MUSiiC ToolKit 2.0:

Bidirectional Real-time Software Framework for Advanced Interventional Ultrasound Research.

Release 1.00

Hyun-Jae Kang[1], Alexis Cheng[1],Emad M. Boctor[1,2]

August 10, 2012

[1] Department of Computer Science, Johns Hopkins University, Baltimore, MD/USA
[2]Department of Radiology, DMIP, Johns Hopkins Medical Institutions, Baltimore, MD/USA

Abstract

Ultrasound (US) imaging is a popular and convenient medical imaging modality thanks to its mobility, non-ionizing radiation, ease-of-use, and real-time acquisition. Conventional US imaging is frequently integrated with tracking systems and robotic systems in Image Guided Therapy (IGT) systems. Recently, these systems are also incorporating advanced US imaging such as US elasticity imaging, photoacoustic imaging, and thermal imaging. Real-time synchronous data from multiple sources and bidirectional data communication are essential for integrating components in advanced US IGT research. We previously proposed the MUSiiC ToolKit [1], a modular real-time software toolkit, and OpenIGTLinkMUSiiC [2], a standard communication protocol extended from the OpenIGTLink library [3, 4]. However, this software framework only supported real-time synchronous data from at most two sources and unidirectional communication at the software module level and class level.

In this paper, we propose MUSiiC ToolKit 2.0, an upgraded software framework for interventional advanced US research supporting bidirectional communication, real-time US data processing, and real-time data synchronization from multiple sources. MUSiiC ToolKit 2.0 consists of OpenIGTLink 2.0, OpenIGTlinkMUSiiC 2.0, MUSiiCNotes 2.0, and a collection of executable programs designed for US research. OpenIGTLink 2.0 is a standard TCP/IP-based protocol for the integration of medical imaging and IGT systems. OpenIGTLinkMUSiiC 2.0 is the upgraded version of OpenIGTLinkMUSiiC with new active multi-task classes, data interfaces for supporting bidirectional communication and parallel data processing. MUSiiCNotes 2.0 provides US research-oriented task classes, such as US data acquisition, beamforming, envelope detection, scan conversion, and data synchronization. Graphic User Interface (GUI) units are also available for the executable programs in MUSiiCNotes 2.0. Finally, we introduce advanced US applications based on this new software framework.

Contents

1 Introduction

Ultrasound (US) imaging has many qualities that make it a popular and convenient medical imaging modality. These qualities include its mobility, non-ionizing radiation, ease-of-use, and real-time data acquisition. US imaging systems are often used in the operating room and the emergency room because of these features. Moreover, conventional US imaging is frequently integrated with other medical imaging modalities such as pre-operative models (CT, MRI), tracking systems and robotic systems for Image Guided Therapy (IGT) [5-7]. These systems are also exploring the use of advanced US imaging such as US elastography, photoacoustic imaging, and thermal imaging [5, 8, 9]. Several software frameworks and toolkits have been developed to integrate US data acquisition, processing and displays with existing IGT systems [1, 2, 6, 10, 11]. In our previous work [1, 2, 12], we proposed a real-time software framework for interventional ultrasound research. It consisted of *MUSiiC ToolKit*, a modular real-time software toolkit, and *OpenIGTLinkMUSiiC*, a standard communication protocol extended from the *OpenIGTLink* library [3, 4].

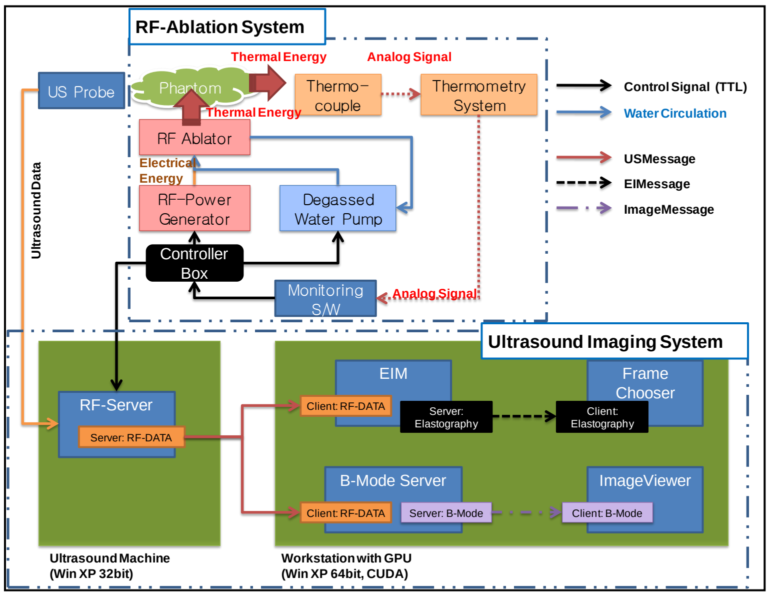

Figure 1. The System configuration of ultrasound thermal monitoring system based on *MUSiiC ToolKit1.0* and *OpenIGTLinkMUSiiC 1.0* [2]

Figure 1 shows the system configuration of a US thermal monitoring system based on *MUSiiC ToolKit 1.0* and *OpenIGTLinkMUSiiC 1.0*. In the figure, all components in the block diagram are connected with a unidirectional communication mechanism. This means that the data source module, *RF-Server*, cannot receive any feedback information from its client modules, *EIM* or *B-Mode server*. In this block diagram, an end user of *ImageViewer* will not be able to set control parameters of *RF-Server* that affect ultrasound data acquisition. As shown in this example, our previous software framework did not support closed-loop feedback control by not providing bidirectional communication between software modules or task classes.

IGT procedures typically require many sources of information such as surgical instrument tracking information, pre-operative medical images, intra-operative medical images, or multiple medical imaging modalities. This presents a need for an efficient and interactive communication mechanism between the various data sources. With continued research on US IGT systems, there is a growing demand for real-time US data acquisition and processing, and synchronization of data from multiple sources such as tracking data or temperature data. There is also a demand for a bidirectional communication mechanism between modules in an IGT system.

To address the requirements above, we propose *MUSiiC ToolKit 2.0*, an upgraded software framework for interventional advanced US research in IGT systems. In this paper, we will focus on the following new features of our upgraded software framework: (1) Bidirectional communication mechanism at application level and task-class level. (2) Real-time US data acquisition and processing method. (3) Real-time data synchronization from multiple data sources.

This paper is organized as follows: Section 2 provides an overview of *MUSiiC ToolKit 2.0* and a detailed explanation of the new functionalities in this software framework. In section 3, we introduce an advanced US research applications based on *MUSiiC ToolKit 2.0*. We conclude with a discussion on possible future improvements and directions in section 4.

2 MUSiiC ToolKit 2.0

Figure 2 shows an overview of *MUSiiC ToolKit 2.0*. Our software framework can be classified into four categories: *OpenIGTLink 2.0* [3, 4], *OpenIGTLinkMUSiiC 2.0*, *MUSiiC Notes 2.0*, and *MUSiiC Modules*, a collection of executable programs. *OpenIGTLink 2.0* is the software library of standard TCP/IP-based message protocol for the integration of medical IGT system, which is proposed by J. Tokuda [3]. In this library, multiple data types are defined as TCP/IP messages for real-time communication between subsystems of an IGT system, and there are serialization and deserialization mechanisms for each message.

OpenIGTLinkMUSiiC is the extended version of *OpenIGTLink* by adding a special ultrasound data message and other message types (*GenMessage*, *ArgMessage*, and *FileMessage*) for advanced ultrasound research [2]. In this research, we upgrade this software library to *OpenIGTLinkMUSiiC 2.0* by adding active task classes that have their own independent task thread. These task threads are based on multithreaded techniques and thread-safe inter-process communication (IPC) [13] for efficient real-time ultrasound data computation. We also implemented the Observer design pattern [14] in this library for bidirectional communication between applications or active task class components. Since *OpenIGTLinkMUSiiC 2.0* provides the basic functionalities such as abstract active-task classes (*MUSiiCTaskObject*, *MUSiiCTaskAbstract*, and *MUSiiCTaskInterfaceAbstract*), thread-safe data interfaces (*MUSiiCVector* and

MUSiiCVectorSet), and callback interfaces (*MUSiiCCallbackInterface*, *MUSiiCCallbackInterface-Control*), it is therefore considered as the fundamental library in *MUSiiC ToolKit 2.0*.

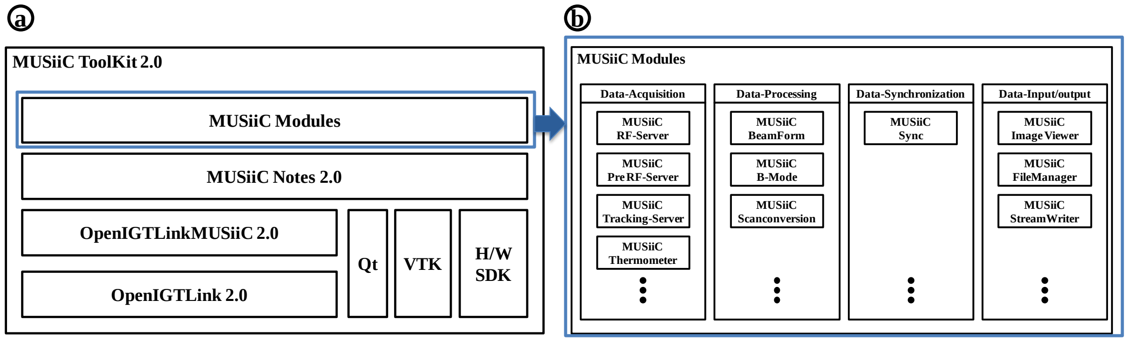

Figure 2. Overview of *MUSiiC ToolKit 2.0*

MUSiiCNotes 2.0 is the component based library in our software framework. It provides US research-oriented task classes, such as US data acquisition, beamforming, envelope detection, scan conversion, data synchronization, and Graphic User Interface (GUI) units for each task class or executable program. As shown in figure 2, *MUSiiC Notes 2.0* depends on *OpenIGTLink 2.0* [3, 4], *OpenIGTLinkMUSiiC 2.0*, the Qt SDK [15], the Visualization ToolKit (VTK) [16], and hardware-dependent SDKs such as the Ultrasonix SDK (Ultrasonix Co.) [17] and the 3D Guidance medSAFE/driveBay SDK (Ascension Technology Co.) [18].

MUSiiC Modules is a collection of executable programs related to US research. All of these programs are based on a network distributed computing system to improve the performance of US data processing and the flexibility to reconfigure the US research system. To support these features, all programs have their own network classes (*MUSiiCTCPServer* and *MUSiiCTCPClient*) and file input/output (I/O) classes (*MUSiiCFileIO*). Figure 2 also shows that there are four types of executable modules: Data acquisition modules, Data Processing modules, Data synchronization module, and Data I/O modules. Different data acquisition modules support collecting US pre-beamformed and post-beamformed RF data, tracking information of a medical device, or temperature information in real-time. The data synchronization module can combine different data from multiple data sources using the timestamps of each data. We can also add extra information such as ultrasound calibration information [11] to the synchronized data at this module. The data I/O modules allow us to efficiently display the data in various formats on the computer monitor. It also gives us the capability to read and write the data to files on the local hard disk in real-time.

2.1 MUSiiC ToolKit 2.0: Real-time US Data-Computation

A significant advantage of US in IGT systems is its ability to provide intra-operative data such as B-mode images, US strain images, or photoacoustic images [1, 9, 12, 19]. However, in general, US data processing is computationally expensive [20]. Therefore, a well-defined task abstract class that supports multithreaded programming is an essential part of a software framework for US research.

To fulfill these requirements, we built an active task object class, *MUSiiCTaskObject*, to run a task function with an independent thread, and two task abstract classes, *MUSiiCTaskAbstract* and *MUSiiCTask-InterfaceAbstract*, to manage multiple task objects efficiently. Also, we made a thread-safe data interface, *MUSiiCVector*, for transferring data between task objects or task abstract classes and a data interface manager, *MUSiiCVectorSet*. Both of these classes define "*concurrent_pushback*", "*concurrent_pop*", and “*concurrent_get”* functions that allow the data to be safely written, deleted, or read by multiple threads. Figure 3 represents the Unified Modeling Language (UML) class diagrams for *MUSiiCTaskObject*, *MUSiiCTaskAbstract*, *MUSiiCTaskInterfaceAbstract*, *MUSiiCVector*, and *MUSiiCVectorSet*.

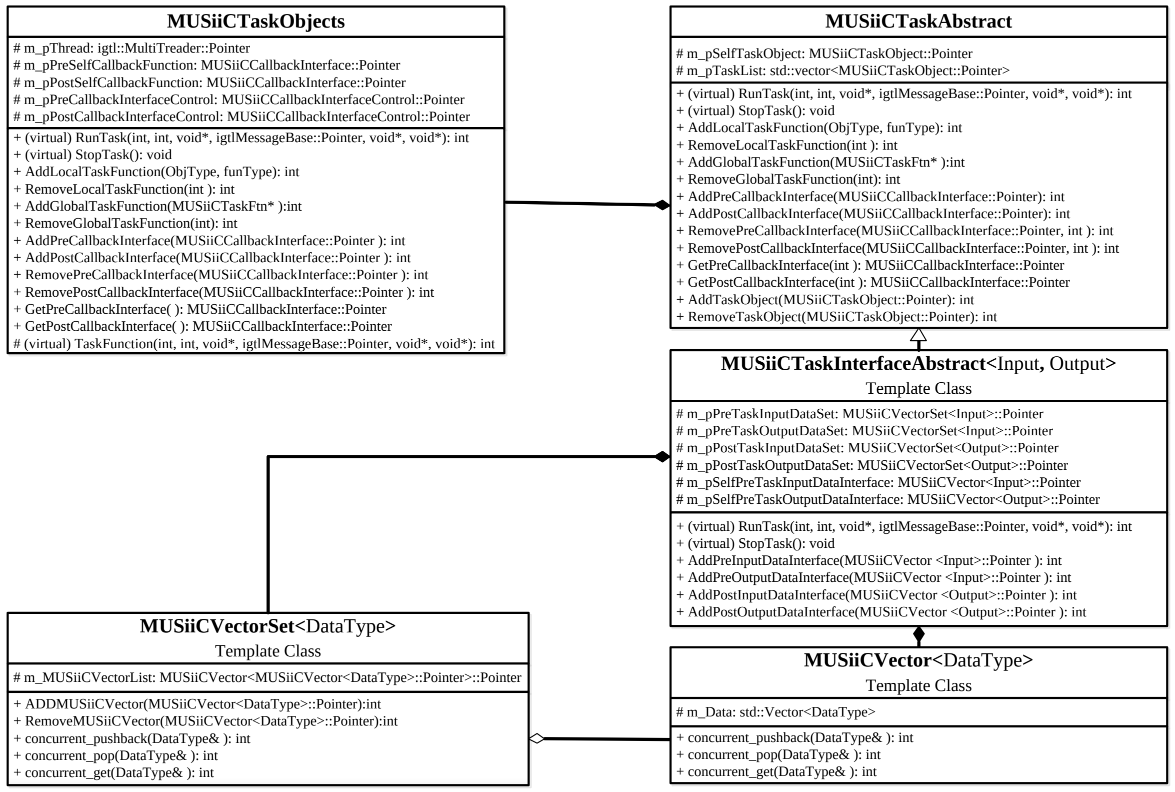
MUSiiCTaskObjects
# m_pThread: igtl::MultiTreader::Pointer
# m_pPreSelfCallbackFunction: MUSiiCCallbackInterface::Pointer
# m_pPostSelfCallbackFunction: MUSiiCCallbackInterface::Pointer
# m_pPreCallbackInterfaceControl: MUSiiCCallbackInterfaceControl::Pointer
# m_pPostCallbackInterfaceControl: MUSiiCCallbackInterfaceControl::Pointer
+ (virtual) RunTask(int, int, void*, igtlMessageBase::Pointer, void*, void*): int
+ (virtual) StopTask(): void
+ AddLocalTaskFunction(ObjType, funType): int
+ RemoveLocalTaskFunction(int): int
+ AddGlobalTaskFunction(MUSiiCTaskFtn*):int
+ RemoveGlobalTaskFunction(int): int
+ AddPreCallbackInterface(MUSiiCCallbackInterface::Pointer): int
+ AddPostCallbackInterface(MUSiiCCallbackInterface::Pointer): int
+ RemovePreCallbackInterface(MUSiiCCallbackInterface::Pointer): int
+ RemovePostCallbackInterface(MUSiiCCallbackInterface::Pointer): int
+ GetPreCallbackInterface( ): MUSiiCCallbackInterface::Pointer
+ GetPostCallbackInterface( ): MUSiiCCallbackInterface::Pointer
# (virtual) TaskFunction(int, int, void*, igtlMessageBase::Pointer, void*, void*): int
MUSiiCTaskAbstract
# m_pSelfTaskObject: MUSiiCTaskObject::Pointer
# m_pTaskList: std::vector<MUSiiCTaskObject::Pointer>
+ (virtual) RunTask(int, int, void*, igtlMessageBase::Pointer, void*, void*): int
+ (virtual) StopTask(): void
+ AddLocalTaskFunction(ObjType, funType): int
+ RemoveLocalTaskFunction(int): int
+ AddGlobalTaskFunction(MUSiiCTaskFtn*):int
+ RemoveGlobalTaskFunction(int): int
+ AddPreCallbackInterface(MUSiiCCallbackInterface::Pointer): int
+ AddPostCallbackInterface(MUSiiCCallbackInterface::Pointer): int
+ RemovePreCallbackInterface(MUSiiCCallbackInterface::Pointer, int): int
+ RemovePostCallbackInterface(MUSiiCCallbackInterface::Pointer, int): int
+ GetPreCallbackInterface(int): MUSiiCCallbackInterface::Pointer
+ GetPostCallbackInterface(int): MUSiiCCallbackInterface::Pointer
+ AddTaskObject(MUSiiCTaskObject::Pointer): int
+ RemoveTaskObject(MUSiiCTaskObject::Pointer): int
MUSiiCTaskInterfaceAbstract<Input, Output>
Template Class
# m_pPreTaskInputDataSet: MUSiiCVectorSet<Input>::Pointer
# m_pPreTaskOutputDataSet: MUSiiCVectorSet<Input>::Pointer
# m_pPostTaskInputDataSet: MUSiiCVectorSet<Output>::Pointer
# m_pPostTaskOutputDataSet: MUSiiCVectorSet<Output>::Pointer
# m_pSelfPreTaskInputDataInterface: MUSiiCVector<Input>::Pointer
# m_pSelfPreTaskOutputDataInterface: MUSiiCVector<Output>::Pointer
+ (virtual) RunTask(int, int, void*, igtlMessageBase::Pointer, void*, void*): int
+ (virtual) StopTask(): void
+ AddPreInputDataInterface(MUSiiCVector <Input>::Pointer): int
+ AddPreOutputDataInterface(MUSiiCVector <Input>::Pointer): int
+ AddPostInputDataInterface(MUSiiCVector <Output>::Pointer): int
+ AddPostOutputDataInterface(MUSiiCVector <Output>::Pointer): int
MUSiiCVectorSet<DataType>
Template Class
# m_MUSiiCVectorList: MUSiiCVector<MUSiiCVector<DataType>::Pointer>::Pointer
+ ADDMUSiiCVector(MUSiiCVector<DataType>::Pointer):int
+ RemoveMUSiiCVector(MUSiiCVector<DataType>::Pointer):int
+ concurrent_pushback(DataType&): int
+ concurrent_pop(DataType&): int
+ concurrent_get(DataType&): int
MUSiiCVector<DataType>
Template Class
# m_Data: std::Vector<DataType>
+ concurrent_pushback(DataType&): int
+ concurrent_pop(DataType&): int
+ concurrent_get(DataType&): int

Figure 3. UML class diagram of *MUSiiCTaskObjects*, *MUSiiCTaskAbstract*, *MUSiiCTaskInterfaceAbstract*, *MUSiiCVector*, and *MUSiiCVectorSet*.

The *MUSiiCTaskObject* class has a *MultiThreader* class pointer from the *OpenIGTLink* library that supports independent threads on multiple operating systems. In this class, the main task function is declared as a virtual function. This means that we can implement our own specific task function in any subclasses of this class. The task function of *MUSiiCTaskObject* follows the function type of *MUSiiCTaskFunction*/ *MUSiiCCallbackFunction* as shown in Table 1. Multiple input parameters are defined in these function types, allowing us to delicately control a task function. Moreover, we can build a task sequence by adding local or global functions of *MUSiiCTaskFunction* type to *MUSiiCTaskObject*.

```
/// The function type of MUSiiCTaskFunction/MUSiiCCallbackFunction
///  Input arguments:
///                 - command : Represents a changed status or any command
///                 - taskInfo:  Describes the information of task
///                 - ptr: The pointer of caller
///                 - msg: Message type of OpenIGTLink  and OpenIGTLinkMUSiiC
///                 - data1: User-defined data
///                 - data2: User-defined data
/// Output arguments:
///                 return value is integer-type
///                  -1   : failed
///                 ≥0   : successes.

typedef  int (*MUSiiCTaskFtn) (int command, int  taskInfo, void* ptr, igtl::MessageBase::Pointer  msg, void* data1, void* data2)
```

Table 1. The function type of *MUSiiCTaskFunction*/*MUSiiCCallbackFunction*

Although we can run a task thread independently using an instance of *MUSiiCTaskObject*, multiple task threads are needed to improve the performance of US data processing. *MUSiiCTaskAbstract* was designed to address this requirement. In figure 3, *MUSiiCTaskAbstract* has a container variable and several task control functions to control multiple *MUSiiCTaskObject* instances and their own task-functions. *MUSiiCTaskInterfaceAbstract* is a subclass of *MUSiiCTaskAbstract*, which is designed for thread-safe data communication between *MUSiiCTaskInterfaceAbstract* objects. We designed *MUSiiCVector*, a concurrent data interface, based on the mutual exclusion method and *MUSiiCVectorSet*, a management class for this data interface. Both of these classes are based on the template design pattern and will allow any kind of data type to be transferred efficiently in our data interface.

Figure 4 shows the flexibility in our software framework to support a number of different multitasking patterns. The combination of *MUSiiCTaskObject*, *MUSiiCTaskAbstract* and *MUSiiCTasknterfaceAbstract* allow us to implement the cases shown in Figure 4(a), 4(b), 4(c), and 4(d) respectively. These cases are the combinations of single or multiple threads and a single task or a sequence of tasks.

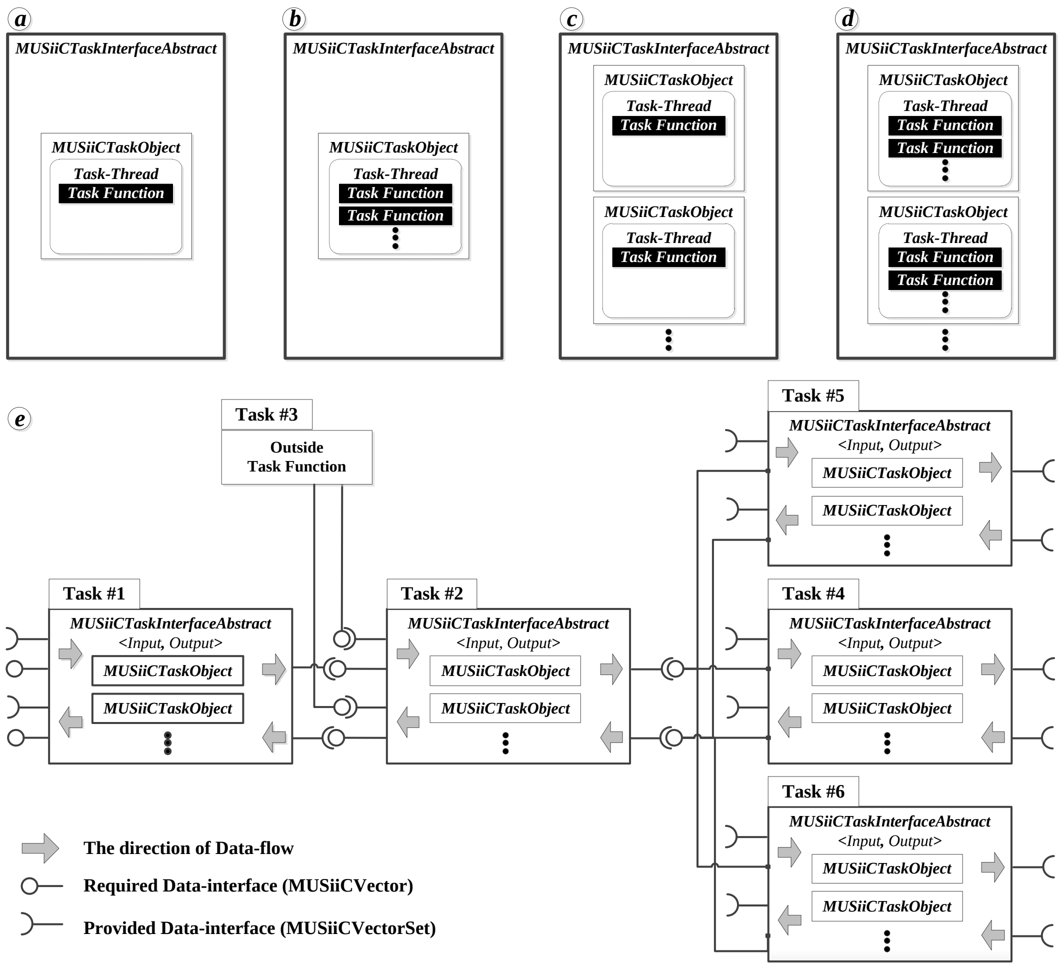
a
b
c
d
MUSiiCTaskInterfaceAbstract
MUSiiCTaskObject
Task-Thread
Task Function
e
Task #1
Task #2
Task #3
Outside Task Function
Task #4
Task #5
Task #6
<Input, Output>
The direction of Data-flow
Required Data-interface (MUSiiCVector)
Provided Data-interface (MUSiiCVectorSet)

Figure 4. Examples of Multitasking pattern using *MUSiiCTaskObject* and *MUSiiCTaskInterfaceAbstract* . (a) Single thread of single task, (b) Single thread of task sequence, (c) Multiple threads of single task, (d) Multiple threads of task sequence, and (e) Network diagram of task classes based on *MUSiiCTask-InterfaceAbstract*.

Moreover, *MUSiiCTaskInterfaceAbstract* has instances of *MUSiiCVectorSet*, the required data interface, and *MUSiiCVector*, the provided data interface, for data communication between task class objects. As seen in figure 3, since *MUSiiCVectorSet* can control multiple *MUSiiCVector* objects concurrently, each task class can receive multiple data from different data providers. The result of a task class can also be broadcasted to multiple data consumers in parallel. Figure 4(e) shows the network of task classes that are based on *MUSiiCTaskInterfaceAbstract*. As shown in the figure, tasks 1 and 2 form a serial connection, while tasks 4, 5, and 6 are connected to task 2 in parallel. Task 2 and 3 demonstrate a serial connection between *MUSiiCTaskInterfaceAbstract* and any data provider or consumer that is not based on *MUSiiCTaskInterfaceAbstract*. This example shows that *MUSiiCTaskInterfaceAbstract* provides high flexibility in the connectivity and data communication between task classes.

2.2 MUSiiC ToolKit 2.0: Bidirectional Communication Mechanism

An interactive communication mechanism between the application level and task class component level is necessary to integrate US systems and IGT systems. Figure 4 shows that asynchronous bidirectional communication between the task classes using the instances of *MUSiiCVector* and *MUSiiCVectorSet* is possible. We describe a synchronous bidirectional communication mechanism between task classes and an asynchronous communication method between applications.

2.2.1 Bidirectional Communication Mechanism at software class level.

We apply the observer software design pattern in *OpenIGTLinkMUSiiC2.0* by adding instances of the *MUSiiCallbackInterface* and *MUSiiCCallbackInterfaceControl* classes in the *MUSiiCTaskObject* and *MUSiiCTaskAbstract* classes.

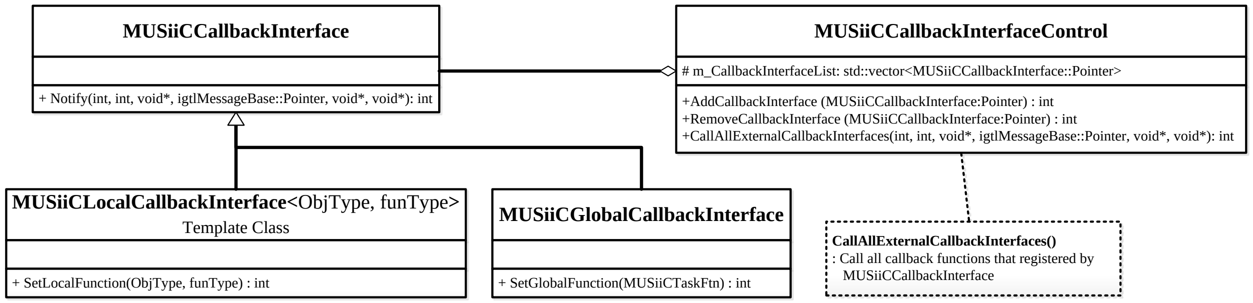
MUSiiCCallbackInterface
+ Notify(int, int, void*, igtlMessageBase::Pointer, void*, void*): int
MUSiiCCallbackInterfaceControl
# m_CallbackInterfaceList: std::vector<MUSiiCCallbackInterface::Pointer>
+AddCallbackInterface (MUSiiCCallbackInterface:Pointer) : int
+RemoveCallbackInterface (MUSiiCCallbackInterface:Pointer) : int
+CallAllExternalCallbackInterfaces(int, int, void*, igtlMessageBase::Pointer, void*, void*): int
MUSiiCLocalCallbackInterface<ObjType, funType>
Template Class
+ SetLocalFunction(ObjType, funType) : int
MUSiiCGlobalCallbackInterface
+ SetGlobalFunction(MUSiiCTaskFtn) : int
CallAllExternalCallbackInterfaces()
: Call all callback functions that registered by MUSiiCCallbackInterface

Figure 5. UML class diagram of *MUSiiCCallbackInterface* and *MUSiiCCallbackInterfaceControl*

Figure 5 represents the UML class diagram of *MUSiiCCallbackInterface*, *MUSiiCLocalCallbackInterface*, *MUSiiCGlobalCallbackInterface* and *MUSiiCCallbackInterfaceControl*. The *MUSiiCCallbackInterface* class provides basic observer functions: "*Notify"* that sends notifications or messages to a pre-registered callback function, "*IsCallbackFunction"* that checks whether any callback function has been registered in this interface. As shown in figure 5, this class has two child classes. *MUSiiCLocalCallbackInterface* is based on the template class design pattern and *MUSiiCGlobalCallbackInterface* can register any local or global function as a callback function of this interface class as long as it is of type *MUSiiCTaskFunction* or *MUSiiCCallbackFunction*. Moreover, *MUSiiCCallbackInterfaceControl* is designed to control multiple *MUSiiCCallbackInterface* objects efficiently. It has dedicated functions to add or remove a predefined instance of *MUSiiCCallbackInterface* to or from this class. All predefined callback functions registered in this class are notified or updated by "*CallAllExternalCallbackInterface"*.

As in figure 3, *MUSiiCTaskObject* has instances of *MUSiiCCallbackInterface* and *MUSiiCCallbackInterfaceControl*, and *MUSiiCTaskAbstract* provides several functions allowing access to these instances. With this design, we can realize a synchronous bidirectional communication based on the observer design pattern between the *MUSiiCTaskObject* instances in *MUSiiCTaskAbstract* or *MUSiiCTaskAbstract* classes. A network diagram of callback interfaces between task classes based on this software design is shown in figure 6. Although this network diagram is very similar to the network diagram in figure 4(e), it demonstrates synchronous bidirectional communication using callback interfaces as opposed to asynchronous bidirectional communication using concurrent data interfaces. This shows that our software framework is capable of providing asynchronous and synchronous bidirectional communication at the software task class level.

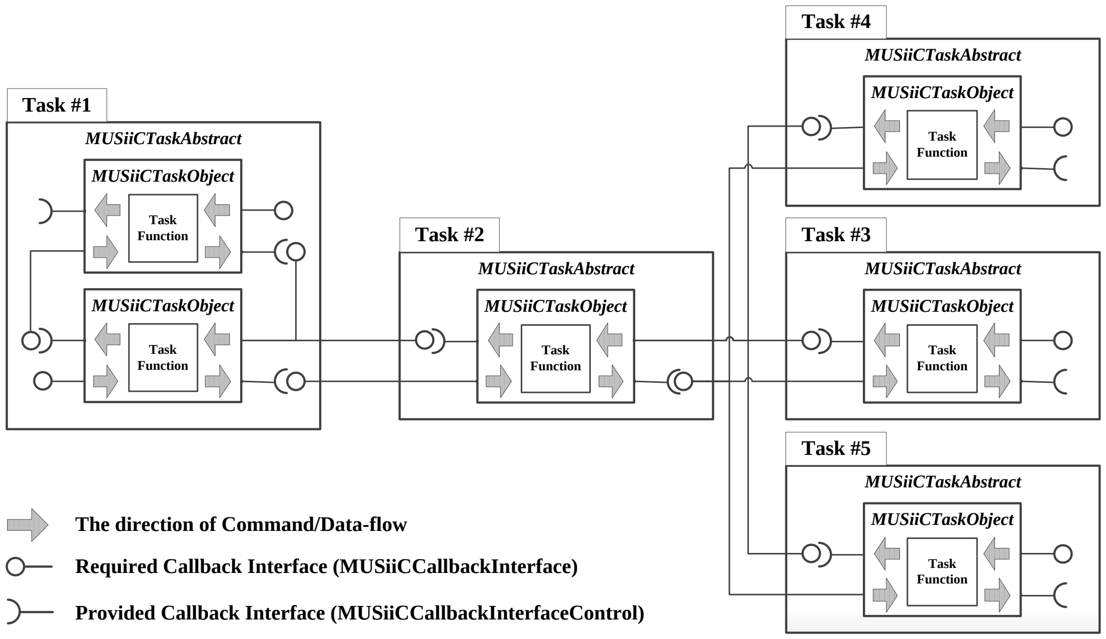
Task #1
Task #2
Task #3
Task #4
Task #5
MUSiiCTaskAbstract
MUSiiCTaskObject
Task Function
The direction of Command/Data-flow
Required Callback Interface (MUSiiCCallbackInterface)
Provided Callback Interface (MUSiiCCallbackInterfaceControl)

Figure 6. Network diagram callback interfaces between task classes based on *MUSiiCTaskAbstract.*

2.2.2 Bidirectional Communication Mechanism at application level.

Since US IGT subsystems are often connected at the application level, software frameworks for these systems must support bidirectional communication. With the increasing availability of Ethernet in IGT systems, TCP/IP is a common communication mechanism [3]. However, the TCP/IP socket provided by *OpenIGTLink2.0* only supports synchronous communication. There is a need to improve this method and the performance of TCP/IP data transmission.

We built custom TCP/IP network I/O classes (*MUSiiCTCPServer* and *MUSiiCTCPClient*) based on I/O completion ports [21] for Windows operating system. We intend to implement asynchronous TCP/IP network I/O classes using epoll [22] and Kqueue [23] for Linux and Mac OS X operating systems respectively in the near future. They provide an efficient threading model for handling multiple asynchronous I/O requests in a program and for supporting bidirectional communication between multiple clients at the application level. The block diagram for our network classes is shown in figure 7. Each of these classes has three instances of *MUSiiCTaskObject* to run independent tasks. The *MUSiiCTCPServer* class has Listening-Task, Data-Sending, and Data-Receiving *MUSiiCTaskObject* instances. The *MUSiiCTCPClient* class has Creating-client socket, Data-Sending, and Data-Receiving *MUSiiCTaskObject* instances. The task thread of Listening-Task in *MUSiiCTCPServer* provides multi-client connections and the Creating-client socket task thread in *MUSiiCTCPClient* can create multiple client sockets in a single *MUSiiCTCPClient* instance. The data communication of these classes is based on the *igtlMessageBase* data type, which is also the parent of all message data types in *OpenIGTLink2.0* [2, 4]. This allows our network classes to send or receive any kind of message type defined in *OpenIGTLink2.0.* As shown in figure 7, required (*MUSiiCCallbackInterface* or *MUSiiCVector*) and provided (*MUSiiCCallbackInterfaceControl* or *MUSiiCVectorSet*) interfaces allow us to send and receive data asynchronously with these network classes. In addition, executable modules with these network classes are capable of asynchronous bidirectional TCP/IP data communication with subsystems that use message types supported in OpenIGTLink2.0, such as 3D Slicer or other tracking devices [4, 24].

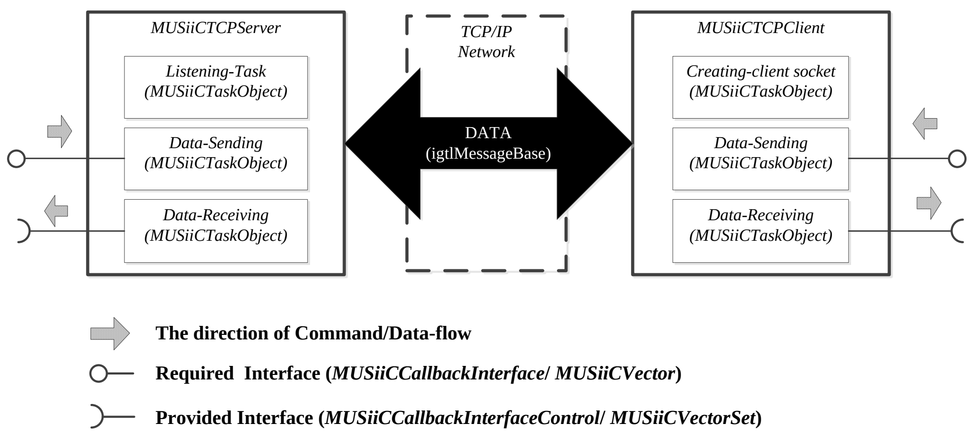
MUSiiCTCPServer
Listening-Task (MUSiiCTaskObject)
Data-Sending (MUSiiCTaskObject)
Data-Receiving (MUSiiCTaskObject)
TCP/IP Network
DATA (igtlMessageBase)
MUSiiCTCPClient
Creating-client socket (MUSiiCTaskObject)
Data-Sending (MUSiiCTaskObject)
Data-Receiving (MUSiiCTaskObject)
The direction of Command/Data-flow
Required Interface (MUSiiCCallbackInterface/ MUSiiCVector)
Provided Interface (MUSiiCCallbackInterfaceControl/ MUSiiCVectorSet)

Figure 7. Block diagram of *MUSiiCTCPServer* and *MUSiiCTCPClient.*

2.3 MUSiiC ToolKit 2.0: Real-time Multiple Data Synchronization

US IGT systems can be composed of many subsystems including a robotic system, tracking devices, and medical imaging systems. Real-time synchronization between multiple data sources is therefore necessary to integrate the US system with other IGT subsystems.

We built a specific task class, *MUSiiCSync*, to solve this requirement. Our synchronization method is closest data combining based on the timestamp of each data. The block diagram of *MUSiiCSync* is shown in figure 8(a). There are two task objects, Data-Collect and Data-Combine, in this class. The Data-Collect task thread receives multiple data from different data sources through the TCP/IP network and generates a *MUSiiCSyncData* instance designed to contain a reference data and multiple data of other types during the time of data collection, *Tc* (see figure 8(b)). Generally, *Tc* is two divided by the frame rate of the reference data in seconds. The Data-Combine task thread finds the data from each data group with the closest timestamp to the timestamp of the reference data. If *TrackingDataMessage* and *ImageMessage* data of *OpenIGTLink2.0* are present in the instance of *MUSiiCSyncData*, this task thread updates the tracking information of *ImageMessage* with the information from *TrackingDataMessage*. At this time, we can apply extra information such as calibration data from the US transducer.

There is a latency time (*Tl*) in *MUSiiCSync* to generate the first set of synchronized data. The set of synchronized data from *MUSiiCSync* will be delayed by at most *Tc*. In general, US data is the reference data in *MUSiiCSync*, and the frame rate is 30 frames per second. In this case, *Tc* will be 60 ms and *Tl* will be less than 60 ms. We feel that this delay is acceptable in US IGT systems. The set of synchronized data from *MUSiiCSync* can be sent to another task class through an instance of *MUSiiCCallbackInterface* or *MUSiiCVector*. In this case, the data can be saved to the local hard disk using the *MUSiiCFileIO* class or sent to the TCP/IP network with *MUSiiCTCPClient*.

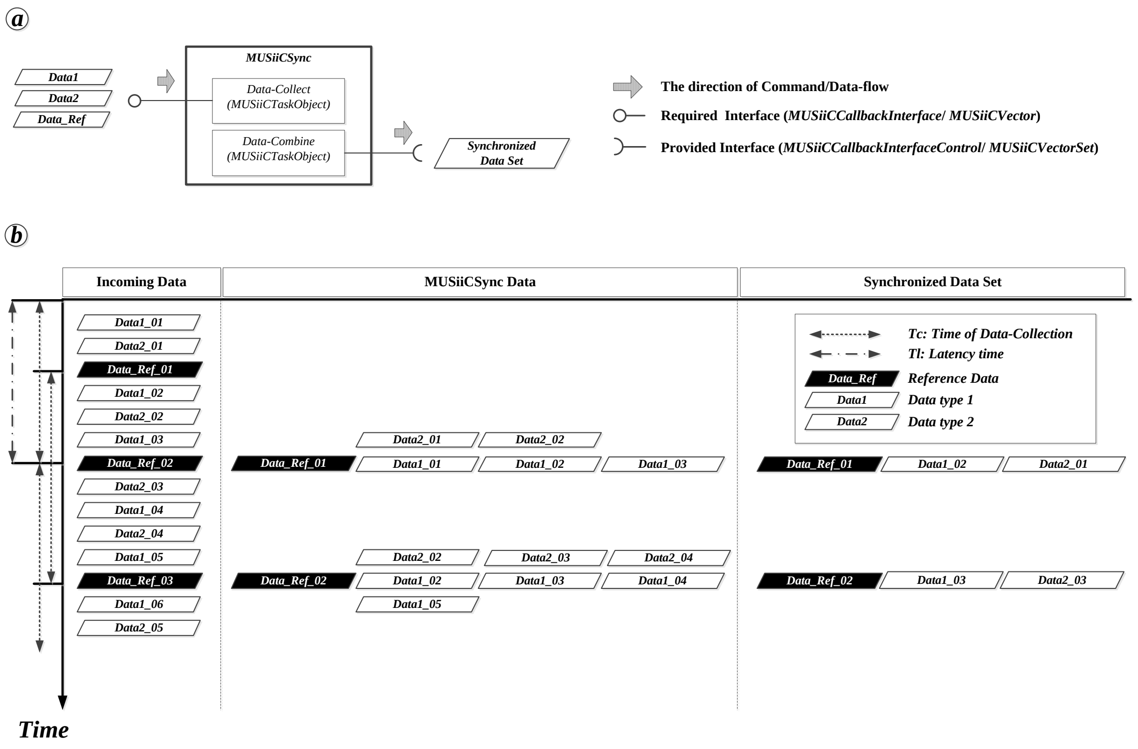

Figure 8. Block diagram of (a) *MUSiiCSync*, and (b) Timeline of *MUSiiCSync*.

3 Application for Advanced Ultrasound Research using MUSiiC ToolKit 2.0

We built an upgraded US thermal monitoring system using *MUSiiC ToolKit 2.0*. A block diagram of this system is presented in figure 9. There are two US elastography modules based on thermal strain imaging for monitoring the change of tissue temperature: *MUSiiC_NCC_Elastrography* [20] and *MUSiiC_TrUE_Elastography*. Since *MUSiiC_TrUE_Elastography* requires synchronized US RF data and US transducer tracking information to generate a tracked elastography [7], a *MUSiiCSync* module is added to the system.

The block diagram in figure 9 is very similar to the one shown in figure 1. Our previous software framework only supported unidirectional communication making it difficult to realize interactive communication between subsystems. In contrast, our new system based on *MUSiiC ToolKit2.0* provides bidirectional communication methods between software modules and task class levels. This communication mechanism allows us to change control parameters in *MUSiiC RF-Server 2.5*, *MUSiiC B-Mode* or *MUSiiC_-NCC_Elastography* from the *MUSiiC ImageViewer* and check the results immediately on a screen.

Instances of *MUSiiCTCPServer*, *MUSiiCTCPClient*, and *MUSiiCFileIO* are implemented in all software modules. This means that we can send out the results of each module to real-time US IGT system or save the data to the local hard disk.

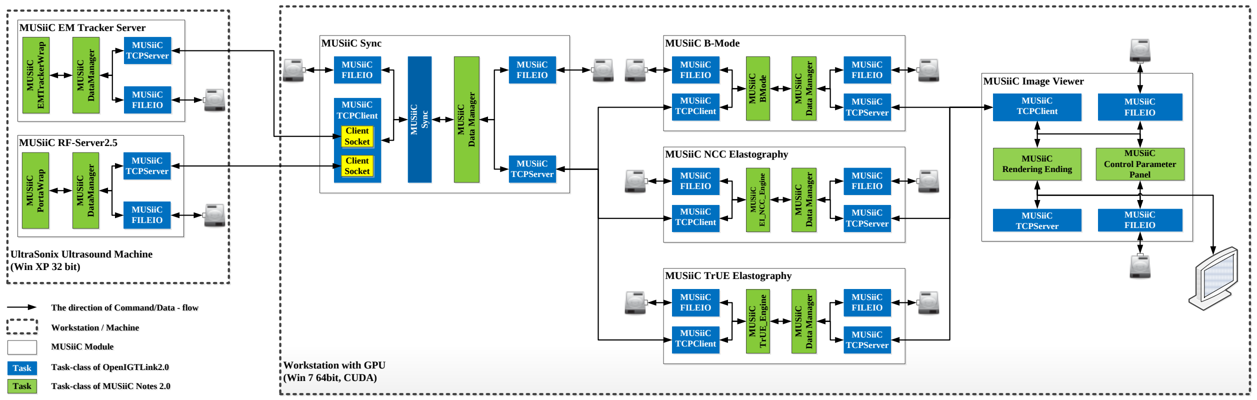

Figure 9. Block diagram of upgraded version of ultrasound thermal monitoring system.

4 Conclusions

In this work, we presented *MUSiiC ToolKit 2.0* as a software framework for advanced US IGT systems. We describe the infrastructure to enable three main features: real-time data processing, bidirectional data communication, and real-time data synchronization. We also present an upgraded application using this new software framework and describe its advantages over the previous version. Future work will focus on developing new modules in *MUSiiC Notes 2.0* and extending *MUSiiC ToolKit 2.0* to support applications such as photoacoustic imaging. In the future, we plan to release our software framework as open source software under the New BSD license [25].

Acknowledgements

The authors would like to thank Pezhman Foroughi and Nishikant Deshmukh from Johns Hopkins University for their cooperation. The authors also thank Dr. Russell H. Talyor of NSF ERC-CISST at the Johns Hopkins University for his valuable advice and discussion. This work was supported by NIH NCI SBIR funding R44CA134169.

References

[1] P. J. Stolka, H. J. Kang, and E. M. Boctor, "The MUSiiC Toolkit: Modular Real-Time Toolkit for Advanced Ultrasound Research," presented at the MICCAI 2010, International Workshop on System and Architectures for Computer Assisted Interventions, 2010.

[2] H. J. Kang, P. J. Stolka, and E. M. Boctor, "OpenITGLinkMUSiiC: A Standard Communications Protocol for Advanced Ultrasound Research," presented at the MICCAI 2011, International Workshop on System and Architectures for Computer Assisted Interventions, 2011.

[3] J. Tokuda, G. S. Fischer, X. Papademetris, Z. Yaniv, L. Ibanez, P. Cheng, H. Liu, J. Blevins, J. Arata, and A. J. Golby, "OpenIGTLink: an open network protocol for image-guided therapy environment," ***The International Journal of Medical Robotics and Computer Assisted Surgery,*** **vol. 5, pp. 423-434, 2009.**

[4] ***Open IGT Link Protocol (Ver. 2) , http://www.na-mic.org/Wiki/index.php/OpenIGTLink/ProtocolV2*****.**

[5] E. M. Boctor, P. Stolka, H. J. Kang, C. Clarke, C. Rucker, J. Croom, E. C. Burdette, and R. J. Webster III, "Precisely shaped acoustic ablation of tumors utilizing steerable needle and 3D ultrasound image guidance," in *SPIE Medical Imaging 2010*, San Diego, CA/USA, 2010.

[6] J. Boisvert, D. Gobbi, S. Vikal, R. Rohling, G. Fichtinger, and P. Abolmaesumi, "An open-source solution for interactive acquisition, processing and transfer of interventional ultrasound images," presented at the MICCAI 2008, International Workshop on System and Architectures for Computer Assisted Interventions, 2008.

[7] P. Foroughi, C. Csoma, H. Rivaz, G. Fichtinger, R. Zellars, G. Hager, and E. Boctor, "Multi-modality fusion of CT, 3D ultrasound, and tracked strain images for breast irradiation planning," in *SPIE Medical Imaging 2009*, Lake Buena Vista, FL/USA, 2009, p. 72651B.

[8] S. Billings, N. Deshmukh, H. J. Kang, R. Taylor, and E. M. Boctor, "System for robot-assisted real-time laparoscopic ultrasound elastography," in *SPIE Medical Imaging 2012*, San Diego, CA/USA, 2012, p. 83161W.

[9] H. Sen, N. Deshmukh, R. Goldman, P. Kazanzides, R. H. Taylor, E. Boctor, and N. Simaan, "Enabling technologies for natural orifice transluminal endoscopic surgery (NOTES) using robotically guided elasticity imaging," in *SPIE Medical Imaging 2012*, San Diego, CA/USA, 2012, p. 83161Y.

[10] E. M. Boctor, A. Viswanathan, S. Pieper, M. A. Choti, R. H. Taylor, R. Kikinis, and G. Fichtinger, "CISUS: an integrated 3D ultrasound system for IGT using a modular tracking API," 2004, p. 27.

[11] Z. Yaniv, P. Foroughi, H. J. Kang, and E. Boctor, "Ultrasound calibration framework for the image-guided surgery toolkit (IGSTK)," 2011, p. 79641N.

[12] H. J. Kang, N. P. Deshmukh, P. Stolka, E. C. Burdette, and E. M. Boctor, "Ultrasound imaging software framework for real-time monitoring of acoustic ablation therapy," in *SPIE Medical Imaging 2012*, San Diego, CA/USA, 2012, p. 83201E.

[13] W. R. Stevens, *UNIX Network Programming: Interprocess Communications* vol. Volume 2, 1998.

[14] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1994.

[15] *Qt SDK*, http://qt.nokia.com/products/qt-sdk/.

[16] *The Visualization ToolKit (VTK)*, www.vtk.org.

[17] *Ultrasonix Wiki*, http://www.ultrasonix.com/wikisonix/index.php/Main_Page.

[18] *Ascension Technology Copoeration.* http://www.ascension-tech.com/technical/index.php.

[19] H. J. Kang, N. Kuo, X. Guo, D. Song, J. U. Kang, and E. M. Boctor, "Software framework of a real-time pre-beamformed RF data acquisition of an ultrasound research scanner," in *SPIE Medical Imaging 2012*, San Diego, CA/USA, 2012, p. 83201F.

[20] N. Deshmukh, H. Rivaz, and E. Boctor, "GPU-based elasticity imaging algorithms," in *MICCAI-GRID 2009 - International Conference on Medical Image Computing and Computer Assisted Intervention*, London/UK, 2009.

[21] *I/O Completion Port*, http://msdn.microsoft.com/en-us/library/windows/desktop/aa365198(v=vs.85).aspx.

[22] ***epoll, http://en.wikipedia.org/wiki/Epoll.***

[23] ***Kqueue, http://en.wikipedia.org/wiki/Kqueue.***

[24] ***3D Slicer, http://www.slicer.org/.***

[25] ***New BSD license, http://opensource.org/licenses/bsd-license.php.***