Augmented Reality for Ultrasound Imaging

An Major Qualifying Project Report
submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science
Biomedical Engineering, Computer Science, and Robotics Engineering

By:
Mary Barsoum, Sarah Lombardi, and Ian Scott
Date: April 28th, 2022

Report Submitted to:
Yihao Zheng, Department of Mechanical Engineering
Haichong Zhang, Department of Computer Science

Table of Contents

Abstract **2**

Introduction **3**

Literature Review **4**

Diagnostic Medical Ultrasound 4

Ultrasound Training 5

Augmented Reality 6

Position Sensors 7

Existing Solutions 9

Methods **11**

Clarius Ultrasound Probe 12

Polhemus Position Sensor 13

Unreal Engine 14

Microsoft Hololens 2 15

Discussion **16**

Achievements and Challenges 16

Future Extensions 18

Broader Impacts 19

Acknowledgments **21**

References **22**

Appendix **25**

Appendix A. UE4_Projection_Actor 25

Projection.h 25

Projection.cpp 26

Appendix B. Primary ROS Script for Position Sensor 32

Data_expoter.py 32

Appendix C. Clarius Cast API Image Streaming 33

Abstract

Ultrasound imaging is a widely used method to painlessly visualize internal structures of the human body through the propagation and reflection of inaudible sound waves. Currently, a sonographer must position and orient the probe correctly on the patient, while looking away at a separate monitor. This ultrasound imaging practice requires high level hand-eye coordination and the ability to conceptualize the 2D screen images into the patient's 3D anatomy. This kind of skill set can take up to two years to develop. The goal of this project is to use augmented reality (AR) to visualize ultrasound images over a patient's respective anatomy, in correlation with the ultrasound probe. We aimed to use the Microsoft Hololens 2 AR headset to display the ultrasound images gathered by a Clarius ultrasound probe in real time on the patient's respective anatomy. The team successfully gathered ultrasound images and position data in real-time. However, due to networking issues, real-time streaming of both the data simultaneously onto the Microsoft Hololens 2 could not be accomplished. In its current state, a single ultrasound image can be moved with real time transformation data over the patient's anatomy, respectively. Future extensions include resolving the WiFi connectivity challenge, as well as calibrating the transformation data and the positioning of the ultrasound image on the patient's anatomy.

Introduction

Ultrasound technology has been around for decades, improving the lives of millions. This technology has a plethora of uses, from prenatal care to diagnosis. Although several advances in ultrasound technology have been made there still lacks a simple, straightforward way to view images while conducting an ultrasound. Currently, technicians must perform the ultrasound and refer to an external monitor, in a different direction, for the images, which is not ideal.

Additionally, ultrasound technicians require two years of training, which we believe could be significantly decreased if there was an easier way to conceptualize the anatomy. Current ultrasound imaging practice requires high level hand-eye coordination and the ability to conceptualize the 2D screen images into the patient's 3D anatomy. Our goal is to use augmented reality (AR) to visualize ultrasound images over a patient's respective anatomy, in correlation with the ultrasound probe.

The use of AR will continue to grow because of its versatility and ability to meld the real and virtual world. This technology has the potential to expand and grow into even more industries, such as aerospace, architecture, and data science.

In our project, we aimed to create not only a solution using AR for gathering ultrasound images, but a baseline for the future expansion of internal system imaging. There are many ways that this system can be built upon that will be beneficial to future technicians and patients. A feature that could be added to the application we created is a way to color code different blood clots, arteries, and abnormalities in the body. Important vitals could be displayed live on the screen to reduce the time spent collecting them prior to imaging. These are just a few examples of the expansions that could be made for this technology. Truly, the possibilities are endless when you have such a powerful technology.

Literature Review

In order to determine the approach we would pursue when creating our solution, we had to fully immerse ourselves in research. The research we completed during the course of our project focused on ultrasounds and training, augmented reality, position sensors, and existing solutions. We chose to research these topics in order to fully understand the problem we had at hand. We also learned about the existing solutions for this technology, which helped us better understand the potential expansions and future extensions we could pursue with our system.

Diagnostic Medical Ultrasound

Ultrasounds are the most widely used tool for non-invasive imaging of the internal organs. According to the Radiology Center at Harding, ultrasounds were initially invented to view the manufacturing flaws of industrial ships. It was not until 1956 that ultrasounds were used in a clinical setting. They are popular due to their versatility in what organs they can image, the sheer number of medical attributes they can analyze, and for causing no negative effects from use. There are two types of medical ultrasounds: therapeutic and diagnostic.

Therapeutic ultrasounds use inaudible sound waves of over 20KHz to modify and destroy specific tissues in the body. The non-invasive method makes it possible to dissolve blood clots, and destroy tissues that contain tumors (*NIH*).

The Mayo Clinic describes a diagnostic ultrasound as a noninvasive way to get an image inside of the body with no negative effects. The physical ultrasound probe acts as a transducer and produces inaudible sound waves. The sound waves that are not absorbed by the tissues and bones bounce back, creating the image.

If the body is experiencing pain, swelling, or infection, ultrasound may be a suitable method to evaluate the concern further due to its wide range of properties it can evaluate. These include but are not limited to pregnancy, blood flow, tumor/lumps, joint inflammation, and metabolic bone disease. Ultrasounds can examine many organs including the heart, liver, pancreas, uterus/fetus, and many more. Ultrasounds can also guide different procedures like needle biopsies (*NIH*).

Ultrasound Training

Individuals that conduct ultrasounds are known as diagnostic medical sonographers. Sonographers are trained on imaging devices that use sound waves to produce an image. To become a sonographer, a two-year associate's degree from an accredited sonography program must be obtained.

When an ultrasound is performed, A sonographer applies gel to the skin to prevent air pockets from forming between the skin and the transducer. Air pockets will block the sound waves from entering the body. While the patient is laying down, the sonographer must move and angle the ultrasound probe to better examine the specific organ. The images are displayed on a separate monitor, requiring the sonographer to turn their head to view the results as seen in figure 1 (*Mayo Clinic*).

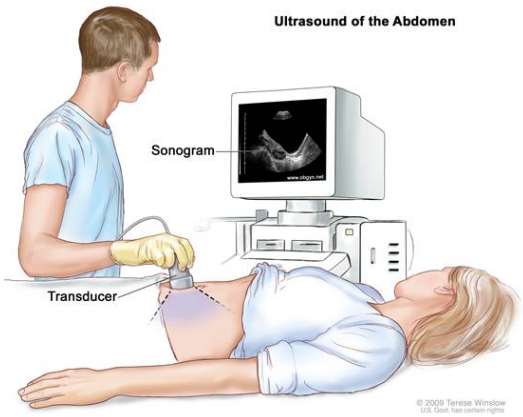

Figure 1. Current ultrasound setup (*NIH*)

It is difficult to angle and orient the probe correctly while looking at a monitor in a different direction. The hand-eye coordination required for a sonographer takes years of training to do correctly.

Augmented Reality

Unlike virtual reality, which is a completely immersive experience, Augmented Reality (AR) is an interactive experience where virtual objects such as images and videos are overlaid on the real world environment. AR can be defined as a system which incorporates the melding of real and virtual worlds, live interaction within that melded world, and registration of positionally accurate 3D virtual and real objects.

The first AR headset was created in 1968 by Ivan Sutherland, a Harvard professor. The headset was called The Sword of Damocles, and it displayed computer generated graphics over the real world environment. Since then the idea of overlaying graphics onto the real world environment have become increasingly popular. This phenomenon can even be seen in sports games. In 2016 the Microsoft Hololens was released, reinventing AR and showing how it can be integrated in our everyday lives.

AR has many use cases, such as in the medical industry, for flight training, video games, visual art, and architecture (. One of the earliest cited examples of AR was using the technology to support surgery by providing virtual overlays to guide medical practitioners. The image on the right displays what the subject is viewing through AR. In the real world, there are no holograms or visuals that can be seen by anyone who is not looking through the hololens.

Position Sensors

There are several ways to gather positional data. One method of gathering an object's location in space is to gather data from an inertial measurement unit (IMU). IMUs are composed of several different sensors like the accelerometer which measures acceleration, gyroscope which measures velocity, and magnetometer, which measures magnetic field strength. Data from this sensor is usually fused to determine motion, orientation, and heading. An IMU can include more than one of the listed sensors depending on the type of transformation data that wishes to be collected. IMU's can collect the following data: gravity, linear acceleration, orientation, and rotation vectors. Also, IMU's gather their transformation data based on a relative data point. For example, the orientation of an object measured using IMU's will be given relative to starting orientation value. While IMU's are great for managing large spatial changes and don't require a physical piece of hardware as a reference frame, they are not particularly suited for micro adjustments and low-acceleration devices. This is because IMUs tend to drift under low acceleration loads due to minute but non-zero accelerations being read and thus translating to slow position drift. Compensation for this issue is the source of a lot of software development and research (*Dawadi* 2017).

Another way to gather positional data is with a position sensor. Position sensors are "devices that can detect the movement of an object or determine its relative position measured from an established reference point". These sensors can be used to detect the presence of an object or its absence, respectively (Edwards, n.d). There are many different sensor types which serve similar purposes to position sensors which we will briefly review. Motion sensors, which deterc movement of an object can be used to trigger actions such as turning on a light or playing an alarm. Proximity sensors can be used to detect objects which are in a certain range of the sensor. These sensors can be considered as a "special type" of position sensor, but they do not achieve the goal that we are trying to meet. The distinction between these sensors and the sensor that we use is that they are concerned not only with the detection of an object, but also with the recording of its position in the real world in comparison to the sensor. Therefore, these sensors must involve the use of a feedback signal which contains the object's positional information.

Although there are many ways to sense positional data, we felt that position sensors were the best option for our team. Position sensors can be divided into three different classes which are described as: linear position sensors, rotary position sensors, and angular position sensors. There are specific technologies which divide these sensors into these categories that can be used to achieve different results. Primary types of position sensors include but are not limited to: Electromagnetic, Potentiometric (resistance-based), Inductive, Eddy Current-Based, Capacitive, Magnetostrictive, Hall Effect-Based Magnetic, Fiber-Optic, Optical, and Ultrasonic. For our purposes, we strictly reviewed the Electromagnetic type of position sensor.

Electromagnetic tracking (EMT) systems utilize a source that acts as a transmitter, which emits an electromagnetic dipole field. The size of the field varies, depending on the size of the source used; the larger the magnetic field, the larger the tracking range, or coverage area. These

devices use Faraday's Law to track the location of a probe. The tracking is done by generating a magnetic field with a base device and moving a sensor probe (constructed of conductive coils) through the field. This effect provides an extremely precise 6-DOF measurement of the probe relative to the field emitter (*Electromagnetic* 2017). However, these EMTs suffer from a few drawbacks. Firstly, they are very fragile. Both the emitter and sensor are made of very fine conductive coils. If there is any damage to these coils, the magnetic field will interact in the same predictable way and thus measurements will become useless. In addition, the magnetic field itself is a limitation. Firstly, the field has a very finite range from the emitter and thus the position sensor is limited by this range. Additionally, the electromagnetic field is highly vulnerable to disturbance from electronic devices. The electromagnetic output from a computer or phone is enough to disrupt position readings from a nearby sensor.

We decided to use the Electromagnetic position sensor because in controlled, confined settings it will provide extremely accurate position readings and suffer none of the issue of drift from an IMU. Since our project involves a very small frame of reference for the position sensor to work with, we decided that this was our best option. However, if motion over a larger range is required or a base reference site is not acceptable, an IMU is by far the better option. For our project, we will use the Viper Position sensor from Polhemus to track the location of the ultrasound probe in relation to the Microsoft Hololens 2.

Existing Solutions

Multiple research institutions in Germany have successfully implemented AR in combination with ultrasound imaging. Although ultrasounds are most commonly known for analyzing and diagnosing medical attributes, they are also used for guiding biopsies that use

needles. These research institutions sought out to create a better way to guide these biopsies. Ultrasound imaging allows for medical professionals to see where they are guiding and orienting a needle inside the patient's body (*Rüger, C. and Tectales*)

The Fraunhofer Institute for Computer Graphics Research IGD used AR glasses to display the sectional planes from the ultrasound image on the patient, where the anatomy was represented. Seeing the image structure real time allowed the doctor to see the needle during the biopsy and adjust it accordingly. The position of the image is determined by using the tracking system from both the ultrasound probe and the AR glasses system. The software that Fraunhofer IGD is using evaluates the positions of the probe and the glasses in relation to either other. The data is then used to display the image on the patient.

Fraunhofer IGD found that there were higher success rates when learning the ultrasound guided biopsy procedure if the medical professionals were equipped with the AR headset. The higher success rate can be attributed to a lack of 2D to 3D spatial relation. The medical professional no longer has to look at a monitor that is placed in a different direction than the patient, and this creates a better experience for everyone involved. As of April 2020, hospital trials plan to be conducted with the AR headset for the purposes of ultrasound guided biopsies.

Our project will focus more on how we can enhance regular medical analysis and diagnostics through AR, rather than ultrasound guided biopsies. However, the positive success rate from the studies done in Germany are promising for our solution and our ability to successfully enhance medical ultrasound diagnosis.

Methods

To create a system that implements AR for ultrasound imaging, many systems must work in unison. Our system is shown below in Figure 2. The most important component of this system is the ultrasound technician. The ultrasound technician will wear the Hololens and use the Clarius ultrasound probe to perform an ultrasound on the patient. What they will see on their end is a superimposed ultrasound image that is displayed over the patient's respective anatomy. This ultrasound image will update in real time corresponding to the patient's internal systems. As the technician performs the ultrasound, they will be able to better understand the location of any abnormalities, and therefore will be able to create diagnoses with more confidence.

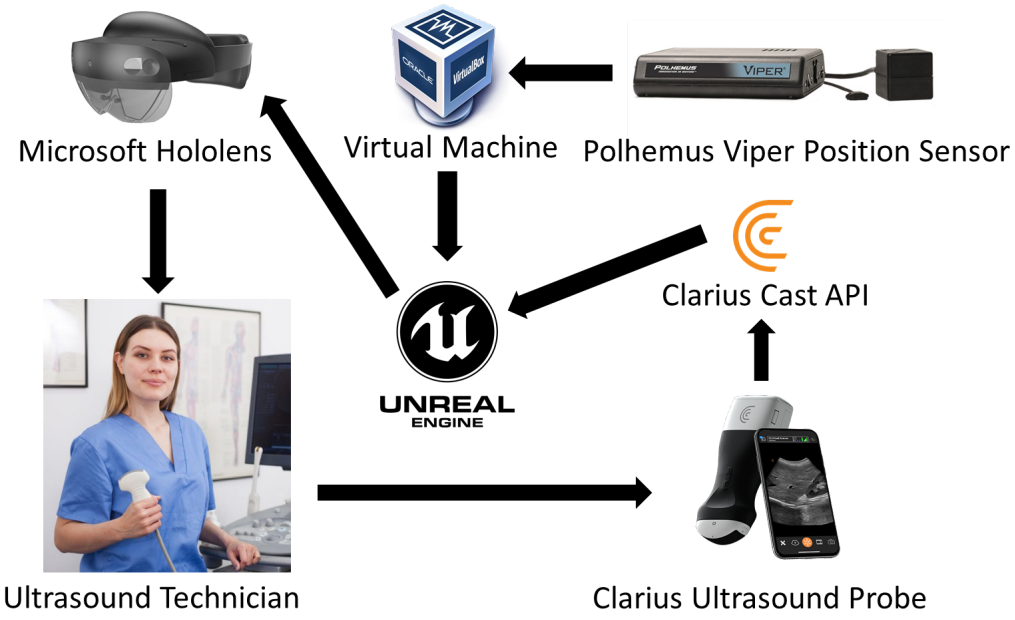

Figure 2. Flow of system design

As you can see from the diagram above, this system contains several different pieces which all make up our solution. We decided to set it up this way for several reasons. The first

being that the Clarius probe uses a wifi network in order to connect to its mobile app and the program we created that streams the images into the Hololens. The second being that the position sensor only operates in Linux, so we would need to run the code through a virtual machine program. The third being that the Hololens needs to be connected to the internet in order to connect to the Unreal Engine. These systems all connect through the Unreal Engine, which exports data into the Hololens. In order to get all these systems working in unison, we had to design our system in this manner.

Clarius Ultrasound Probe

The system must have some way of streaming in ultrasound data gathered by the sonographer. To capture these ultrasound images the team utilized the Clarius Ultrasound probe. This hand-held wireless ultrasound probe is capable of capturing and exporting images in real-time. We chose to use this probe because of its versatility. The fact that it does not need to be connected to an external system other than the mobile app was very appealing for our team. This is because our goal is to simplify the ultrasound process and make it more straightforward for all parties involved. The Clarius probe is easy to understand, and does not require technicians to adjust image quality manually. This is because it has an Artificial Intelligence that automatically determines the best settings for image clarity.

Clarius Cast is an application programming interface (API) that allows developers to create software that utilizes the Clarius probe in some manner. APIs are defined as “a software intermediary that allows two applications to talk to each other” (Mulesoft, n.d). Essentially this means that the functionality of the probe is easily accessible to developers through the use of the API. The Cast API helped the team develop an application that could receive and process the

ultrasound images as they were gathered. This application was created in the QT Creator integrated development environment (IDE). An IDE is “software for building applications that combines common developer tools into a single graphical user interface” (Red Hat, 2018). We chose to use the QT creator IDE because the libraries that were accessed in the API documentation made it necessary for us to use the IDE. The QT creator IDE also helped provide the team with several functions that simplified the construction of our user interface. In the Cast API, there were several example programs which we used to base our code off of. We chose to use the “caster” example code as a base to build upon for our particular application. The program that we created can access the live ultrasound images by establishing a connection to the probe via its ip address and port number. Once the probe starts imaging, the application exports each captured frame into a .raw binary file inside the local file system. These images are used in correlation with our Unreal Engine program that will stream images to the Hololens.

Polhemus Position Sensor

In order to position the ultrasound image in the proper location, the team had to track the motion of the ultrasound probe. The proper location corresponds to the patient’s anatomy. For example, if you are scanning someone’s forearm, you expect to see the forearm’s veins in the ultrasound image located in an area that is consistent with where you are aiming the probe.

To help us achieve the ultrasound image overlay, we would need a means for collecting positional data from the ultrasound probe. Our team decided to use a Polhemus Viper position sensor for gathering this data. Since this positional data could only be gathered through drivers that ran in the Robot Operating System (ROS), the team had to use an Ubuntu Linux virtual machine. A program was created to read the data published in ROS and write it to a .CSV file.

This .CSV file was then made accessible both in the virtual machine and the parent Windows operating system through file sharing. Then, we could retrieve this data in our final solution to gather position data from the ultrasound probe. The refresh rate at which this image was updated was set to 10 Hz. It was found that if the program tried to write data as fast as it was published, the software would bottleneck. The 10 Hz data refresh rate was found to avoid this issue while providing the end user with a relatively smooth experience.

Unreal Engine

Game engines are the primary tool used to create applications for an AR or VR device. Developers generally use these pre-built 3D graphics engines to render out the correct objects in 3D space. Game engines are the primary tool for this task due to their powerful real time 3D rendering capabilities. Unreal Engine was chosen because one of our team members had experience working in it. Applications developed for it are created in C++. In addition, the Hololens has pre-built libraries for the Unreal Engine. We also planned to bundle the Clarius API application (built in C++) with the Unreal Engine hololens application to allow for easy transfer of data between the two.

For construction of the 3D scene, the Clarius Ultrasound data and Polhemus position sensor data were used in unison. Every frame, the most up-to-date ultrasound data would be imported into the engine. The ultrasound data was then converted into a 2D texture and applied to a plane object in the 3D scene as a material. The transformation data from the position sensor was used to set the position and orientation of the 3D plane. By doing this process every frame, and with the rapid refresh rate, the end result was a 2D plane transforming in real-time in the 3D scene using the data gathered. There was virtually no input lag.

Microsoft Hololens 2

The Microsoft Hololens 2 was chosen as our means for augmented reality. The final stage of this system was to upload it onto the Microsoft Hololens 2. The Microsoft Hololens 2 is “an ergonomic, untethered self-contained holographic device with enterprise-ready applications to increase user accuracy and output” (Microsoft, n.d). This device can be used to overlay 3D objects over the real world, such as our ultrasound images. We chose to use the Microsoft Hololens because we believe that it is the best augmented reality headset on the market. Although augmented reality has been around for several years, it is only now being truly improved upon. Microsoft is taking the lead while other companies follow, so we figured that the Hololens would have the best, most accurate representation of augmented reality that we could find.

In order to get the program uploaded onto the Hololens, we would need to establish a connection to the wifi network so that we could stream the program into the Hololens environment. This is because the Hololens requires a wifi connection in order to be streamed to. This system works as follows: The Hololens headset connects to Unreal Engine via wifi in order to stream the 3D scene from the engine. Then, our program calculates the proper position and calibrates the Hololens IMU with the Polhemus position sensor. After this, the images are gathered by the Clarius probe and sent into the Unreal Engine 3D environment. These images are sent to the hololens in real time as we stream from the engine. One issue we ran into when developing this program was the fact that people are different sizes. Although everyone has similar anatomy, not every arm is the same length. In order for us to feel fully confident in our system, we would require a pre-calibration process so that the images displayed in the proper locations without fault.

Discussion

Achievements and Challenges

Our solution was achieved through incorporating several different systems into one. During our project, we accomplished several different aspects of the overall solution. We successfully set up the Clarius Cast API and were able to run the example code. This required us to work through some issues with the documentation and installation of some libraries. We used the QT Creator IDE to develop this project, and it was chosen because that is how the Clarius team created their example programs. We also successfully debugged an issue we were having with our application's connection to the Clarius wifi network. After we fixed our issues, the team was able to successfully implement the Cast API and stream the images onto a PC. These images are an imperative part of our system, because without them we would have no real product. Achieving the image streaming allowed us to continue development and eventually upload an image into the Unreal Engine.

The team was also able to collect live transformation data from the Polhemus Viper position sensor. A script written in ROS was provided to the team to use. Using Ubuntu-Linux, the script was compiled and run to collect live transformation data. The data was then uploaded in real time to a shared folder between Ubuntu-Linux and Windows. From there, the data was uploaded to the Microsoft Hololens 2. This positional data allowed the team to collect information to properly position the image. This is important because in order to overlay the image onto a patient's respective anatomy, we must first determine the position of the ultrasound probe. Without this positional data, we would not be able to determine where the patient, probe, or technician was in relation to the virtual environment in Unreal Engine. Live transformation

data collected was successfully used to orient a single ultrasound image over the patient's respective anatomy through the Microsoft Hololens 2.

Although the team was successful with the vast majority of tasks that they set out to accomplish, they did run into some significant challenges. The biggest issue that we ran into was that the The Cast API provided by Clarius was very poorly documented. The biggest issue we ran into was having to install QT binaries. QT Creator is the IDE we used to develop this project, and it was chosen because of this issue. We had to use the IDE in order to properly run the example code to start. However, when we did this, we realized that the connection to the Clarius probe was not being properly established. This issue persisted for a couple weeks, as we tried different solutions and adapted our code accordingly. Eventually we had to post an issue to the Clarius Github forum and worked with one of their representatives to get the program running. We discovered that we needed to disable our computer's firewall, because the connection to the probe was not secure. The Windows security system is very specific about the connections you can make to networks, and it was blocking us from connecting to the probe's wifi network. After this, the team was able to successfully implement the Cast API and stream the images onto a PC. These images were then saved into a folder and uploaded to the Microsoft Hololens 2.

The system that we created has two primary limitations. Firstly, the ultrasound probe creates its own wifi network to connect with the cast API. Unfortunately, this network cannot also be shared with the Hololens because it requires a wifi connection to stream the 3D environment from Unreal Engine. To remedy this problem, the team attempted to implement two wifi networks through use of a Raspberry Pi, a wifi card, and ethernet. However, the Clarius probe was unable to be connected as a secondary wifi network rather than a primary one. The team was unable to overcome this limitation in the time provided. This resulted in the inability to

stream real time ultrasound data into the Hololens. We were still able to stream this data into Unreal Engine and apply the appropriate transformations, but in order to have a working prototype with the AR headset, a pre-recorded set of ultrasound images was used.

The second limitation the team ran into was the inability to correctly calibrate the Hololens spatial mapping system to the patient's anatomy. The team heavily relied upon functionality packaged with the Hololens to map the virtual world to the real world. This was to avoid the need to perform complicated transformations from the position sensor to the Hololens every frame. Instead, the position sensor data was piped into the virtual environment in Unreal Engine and we relied on the Hololens to map that appropriately to the real world. The accuracy of the default mapping we used seemed to vary based on user and degradation over time occurred if users swapped in the middle of program execution.

Overall, the team was able to construct a system that combined all of the different parts required to create a powerful AR ultrasound diagnostic tool. We innovated upon pre-existing ultrasound technology in order to make training more straightforward and assist sonographers with developing the necessary hand-eye coordination to perform exams. We also created a functional prototype that included most of the capabilities required. While there is more work that needs to be done to bridge the network issues and improve the spatial mapping features of this software, the team created a strong foundation for future projects utilizing AR and medical imaging.

Future Extensions

Our system is the foundation for future advancements in the visualization of image analysis. We hope that this project will be expanded into a system which can create several key

visualizations of a patient's anatomy. AR is such a new technology, and it can be used for a plethora of different visualizations. There are a lot of potential routes we can take to expand upon this system. For example, 3D rendering and visualization of internal systems. We theorize that this system can be achieved through multiple swipes over a patient's anatomy which can be layered and transposed to convey a deeper meaning. Another application that we hope to implement is generating 3D visualizations of internal systems in real time. This will provide ultrasound technicians with a greater understanding of the internal systems and will improve accuracy of diagnosis. A visual for these future expansions is provided below in Figure 3.

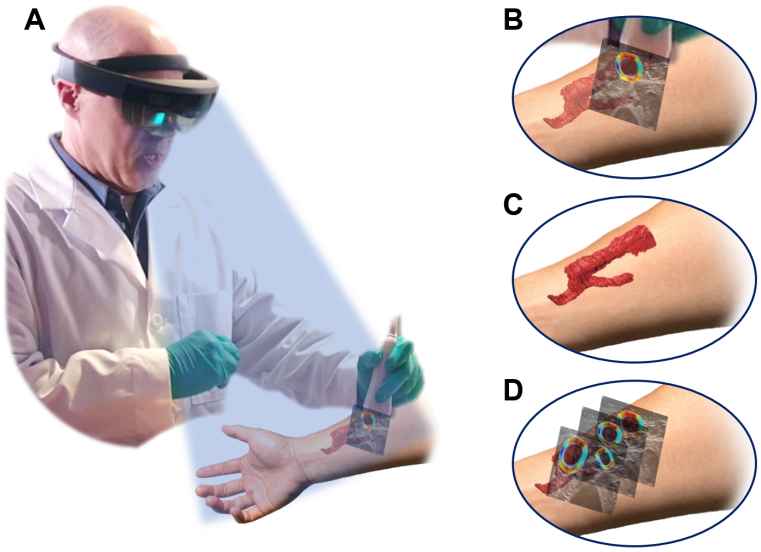

Figure 3. Potential future expansion of AR and ultrasound imaging

Broader Impacts

Although there are private companies and institutions that aim to use AR for ultrasound imaging, our goal was to create a baseline system to provide to the community. We believe that AR can provide ultrasound technicians with a greater understanding of internal human systems. Continuing implementation of AR in ultrasound imaging can lead to advanced diagnostic

techniques. This system can be expanded to include a color gradient in the ultrasound image showing blood flow or pressure, or a highlight of a tumor. These enhancements in ultrasound imaging will improve the accuracy and diagnosis, ultimately improving patient experience. We hope to positively impact the community by providing our solution to them free of charge. In our efforts, we also hope to encourage other corporations to make their software available to individuals who are curious just like us.

Acknowledgments

We would like to say thank you to our advisor, Professor Yihao Zheng, for all his help in the development of this project. A special thank you to Rohit Dey for his assistance in working with the Polhemus position sensor and Clarius probe. We are also grateful for the Office of Information Technology for providing us with the Microsoft Hololens 2.

References

Augmented reality brings ultrasound into a new dimension. (n.d.). Retrieved April 26, 2022, from https://tectales.com/ar-vr/augmented-reality-brings-medical-ultrasound-into-a-new-dimension.html

Clarius | Portable Pocket Handheld Ultrasound Scanners. (n.d.). Retrieved April 12, 2022, from https://clarius.com/

Clariusdev, C. (2021, March 2). Clariusdev/Cast: Cast Api and associated artifacts. GitHub. Retrieved April 28, 2022, from https://github.com/clariusdev/cast

Divecha, Devin (2011) “Augmented Reality (AR) used in architecture and design” Retrieved April 11th, 2022.

Edwards, E. E. (n.d.). All About Position Sensors (Types, Uses and Specs). ThomasNet. Retrieved April 16, 2022, from https://www.thomasnet.com/articles/instruments-controls/all-about-position-sensors/

Hawkins, Mathew (2011). “Augmented Reality Used To Enhance Both Pool And Air Hockey Game Set Watch” Retrieved April 11th, 2022. (Hawkins, 2011; Lintern, 1980; Marr, n.d)

HoloLens 2—Overview, Features, and Specs | Microsoft HoloLens. (n.d.). Retrieved April 12, 2022, from https://www.microsoft.com/en-us/hololens/hardware

Lintern, Gavan (1980). "Transfer of landing skill after training with supplementary visual cues". Human Factors. 22 (1): 81–88. doi:10.1177/001872088002200109. PMID 7364448. S2CID 113087380.

Marr, B. (n.d.). *The Fascinating History And Evolution Of Extended Reality (XR) – Covering AR, VR And MR*. Forbes. Retrieved April 26, 2022, from

https://www.forbes.com/sites/bernardmarr/2021/05/17/the-fascinating-history-and-evolution-of-extended-reality-xr--covering-ar-vr-and-mr/

Mulesoft. (n.d.). What is an API? (Application Programming Interface). Retrieved April 10, 2022, from https://www.mulesoft.com/resources/api/what-is-an-api

Radiology (ACR), R. S. of N. A. (RSNA) and A. C. of. (n.d.). *General Ultrasound*. Radiologyinfo.Org. Retrieved April 12, 2022, from https://www.radiologyinfo.org/en/info/genus

Red Hat. (2018, January 8). What is an IDE? Retrieved April 15, 2022, from https://www.redhat.com/en/topics/middleware/what-is-ide

Rüger, C., Feufel, M. A., Moosburner, S., Özbek, C., Pratschke, J., & Sauer, I. M. (2020). Ultrasound in augmented reality: a mixed-methods evaluation of head-mounted displays in image-guided interventions. *International Journal of Computer Assisted Radiology and Surgery*, *15*(11), 1895–1905. https://doi.org/10.1007/s11548-020-02236-6

The History of Fetal Ultrasound Treatment | Ultrasound Morristown NJ. (n.d.). Harding Radiology. Retrieved April 26, 2022, from https://hardingradiology.com/

Ultrasound. (n.d.). Retrieved April 12, 2022, from https://www.nibib.nih.gov/science-education/science-topics/ultrasound

Ultrasound - Mayo Clinic. (n.d.). Retrieved April 12, 2022, from https://www.mayoclinic.org/tests-procedures/ultrasound/about/pac-20395177

VIPER - Real Time Tracking Redefined. (n.d.). Retrieved April 12, 2022, from https://polhemus.com/viper/

Dawadi, R., Asghar, Z., & Pulli, P. (2017). *Internet of things controlled home objects for the elderly*. SCITEPRESS. Retrieved April 26, 2022, from https://www.scitepress.org/Papers/2017/61098/

*, N. (2017, July 14). *Electromagnetic tracking systems explained: Behind the scenes of VR/AR*. Premo S.L. Retrieved April 26, 2022, from https://3dcoil.grupopremo.com/blog/electromagnetic-tracking-systems-virtual-reality/

Appendix

Appendix A. UE4_Projection_Actor

Projection.h

```
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once



#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "Projection.generated.h"

UCLASS()
class MQP_427_API AProjection : public AActor
{
	GENERATED_BODY()

public:
	// Sets default values for this actor's properties
	AProjection();

	UPROPERTY(EditAnywhere)
	USceneComponent* m_root;
```

```
	UPROPERTY(EditAnywhere)
	UStaticMeshComponent* m_plane;



protected:
	// Called when the game starts or when spawned
	virtual void BeginPlay() override;

public:
	// Called every frame
	virtual void Tick(float DeltaTime) override;
private:
	void load_image(const char* path);
	void load_coordinates(const char* path);

};
```

Projection.cpp

```
// Fill out your copyright notice in the Description page of Project Settings.


#include "Projection.h"
#include "HAL/PlatformFilemanager.h"
```

```
	#include "Misc/FileHelper.h"
	#include "HAL/UnrealMemory.h"
	#include "PixelFormat.h"
	#include "Math/UnrealMathUtility.h"

	#include <cmath>
	#include <string>
	#include <stdio.h>
	// Sets default values
	AProjection::AProjection()
	{
		// Set this actor to call Tick() every frame.  You can turn this off to improve performance if you don't need it.
		PrimaryActorTick.bCanEverTick = true;


		m_root = CreateDefaultSubobject<USceneComponent>(TEXT("root"));
		m_plane = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("plane"));
		m_plane->AttachToComponent(m_root, FAttachmentTransformRules::KeepRelativeTransform);

	}

	// Called when the game starts or when spawned
	void AProjection::BeginPlay()
```

```
{
	Super::BeginPlay();
}

// Called every frame
void AProjection::Tick(float DeltaTime)
{
	load_image("/Game/working_ultrasound.raw");
	//this->SetActorRotation(FRotator(0, 90, 90));

	load_coordinates("test");
	Super::Tick(DeltaTime);
}
void AProjection::load_image(const char* path)
{
	FString file = FPaths::ProjectContentDir();
	file.Append(TEXT("working_ultrasound.raw"));
	// We will use this FileManager to deal with the file.
	IPlatformFile& FileManager =
FPlatformFileManager::Get().GetPlatformFile();
	TArray64<uint8> data;
	if (FileManager.FileExists(*file))
	{
//		UE_LOG(LogTemp, Warning, TEXT("%s"), *file)

		if (FFileHelper::LoadFileToArray(data, *file))
		{
```

```
                        UTexture2D* texture =
UTexture2D::CreateTransient(640, 480);
                        FTexture2DMipMap mip =
texture->PlatformData->Mips[0];
                        mip.BulkData.Lock(LOCK_READ_WRITE);
                        uint8* ImageData = (uint8*)
mip.BulkData.Realloc(data.Num() * sizeof(uint8));
                        uint8* loaded_data = (uint8*)data.GetData();

                        FMemory::Memcpy(ImageData, loaded_data,
data.Num());
                        //UE_LOG(LogTemp, Warning, TEXT("Image data size:
%d, "), sizeof(*RawImageData / 8))

                        mip.BulkData.Unlock();
                        texture->UpdateResource();

                        UMaterialInstanceDynamic* mat =
m_plane->CreateDynamicMaterialInstance(0);
                        if (mat != nullptr && texture != nullptr)
                        {


mat->SetTextureParameterValue(FName("input_texture"), texture);
                        }
                        else
                        {
```

```
                              UE_LOG(LogTemp, Warning, TEXT("File Exists
and array didn't work"))
                        }
                  }


            }
            else
            {
                  UE_LOG(LogTemp, Warning, TEXT("File Doesn't Exist"))
            }


      }
      void AProjection::load_coordinates(const char* path)
      {
            FString file =
FString(TEXT("../../../../../../vm_share/test.csv"));
            IPlatformFile& FileManager =
FPlatformFileManager::Get().GetPlatformFile();
            FString data;
            if (FileManager.FileExists(*file))
            {
                  if (FFileHelper::LoadFileToString(data, *file))
                  {
                        TArray<FString> char_array;
                        data.ParseIntoArray(char_array, TEXT(","), false);
                        float positions[6];
                        for (int i = 0; i <6; i++)
```

```
                        {
                              positions[i] =
FCString::Atof(*char_array[i]);
                              //UE_LOG(LogTemp, Warning, TEXT("%f"),
positions[i])
                        }
                        UE_LOG(LogTemp, Warning, TEXT("%f, %f, %f"),
positions[0], positions[1], positions[2])
                        float sqrt2 = sqrt(2) / 2;

this->SetActorLocation(FVector(100*(sqrt2*positions[0]+sqrt2*positions[1])
, 100 * (-sqrt2 * positions[0] + sqrt2 * positions[1]) , -100 *
positions[2]-5));
                        /*FRotator r =
FMath::RadiansToDegrees(FRotator(positions[3], positions[4],
(positions[5])));
                        r.Yaw += 90;
                        r.Roll += 45;*/
                        this->SetActorRotation(FRotator(0, 90, 90));

                  }
            }
            else
            {
                  UE_LOG(LogTemp, Warning, TEXT("FILE Doesn't exist"))
            }
      }
```

Appendix B. Primary ROS Script for Position Sensor

Data_expoter.py

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String
import tf2_ros
import csv
import tf2_msgs.msg
from pathlib import Path
from tf.transformations import euler_from_quaternion



def callback(data):
    now = rospy.get_time()
    # rospy.loginfo(rospy.get_caller_id() + "I heard %s", now)
    if now - callback.time >= 0.1:
        rospy.loginfo(rospy.get_caller_id() + "I heard %s", data)
        path = Path("/mnt/hgfs/vm_share")
        path = path / "test.csv"
        with open(path, 'w') as csvfile:
            csvwriter = csv.writer(csvfile)
            t = data.transforms[0].transform
            l_quat = [t.rotation.x, t.rotation.y, t.rotation.z,
t.rotation.w]
            (roll, pitch, yaw) = euler_from_quaternion(l_quat)
```

```
                csvwriter.writerow([t.translation.x, t.translation.y,
t.translation.z, roll, pitch, yaw])
            callback.time = now


    def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
        rospy.init_node('listener', anonymous=True)
        callback.time = rospy.get_time()
        rospy.Subscriber("tf", tf2_msgs.msg.TFMessage, callback)

        # spin() simply keeps python from exiting until this node is
stopped
        rospy.spin()


    if __name__ == '__main__':
        listener()
```

Appendix C. Clarius Cast API Image Streaming

```
#include <stdio.h>
```

```
#include <string>
#include <iostream>
#include <atomic>
#include <thread>
#include <fstream>
#include <string>
#include <string.h>

#ifdef _MSC_VER
#include <boost/program_options.hpp>
#else
#include <unistd.h>
#endif
#include <memory>
#include <vector>

#include <cast/cast.h>

#define PRINT           std::cout << std::endl
#define PRINTSL         std::cout << "\r"
#define ERROR           std::cerr << std::endl
#define FAILURE         (-1)
#define SUCCESS         (0)

static char* buffer_ = nullptr;
static int szRawData_ = 0;
static int counter_ = 0;
```

```
static std::vector<char> _image;


/// callback for error messages
/// @param[in] err the error message sent from the casting module
void errorFn(const char* err)
{
    ERROR << "error: " << err;
}

/// callback for freeze state change
/// @param[in] val the freeze state value, 1 = frozen, 0 = imaging
void freezeFn(int val)
{
    PRINT << (val ? "frozen" : "imaging");
    counter_ = 0;
}

/// callback for button press
/// @param[in] btn the button that was pressed, 0 = up, 1 = down
/// @param[in] clicks # of clicks used
void buttonFn(int btn, int clicks)
{
    PRINT << (btn ? "down" : "up") << " button pressed, clicks: " <<
clicks;
}
```

```
/// callback for readback progress
/// @param[in] progress the readback progress
void progressFn(int progress)
{
    PRINTSL << "downloading: " << progress << "%" << std::flush;
}

/// prints imu data
/// @param[in] npos the # of positional data points embedded with the
frame
/// @param[in] pos the buffer of positional data
void printImuData(int npos, const ClariusPosInfo* pos)
{
    for (auto i = 0; i < npos; i++)
    {
        PRINT << "imu: " << i << ", time: " << pos[i].tm;
        PRINT << "accel: " << pos[i].ax << "," << pos[i].ay << "," <<
pos[i].az;
        PRINT << "gyro: " << pos[i].gx << "," << pos[i].gy << "," <<
pos[i].gz;
        PRINT << "magnet: " << pos[i].mx << "," << pos[i].my << "," <<
pos[i].mz;
    }
}

/// callback for a new pre-scan converted data sent from the scanner
/// @param[in] newImage a pointer to the raw image bits
```

```
/// @param[in] nfo the image properties
/// @param[in] npos the # of positional data points embedded with the
frame
/// @param[in] pos the buffer of positional data
void newRawImageFn(const void* newImage, const ClariusRawImageInfo* nfo,
int npos, const ClariusPosInfo* pos)
{
#ifdef PRINTRAW
    if (nfo->rf)
        PRINT << "new rf data (" << newImage << "): " << nfo->lines << " x
" << nfo->samples << " @ " << nfo->bitsPerSample
          << "bits. @ " << nfo->axialSize << " microns per sample. imu
points: " << npos;
    else
        PRINT << "new pre-scan data (" << newImage << "): " << nfo->lines
<< " x " << nfo->samples << " @ " << nfo->bitsPerSample
          << "bits. @ " << nfo->axialSize << " microns per sample. imu
points: " << npos << " jpeg size: " << (int)nfo->jpeg;

    if (npos)
        printImuData(npos, pos);
#else
    (void)newImage;
    (void)nfo;
    (void)npos;
    (void)pos;
#endif
```

```
}

int imageNum = 0;

/// callback for a new image sent from the scanner
/// @param[in] newImage a pointer to the raw image bits
/// @param[in] nfo the image properties
/// @param[in] npos the # of positional data points embedded with the
frame
/// @param[in] pos the buffer of positional data
void newProcessedImageFn(const void* newImage, const
ClariusProcessedImageInfo* nfo, int npos, const ClariusPosInfo* pos)
{
    int sz = nfo->imageSize;
    // we need to perform a deep copy of the image data since we have to
post the event (yes this happens a lot with this api)
    if (_image.size() < static_cast<size_t>(sz))
        _image.resize(sz);
    memcpy(_image.data(), newImage, sz);

    // !!! filepaths
    std::string stringFN;
    std::string fn = "images/clariusimage";
    std::string numString = std::to_string(imageNum);
    std::string ext = ".raw";

    stringFN += numString + ext;
```

```
    int n = stringFN.length();

    // declaring character array
    char filename[n + 1];

    // copying the contents of the
    // string to char array
    strcpy(filename, stringFN.c_str());

    //char oldname[] =
"C:/Users/sadie/Documents/MQP/build-caster-Desktop_Qt_6_2_1_MinGW_64_bit-D
ebug/images/clariusimage.raw";
    //char newname[] =
"C:/Users/sadie/Documents/MQP/movedImages/clariusimage.raw";

    // !!!code
    FILE * fp;
    fp = fopen(filename, "wb");

    fwrite(newImage, sizeof(int32_t), nfo->width * nfo->height, fp);
    imageNum++;


    /*     deletes the file if exists
    if (rename(oldname, newname) != 0){
        perror("Error moving file");
```

```
    }
    */

    std::cout << "filename: " << filename;

    fclose(fp);


    std::cout << "file closed";

    PRINTSL << "new image (" << counter_++ << "): " << nfo->width << " x "
<< nfo->height << " @ " << nfo->bitsPerPixel << " bpp. @ "
            << nfo->imageSize << "bytes. @ " << nfo->micronsPerPixel << "
microns per pixel. imu points: " << npos << std::flush;
}

/// callback for a new spectral image sent from the scanner
/// @param[in] newImage a pointer to the raw image bits
/// @param[in] nfo the image properties
void newSpectralImageFn(const void* newImage, const
ClariusSpectralImageInfo* nfo)
{
    (void)newImage;
    PRINTSL << "new spectrum: " << nfo->lines << " x " << nfo->samples <<
" @ " << nfo->bitsPerSample
          << "bits. @ " << nfo->period << " sec/line." << std::flush;
}
```

```
/// saves raw data from the current download buffer
/// @return success of the call
bool saveRawData()
{
    if (!szRawData_ || !buffer_)
        return false;

    auto cleanup = []()
    {
        free(buffer_);
        buffer_ = nullptr;
        szRawData_ = 0;
    };

    FILE* fp = nullptr;
    // save raw data to disk as a compressed file
    #ifdef _MSC_VER
        fopen_s(&fp, "raw_data.tar", "wb+");
    #else
        fp = fopen("raw_data.tar", "wb+");
    #endif
    if (!fp)
    {
        cleanup();
        return false;
    }
```

```
    fwrite(buffer_, szRawData_, 1, fp);
    fclose(fp);
    cleanup();
    return true;
}

/// processes the user input
/// @param[out] quit the exit flag
void processEventLoop(std::atomic_bool& quit)
{
    std::string cmd;

    while (std::getline(std::cin, cmd))
    {
        if (cmd == "Q" || cmd == "q")
        {
            quit = true;
            break;
        }
        else if (cmd == "F" || cmd == "f")
        {
            if (cusCastUserFunction(USER_FN_TOGGLE_FREEZE, 0, nullptr) <
0)
                ERROR << "error toggling freeze" << std::endl;
        }
        else if (cmd == "D")
```

```
{
    if (cusCastUserFunction(USER_FN_DEPTH_INC, 0, nullptr) < 0)
        ERROR << "error incrementing depth" << std::endl;
}
else if (cmd == "d")
{
    if (cusCastUserFunction(USER_FN_DEPTH_DEC, 0, nullptr) < 0)
        ERROR << "error decrementing depth" << std::endl;
}
else if (cmd == "G")
{
    if (cusCastUserFunction(USER_FN_GAIN_INC, 0, nullptr) < 0)
        ERROR << "error incrementing gain" << std::endl;
}
else if (cmd == "g")
{
    if (cusCastUserFunction(USER_FN_GAIN_DEC, 0, nullptr) < 0)
        ERROR << "error decrementing gain" << std::endl;
}
else if (cmd == "R" || cmd == "r")
{
    if (cusCastRequestRawData(0, 0, [](int sz)
    {
        if (sz < 0)
            ERROR << "error requesting raw data" << std::endl;
        else if (sz == 0)
        {
```

```
                    szRawData_ = 0;
                    ERROR << "no raw data buffered" << std::endl;
                }
                else
                {
                    szRawData_ = sz;
                    PRINT << "raw data file of size " << sz << "B ready to
download";
                }

            }) < 0)
                ERROR << "error requesting raw data" << std::endl;
        }
        else if (cmd == "Y" || cmd == "y")
        {
            if (szRawData_ <= 0)
                ERROR << "no raw data to download" << std::endl;
            else
            {
                buffer_ = (char*)malloc(szRawData_);

                if (cusCastReadRawData((void**)(&buffer_), [](int ret)
                {
                    if (ret == SUCCESS)
                    {
                        PRINT << "successfully downloaded raw data" <<
std::endl;
```

```
                        saveRawData();
                    }
                }) < 0)
                    ERROR << "error downloading raw data" << std::endl;
            }
        }
        else
        {
            PRINT << "valid commands: [q: quit]";
            PRINT << "       imaging: [f: freeze, d/D: depth, g/G: gain]";
            PRINT << "      raw data: [r: request, y: download]";
        }
    }
}

int init(int& argc, char** argv)
{
    const int width  = 640;
    const int height = 480;
    std::string keydir, ipAddr;
    unsigned int port = 0;

    // ensure console buffers are flushed automatically
    setvbuf(stdout, nullptr, _IONBF, 0) != 0 || setvbuf(stderr, nullptr,
_IONBF, 0);
```

```
    // Windows: Visual C++ doesn't have 'getopt' so use Boost's
program_options instead
#ifdef _MSC_VER
    namespace po = boost::program_options;
    keydir = "c:/";

    try
    {
        po::options_description desc("Usage: 192.168.1.21", 12345);
        desc.add_options()
            ("help", "produce help message")
            ("address", po::value<std::string>(&ipAddr)->required(), "set
the IP address of the host scanner")
            ("port", po::value<unsigned int>(&port)->required(), "set the
port of the host scanner")
            ("keydir",
po::value<std::string>(&keydir)->default_value("/tmp/"), "set the path
containing the security keys")
        ;

        po::variables_map vm;
        po::store(po::command_line_parser(argc,
argv).options(desc).allow_unregistered().run(), vm);

        if (vm.count("help"))
        {
            PRINT << desc << std::endl;
```

```
            return FAILURE;
        }

        po::notify(vm);
    }
    catch(std::exception& e)
    {
        ERROR << "Error: " << e.what() << std::endl;
        return FAILURE;
    }
    catch(...)
    {
        ERROR << "Unknown error!" << std::endl;
        return FAILURE;
    }
#else // every other platform has 'getopt' which we're using so as to not
pull in the Boost dependency

    //  !!!ip and port
    ipAddr = "192.168.1.1";
    port = 45921;
    keydir = "C:/keys";

    // check command line options
//    while ((o = getopt(argc, argv, "k:a:p:")) != -1)
//    {
//        switch (o)
```

```
//        {
//        // security key directory
//        case 'k': keydir = optarg; break;
//        // ip address
//        case 'a': ipAddr = optarg; break;
//        // port
//        case 'p':
//            try { port = std::stoi(optarg); }
//            catch (std::exception&) { PRINT << port; }
//            break;
//        // invalid argument
//        case '?': PRINT << "invalid argument, valid options: -a [addr],
-p [port], -k [keydir]"; break;
//        default: break;
//        }



    if (!ipAddr.size())
    {
       ERROR << "no ip address provided. run with '-a [addr]" <<
std::endl;
        return FAILURE;
    }

    if (!port)
    {
```

```
        ERROR << "no casting port provided. run with '-p [port]" <<
std::endl;
        return FAILURE;
    }
#endif

    PRINT << "starting caster...";

    // initialize with callbacks
    if (cusCastInit(argc, argv, keydir.c_str(), newProcessedImageFn,
newRawImageFn, newSpectralImageFn, freezeFn, buttonFn, progressFn,
errorFn, width, height) < 0)
    {
        ERROR << "could not initialize caster" << std::endl;
        return FAILURE;
    }
    if (cusCastConnect(ipAddr.c_str(), port, [](int ret)
    {
        if (ret == FAILURE)
            ERROR << "could not connect to scanner" << std::endl;
        else
            PRINT << "...connected, streaming port: " << ret;

    }) < 0)
    {
        ERROR << "connection attempt failed" << std::endl;
        return FAILURE;
```

```
    }

    return 0;
}

/// main entry point
/// @param[in] argc # of program arguments
/// @param[in] argv list of arguments
int main(int argc, char* argv[])
{
    int rcode = init(argc, argv);

    if (rcode == SUCCESS)
    {
        std::atomic_bool quitFlag(false);
        std::thread eventLoop(processEventLoop, std::ref(quitFlag));
        eventLoop.join();
    }

    cusCastDestroy();
    return rcode;
    return 0;}
```