Multimodal Graphical User Interface for Ultrasound Machine Control via da Vinci Surgeon Console: Design, Development, and Initial Evaluation

by

Samuel Radiant Tatasurya

Supervisor: Dr. Septimiu Salcudean

EECE 496: Undergraduate Thesis

Electrical and Computer Engineering

THE UNIVERSITY OF BRITISH COLUMBIA

Vancouver

August 2015

Abstract

The integration of hands-free modalities for controlling an ultrasound machine from the surgical robot console has the potential to assist the surgeon in manipulating the ultrasound image independently during robot-assisted minimally-invasive surgery (MIS). Hands-free control is especially useful when both of the surgeon's hands are occupied with the control of surgical tools and no assistant is present to help the surgeon operating the ultrasound machine. This thesis presents the design and development of a graphical user interface (GUI) for da Vinci classic system which enables its user to interact with the SonixTouch ultrasound machine directly from the surgeon console using eye gaze and voice commands. An initial user study was performed as a part of iterative design process in order to assess the GUI usability, and the task chosen for this purpose was an image optimization task. Although current iteration of the GUI resulted in relatively long completion time and high speech recognition error ratio, it demonstrated that integrating such technology in robot-assisted MIS is feasible.

Preface

This thesis is original, unpublished, independent work by the author, Samuel Radiant Tatasurya.

Table of Contents

List of Tables

List of Figures

Acknowledgements

First of all, the author would like to thank Dr. Septimiu Salcudean from the Department of Electrical and Computer Engineering at the University of British Columbia (UBC) for giving the author the opportunity to work on this project and providing technical guidance for the completion of this project. The author would also like to thank the graduate students from UBC Robotics and Control Laboratory, especially Irene Tong, Yasmin Halwani, Caitlin Schneider, and Angelica Ruszkowski, for their willingness to share their technical knowledge with the author. Finally, the author would like to extend his gratitude to Dr. Christopher Nguan from the Department of Urologic Sciences for giving the author the opportunity to participate as an observer in a robotic kidney surgery and providing invaluable advices and suggestions for the improvement of the technology under development.

Chapter 1

Introduction

1.1 Background

The superiority of eye gaze in revealing a person's intention and spatial focus has compelled many researchers to implement eye and gaze tracking in various applications [1]. As a first example, in marketing, the analysis of consumers' eye movement led to the understanding of effective advertisement placement [2]. As a second example, the use of gaze tracking in automotive field enabled researchers to build a system which was able to detect a driver's attention and raise a warning whenever the driver became inattentive [3]. The first example shows a passive implementation of gaze tracking, where the user's gaze is continuously recorded during the experiment and the gaze pattern analysis is performed at a later time. The second example is categorized as active gaze tracking, since the user's gaze was not only recorded during the experiment, but also used to generate a real-time response. The active use of gaze is made possible due to the fast movement of the gaze itself, which can precede voluntary actions by up to one second [4]. In the field of Human-Computer Interaction (HCI), this superior movement speed of the gaze ultimately makes gaze tracking a feasible input modality, complementing conventional hand-based modalities, such as mouse and keyboard.

The remote nature of gaze tracking opens a pathway for its introduction in surgical settings. Currently, a technological need exists for hands-free interaction with imaging technology to maintain sterile condition during surgery [5]. Imaging technology used intra-operatively can minimize the damage to the tissues surrounding the area of interest, thereby improving the patient's quality of life [6]. Gaze tracking has the potential to be the hands-free input modality that is able to answer the ever-increasing need mentioned above.

Aside from the need to maintain the sterile conditions in the operating room, other scenarios exist where hands-free interaction in surgical settings is required. Most of the time during the surgery, both of the surgeon's hands are occupied with surgical tools, but there are times when the surgeon needs the direct control to the imaging machine to use it intra-operatively. In a typical situation, an assistant will help the surgeon to operate the imaging machine until the desired image is obtained. However, this practice introduces potential communication problems between the surgeon and the assistant, which eventually can disrupt the procedure flow and prolong the procedure duration [7]. If the surgeon is provided with a hands-free modality such as gaze tracking to control the machine, this potential problem can be prevented.

1.2 Research Direction

The goal of this work is to develop a prototype of a graphical user interface (GUI) to be used in robot-assisted minimally-invasive surgery (MIS), which allows for hands-free operation of an ultrasound machine via operator's eye gaze and voice. This technology will facilitate the surgeon to manipulate some of the ultrasound machine parameters directly from the robot surgeon console.

Robot-assisted MIS offers the capability of open surgery in enabling the surgeon to control the surgical instruments precisely and intuitively, while at the same time leads to low morbidity, which is one of the advantages of MIS [8]. The robotic system, which is used for the design, development, and initial evaluation in this work, is the standard da Vinci Surgical System (Intuitive Surgical, Inc., Sunnyvale, CA).

The imaging technology that is chosen to be the focus of this work is ultrasound imaging. Ultrasound imaging offers fast, economical, real-time imaging which is suitable for interventional radiology [9]. This work focuses on the intra-operative application of ultrasonography instead of the purely diagnostic one. Therefore, the ultrasound machine features that will be integrated to the GUI are the ones which are identified as the commonly used features during the surgery. The ultrasound machine that is used for the development and initial evaluation in this work is SonixTouch (Ultrasonix Medical Corporation, Richmond, BC), due to the existence of the Ulterius Software Development Kit (SDK) as the framework for remote communication with the machine and the machine availability in the UBC Robotics and Control Laboratory (RCL).

The human-computer modalities used for interacting with the GUI are eye gaze and speech. Various research papers have explained the Midas Touch problem when only eye gaze is used as the input modality [10, 11, 12]. The Midas Touch problem is a situation where the gaze tracker cannot differentiate between the operator's gaze for normally looking around and for triggering an action, meaning that everywhere the operator looks, a command is triggered. One solution to this problem is to track the dwell time of the gaze. If the operator looks at a certain area longer than a certain period of time, it means that the operator intends to trigger an action and the system will issue that action. However, implementing only dwell-time method raises a trade-off problem. If the dwell time required to issue a command is too short, then the Midas Touch problem will practically still exist. On the other hand, if it takes a long time for the operator to look at a certain area and issue a command, it will defeat the purpose of using the eye gaze as a fast input modality. The addition of speech as the activation modality in this work is expected to overcome the problems outlined above.

1.3 Related Work

The ability of eye gaze to reveal a person's attention to a certain object right before the person performing a voluntary action to that object, led many researchers to explore its potential use as a menu selection tool. The work done by faculty and students at the

University of Virginia resulted in the realization of a device called Erica, which was able to accept the user's eye gaze as the input for its user interface [13]. This device aimed to help people with physical and vocal disability to regain some degree of independent control through its specially-designed menu selection feature, which responded to the user's gaze after the user stared at the desired option for a fixed amount of time. Therefore, Erica's selection method could be categorized as dwell-time method. The paper mentioned some limitations of the system, such as the need for the user to maintain the position of the head and the need to improve the accuracy and precision of the gaze tracker. Both limitations are common to gaze-based human-computer interaction, as pointed out by Robert Jacob [12].

In his observation, Robert Jacob explained another considerable limitation of unimodal, gaze-based menu interface: the Midas Touch problem, which has been discussed in details in the previous sub-section. He further explained that item execution using dwell-time method typically resulted in unnatural stare by the user and proposed the use of a button for the menu confirmation. This observation raised the possibility for the development of multimodal user interface.

Some researchers decided to implement multimodal interaction in their work and compare its performance with the unimodal one [14]. The input modalities they employed for their multimodal interaction are gaze and speech, whereas only gaze was utilized for their unimodal interaction. Furthermore, they designed their user interface to have standard-sized objects, as opposed to the large objects and spacings that are typically used to accommodate calibration errors and the accuracy of the gaze input. Their work reported that the mean error rate of the gaze-only interaction was 51.1% and the use of multimodal interaction greatly reduced the error rate down to 17.4%. Furthermore, the use multimodal interaction decreased the mean movement time by 24%. This work showed that multimodal implementation was superior compared to the unimodal, gaze-only implementation.

Another group of researchers also investigated the effectiveness of multimodal implementation for menu selection [15]. They explained the distribution of labour between the eye gaze and speech input, where the former was used as a pointing element and the latter acted as the selection element. Contrary to the findings in [14], they found that the multimodal implementation resulted in slower task completion time compared to the gaze-only implementation. Two possible causes to this slow task completion time were the existence of eye-voice span, which resulted in the temporal gap of around 500 milliseconds, and the latency from speech recognition software.

The work in [15] reported not only the significance of modality choice, but also the effectiveness of menu layout design. They compared three different menu layouts: linear, full-circle, and semi-circle. Semi-circle menu design was found to have the highest accuracy rate, fastest task completion time, and lowest cognitive load for the user. Therefore, carefully considered layout serves as a major part in the design and development of hands-free menu interface.

The application of hands-free menu selection in laparoscopic surgery became the focus of the work by Geoffrey Tien [16]. He decided to use eye gaze as the only input modality in his design. The menu selection in the final design was performed using the dwell-time method. The reason behind this unimodal approach was that the use of voice commands during the surgery might be ineffective due to the high level of ambient noise from different machines in the operating room. Moreover, the voice commands from the surgeon might be mistakenly heard by the rest of the surgical team as a regular communication. Regardless of these design decisions, he specified that further user studies in realistic operating room settings were needed.

Chapter 2

Design

This part of the paper will explain the design considerations and decisions made for the prototype of the hands-free multimodal GUI for controlling the ultrasound machine. The GUI design process follows iterative design methodology, in which a cycle consisting of prototyping, testing, and user evaluation is repeated in order to deliver an interface that has higher usability than the previous iteration [17]. Therefore, before the user interface is ready to be used in the actual surgical settings, it will require a couple of iterations to refine its features and further improve the overall user experience.

2.1 Gaze Tracking

The use of eye gaze as one of the input modalities in the GUI requires the operator to go through a calibration process. The process is used to map the operator's pupil-glint vectors to their respective pixel coordinates on the screen. The calibration method chosen for this purpose is nine-point calibration, in which the operator has to look at the nine points shown in Fig. 1 in succession. The entire calibration process takes less than a minute, since the operator needs to fixate on each point for 2.5 seconds and the transition time to allow the operator's gaze to move on to the next point is 3 seconds.

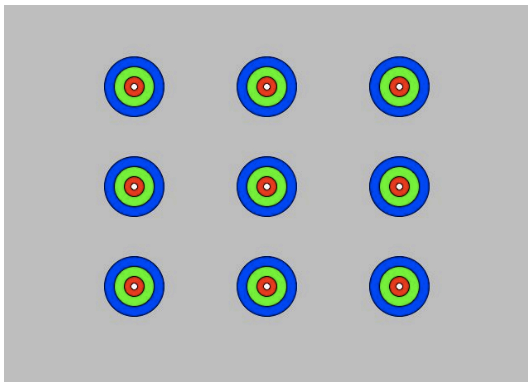


Figure 1. Nine-point calibration design

When using eye gaze as an input modality for the menu selection, two general approaches exist. The first approach is to use eye gaze as both pointing and selection elements. However, one limitation of eye gaze as an input modality is that, unlike touch-based input modalities such as mouse and touchscreen, eye gaze does not possess an intrinsic clicking method [10]. Therefore, dwell time and blinking are typically employed as clicking methods in this approach so that the eye gaze can be used for both normal vision and computer input.

The second approach is to use eye gaze only as a pointing element. The interface continuously tracks the operator's eye gaze and checks if the gaze points to a certain menu elements, but the gaze does not have any ability to make the selection by itself. This approach is more natural compared to the first approach, since in the first approach the perceptual channel is overloaded with a motor control task [18].

The current iteration of the GUI utilizes the combination of two approaches mentioned above. Generally, the eye gaze will be only used as a pointing element. However, on some menu elements that require the operator to perform repeated tasks, switching between eye gaze and another input modality to make a selection is cumbersome. Therefore, the eye gaze will also be able to make a selection using the dwell-time method on these menu elements.

Other than those two approaches, the GUI is also able to detect if the operator's gaze is on the edge of the screen in order to change the display. This method is known as gaze contingency, in which the screen functionalities are changing depending on where the operator is looking. This feature is added in the GUI as an alternative to the speech recognition method for switching between displays, in the case that the high level of ambient noise greatly affects the performance of the speech recognition. The implementation for the gaze contingency is explained in details in Chapter 3.

2.2 Speech Recognition

Since eye gaze is only used as a pointing element, another modality needs to act as the selection element. Speech recognition is chosen to complement the eye gaze as a menu selection input modality due to its hands-free nature and its ability to emulate the human action in requesting an object. The natural way for a person to request an object which is out of his/her reach to another person is by pointing to that object and expressing the request to pass that object verbally. In this GUI, the menu selection is designed to replicate those actions, with the eye gaze accommodating the pointing action and the speech recognition relaying the operator's request to the computer system. This parallel representation with human skills is important for the development of GUI since it directly affects the predictability and controllability of the system [19].

Despite the superiority of speech recognition mentioned previously, the high level of ambient noise from different machines in the operating room may reduce the effectiveness of speech recognition usage in surgical settings. Therefore, the use of speech recognition in the current iteration of GUI is kept to a minimum, both to attenuate the problem above and minimize the cognitive load of the operator.

2.3 Graphical User Interface

Designing an intuitive GUI requires the author to give careful thought and find the right balance of many different factors, starting from the menu layout to low-level details, such as button dimensions and button effects. One important constraint with any GUI is the screen dimensions, which in this work are 640 x 480 pixels. Regarding this constraint, Ben Shneiderman from the University of Maryland acknowledged in his paper that "screen space is a scarce resource" [20].

In this GUI, the menu for the ultrasound machine control occupies the same screen space as the video stream, essentially acts as the overlay for the video stream. Therefore, the menu layout chosen has to enable the operator to get the idea of what is happening on the video stream. One way to achieve this outcome is by grouping the menu items into comprehensive clusters on the screen [20]. Aside from the fact that grouping the menu items can lead to greater amount of free screen space, it also leads to more efficient search behaviour by the operator [1]. The reason behind this effect is that the users inherently expect buttons which share common functionalities to be grouped, and meeting this expectation in the GUI may result in shorter scanpaths and smaller search areas. Ultimately, the short scanpaths and small search areas are desirable in a system where eye gaze is utilized as one of the input modalities.

The grouping of the menu items is in line with Fitts' Law, which is used in HCI to measure the performance of target selection task:

$$MT = a + b \cdot ID = a + b \cdot log_2\left(\frac{2D}{W}\right)$$

where *MT* is the movement time, *ID* is the index of difficulty, *D* is the distance to the target, W is the width of the target, and a and b are the model constants [21]. The shorter the movement time, the more desirable the system for performing the target selection task. Hence, grouping the menu items essentially reduces the distance between targets, which in turn lowers the index of difficulty and ultimately the movement time.

However, minimizing the index of difficulty in this GUI proves to be a challenge, since there is a limit on how close the menu items can be placed next to each other before the limited accuracy of the gaze tracker makes it difficult for the operator to select a particular item on the screen. The mean error of the gaze tracker used in this work is 0.88 degrees of viewing angle, and its standard deviation is 0.69 degrees [22]. With the da Vinci screen size of 14 inches, the screen resolution of 640 x 480 pixels, and the assumption that the distance from the operator's eyes to the screen is 18 inches [23], the pixel accuracy can be calculated. Based on the result from the calculation of mean pixel error in Appendix A, the minimum gap surrounding each menu item is 20 pixels.

One way to compensate for the limitation explained in the previous paragraph is by increasing the dimensions of the menu items, thereby effectively increasing the width of the target in the Fitts' Law mentioned previously. Moreover, large on-screen objects, which are typically more than one degree of viewing angle, can also compensate for the relatively low accuracy of gaze-based pointer and the calibration errors [24]. Therefore, the dimensions for a menu item are chosen to be 140 x 140 pixels. The button concept in the current iteration of GUI can be seen in Fig. 2 below.

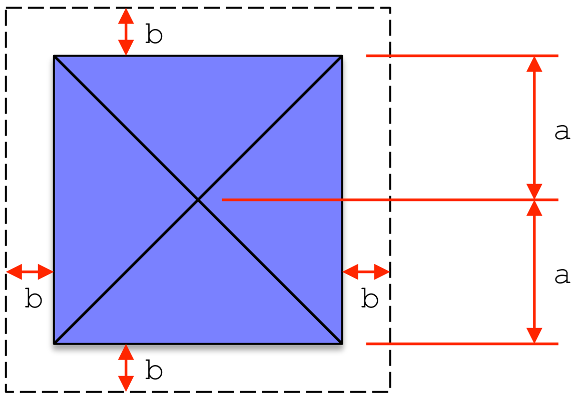


Figure 2. Button dimensions and surrounding gap (a = 70 pixels, b = 20 pixels)

Aside from finding the right balance between the button dimensions and the gaze tracker accuracy, the addition of visual feedback in the GUI can improve the overall user experience [15, 16]. Therefore, in this GUI two forms of visual feedback are added to the buttons: button highlighting and button press animation, as seen in Fig. 3. When the eye gaze of the operator hovers on top of a button, the button will change its colour, indicating that the operator is able to interact with that button. Moreover, when the operator makes a selection to a button, the button will appear to be sinking from its initial raised state, indicating that the operator has performed an action to that button. These visual feedbacks keep the operator informed about what is happening in the GUI and indirectly give the operator a sense of control [25].

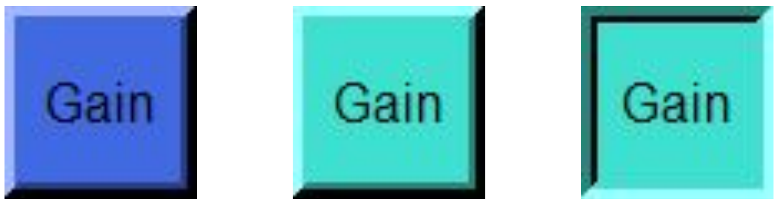


Figure 3. Three states of a button (from left to right): default, highlighted, and pressed

Chapter 3

Implementation

This part of the paper will focus on the actual implementation of the design explained in Chapter 2. The outcome of this implementation is the prototype of the GUI to be used in da Vinci Surgical System for manipulating the parameters of SonixTouch using the operator's eye gaze and voice.

3.1 Human-Computer Modalities

3.1.1 Eye Gaze

The gaze tracking hardware used in this work, as seen in Fig. 4, is the gaze tracker developed in UBC RCL by Irene Tong, a Master's student supervised by Dr. Salcudean. It consists of four major parts: 1) two OV5640 USB cameras, 2) two infrared (IR) bandpass filters installed on top of the cameras to allow only the reflection from IR light to be captured by the cameras, 3) two IR LEDs with the typical peak emission wavelength of 890 nm, and 4) a 3D-printed eyepiece to mount the gaze tracker on the surgeon viewing console.

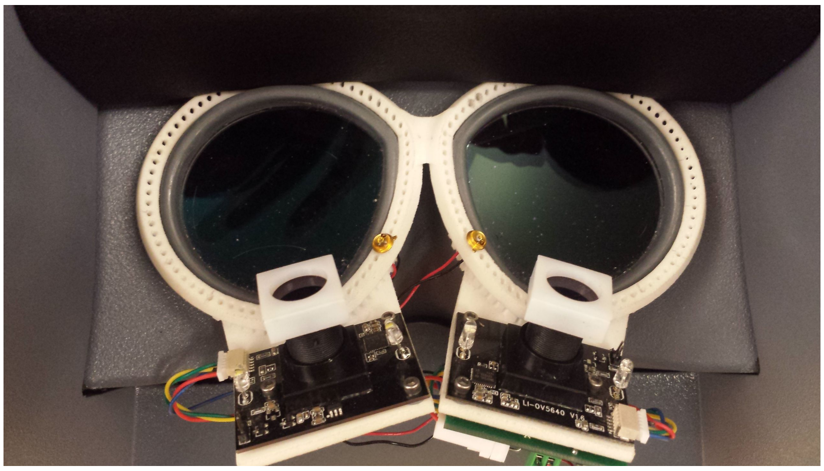


Figure 4. UBC RCL gaze tracker

The gaze tracking software, which was initially developed in Python by Irene Tong and then slightly modified by the author to fit the goal of this work, continuously tracks the operator's point of gaze based on the video frames captured by the gaze tracking cameras. After the operator completes the calibration explained in Section 2.1, a pair of pixel coordinates is mapped to the GUI, where each coordinate represents the gaze from each eye. The arithmetic mean of the two pixel coordinates is used as the absolute point of gaze of the operator in the GUI. This absolute point of gaze coordinate is used to check if the operator fixes his/her gaze on a particular menu element.

The GUI also checks whether the operator's point of gaze is on the edge of the screen or not. If the operator's point of gaze is on the edge of the screen for a certain period of time, the display will change accordingly. For example, if the GUI is currently showing the surgical scene and the operator is focusing his/her gaze on the right edge of the screen for 800 milliseconds, the GUI is switching to display the ultrasound image instead. On the other hand, if the GUI is currently showing the ultrasound image and the operator is looking at the left edge of the screen for 800 milliseconds, the GUI is switching back to display the surgical scene. The sample implementation of this feature can be seen in Fig. 5. Therefore, as an input modality, the operator's eye gaze not only acts as a cursor to select a menu element, but also follows the gaze contingency paradigm in order to change the screen functionalities depending on where the operator is looking.

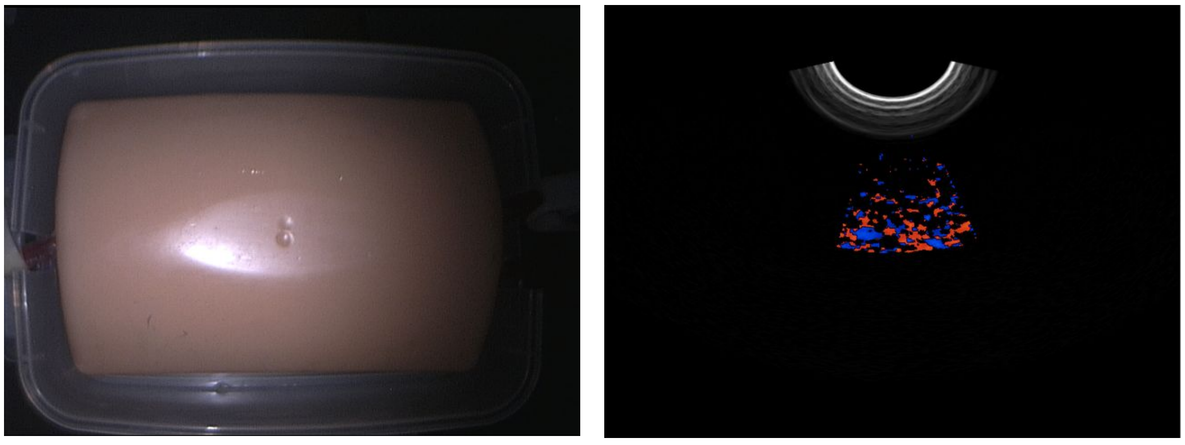


Figure 5. Switching between surgical scene (left) and ultrasound image (right)

3.1.2 Speech Input

During the initial development of the speech recognition for the GUI, Google Speech Recognition was chosen as the speech recognition application programming interface (API) and engine. The API was implemented in Python through the use of the *SpeechRecognition* package developed by Anthony Zhang. Despite the fact that the engine was able to interpret the speech input accurately, the computer needed to be constantly connected to the Internet during the operation and it had to wait for a few seconds to get the reply from the server. Therefore, this type of implementation is not suitable for the purpose of this work, since fast response time from the system is essential in surgical settings.

The current iteration of the GUI utilizes Microsoft Speech for its speech recognition engine. The API is also implemented in Python, using the *speech* package developed by Michael Gundlach. However, the original source file uses the Shared recognition engine, where multiple applications are allowed to use the engine's resources concurrently. This type of engine occasionally causes unwanted responses from the computer, such as running another application or performing Microsoft Windows native commands. Therefore, the author has modified the original source file to utilize the InProc recognition engine instead, where the speech recognition's resources are dedicated only to the GUI application. The details of the changes can be found in Appendix B.

For the GUI, the speech recognition runs on a background thread. When a certain word or phrase is heard, the speech recognition thread will notify the main thread and pass the recognized word or phrase to it. Then, the main thread will run a particular function depending on which word or phrase gets passed on. The words chosen have to be distinguishable from each other in order to avoid any incorrect function being called by the operator. The details of the speech recognition implementation in the GUI can be seen in Appendix C.

The current iteration of the GUI recognizes six basic commands: 1) *display*, to show or hide the menu buttons on the screen, 2) *switch*, to toggle the display between surgical screen and ultrasound image, 3) *select*, to select the menu button pointed by the operator's gaze, 4) *stop*, to take the cursor focus away from any button, 5) *voice only*, to activate voice only mode that sets speech recognition as the only input modality and temporarily disables the gaze tracking pointing capability, and 6) *use gaze*, to give the pointing capability back to the gaze tracking system. Aside from these basic commands, when the voice only mode is activated, each button has an inherent voice command that will move the cursor to that particular button if it is recognized by the GUI. The voice only mode is added to the GUI as a countermeasure if the gaze tracking system fails during the operation. During this mode, the voice command for each button is displayed on the button itself, both for providing the operator with a visual feedback and helping the operator carrying out the commands. The comparison between normal operation mode and voice only mode can be seen in Fig. 6.

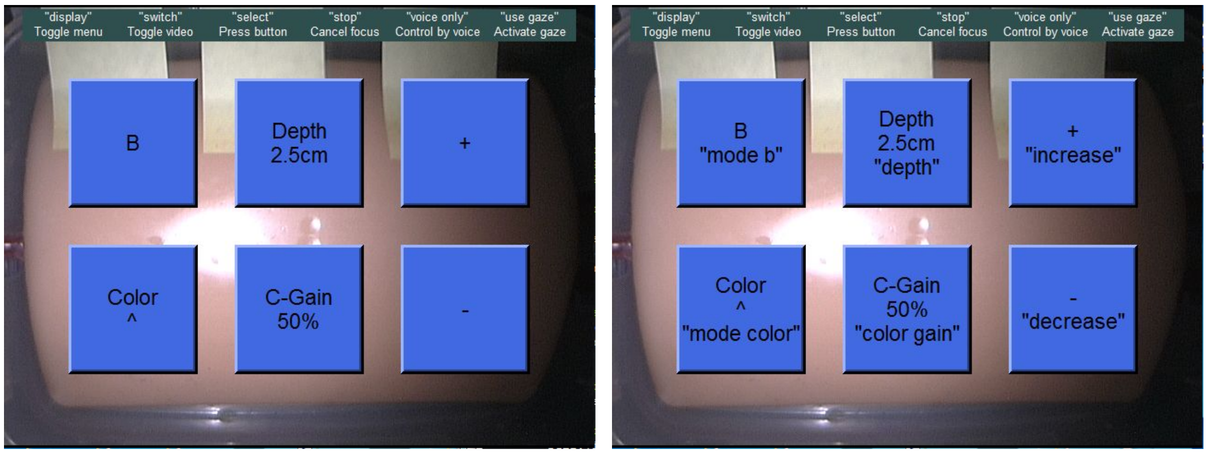


Figure 6. Display comparison between normal operation (left) and voice only mode (right)

3.2 Graphical User Interface

The GUI for this work was developed in Python using the *Tkinter* package. It consists of multiple layers, as seen in Fig. 7. The first layer, Toplevel Window, is the GUI application window and contains the rest of the layers. In this work, it is set to have a fixed width of 640 pixels and a fixed height of 480 pixels, in order to match the screen size of the surgeon console display. The second layer, Base Canvas, acts literally as a canvas that holds the objects generated in third, fourth, and fifth layers. The third layer, Video Layer, contains the live video stream either from the stereo camera or the ultrasound machine. The image frames are captured continuously on a separate thread and passed into a First In, First Out (FIFO) queue. Then, the main thread of the GUI obtains these frames from the queue and displays them on Video Layer. The fourth layer contains buttons to interact with the ultrasound machine. Finally, the fifth layer, Help Layer, continuously displays multiple help labels, where each label contains a keyword for basic voice commands along with the short description of its functionality. This last layer aims to reduce the cognitive load of the operator by letting the operator to use the GUI without having to memorize all of the voice commands beforehand.

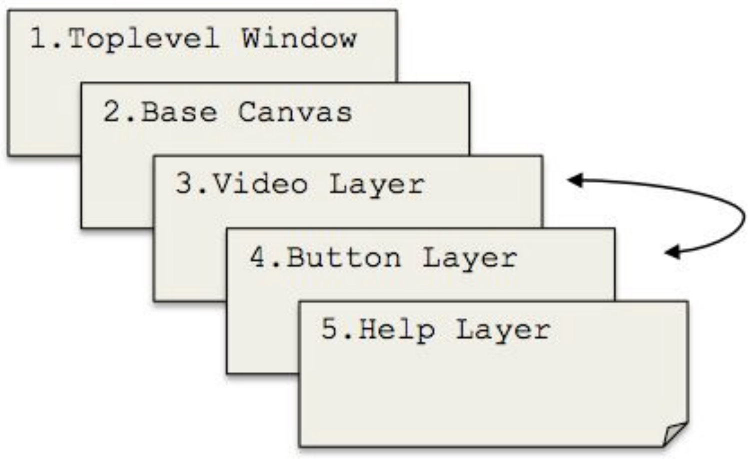


Figure 7. GUI layer organization

As mentioned briefly in Section 3.1.2, the GUI has the ability to show or hide the menu buttons. Technically, this feature is achieved by switching the display order of Video Layer and Button Layer, as illustrated in Fig. 7. When the menu buttons are shown to the operator, the Video Layer is placed on the third layer and the Button Layer is placed on the fourth layer. However, when the menu buttons need to be hidden from the operator, the

Video Layer is placed on the fourth layer and the Button Layer is placed on the third layer. In the current iteration of GUI, the operator can perform this switching mechanism either through the speech recognition or the gaze contingency method. The speech recognition method is carried out by passing the command *display* to the speech recognition engine, whereas the gaze contingency method is carried out by looking at the upper edge of the screen for 800 milliseconds to display the menu buttons and the lower edge of the screen for the same period of time to hide the menu buttons. The sample implementation of this feature can be seen in Fig. 8.

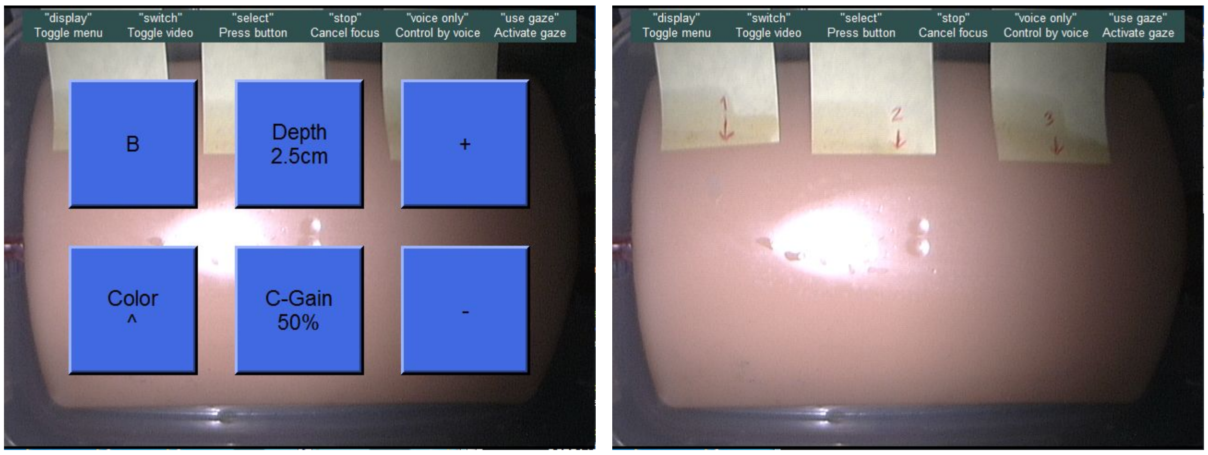


Figure 8. Menu buttons can be shown (left) or hidden (right) from the operator

The framework for the GUI provides four different button types related to the ultrasound machine control: *active*, *passive*, *latched*, and *mode*. Any active command sent to the ultrasound machine, which is not dependent on other parameters or not directly affecting other parameters, is going to take the form of an *active* button. The sample implementations of this button type are saving the screen image and toggling the freeze state of the ultrasound machine. The *passive* button type is used for generating a button that passes a parameter identifier to the GUI, which can be incremented or decremented later by the operator. The parameter passed to this type of button has to be an imaging parameter that holds an integer value, such as B-mode depth, B-mode gain, and Color Doppler gain. Moreover, only one parameter can be passed to the GUI in one moment.

The third button type, the *latched* button, is used to pass a command that depends on the parameter passed by the *passive* button; hence its name *latched* button since it depends on a latched value in order for it to be functional. Buttons that are used to increment and decrement a parameter value take the form of *latched* buttons. Each of these buttons also has a dwell-time attribute, so that the operator's eye gaze is able to select the button by focusing his/her gaze for a fixed period of time. Therefore, the operator does not have to repeatedly say *select* if he/she wants to increment or decrement a parameter value multiple times. This dwell-time attribute is dynamic, meaning that the time required for the button activation decreases every time the button has been successfully triggered. For example, the first time the operator focuses his/her gaze on a *latched* button, the time needed to activate the button is 1 second. Then, if the operator decides to keep staring at that button, the next duration before the button activation will be 0.5 seconds. This halving pattern will continue until the operator's gaze is no longer focusing on that button. However, a lower limit of 0.2 seconds is currently set to avoid the parameter value being incremented or decremented too fast.

The last button type, the *mode* button, acts as a template to generate buttons which are used to change the ultrasound imaging mode. It also has the ability to show and hide certain *passive* buttons, depending on which imaging mode is chosen. For example, selecting B-mode *mode* button shows the *passive* buttons for B-mode depth and B-mode gain and hides any other *passive* buttons.

The decision to show only the *passive* buttons which are directly related to the selected imaging mode not only maintains as much empty space as possible on the screen, but also provides the operator with the right level of control. The latter concept is known as progressive disclosure, which aims to help novice operators familiarize with the interface without being overwhelmed by the amount of features displayed on the screen, while at the same time provides advanced operators with greater degree of control [25]. The menu hierarchy in the current iteration of GUI can be found in Fig. 9.

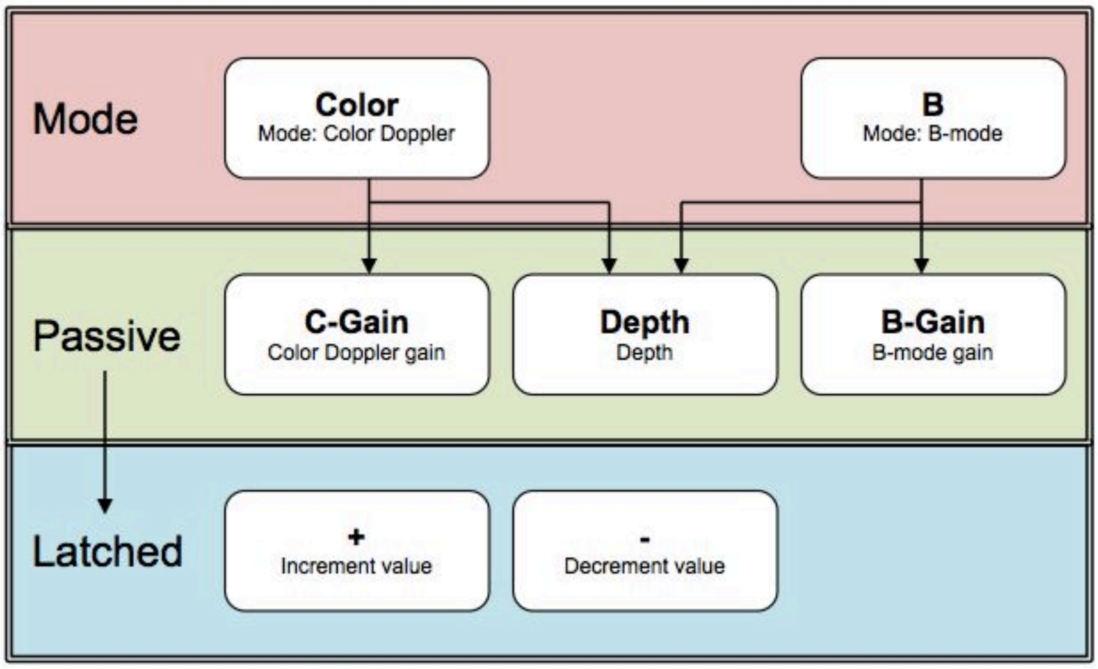


Figure 9. GUI menu hierarchy

Each button on the GUI has a feature called snap-to-center. Whenever the operator's point of gaze enters the effective area of a button, the gaze coordinate that is passed to the GUI is automatically converted from the absolute point of gaze coordinate to the centroid of that button. This feature can theoretically compensate for the relatively low accuracy of gaze-based pointer, thereby helping the operator in selecting a button on the screen.

3.3 Ultrasound Machine Communication

The Ulterius SDK provides an external computer system the capability to control an Ultrasonix ultrasound machine remotely via a TCP/IP connection. Once the computer and the ultrasound machine are connected, the computer can perform various actions to the machine, such as obtaining different parameter values, setting a parameter to a new value, changing the ultrasound image acquisition mode, and freezing the image frame. Further details of Ulterius capabilities can be found in its SDK documentation.

The Ulterius SDK was developed for use in C or C++ programming languages. However, this work uses Python as the main development language. Therefore, a wrapper is needed to implement the SDK in the GUI. The author decided to generate a Python wrapper for the SDK using Simplified Wrapper and Interface Generator (SWIG), called *pyUlterius*. In order to generate the wrapper, a SWIG interface file was needed. Then, this interface file was compiled together with the required dynamic-link libraries (DLLs) and header files, resulting in a Python module, pyUlterius.py, and a Python DLL, _pyUlterius.pyd.

After the Python wrapper module and DLL have been generated, a path configuration file is needed to allow the module to find the corresponding DLL. Therefore, pyUlterius.pth was generated to link the module and the DLL. The system setup to enable a Python program to use *pyUlterius*, along with the sample content of this path configuration file, can be found in Appendix D.

The wrapper, *pyUlterius*, was able to perform Ulterius functionalities needed for this work. However, the function call of some member functions from the original SDK had to be modified in order to comply with the Python underlying mechanism. Moreover, a helper class called *IntByRef* and a helper function called *getImageData* were created.

Class *IntByRef* was created to facilitate one of the function calls to *getParamValue* in *ulterius* class that required a reference to an integer variable. It has one member variable called val, which stores an integer value, and three member functions: 1) get( ), 2) set(num), where num is a Python integer value, and 3) access_ref( ). The function get( ) is used to obtain the value of val, whereas set(num) is used to set the value of val to num. Finally, access_ref( ) is used to get the reference of val.

The helper function *getImageData* in *ulterius* class converts an ultrasound image data type from PyCapsule to bytearray. This data in the form of bytearray can later be passed to *frombuffer* method of Python Imaging Library (PIL) *Image* class to form a bitmap image. The code in Fig. 10 is the sample implementation of these functions.

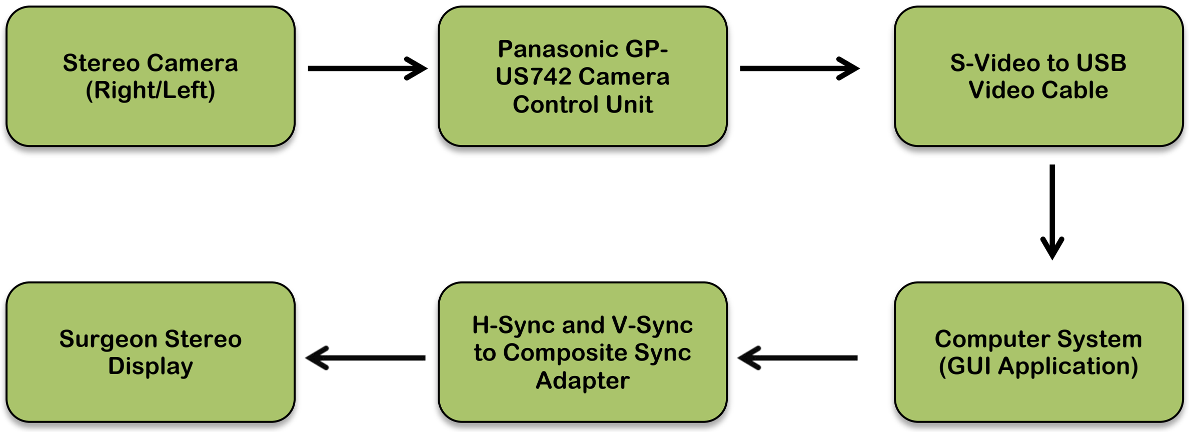
```
ultrasound = pyUlterius.ulterius()
temp = None

try:
    # data: the image data in the form of PyCapsule
    # size: the length of the image data, specifying the number of pixels
    temp = ultrasound.getImageData(data, size)
except TypeError as e:
    print 'Error', e

if temp is not None:
    # width: the width of the ultrasound image, in pixels
    # height: the height of the ultrasound image, in pixels
    img = Image.frombuffer("L", (width, height), temp)
    img.save("my_image.bmp")
```

Figure 10. Sample image data conversion

The list of member functions of the *ulterius* class that are specific to *pyUlterius*, along with their implementation details, can be found in Appendix E. Some functions that were not used in this work have not been fully tested, as seen in Appendix F. Therefore, *pyUlterius* is currently open for further testing and development in order for it to become a robust and complete wrapper for the Ulterius SDK.

3.4 Surgeon Console Display

The surgeon console display consists of right and left cathode ray tube (CRT) screens, and in the default setting, each of the screens shows different video streams from the right and left cameras of the stereo camera. The position offset between the right and left cameras gives offset images on the display, thereby providing depth perception to the surgeon. In order to display the surgical scene along with the menu overlay on the surgeon console display, the video streams from the stereo camera need to be fed into the Python GUI application and then this application needs to be displayed on the stereo display. The full implementation of these settings can be seen on Fig. 11.



Stereo Camera (Right/Left)
Panasonic GP-US742 Camera Control Unit
S-Video to USB Video Cable
Computer System (GUI Application)
H-Sync and V-Sync to Composite Sync Adapter
Surgeon Stereo Display

Figure 11. Video implementation diagram

One thing to be noted that in this prototype, only one of the video streams (either from right or left camera) is fed to the application, and the resulting display on one of the CRT screens is also used in the other screen. Therefore, the sense of depth normally found on the surgeon display is not present in this current prototype. In the future iteration of the GUI, the addition of another S-Video to USB video cable, along with slight modification to the GUI application will make it possible to show different video streams on the stereo display, thereby recovering the depth factor of the display. The display output, which contains both the video stream from one of the camera and the menu overlay, can be seen in Fig. 12.

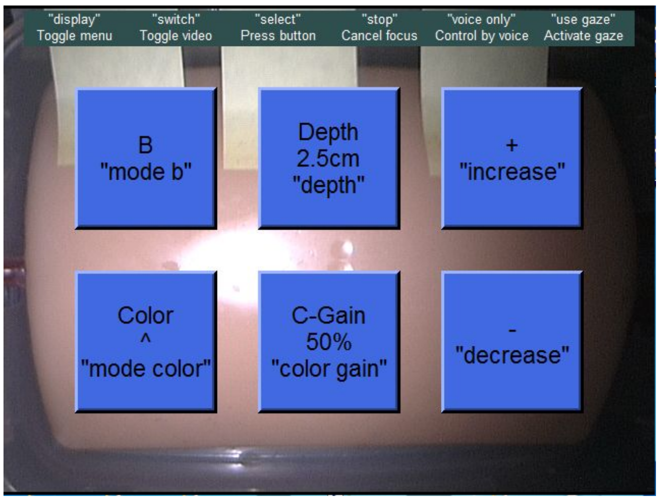


Figure 12. Non-stereo display output

Chapter 4

Evaluation

4.1 Setup

The gaze tracking and the GUI software were run on the same computer. The computer specifications can be seen in Table 1. The gaze tracker used in this study was the UBC RCL gaze tracker explained in Section 3.1.1. The microphone used to receive voice input from the operator was a USB microphone manufactured by Dynex, DX-USBMIC13. Lastly, the S-Video to USB video cable that was used to feed the GUI with the video stream from the surgical scene was StarTech.com SVID2USB2NS. The hardware setup on the surgeon console can be seen in Fig. 13.

Operating System	Windows 7 Professional 64-bit, Service Pack 1
Processor	AMD FX-8350, 8-Core, 4.00 GHz
RAM	16.0 GB

Table 1. Computer specifications for the user study

The GUI communicated with Sonix RP version 6.1.2 on SonixTouch via a TCP/IP connection established within UBC RCL. This communication included ultrasound image transfer from Sonix RP to the GUI, and ultrasound parameter acquisition (imaging mode, depth value, gain values) and manipulation (increment and decrement values) by the GUI.

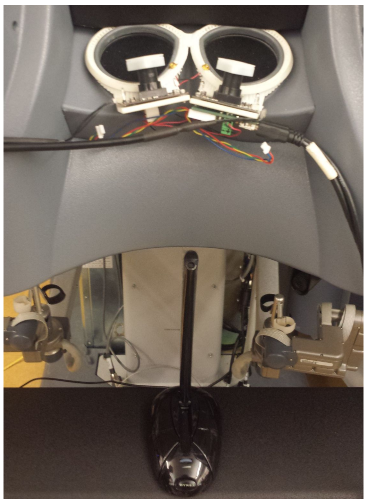


Figure 13. Hardware setup on surgeon console

For the user study, the "pick-up" ultrasound probe developed by Caitlin Schneider was used to generate the ultrasound image on SonixTouch [26]. The object of the study was a vessel flow phantom (Blue Phantom, Redmond, WA) placed on the patient-side cart. The phantom had a single vessel on one end, vessel bifurcation around its middle part, and two vessels on the other end. A mini-pump variable flow from Fisher Scientific controlled the liquid flow inside the vessel. In this study, the pump was set to operate in forward direction, fast setting, and the speed of 5. The photographs of the probe and the patient-side setup can be seen in Fig. 14.

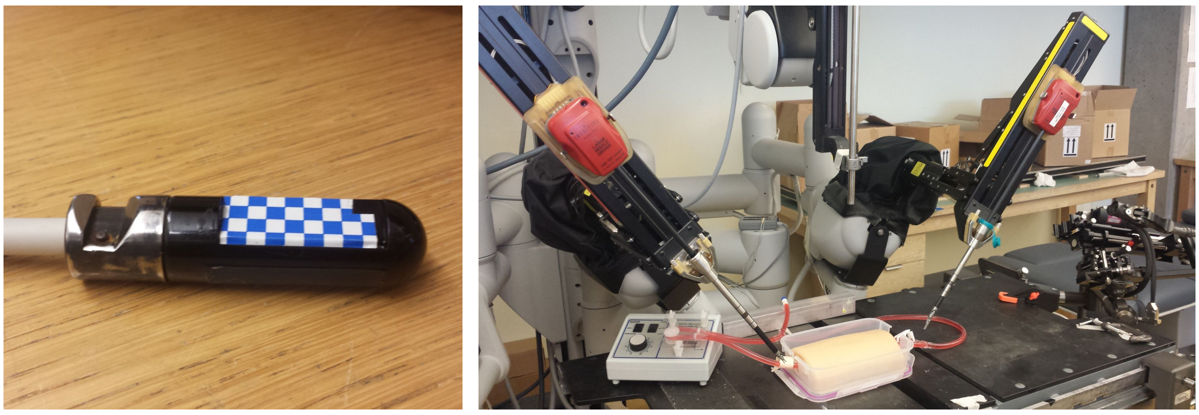


Figure 14. Ultrasound probe (left) and patient-side setup (right) for the user study

4.2 User Study

The image optimization task was chosen for the user study. Each operator had to reproduce three Color Doppler images: 1) the single vessel, 2) the vessel bifurcation along with the bottom of the phantom, and 3) two vessels. The images had to be similar in quality as the images used as the standard of this study, as seen in Fig. 15. In order to achieve that goal, the operator had to change the values of the depth, B-mode gain, and Color Doppler gain using the GUI developed in this work. The operator was allowed to use any combination of eye gaze control and voice commands provided by the GUI. Before the study, each operator was introduced to the GUI and its features, and also the ultrasound machine controls that were related to this study. The operator was also given some time to familiarize with the overall control of the system by trying out the GUI and the da Vinci teleoperation mode directly.

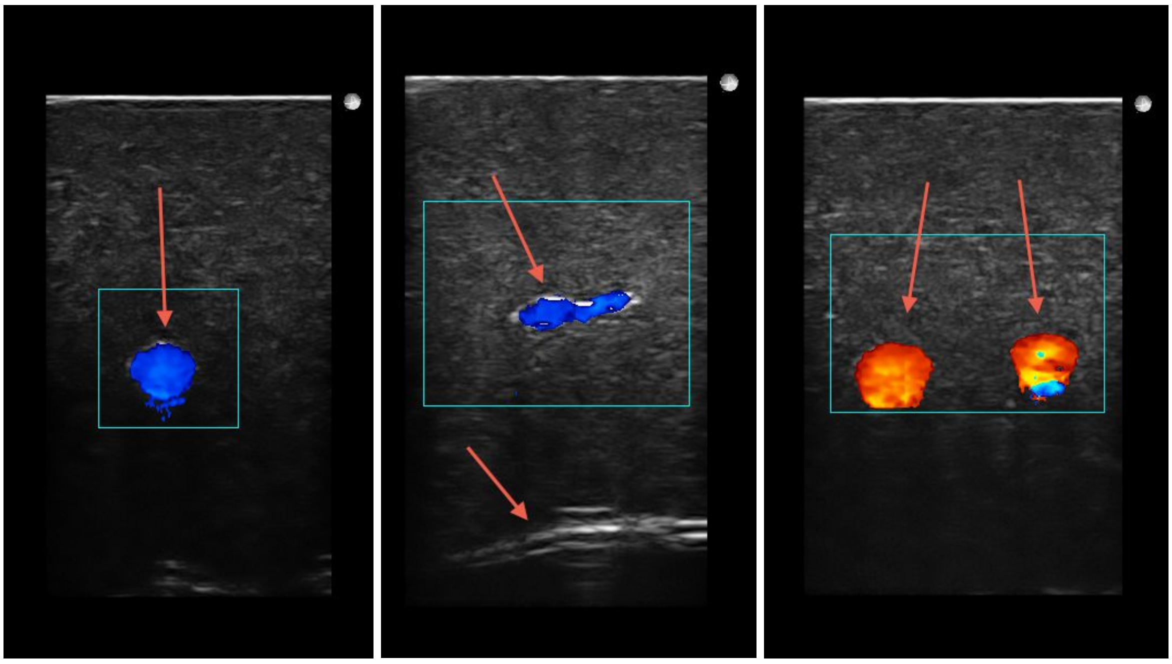


Figure 15. Standard images for the user study

The task was split into three sessions, one for each image. Before each session, the operator had to go through a calibration process for the eye gaze and the ultrasound parameter values were set back to an initial condition of Depth = 2.5cm, B-mode gain = 50%, and Color Doppler gain = 50%. Then, using the da Vinci teleoperation mode, the operator placed the probe according to the indicators on the phantom, which corresponded to the session number, as seen in Fig. 16. The probe placement was considered as a pre-session setup and not included in the actual session to nullify the effect of varying proficiency in using the da Vinci robot between subjects in the results. Once these pre-session setups had been completed, an observer signalled the operator to start the ultrasound image manipulation. The observer continually monitored the ultrasound image quality displayed on SonixTouch and verbally guided the operator if the operator was not sure what he/she would do next. For example, if the ultrasound image was too dark and needed more contrast, the observer would ask the operator to increase the B-mode gain. Once the desired image had been achieved, the observer would record the time required for the operator to complete the session. Throughout the session, another observer also counted the number of speech errors. The details of the speech errors will be explained in Section 4.3.

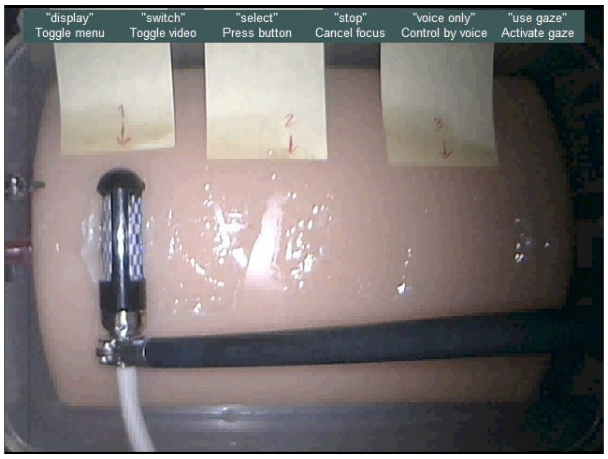


Figure 16. Probe placement for the first session

After completing the task using the GUI, each subject was asked to perform the same task directly using the ultrasound machine, while still sitting in front of the surgeon console and placing the probe using the console. This scenario was not a realistic approach to the ultrasound machine control performed in actual surgery, as in typical surgical settings, an operator other than the surgeon controlling the da Vinci robot would be in charge of the ultrasound machine. Same as the task procedure using the GUI, before each session the ultrasound parameter values were also set back to the initial condition of Depth = 2.5cm, B-mode gain = 50%, and Color Doppler gain = 50%. The placement of the ultrasound machine with respect to da Vinci surgeon console can be seen in Fig. 17.

Initially, five subjects were selected for the study, all of which have an engineering background and considered as novice users of the ultrasound machine. However, one subject had difficulty in proceeding with the task due to the poor ability of the speech recognition engine in recognizing one of the words spoken by the subject. The engine kept recognizing the command *select* spoken by the subject as *gain*. Therefore, Section 4.3 will report the results from the other four subjects, whereas the observation from the subject that was excluded from this study will be discussed in-depth in Section 5.2.

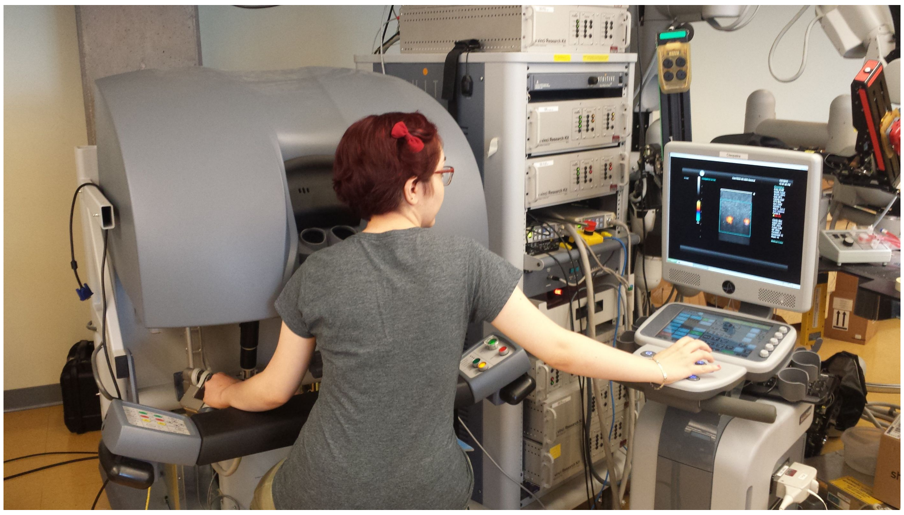


Figure 17. Ultrasound machine direct control setup

4.3 Results

For each subject, the completion time for each session was recorded. The results can be seen in Table 2. The completion time was the time required for each subject in each session to optimize the ultrasound image by changing the values of the depth, B-mode gain, and Color Doppler gain from their initial values. This metric did not include the time required to complete the calibration process and probe placement, as these actions were considered as parts of pre-session setups.

	Session 1	**Session 2**	**Session 3**
Subject 1	3 min 45 sec	2 min 51 sec	2 min 49 sec
Subject 2	2 min 41 sec	1 min 30 sec	2 min 16 sec
Subject 3	3 min 42 sec	2 min 17 sec	1 min 53 sec
Subject 4	1 min 46 sec	2 min 41 sec	2 min 25 sec
Mean	**2 min 59 sec**	**2 min 20 sec**	**2 min 21 sec**
Standard Deviation	**57 sec**	**36 sec**	**23 sec**

Table 2. Completion time for each session using the GUI

Table 3 contains the results for the task completed by direct control. Direct control was the scenario where the subject performed the image optimization task directly using the ultrasound machine, while still sitting in front of the surgeon console and placing the probe using the console. These values cannot be fairly compared with the values on Table 2 to completely assess the performance and the overall benefits of the GUI. The setup for the user study ruled out the time required to place the ultrasound probe correctly on the phantom in order to nullify the variations of proficiency in using the da Vinci robot between subjects in the results. However, the continuous control interaction with the da Vinci console and the ability of the operator in keeping his/her hands on the master tool manipulators, while using the GUI to control the ultrasound machine, are the apparent benefits of the GUI that could not be captured quantitatively through this image optimization task. The image optimization task only aimed to assess the general usability of the GUI, and further study needs to be performed in order to assess the usability of the GUI in the actual surgical settings. Therefore, the results presented in Table 3 are provided here solely as a baseline for the image optimization task.

	Session 1	**Session 2**	**Session 3**
Subject 1	30 sec	25 sec	23 sec
Subject 2	26 sec	27 sec	38 sec
Subject 3	33 sec	26 sec	30 sec
Subject 4	30 sec	28 sec	20 sec
Mean	**30 sec**	**27 sec**	**28 sec**
Standard Deviation	**3 sec**	**1 sec**	**8 sec**

Table 3. Completion time for each session by direct control

Another parameter that was analyzed from the user study was the speech recognition error ratio. This ratio is defined as the number of mistakes divided by the total number of commands recognized by the GUI. The mistakes included the situation where the subject said different commands or words from what was recognized by the GUI and the situation where the subject did not mention any command but the GUI picked up background voice as an input. The results from this analysis can be found in Table 4.

	Session 1	**Session 2**	**Session 3**	**Mean per Subject**
Subject 1	28.6%	15.6%	26.9%	23.7%
Subject 2	8.9%	18.5%	19.4%	15.6%
Subject 3	18.9%	14.7%	55.6%	29.7%
Subject 4	9.4%	24.1%	12.5%	15.3%
			Grand Mean	**21.1%**

Table 4. Speech recognition error ratio

Lastly, Table 5 shows the percentage of time spent using voice only mode. Since each subject was allowed to develop his/her own strategy in completing the task, great variations existed between subjects. For example, Subject 4 preferred to use voice only mode rather than the combination of gaze and voice, proven by the high percentages of time spent using voice only mode. On the other hand, Subject 2 in Session 2 and Subject 3 in Session 3 did not use voice only mode at all.

	Session 1	**Session 2**	**Session 3**	**Mean per Subject**
Subject 1	36.4%	22.8%	9.5%	22.9%
Subject 2	40.4%	0.0%	26.5%	22.3%
Subject 3	23.4%	73.0%	0.0%	32.1%
Subject 4	89.6%	66.5%	92.4%	82.8%
			Grand Mean	**40.0%**

Table 5. Percentage of time spent using voice only mode

All subjects found that the dynamic dwell-time attribute on the buttons that were used to increment and decrement parameter values were helpful in changing the values of B-mode gain and Color Doppler gain, but made it more difficult to change the value of the depth. Two of the subjects pointed out that when the menu display was on, the large dimensions of the buttons prevented them from clearly see the changes they made on the ultrasound image. Finally, all subjects gave positive feedbacks on the addition of voice only mode, as it helped them to continue with the task when their gaze's accuracy decreased over time. The discussion regarding these observations and the results of the user study can be found in Chapter 5.

Chapter 5

Discussion

User evaluation is an essential part in iterative design methodology followed in this GUI development. The author can receive invaluable feedbacks to improve upon the current interface, so that the next iteration will have higher usability than the interface under evaluation. The initial evaluation performed on the current iteration of the GUI proved to be very effective, as it pointed out the features that needed to be improved.

5.1 Gaze Tracking

The first issue came from the pre-session setups. Before going through each session, all subjects had to go through the gaze calibration process. They were allowed to repeat the calibration until they declared that current calibration result was the best for their gaze tracking purpose. Moreover, different iris color, eye shape, and skin tone of each subject required the observer to adjust some of the gaze tracking parameters in order for the gaze tracker to detect the subject's point of gaze accurately. As a consequence, this manual fine-tuning prolonged the actual total duration of the user study. In the future, the addition of image processing algorithm in the gaze tracking software to automatically perform the fine-tuning will speed up the calibration process.

Even if the gaze is calibrated accurately, the calibration will deteriorate over time [24]. This trend was also observed during the user study, where all subjects decided to use the voice only mode some time in the middle of the sessions. The deterioration was partly caused by the intrinsic anatomical (fovea coverage of approximately one degree of viewing angle) and physiological limitations (slow drift, micro-saccades, and micro-tremor) of the eyes [10, 27, 28]. Another source of the deterioration came from the head movement of the subject.

Although in the beginning of the study the observer had already told all subjects to minimize their head movement, they would not realize that they might have slightly changed their head position, since they interacted continuously with the GUI during the sessions. In the future, the addition of head frame on the da Vinci surgeon console or the development of head-mounted gaze tracker may potentially minimize the relative motion between the gaze tracking cameras and the operator's head.

5.2 Speech Recognition

The speech recognition engine worked well in recognizing the voice commands from all subjects included in Section 4.3. However, one subject from the initial five subjects was excluded from that result analysis. The subject could not complete the task due to the poor ability of the speech recognition engine in recognizing the command *select* from the subject. Due to the existence of accent in the subject's voice, the command *select* was recognized as *gain*. As a result, the subject could barely select any of the buttons displayed on the GUI.

Based from this case, two improvements can be made to the speech recognition in the GUI. First, the list of words for the speech recognition can be dynamically updated depending on which mode and program layer the GUI is currently in, so the GUI will only recognize a certain set of commands while in a particular mode. Current implementation registered all of the commands during the GUI initialization. Hence, it only prevented the GUI from executing the command when the current mode did not have that command displayed, but not prevented the GUI from recognizing that command. The second improvement for the speech recognition can be in the form of user-specific calibration, similar to that of the gaze tracking. Before using the GUI, the operator can be given a sentence to train the speech recognition engine to adapt with the operator's voice.

Based on the results shown in Table 4, the grand mean for the speech recognition error ratio was 21.1%, while the maximum error ratio could reach up to 55.6%. This high error ratio was mainly due to the fact that aside from the actual commands, the GUI also picked up the following two things as its input: 1) normal conversation between the subject and the observer (when the subject clarifying the image quality to the observer) and 2) background voice. The first source of error is expected to occur even in the actual surgical settings, since the surgeon needs to constantly communicate with the rest of the team. Therefore, it is considered as an external factor that the GUI has little or no control over. However, the second source of error can be greatly reduced by decreasing the gain of the microphone further to only pick up the operator's voice as its input. Moreover, the addition of trigger word to start the speech recognition engine in the next iteration of the GUI can potentially nullify both sources of error and consequently reduce the error ratio.

5.3 Graphical User Interface

The addition of voice only mode in the GUI was proven to be effective, proven by the grand mean of 40.0% for the percentage of time spent using the voice only mode. It allowed the subject to continue with the task as the gaze accuracy deteriorated over time. One subject even dominantly used this feature instead of the combination between gaze and voice inputs.

On the other hand, all subjects almost never used the gaze contingency feature on the edge of the screen. The lack of visual cues regarding this feature was most likely the cause of this low usage. Since each subject had to perform multiple subtasks during the image optimization task, the subjects tended to forget about this feature. In the future, the addition of visual feedback for the gaze contingency implementation may help the user in recognizing this feature on the GUI.

The dynamic dwell-time attribute on *latched* buttons has helped the subject in changing the values of B-mode gain and Color Doppler gain faster, since those parameters have a wide range of values. However, this attribute also made it more difficult for the subject to gain a fine control over the depth value. When trying to increase the depth, the subject tended to overshoot the value. Similarly, when the subject tried to decrease the depth, the final value appeared to be too low. One possible fix to this problem is to add fine and coarse options to the dwell-time attribute. When fine option is selected, the dynamic property can be deactivated or the increment can be set to smaller value. On the other hand, when coarse option is selected, the dynamic property can be activated or the increment can be set to larger value.

Nearly all of the features in the GUI can be activated using either the eye gaze or the voice command. For example, showing the menu buttons on the screen can be done either by using the voice command *display* or looking on the upper edge of the screen, switching the video stream from the surgical scene to the ultrasound image can be done either by using the voice command *switch* or looking on the right edge of the screen, and pointing on a button can be done by directly looking at the button or calling out the inherent voice command on the button during the voice only mode. However, confirming the selection of a button, which is a crucial feature in this GUI, can only be done through the use of voice command *select.* This limitation posed a problem during the user study, which caused the fifth subject to not complete the task. Therefore, the addition of button confirmation method using the eye gaze or other non-voice input in the next iteration of the GUI can eliminate similar problems in the future. From the user study, it can be concluded that the addition of the counterpart for each modality in this multimodal environment is essential in helping the subject completing the task, in case one of the modalities fails to perform reliably during the task.

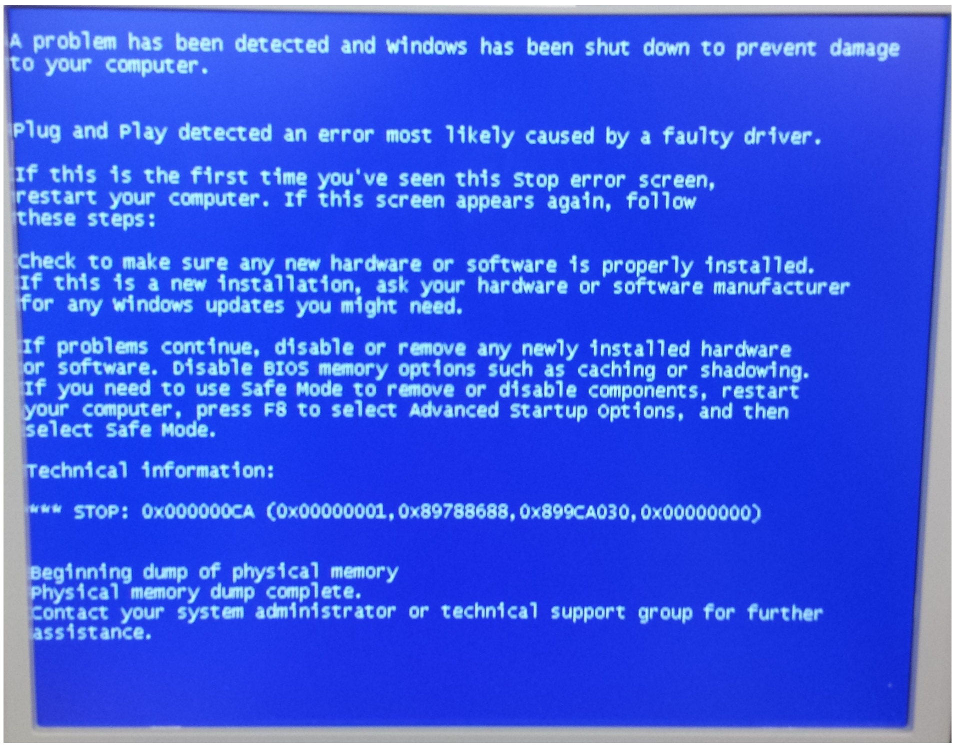
Regarding the menu layout, two subjects noticed that the value of the selected parameter, which was displayed on the parameter button itself, was situated at their peripheral vision when they were trying to increment or decrement the value. As a result, when the subjects wanted to change the parameter value to a specific number, they had to look back and forth between the increment or decrement button and the parameter button. This gesture ultimately prolonged the duration of the image optimization task. In the future, the menu layout should be redesigned in order to place the parameter value around the central vision of the operator, when the operator is incrementing or decrementing the value. Alternatively, gaze contingency can also be implemented when the ultrasound image is displayed on the screen, so that the operator can change the value of the depth by focusing his/her gaze on a certain area of the ultrasound image.

Some subjects also pointed out that due to the large dimensions of the menu buttons, they were unable to clearly see the changes they made on the ultrasound image. As discussed in Section 2.3, finding the right balance between the free space for the video stream and the button dimensions that were able to accommodate the limited accuracy of the gaze tracker, on a screen with the resolution of 640 x 480 pixels, required a couple cycles of design and user evaluation. An alternative to this trade-off would be the addition of auxiliary display such as TilePro, which is available in other versions of da Vinci system, but not on the da Vinci classic system (the system used for this work).

5.4 Technical Issues

Due to hardware limitations, the setup used in this initial user study was not the same as what the author had initially planned. Ideally, the gaze tracking and the GUI software should run directly on SonixTouch. However, due to the driver error, when one of the gaze tracking cameras (Leopard Imaging LI-OV5640-USB-72) was plugged in to the front USB port of

SonixTouch and the other camera was plugged in to the USB port at the back of SonixTouch, an error screen as shown in Fig. 18 occurred. After a thorough investigation regarding this issue using a software package called BlueScreenView by NirSoft and Bug Check Code Reference from Microsoft Corporation, the cause of this error was found to be Duplicate PDO. The details of the error code can be seen in Fig. 19.

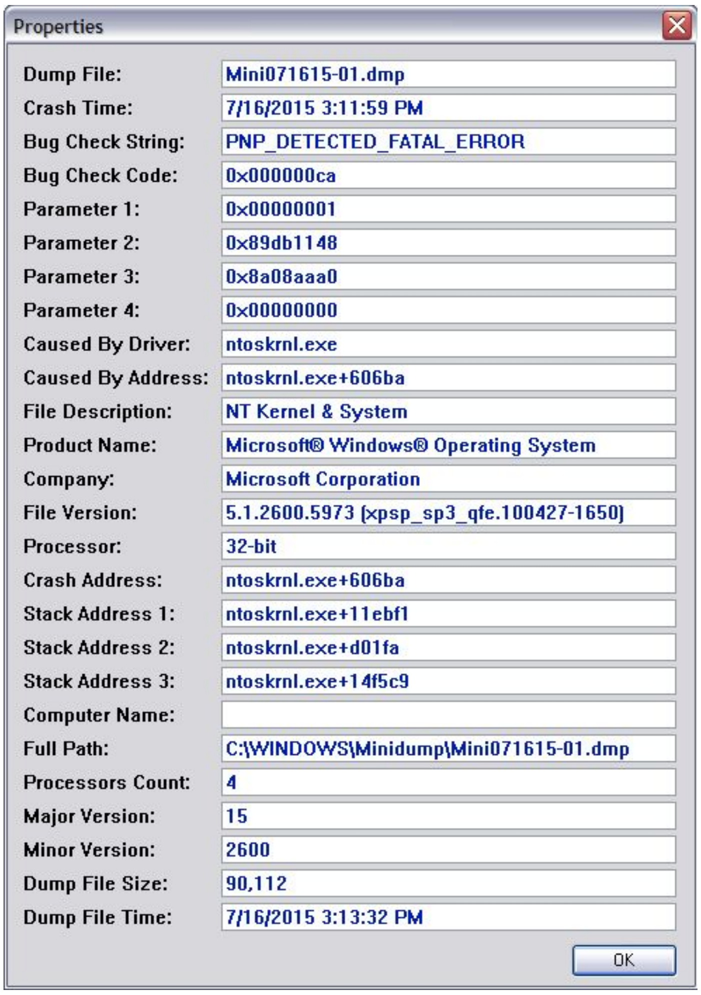


```
A problem has been detected and windows has been shut down to prevent damage
to your computer.

Plug and Play detected an error most likely caused by a faulty driver.

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

*** STOP: 0x000000CA (0x00000001,0x89788688,0x899CA030,0x00000000)


Beginning dump of physical memory
Physical memory dump complete.
Contact your system administrator or technical support group for further
assistance.
```

Figure 18. Blue screen appeared on SonixTouch

On the other hand, if both gaze tracking cameras were connected to the front USB ports of SonixTouch, no error screen occurred. However, when the gaze tracking software was running, OpenCV (computer vision library used in this work) could not access both cameras simultaneously. Based on this camera interfacing issue, it was concluded that the gaze tracking software had to run on another computer.

Another source of technical issues was the video interface. Initially, the author thought that the video signals from SonixTouch could be fed to the CRT screens of da Vinci system in order to show the GUI on the surgeon console. Using this setup, the GUI software could run directly on SonixTouch and the ultrasound image frames would not have to be sent via a TCP/IP connection, thereby reducing the time required to display the ultrasound image on the GUI. However, the author did not manage to interface the digital output of SonixTouch with the analog input of the CRT screens successfully. Even with the use of DVI-D to VGA active adapter in combination with H-Sync and V-Sync to Composite Sync adapter, SonixTouch could not detect the CRT screen as external monitors and consequently the CRT screens on da Vinci surgeon console could not show the video from SonixTouch. As a result, the GUI had to run on another computer, and the ultrasound image frames had to be sent to the GUI via a TCP/IP connection. This setup resulted in a delay when the ultrasound image was displayed on the GUI. The hardware and software connectivity for the user study can be found in Appendix G.



Dump File:	Mini071615-01.dmp
Crash Time:	7/16/2015 3:11:59 PM
Bug Check String:	PNP_DETECTED_FATAL_ERROR
Bug Check Code:	0x000000ca
Parameter 1:	0x00000001
Parameter 2:	0x89db1148
Parameter 3:	0x8a08aaa0
Parameter 4:	0x00000000
Caused By Driver:	ntoskrnl.exe
Caused By Address:	ntoskrnl.exe+606ba
File Description:	NT Kernel & System
Product Name:	Microsoft® Windows® Operating System
Company:	Microsoft Corporation
File Version:	5.1.2600.5973 (xpsp_sp3_qfe.100427-1650)
Processor:	32-bit
Crash Address:	ntoskrnl.exe+606ba
Stack Address 1:	ntoskrnl.exe+11ebf1
Stack Address 2:	ntoskrnl.exe+d01fa
Stack Address 3:	ntoskrnl.exe+14f5c9
Computer Name:	
Full Path:	C:\WINDOWS\Minidump\Mini071615-01.dmp
Processors Count:	4
Major Version:	15
Minor Version:	2600
Dump File Size:	90,112
Dump File Time:	7/16/2015 3:13:32 PM

Figure 19. Error code details

Chapter 6

Conclusions

This thesis presents the design and the development of a hands-free multimodal GUI for controlling an ultrasound machine. A prototype of the GUI was built, which enabled the user to change some parameter values directly from the da Vinci surgeon console using a combination of eye gaze and voice commands. Initial evaluation in the form of user study was performed, and using the GUI, the subjects were able to complete an image optimization task without having to take their hands off the da Vinci master tool manipulators. However, the current implementation of the speech recognition caused one subject unable to complete the task. In an iterative design process, especially for the first iteration of the GUI, this kind of failure is expected and becomes an invaluable feedback. Combined with other user feedbacks, they help the author pointing out not only the good features of the GUI, but also its weak features that need to be redesigned, so that the next GUI iteration will have improved usability.

Based on the results and observations from the user study, the improvements that can be made to the GUI and its overall system in the future include:

1. The implementation of fine-tuning algorithm for the gaze calibration to accommodate different iris color, eye shape, and skin tone of the operator and consequently improve the accuracy of the gaze tracking.
2. The use of a mechanism which minimizes the relative motion between the gaze tracking cameras and the operator's head, such as head frame on the da Vinci surgeon console or head-mounted gaze tracker.
3. The addition of user-specific calibration and trigger word for the speech recognition to reduce its error ratio.

4. The addition of visual cue or feedback for the gaze contingency implementation in order to help the operator in recognizing and using this feature on the GUI.
5. Further research regarding effective menu layout in order to deliver a GUI that is more intuitive to use and ready to be used in actual surgical settings.

Despite its limitations, the current iteration of the GUI shows that controlling the ultrasound machine directly from da Vinci surgeon console during robot-assisted MIS, using a combination of eye gaze and voice commands, is feasible. It allows the surgeon to manipulate the ultrasound parameters independently even when both of the surgeon's hands are occupied with surgical tools. This hands-free GUI has a practical application in the surgical room especially when no assistant is present to help the surgeon operating the ultrasound machine. Future works to improve the GUI usability will substantiate its potential adoption in actual surgical settings.

References

[1] J. H. Goldberg and X. P. Kotval, “Computer interface evaluation using eye movements: methods and constructs,” *Int. J. Ind. Ergonom.*, vol. 24, no. 6, pp. 631–645, 1999.

[2] G. L. Lohse, “Consumer eye movement patterns on Yellow Pages advertising,” *J. Advertising*, vol. 26, no. 1, pp. 61-73, 1997.

[3] L. Fletcher and A. Zelinsky, “Driver inattention detection based on eye gaze-road event correlation,” *Int. J. Robotics Research*, vol. 28, no. 6, pp. 774-801, 2009.

[4] M. F. Land, “Eye movements and the control of actions in everyday life,” *Progress in Retinal and Eye Research*, vol. 25, no. 3, pp. 296-324, 2006.

[5] R. Johnson *et al.*, “Exploring the potential for touchless interaction in image-guided interventional radiology,” in *Proc. SIGCHI Conference on Human Factors in Computing Syst.*, Vancouver, 2011, pp. 3323–3332.

[6] J. N. Afthinos *et al.*, “What technical barriers exist for real-time fluoroscopic and video image overlay in robotic surgery?” *Int. J. Medical Robotics and Comput. Assisted Surgery*, vol. 4, no. 4, pp. 368-372, 2008.

[7] C. Gratzel *et al.*, “A non-contact mouse for surgeon-computer interaction,” *Technology and Health Care: Official J. European Soc. for Eng. and Medicine*, vol. 12, no. 3, pp. 245-257, 2004.

[8] J. Leven *et al.*, “DaVinci canvas: A telerobotic surgical system with integrated, robot-assisted, laparoscopic ultrasound capability,” *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2005*, vol. 8, no. 1, pp. 811-818, 2005.

[9] G. D. Dodd 3rd *et al.*, “Sonography: The undiscovered jewel of interventional radiology,” *Radiographics*, vol. 16, no. 6, pp. 1271–1288, 1996.

[10] H. Drewes, “Eye gaze tracking for human computer interaction,” Ph.D. dissertation, Ludwig Maximilians Univ., Munich, Germany, 2010.

[11] J. van der Kamp and V. Sundstedt, “Gaze and voice controlled drawing,” in *Proc. of 1st Conference on Novel Gaze-Controlled Applicat.*, Karlskrona, 2011, pp. 1-8.

[12] R. J. K. Jacob, “What you look at is what you get: Eye movement-based interaction techniques,” in *Proc. SIGCHI Conference on Human Factors in Computing Syst.*, Seattle, 1990, pp. 11-18.

[13] T. E. Hutchinson *et al.*, “Human-computer interaction using eye-gaze input,” *IEEE Trans. Syst. Man Cybern.*, vol. 19, no. 6, pp. 1527-1534, 1989.

[14] D. Miniotas *et al.*, “Extending the limits for gaze pointing through the use of speech,” *Inform. Technology and Control*, vol. 34, no. 3, pp. 225-230, 2005.

[15] Y. Kammerer *et al.*, “Looking my way through the menu: The impact of menu design and multimodal input on gaze-based menu selection,” in *Proc. 2008 Symp. Eye Tracking Research & Applicat.*, Savannah, 2008, pp. 213-220.

[16] G. Tien, “Building interactive eyegaze menus for surgery,” M.S. thesis, School Comput. Sci., Simon Fraser Univ., Burnaby, Canada, 2009.

[17] J. Nielsen, “Iterative user-interface design,” *IEEE Computer*, vol. 21, no. 11, pp. 32-41, 1993.

[18] S. Zhai *et al.*, “Manual and gaze input cascaded (MAGIC) pointing,” in *Proc. SIGCHI Conference on Human Factors in Computing Syst.*, Pittsburgh, 1999, pp. 246-253.

[19] M. M. Farid and F. Murtagh, “Eye-movements and voice as interface modalities to computer systems,” in *Proc. SPIE*, Galway, Ireland, 2002, pp. 115-125.

[20] B. Shneiderman, “Designing menu selection systems,” *J. Amer. Soc. Inform. Sci.*, vol. 37, no.2, pp. 57-70, 1986.

[21] Y. Tan *et al.*, “Evaluating eyegaze targeting to improve mouse pointing for radiology tasks,” *J. Digital Imaging*, vol. 24, no. 1, pp. 96-106, 2011.

[22] I. Tong, “Model-based eye gaze tracking for robotic surgery,” unpublished.

[23] J. Hallett. (2001, May 1). *3-D imaging guides surgical operations* [Online]. Available: http://www.vision-systems.com/articles/print/volume-6/issue-5/features/medical-imaging/3-d-imaging-guides-surgical-operations.html

[24] O. Spakov and D. Miniotas, “Gaze-based selection of standard-size menu items,” in *Proc. 7th Int. Conference Multimodal Interfaces*, Torento, Italy, 2005, pp. 124-128.

[25] Apple, Inc. (2015, Apr. 8). *OS X human interface guidelines* [Online]. Available: https://developer.apple.com/library/mac/documentation/UserExperience/Conceptual/OSXHIGuidelines/DesignPrinciples.html#//apple_ref/doc/uid/20000957-CH18-SW1

[26] C. Schneider, “Intra-operative ‘pick-up’ ultrasound for guidance and registration to pre-operative imaging,” M.A.Sc. thesis, Dept. Elect. Comput. Eng., Univ. British Columbia, Vancouver, Canada, 2011.

[27] A. Spauschus *et al.*, “The origin of ocular microtremor in man,” *Experimental Brain Research*, vol. 126, no. 4, pp. 556-562, 1999.

[28] R. J. K. Jacob, “Eye tracking in advanced interface design,” in *Virtual Environments and Advanced Interface Design*, W. Barfield and T. A. Furness, III, Eds. New York: Oxford Univ. Press, 1995, pp. 258-288.

[29] C. Hennessey, “Eye-gaze tracking with free head motion,” M.A.Sc. thesis, Dept. Elect. Comput. Eng., Univ. British Columbia, Vancouver, Canada, 2005.

Appendices

Appendix A: Visual Angle Error to Pixel Error Calculation

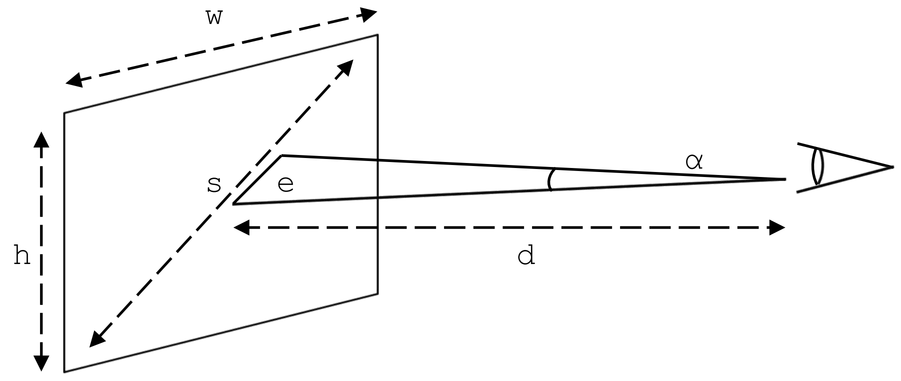


Figure 20. Diagram of the operator's eye to the da Vinci screen

w	= width of the screen = 640 pixels
h	= height of the screen = 480 pixels
s	= diagonal size of the screen = 14 in
d	= distance from the operator's eye to the screen = 18 in
e	= gaze tracker error on the screen (in inches)
α	= gaze tracker error (in degree of viewing angle)

Since the ratio between the width and the height of the screen is 4:3 and the diagonal size of the screen is 14 inches, it is assumed that the width of the screen is 11.2 inches and the height of the screen is 8.4 inches.

These three equations for converting from pixel error to visual angle error are adapted from the paper written by Craig Hennessey [29], with modification necessary for this work:

$$\Delta X_{in} = \Delta X_{pixel} \cdot \frac{11.2\ in}{640\ pixels}$$

$$\Delta Y_{in} = \Delta Y_{pixel} \cdot \frac{8.4\ in}{480\ pixels}$$

$$\alpha = 2 \cdot tan^{-1}\left(\frac{\frac{\sqrt{\Delta X_{in}{}^2 + \Delta Y_{in}{}^2}}{2}}{18\ in}\right)$$

For the following calculation, it is assumed that the horizontal and vertical pixel errors are the same ($\Delta X_{pixel} = \Delta Y_{pixel} = \Delta E_{pixel}$). The gaze tracker error is the sum of mean pixel error (ε) and the standard deviation, σ. Assuming that the error follows a normal distribution, this calculation will cover around 68% of the error caused by the operator's gaze. The final value of the pixel error (ΔE_{pixel}) in this calculation is used to determine the minimum gap surrounding each button.

$$\alpha = \varepsilon + \sigma = 0.88° + 0.69° = 1.57°$$

$$\sqrt{{\Delta X_{in}}^2 + {\Delta Y_{in}}^2} = 2 \cdot 18\, in \cdot tan\left(\frac{\alpha}{2}\right)$$

$$\sqrt{\left(\frac{11.2\, in}{640\, pixels}\right)^2 + \left(\frac{8.4\, in}{480\, pixels}\right)^2}\, \Delta E_{pixel} = 2 \cdot 18\, in \cdot tan\left(\frac{1.57°}{2}\right)$$

$$\Delta E_{pixel} = \lceil 19.93 \rceil = 20\, pixels$$

Appendix B: Source Code Modification to *speech* Python Module

- **Original code:**

  ```
  _recognizer = win32com.client.Dispatch("SAPI.SpSharedRecognizer")
  ```

  Modification:

  ```
  _recognizer = win32com.client.Dispatch("SAPI.SpInProcRecognizer")
  _recognizer.AudioInputStream = win32com.client.Dispatch(
                                 "SAPI.SpMMAudioIn")
  ```

- **Original code:**

  ```
  _ListenerBase = win32com.client.getevents("SAPI.SpSharedRecoContext")
  ```

  Modification:

  ```
  _ListenerBase = win32com.client.getevents("SAPI.SpInProcRecoContext")
  ```

Appendix C: Speech Recognition Implementation in the GUI

- **Word dictionary initialization in the main thread of the GUI:**

```
# Add basic commands to the word dictionary
self.word_dict = {'display':(lambda x:self.showMenu(0)),
                  'select':(lambda x:self.buttonAction(0)),
                  'switch':(lambda x:self.toggleStream(0)),
                  'voice only':(lambda x:self.overrideGaze(0)),
                  'use gaze':(lambda x:self.revertGaze(0)),
                  'stop':(lambda x:self.canvas.coords(
                          self.crosshair,self.width/2,self.height/2))}
# Add words specific to each button
for key,val in self.button_dict.iteritems():
    self.word_dict[val[1].voice_command] = (
        lambda x,y:self.moveCursorByVoice(x, y))
    # Save the X and Y coordinates of the button center and button ID
    self.btn_voice_dict[val[1].voice_command] = (val[1].center[0],
        val[1].center[1],val[1].buttonID)
```

- **Speech notification checking and function invocation depending on which word or phrase gets passed on to the main thread of the GUI, inside the main loop:**

```
try:
    temp_word = self.speech_queue.get(block=False, timeout=0)
except Queue.Empty:
    pass
else:
    self.command_word = temp_word
    # Check whether the command is button-specific or basic command
    if self.command_word in self.btn_voice_dict.keys():
        temp_buttonID = self.btn_voice_dict[self.command_word][2]
        if temp_buttonID not in self.button_hidden_list:
            force_x = self.btn_voice_dict[self.command_word][0]
            force_y = self.btn_voice_dict[self.command_word][1]
            self.word_dict[self.command_word](force_x,force_y)
    else:
        self.word_dict[self.command_word](0)
    self.command_word = ''
```

- **Speech recognition engine initialization in the background thread:**

```
self.listener = speech.listenfor(self.word_list, self.callback)
```

- **Callback function definition in the background thread, where this function is invoked everytime the word or phrase in the list passed to the speech recognition engine is picked up by the computer:**

```
def callback(self, phrase, listener):
    print 'Command detected: ', phrase
    self.queue.put(phrase.lower())
```

Appendix D: System Setup and Path Configuration File for *pyUlterius*

- **System setup for *pyUlterius*:**

  The following system setup assumes that the Python version installed to the system is 2.7, and the Ulterius SDK used is 6.1.1.

  1. Copy the main Python module, pyUlterius.py, and the path configuration file, pyUlterius.pth, into C:\Python27\Lib\site-packages
  2. Create a new directory named _pyUlteriuslib in C:\Python27\Lib\site-packages
  3. Copy the Python DLL, _pyUlterius.pyd, into _pyUlteriuslib
  4. Copy all the contents from the directory \sdk_6.1.1\bin\ into _pyUlteriuslib

- **Sample content of pyUlterius.pth:**

  The path configuration file basically contains the directory name where the DLL of the corresponding module can be found, which based on the above system setup is _pyUlteriuslib. Note that this path configuration file has to be in the same directory as the main Python module.

```
# pyUlterius package configuration
_pyUlteriuslib
```

Appendix E: List of Member Functions for *ulterius* Class of *pyUlterius*

The functions below are meant to be used in Python to access the original functions of Ulterius SDK. The description found in each member function only specifies what the corresponding original function is. The complete description for each function can be found in the original Ulterius SDK documentation. Finally, this list is non-exhaustive, since it only contains functions that are found in *pyUlterius* but not in the original SDK and functions that can be found in the original SDK but have different calling or return method from the original SDK.

- **getActivePreset (sz)**
  - Arguments
    - sz (int): The size of the buffer
  - Returns: **[success, preset]**
    - success (bool) : True if call is successful, False otherwise
    - preset (str) : The character string containing active preset name
  - Description
    - This function performs the same task as *getActivePreset* in the original SDK. However, the function calling method is different.

- **getActiveProbe (sz)**
  - Arguments
    - sz (int): The size of the buffer
  - Returns: **[success, probe]**
    - success (bool) : True if call is successful, False otherwise
    - probe (str) : The character string containing the active probe name
  - Description
    - This function performs the same task as *getActiveProbe* in the original SDK. However, the function calling method is different.

- **getImageData (encap_data, size)**
  - Arguments
    - encap_data (PyCapsule) : The capsule object containing the pointer to the image data
    - size (int) : The total pixel length of the image data
  - Returns: **pixel_array**
    - pixel_array (bytearray) : The array containing the 8-bit pixel information of the ultrasound image
  - Description
    - This function is a helper function to convert an image data type from PyCapsule to bytearray.

- **getLastError (sz)**
    - Arguments
        - sz (int): The size of the buffer
    - Returns: **[success, err]**
        - success (bool) : True if call is successful, False otherwise
        - err (str) : The character string containing the error message
    - Description
        - This function performs the same task as *getLastError* in the original SDK. However, the function calling method is different.

- **getParamValue (id, ref)**
    - Arguments
        - id (str) : The identifier of the parameter
        - ref (IntByRef reference): The reference to a Python integer value
    - Returns: **success**
        - success (bool): True if call is successful, False otherwise
    - Description
        - This function performs the same task as *getParamValue* in the original SDK, which retrieves the value of an integer imaging parameter. However, the function calling method is different.

- **getParamValue (id, size)**
    - Arguments
        - id (str) : The identifier of the parameter
        - size (int) : The size of the buffer
    - Returns: **[success, value]**
        - success (bool) : True if call is successful, False otherwise
        - value (str) : The character string storing the parameter value
    - Description
        - This function performs the same task as *getParamValue* in the original SDK, which retrieves the value of a string imaging parameter. However, the function calling method is different.

- **getPatientInfo (sz)**
    - Arguments
        - sz (int): The size of the buffer
    - Returns: **[success, ptinfo]**
        - success (bool) : True if call is successful, False otherwise
        - ptinfo (str) : The character string containing the patient fields, separated by newline characters (\n)
    - Description
        - This function performs the same task as *getPatientInfo* in the original SDK. However, the function calling method is different.

- **getPresets (sz)**
  - Arguments
    - sz (int): The size of the buffer
  - Returns: **[success, presets]**
    - success (bool) : True if call is successful, False otherwise
    - presets (str) : The character string containing the preset names, separated by newline characters (\n)
  - Description
    - This function performs the same task as *getPresets* in the original SDK. However, the function calling method is different.

- **getProbes (sz)**
  - Arguments
    - sz (int): The size of the buffer
  - Returns: **[success, probes]**
    - success (bool) : True if call is successful, False otherwise
    - probes (str) : The character string containing the probe names, separated by newline characters (\n)
  - Description
    - The function performs the same task as *getProbes* in the original SDK. However, the function calling method is different.

- **set_pyCallback (callback)**
  - Arguments
    - callback (callable) : The callback function that is triggered when new frames arrive or the server has disconnected
  - Returns: **None**
  - Description
    - This function is the wrapper to *setCallback* in the original SDK.
    - callback has to have 5 arguments: data, type, sz, cine, and frmnum.

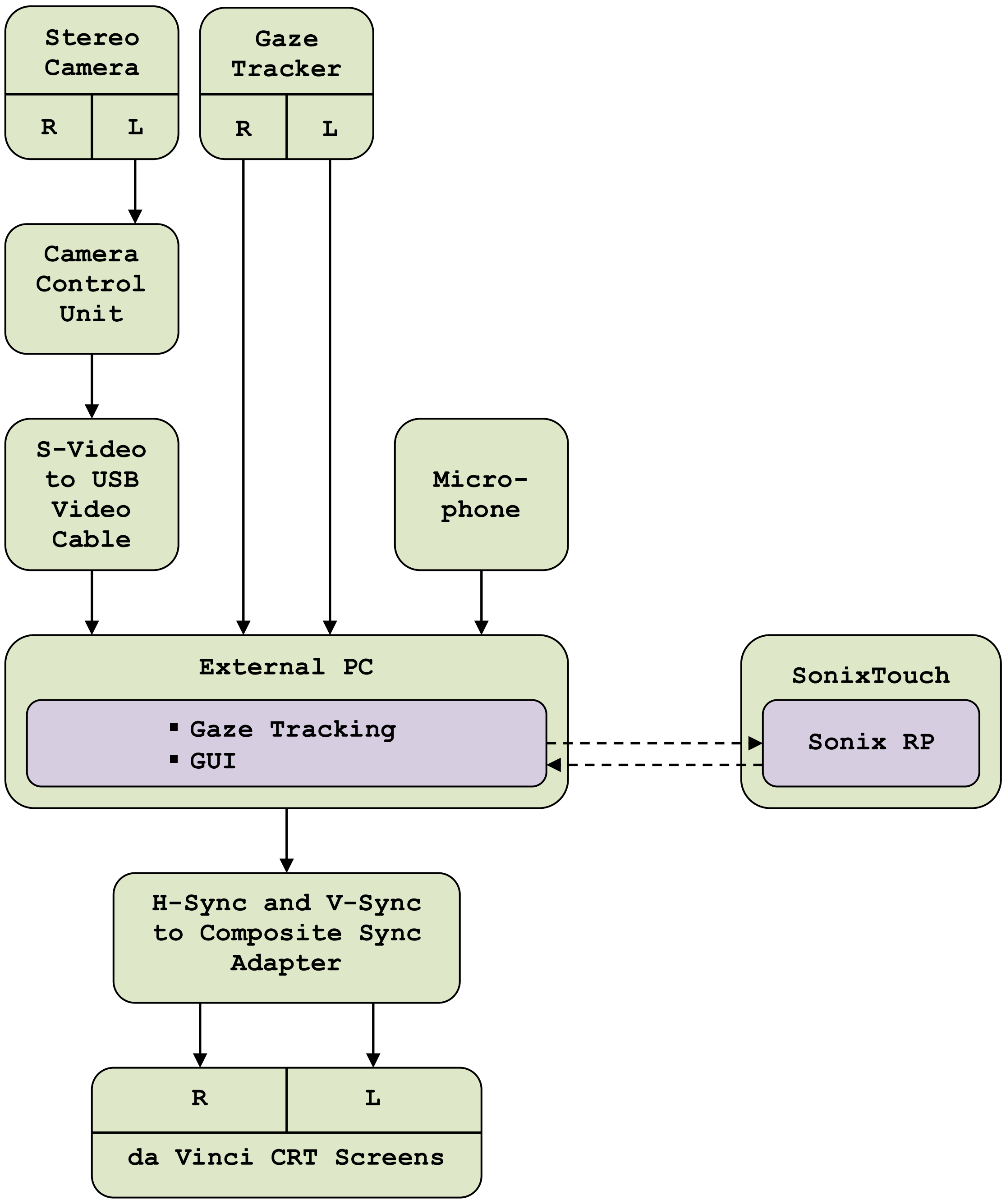
- **set_pyParamCallback (param_callback)**
  - Arguments
    - param_callback (callable) : The callback function that monitors the status of some parameters on the server side
  - Returns: **None**
  - Description
    - This function is the wrapper to *setParamCallback* in the original SDK.
    - param_callback has to have 3 arguments: paramID, ptX, and ptY.

Appendix F: Untested Member Functions for *ulterius* Class of *pyUlterius*

The member functions listed here have not been tested, since their functionalities were not utilized in this work.

- getCineData
- getStreamStatus
- injectImage
- stopStream
- streamScreen

Appendix G: Hardware and Software Setup for User Study



🟩	Hardware
🟪	Software
—	Physical connection
- -	Communication via TCP/IP