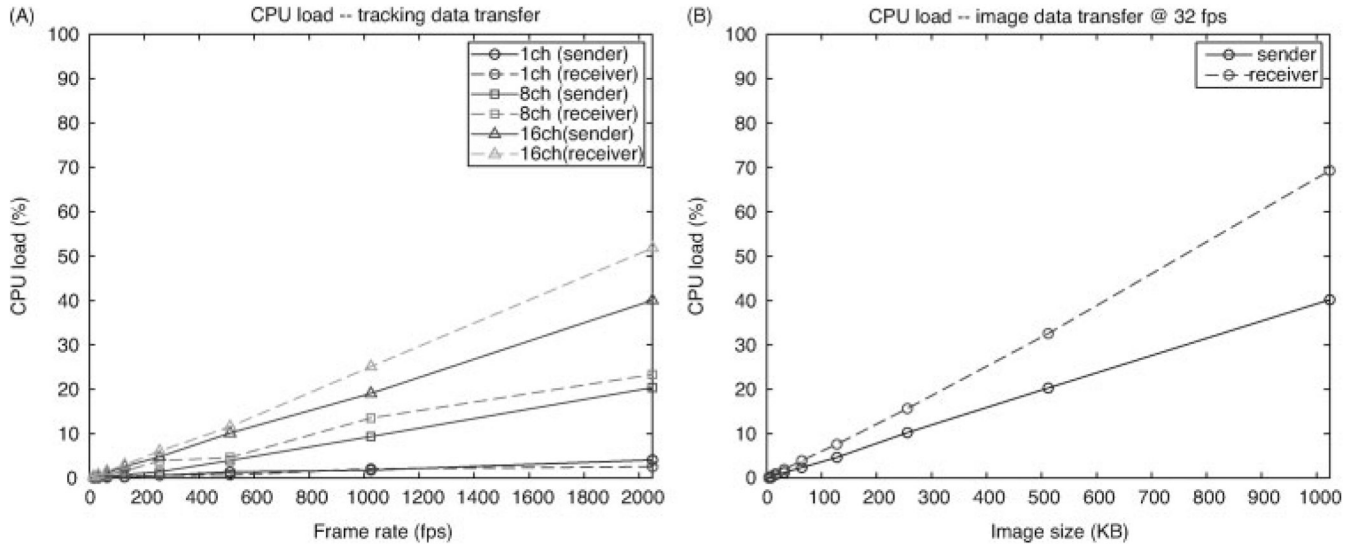


Figure 4.

(A) CPU loads during tracking data transfer are compared among 1-, 8- and 16-channel tracking. Comparison is also made between sender and receiver. (B) CPU loads during the image data transfer are compared between sender and receiver

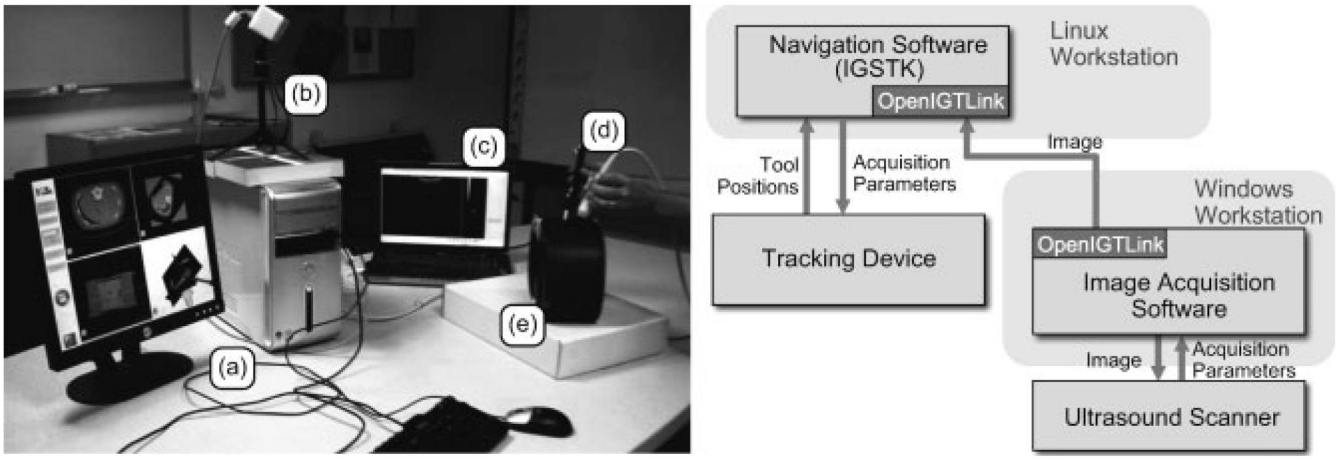


Figure 5.
CT navigation system with ultrasound (US) image acquisition: (a) workstation for navigation system; (b) tracking device; (c) US Scanner; (d) US probe; (e) abdominal phantom

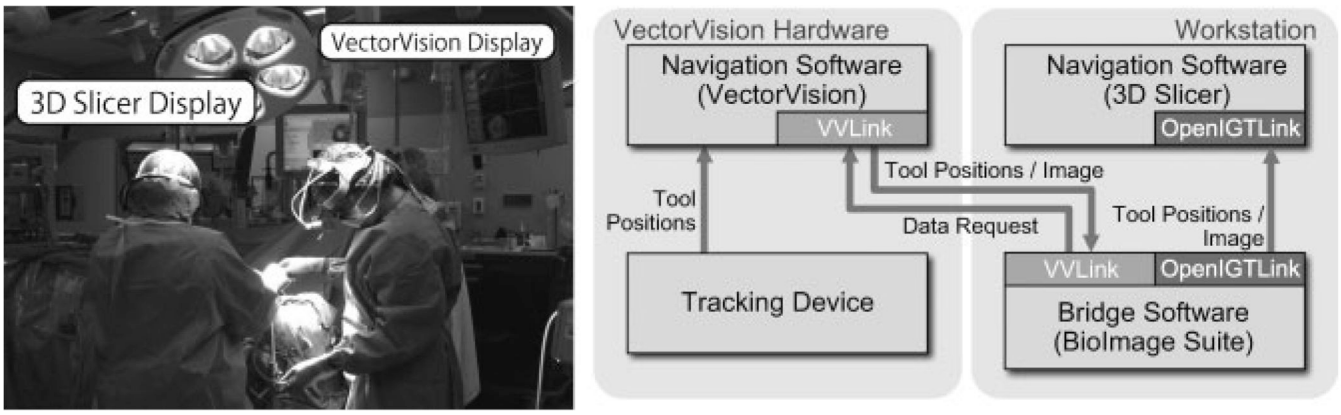


Figure 6.
Overview (left) and system diagram (right) of VectorVision -3D Slicer integration in the operating room. VectorVision provides basic navigation features, including tool tracking and image display, while 3D Slicer provides advanced image processing. The surgeons can check both the VectorVision console and the 3D Slicer user interface on dedicated displays during surgery

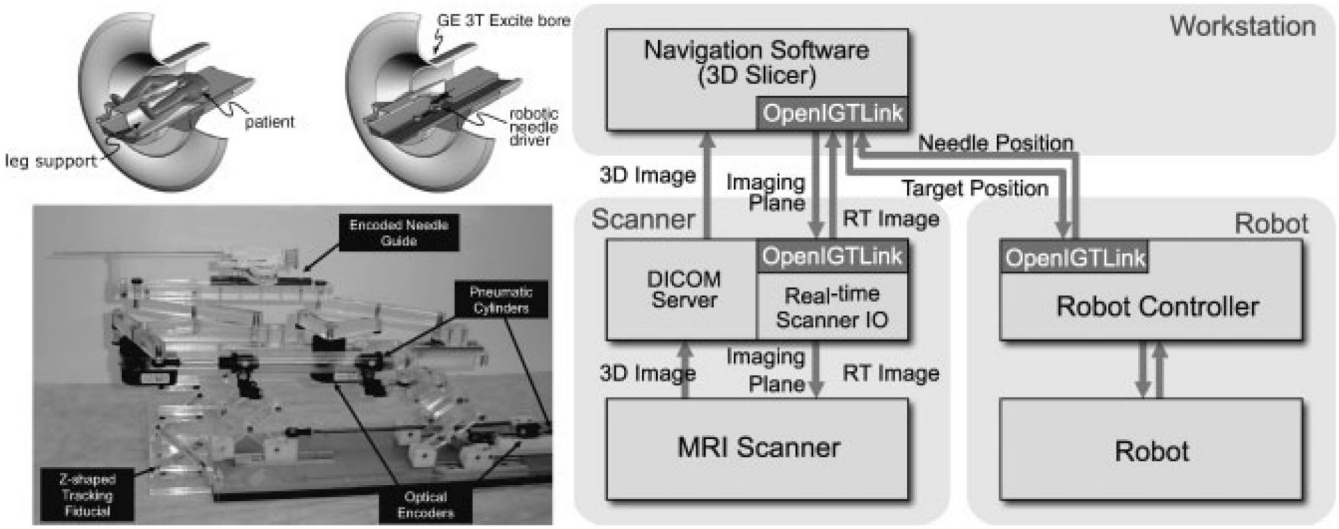


Figure 7.

A robot for transperineal prostate biopsy and treatment (left) and its system configuration (right). Pneumatic actuators and optical encoders allow the robot to be operated inside a closed-bore 3T MRI scanner

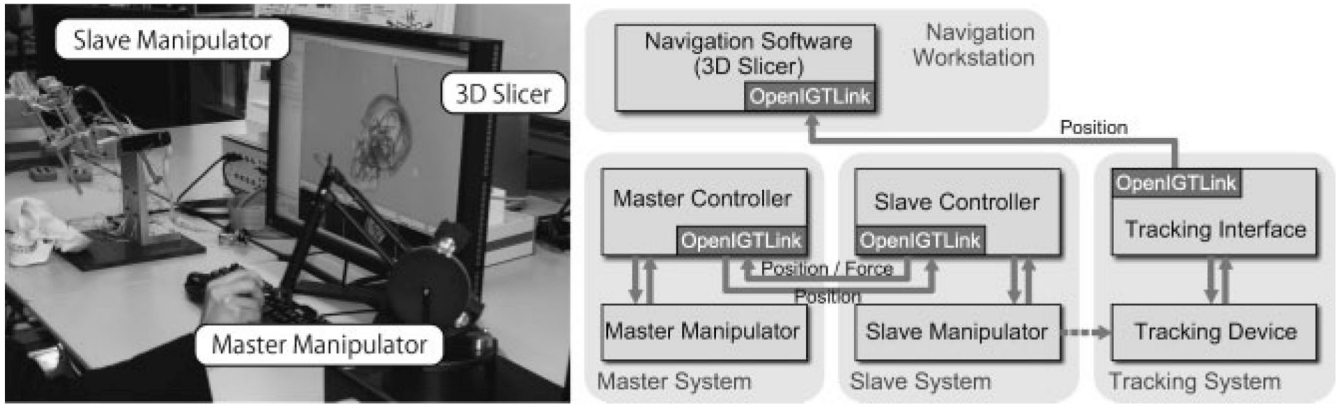


Figure 8.

The optical tracking device and surgical navigation software 3D Slicer were integrated into the master–slave surgical manipulator system developed at Nagoya Institute of Technology, Japan. The OpenIGTLink protocol was used for communication between the tracking device and navigation software. The position of the end-effector was tracked by the optical tracking system and transferred to the surgical navigation software. The end-effector was displayed as a 3D model with a patient model, which was created from pre-operative images

Table 1

Device status defined in OpenIGTLink protocol

Code	Description
0	Invalid packet
1	OK
2	Unknown error
3	Panic
4	Not found
5	Access denied
6	Busy
7	Time out
8	Overflow
9	Checksum error
10	Configuration error
11	Resource error
12	Unknown instruction
13	Device not ready
14	Manual mode
15	Device disabled
16	Device not present
17	Unknown device
18	Hardware failure
19	Shutdown in progress

Table 2

Mean, standard deviation (SD) and maximum latency of the tracking data transfer

Frame rate (fps)	Mean (ms)	SD (ms)	Maximum (ms)
128	0.36	0.01	0.47
512	0.36	0.02	0.52
1024	0.45	0.07	0.83

The table shows only the result from 16-channel tracking, which is the worst case among the conditions we tested in terms of latency

Table 3

Mean, SD and maximum latency of the image data transfer for image sizes 128, 256 and 512 KB

Image size (KB)	Mean (ms)	SD (ms)	Maximum (ms)
128	3.94	0.01	5.16
256	7.68	0.01	8.96
512	15.51	0.03	17.93

Table 4

Latencies of the tracking data transfers during simultaneous data transfer with image transfer frame rates of 2, 16, and 32 fps, to demonstrate how imaging data transfer affects the tracking data transfer performed in the same OpenIGTLink connection. The size rate of image data transfer was fixed at 4096 KB/s

Frame rate (fps)/image size (KB)	Mean (ms)	SD (ms)	Maximum (ms)
2/2048	3.42	10.15	66.19
16/256	0.51	1.40	7.84
32/128	0.31	0.78	4.01

Table 5

Latency of image data transfers during simultaneous data transfer with image transfer frame rates of 2, 16, and 32 fps in the same experiment as Table 4

Frame rate (fps)/image size (KB)	Mean (ms)	SD (ms)	Maximum (ms)
2/2048	66.23	0.27	67.37
16/256	7.71	0.12	8.70
32/128	3.93	0.07	4.37