IMPLEMENTING AN ENHANCED MOTION TRACKING SYSTEM FOR VIRTUAL REALITY USING 'COMMON OFF-THE-SHELF' ITEMS

By Sean Michael O'Hara

A thesis submitted to the faculty of The University of Mississippi in partial fulfillment of the requirements of the Sally McDonnell Barksdale Honors College.

Oxford
May 2016

Approved by:

_____

Advisor: Dr. J. Adam Jones

_____

Reader: Dr. Dawn E. Wilkins

_____

Reader: Dr. Philip J. Rhodes

ABSTRACT
SEAN MICHAEL O'HARA: Implementation of an Enhanced System of Motion Tracking for Virtual Reality Using 'Common Off-the-Shelf' Items
(Under the direction of Dr. J. Adam Jones)

My project was tasked and guided by Dr. Adam Jones of the University of Mississippi's Computer Science Department to design and implement an improved system of infrared motion tracking for virtual reality. The enhancements allow Dr. Jones to use motion tracking for VR in ways that he previously could not, and these improvements include the ability to track multiple objects along with a foundation for expanding my project's use to work with a multiple camera setup for motion tracking. The hardware used in my project made use of Dr. Jones' supply of technology for development and research within the area of Virtual Reality, and in that regard, all software and hardware used within my project were 'common off-the-shelf'. The completed system is an addition to Dr. Jones' setup, especially in research with Augmented Virtual Reality. It contributes to Dr. Jones' research by implementing a customized system of motion tracking that is comparable to that of more expensive, commercial models.

Table of Contents

List of Figures

List of Terms

Application Programming Interface (API) – an application programming interface is a framework of tools and protocols that serve computer programmers in working with software

Common Off-the-Shelf (COTS) – readily available and standard manufactured products as opposed to customized products

Graphical User Interface (GUI) – a graphical user interface is a type of interface used in computer science that allows for interaction between users and electronic devices through graphical icons and visual indicators instead of text-based interaction or command labels

Light Emitting Diode (LED) – is a two-lead semiconductor light source that emits light when enough voltage is carried to power it

Integrated Development Environment (IDE) – an integrated development environment is an application which provides computer programmers a helpful environment to work in software development

Open Source Virtual Reality (OSVR) – a term used to describe hardware and software for open-source use in gaming and development with virtual reality

Virtual Reality (VR) – virtual reality is a computer-simulated environment meant to immerse the user through interaction

1 Introduction

1.1 Problem

Dr. Adam Jones is an assistant professor of Computer Science at the University of Mississippi who researches various aspects of Virtual Reality, and one of his primary areas of research in this field is developing a deeper understanding, as well as enhancing, the general nature of motion tracking. I developed this project in order to create a customized motion tracking system capable of tracking multiple objects for Dr. Jones's research into VR implementation. Also, I researched into the aspects needed for the tracking system to be upgraded even further for a multiple camera usage. With these enhancements merged into the tracking system, Unity's three-dimensional environment can be used for tests and development with tracking data from multiple objects.

1.2 Solution

With the help of Dr. Jones, I developed an enhanced system of motion tracking for use with Virtual Reality. I implemented the motion tracking system by utilizing open-source tracking software in combination with 'common off-the-shelf' items from the VR lab. The new enhanced system of motion tracking pushes the boundaries of the system, giving the ability to distinguish and track the movement of multiple users. This customized implementation involved using relatively inexpensive, infrared cameras in combination with an open-source tracking application to merge and enhance the tracking setup with added functionality. My project also involved researching into the possible application of a multiple camera setup within the tracking system. Although my project

doesn't fully implement the capability for tracking with multiple cameras, it does have specific ideas and places specified within the code where the project could be altered to have multiple camera functionality. However, there were time constraints as wells as some added complexities within the existing code, so I shifted the multiple camera content of the project towards a more research-oriented component for Dr. Jones, giving him a foundation to work with since there wasn't enough time to fully implement.

2 Tools

2.1. FaceTrackNoIR

FaceTrackNoIR is an open-source, just recently turned commercial, program that was created with the idea in mind of making computer gaming more accessible and fun. There have been non-commercial, as well as commercial face-tracking programs around for some time now, but FaceTrackNoIR has a specificity to it which is ideal for this project. While most programs are costly or involve a complicated setup or a predefined system, FaceTrackNoIR is a flexible application which can be used with a variety of different tracking systems and game protocols. It was created with a developmental outlook of accessibility for the user and developer, and anyone with a decent laptop and built-in webcam can make use of its tracking capabilities. So, it was the perfect foundation for my project.

There are other tracking applications we could have used such as free, non-robust applications like Open Track, or commercial positional trackers like Vicon, however, these options were either costly, or incapable of being modified for developing beyond their current capabilities. So FaceTrackNoIR, with it's robust internal system and implementation using common hardware, made it the ideal application.

2.2 Open Source Virtual Reality Positional Tracking Unit

The Open Source Virtual Reality (OSVR) Positional Tracking Unit is an infrared camera that comes along with the OSVR Hacker Development kit. It was created and is distributed by a company called Razor, who strive to be innovative 'for gamers and by gamers'

in the sense that they wish for their OSVR kit and all products to generate a progressively constructive mindset towards VR gaming and development. Not only is the package meant for wide range of uses, but the design of the hardware is also completely open-source, which makes it a perfect fit for building a customized motion tracking program. Its design is a mix of fairly simple system requirements, so the kit can be used with a decent laptop. More importantly, the software and hardware design is an ideal platform for upgrading and enhancing any VR setup. The camera is included within a package intended for gamers and developers who are either well-versed or maybe just interested in Virtual Reality, and it's various open-source features are encouraged by Razor for of any kind of use in VR. So as a student implementing it within Dr. Jones's proposed system, in a customized fashion, I used the camera for exactly what it was intended in terms of design and developmental capabilities.

2.3 Arduino Uno Microcontroller

The Arduino Uno is a relatively simple and cost-effective microcontroller. It makes use of Arduino's own integrated development environment and offers a simple interaction for uploading code. It has a wide range of capabilities, but in my project I used the development board as a communication tool between the users and the tracking program. By using LED three point lighting and uploading code to the Arduinos that controlled their communication, which I will fully explain later, the controllers functioned as a sort of power source for the users. In this manner, readying the microcontroller signifies the beginning of the interaction for the tracking system.

2.4. “common off-the-shelf” technology

Common off-the-shelf, (COTS), technology describes software or hardware that is easily accessible and available for use by the general public. They are items that are designed and built with hopes for cost-reduction and a reduced need for long-term costs of maintenance. In my project, I won't be directly modifying any of the 'common off-the-shelf' technology that is being used. I'll be creating a hybrid model with customized capabilities in order to merge and enhance the tracking system.

In my project, 'common off the shelf' technology was utilized within a prototype framework and limited budget. Once again, none of the actual software or hardware was directly modified, so the project wasn't necessarily a customized COTS product, but the idea for merging and enhancing the tracking system is the integration of existing systems by utilizing a custom software wrapper and a UDP Client/Server system. The end product is a customized addition to the motion tracking system, and this customization process is a fairly common area of programming work as a software consultant. Software engineering in this manner is an interesting way of symbiotic learning between the programmer and client that I thoroughly enjoyed. As a student about to set out into the real world of work in programming, it was refreshing to experience firsthand just how rewarding and fun working with intelligent people can be.

2.5. Java

Java is a programming language that was designed by James Gosling with the intention of giving programmers the power of application implementation with as few dependencies as

possible. Developers are able to write code that is able to be “written once, run anywhere”. Java is also the programming language with which I am most well acquainted. For this reason, I used my Java capabilities to create the UDP server in order to communicate with the the tracking application. Originally, I planned on using C sharp to code the UDP server, since I also used C sharp elsewhere within my project with the Unity script.

However, the code for the UDP Server turned out to be even more complicated than I first expected, especially with figuring out issues in the data being sent from the tracking application. So, I changed my coding platform yet again, taking what I learned from my work in Java in regards to developing the UDP server, and moved on to working with C.

2.6. C

C is an imperative, procedural computer programming language created by Dennis M. Ritchie in 1972 and intended for general-purpose use in the UNIX operating system. C is one of the most popular programming languages for it's low-level interaction within memory usage, and for this reason exactly, I used C within my project's UDP Server to specifically allocate memory for the tracking data transmitted from FaceTrackNoIR. Originally I hadn't even planned to use C in my project, but in the end I learned enough to be able to navigate through fairly complex network coding. Also, by scrapping my old plan and embracing a new reactionary response to the C-heavy shape that my project took, I came away with a satisfied sense of accomplishment for having pushed through an extremely frustrating barrier in my project's development.

2.7. C#

C sharp (C#) is a programming language that was developed by Microsoft to be a multi-paradigm language. It is a strongly typed language with an object-oriented, general purpose approach that emphasizes a long-lasting, efficient, and robust relationship between the programmer and the product.

C sharp is an ideal language for use in developing distributed software components, and this is one of the reasons I chose to use it in my project. It is also the innate language for use in Unity's three dimensional, development environment, and on top of that, Dr. Jones preferred a C sharp implementation on the front end of interaction due to his familiarity and satisfaction with the language in previous research with Virtual Reality.

2.8. UDP Server

User Datagram Protocol is a network transportation protocol that is sometimes used instead of a TCP server for its small, quick, and also, partially reliable handling of data. A UDP Server does not offer an absolute guarantee on its delivery of datagram packets, but my project can afford to lose some data and the general “postcard” nature of UDP makes it ideal for the communication framework. UDP is usually utilized to send relatively small amounts of data with real-time specificity. My project will be keeping a real-time communication with the UDP Server as it constantly receives information from the cameras via the tracking application and subsequently updates the Unity client-side script by sending out data through a shared port.

The motion tracking application, FaceTrackNoIR also has a built in UDP streamer, so

choosing to work with a UDP server for communication with the tracking was an implementation in alignment with FaceTrackNoIR's design.

2.9. LED Point System

In order to avoid the problems of background light, the motion trackers used a three-point tracking system with infrared LEDs installed on top of simple black boxes that served to contain the LEDs' power sources. The circuit itself is a system that connects the three LEDs with a nine volt battery, and the system draws power through a ten ohm resistor on the front end of its wiring. Infrared LEDs were the ideal sources of focus for tracking because they emit an invisible amount of light that can be separated easily from its surroundings.

The LEDs were inexpensive and easily replaceable, and I used the Arduino Uno microcontrollers as a power source for the LEDs and also as a way to differentiate users. This is an aspect which I will further explain later, but for now, an important note in describing the use of LED point systems within my project is that the separate LED setups have a symbiotic nature with one another. By this, I mean that they check for one another to be working, so if one Arduino is not powered, then the other microcontroller will not send power to its LEDs until its partner is also powered and ready.

2.10. Virtual Machine

A virtual machine is a term used in computing that refers to an emulator for a computer system. These virtual machines can function in a multitude of ways, with different

implementations that entail all sorts of hardware and software combinations. In my project, I used a virtual machine known as VirtualBox that ran on my laptop, giving me an integrated, dual-computer setup so that two instances of the tracking application could be used simultaneously.

Originally, we had planned on having both instances of the tracking application running on the same computer, or using two cameras within one instance of the application, however, the setup crashed with both arrangements, thus requiring a virtual machine to make efficient use of the tracking application's shortcomings. VirtualBox is the virtual machine that I've used before so I decide to use it with a Windows operating system in order to provide a functional platform for utilizing two cameras with different instances of the tracking application.

2.11. Eclipse

Eclipse is the integrated development that I first used for developing the UDP Server. It's innate capabilities with Java made it the ideal environment for error-debugging and establishing an initial connection with the server. I considered using the command prompt along with a text editor so that port numbers could be specified before run time, but using a hard-coded implementation of port declaration without the need for user interaction ended up accomplishing the same goal of specified port numbers.

Also, Eclipse was a great help in error-checking and debugging my server. It has an extremely helpful error checking system which displays a stack trace for any errors that are found, and it also gives precautionary warnings for pieces of code which may cause

compilation problems. My project ended up moving away from Eclipse as it's basic foundation for development, but working within it's environment in the beginning of my project was an essential part of development that got me started on the networking areas of programming which I needed to implement for my project to function.

2.12. Unity

Unity is a game engine that gives programmers and gamers an environment for developing all sorts of games and virtual reality. Its design is cross-platform so developers have control over the delivery to different devices, and this accessibility is one of the primary ideals in Unity's design along with the basic idea of making game development possible for anyone. Their goal is to promote a push towards a more generally creative outlook in gaming that relies on the entire community for technological progress rather than just a consumer-based mindset. Unity is a development environment with consistently accessible ideals and is specifically used by independent developers with small budgets because it makes the most of what is available in an extremely efficient manner by finding the right mix of “not too much or too little.”

It functions in a manner such that there is an initial Start function which Unity calls on the programs start-up. It then has an update function which it calls once every frame to update the script within its three-dimensional environment.

3 Design and Development

3.1. Development Environments and Design Overview

I used my laptop with multiple internal environments in developing the project. Eclipse was the first integrated development environment used for developing a communication between the motion tracking application and my UDP server, but then I moved on to a basic text editor for coding the server in C. I also worked with Visual Studio in developing the Unity script part of my project. The hardware development environment which I utilized was a combination of easy-to-use technology including the Arduinos and tracking cameras that I worked with in the multimedia research lab as well as my own home. My laptop also had a virtual machine at the ready so that two instances of the motion tracking application could be functioning simultaneously.

The initial research and work with the tracking application was done on my own so that I could grow some familiarity with the camera setup along with the motion tracking application, FaceTrackNoIR. In this stage, configuring the LED point system with the Arduino turned out to be a challenge. I was completely unfamiliar with any of the hardware or software involved in my project so most of my problems were from a basic lack of understanding for my project's components.

The next stage was developing a UDP server to communicate with the motion tracking application, and in this stage along with the rest of my project, Dr. Jones and I were in direct communication almost constantly through the development process. I would do research and work on my own, but then I would bring in work and findings to Dr. Jones to ensure their

consistency with the plan for development, and also, to further develop my own understanding for his ideas and hopes regarding the project's outcome. Making sense of data being sent from the FaceTrackNoIR application turned out to be one of the most difficult aspects of my project, and I'll explain that in more detail later on.

After this stage of establishing a program in direct communication with the motion tracking application, we moved on to implementing a multiple user setup in the multimedia research lab. This was an extremely interesting part for me since I was spending time in a room stocked full of hardware and gadgets that Dr. Jones uses in his work with Virtual Reality. The multiple users were accounted for in such ways that I'll go into later, and after we developed a framework for detecting the tracking data for multiple users, I had the information needed to reach the final stage of development within Unity.

3.2. Interaction with the FaceTracking application

The motion tracking application, FaceTrackNoIR has a specific configuration needed to communicate effectively with my project. Figure 1 on the following page shows the basic layout and the setup needed within the application's GUI layout. I'll also give a short explanation to describe how the application was set up to communicate with my project, and more specifically, my UDP server. The tracker source used a Point Tracker 1.1 source setting so that the tracking system works by detecting a three point object orientation. The filter needs to be set to none, even though Figure 1 has the Accela filter added on, which ended up causing problems within the multiple user detection. Lastly, the game protocol needs to be set to FaceTrackNoIR's own protocol so that it's UDP streaming capabilities can be accessed.

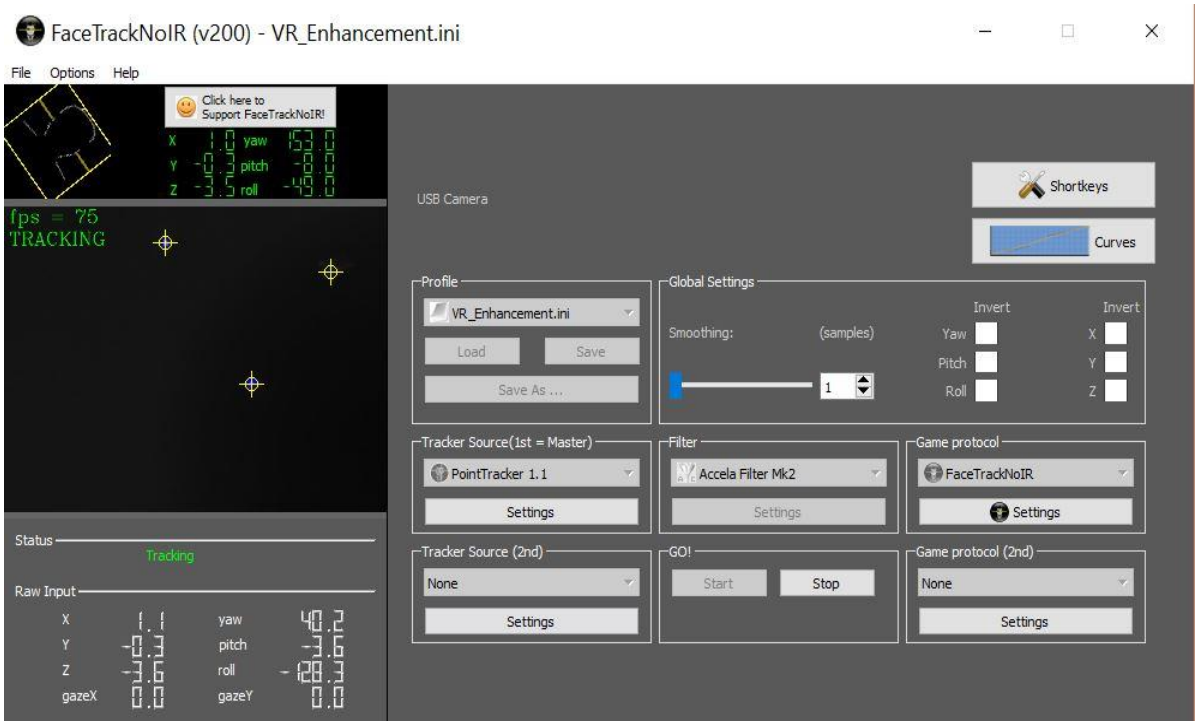


FIGURE 1 – FaceTrackNoIR General Layout and Settings

After experimenting with FaceTrackNoIR and gaining some familiarity in the very early stages of my project, I began using a Java UDP framework to receive data from the motion tracking camera through the application. This stage of development turned out to be one of the most difficult parts of my project, and without the help of Dr. Jones, I honestly don't know if I would have been successful in figuring out the solution. So, I worked on changing the code to fit my project's needs, and in the process, I gave myself a much needed refresher in the aspects of socket programming. My Java server was functional to the point of receiving data from FaceTrackNoIR, but for some reason, there were patterns of zero that made absolutely no sense in the context of the motion tracking application's data. Something was occurring in the conversions, and I think the problem either was caused by Java's API, or there

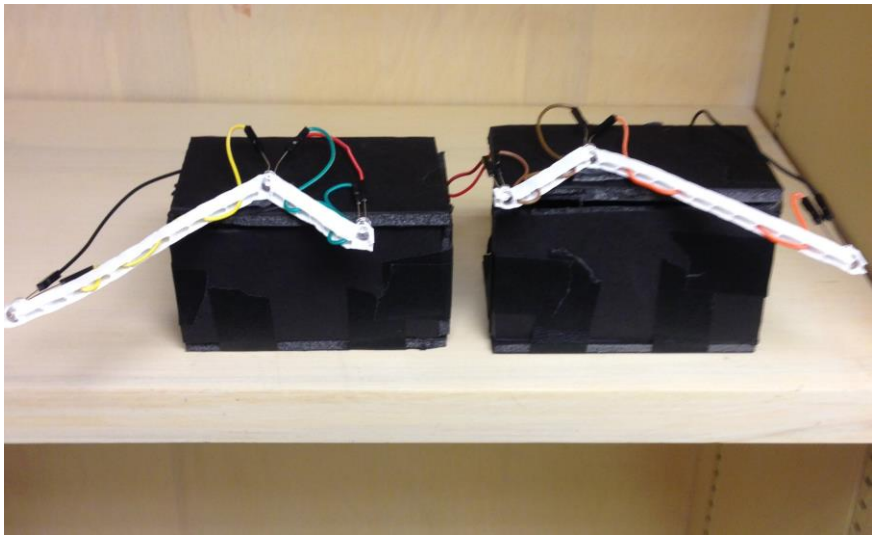
was a small amount of data loss in my UDP connection that my Java server just couldn't handle.

So, with Dr. Jones' help, I developed an entirely new server in the C programming language. I was not as familiar with this language, but thankfully Dr. Jones could help in lending some understanding to the areas where I had trouble. The primary of these areas, which also was the defining reason for C being the ideal language for coding my server as opposed to Java, was the specific memory allocation within the C server that set a certain amount of space in memory to receive the data from FaceTrackNoIR. This packet turned out to be an array of six doubles, with the first three places correlating to the three dimensions of x, y, and z, while the second group of three pertain to the yaw, pitch, and roll. This second group of three numbers is angular data that helps orient an object's tracking points within a three-dimensional area of movement.

3.3. Developing a System for Tracking Multiple Users

The primary enhancement to Dr. Jones's system of motion tracking for Virtual Reality involved merging the capability to detect and track multiple users with the tracking system by utilizing the 'common, off-the-shelf technology' at hand in the VR lab. In order for my project to integrate a system of detecting multiple users, I implemented a system of motion tracking which utilizes infrared tracking along with multiple configurations of LEDs. These LEDs were held in place, with a structure that Dr. Jones created, on top of black boxes that contained the corresponding Arduino microcontrollers that sent alternating power to the LEDs and functioned as a power source within my motion tracking framework.

FIGURE 2 – Black Boxes for Arduino Storage and LED Attachment



I built the boxes which stored the Arduinos and Dr. Jones also gave me an attachment for a portable nine volt power supply. So I cut wire to attach onto the Arduino and folded the entire micro-controller within the box so that it's lights wouldn't interfere with the LED tracking. Infrared tracking was chosen on account of its specificity with LED sources, however there was still some lack of efficiency with the LED point system on account of light distractions interfering with the readings, hence, the black box to contain the Arduinos and to ensure that no surrounding light sources are taken into account. The LEDs' points were still a bit faint in their readings from FaceTrackNoIR after the black box installation, so we went a step further and Dr. Jones cut off some of the covering material which holds the LEDs in place

so that as much of their light could be seen as possible.

With the LED point sources ready for effective use, Dr. Jones helped me with developing a script written in Arduino's own IDE that functioned as a blueprint for the interaction between the two point systems. By alternating the LED's power at a rate of 30 hz, which was specified within the Arduino script, we implemented a system of temporal multiplexing that distinguished each user by corresponding them to to each point system of lights. This caused a system of extremely fast blinking between the two LED point systems so that the FaceTrackNoIR positional readings could alternate.

FIGURE 3 – Arduino Code for Temporal Multiplexing

```
void loop(){
  unsigned long time = millis();

  Mirf.setTADDR((byte *)"serv1");

  Mirf.send((byte *)&toggle);

  while(Mirf.isSending()){
  }
  while(!Mirf.dataReady()){
```

FIGURE 3 - continued

```
Mirf.getData((byte *) &time);
if (toggle==255)
  {
      toggle=0;
      digitalWrite(2, LOW);
  }
else
  {
      toggle=255;
```

3.4. Multiple Camera Setup

My project was also concerned with researching into the possibility of adding a second camera into the tracking system so that the volume of the tracking space could be increased. Originally I had planned on this being a functional implementation within my project, but after working with my UDP server in communication with one instance of FaceTrackNoIR's UDP streamer, it soon became apparent that tracking from two separate cameras would create some problems within my project that were beyond my time constraints. However, my research for implementation is still usable within the context of my project if Dr. Jones wishes to expand upon my work.

In order to have multiple cameras in communication with one another, a virtual machine is needed to run another instance of the tracking application with the second camera.

The port would need to be changed within FaceTrackNoIR, but besides that, the settings would be the same. Next, the code within my C server would need to be updated so that it has ports, sockets, and variables that correspond with the second tracking instance, and most of this could be just simple code duplication with labeling changes. The tracking data itself would also need to translated so that it could be used in combination with the first camera's data, but if the camera is set at a ninety degree angle at the same height as the other camera, then the rotation needed would only involve a swap between the x and z coordinates.

After adjusting for the second camera within the C Server's networking design and data organization, the Unity script could be updated to contain another camera object, or it could just use the second camera's data along with first camera in order to create a larger volume of movement.

3.5. Server Side Code

My UDP Server that received data from FaceTrackNoIR was coded in C after my initial Java framework proved ineffective, but I modeled the connection in a similar manner besides making the adjustments needed for correct memory allocation. The sockets are instantiated and opened in the driver while the blocks of code for retrieving the tracking data and sending it out were placed into their own respective functions and called at the beginning of the server's connection loop. I separated these functions for the sake of neatness and efficient code since these processes involved large amounts of code space.

So I kept the while loop to a fairly simple design while the bulk of the multiple user processing took place in the function for returning data from FaceTrackNoIR's UDP Streamer.

This code is shown in Figure 4. Before the program enters this code block, the memory is allocated and the buffer is read into six pointers that correspond to specific patterns within the UDP's buffer size. Then these points are stored in an array for sending out into Unity.

FIGURE 4 – UDP Server Side Code for Multiple User Detection

```
// if statement since initial read won't have anything previously stored
if(u[0] == null){
    v[1] = 1;
    currentUser = 1;
}
else{
xd = u[0]-v[2];
yd = u[1]-v[3];
zd = u[2]-v[4];
distance = sqrt(xd*xd + yd*yd + zd*zd); // 3-D distance calculation

    if(distance < 1)    // checks for change in distance between,
    {                   //  if less than 1 meter, than it's the same user
        v[1]=currentUser;
    }
    else if(distance > 1)   // an else if for my initial if statement to turn to
    {                       // if the distance is greater than 1, meaning a different object is being detected
        if(currentUser == 1)
            currentUser = 2;
        else
            currentUser =1;
        v[1] = currentUser;


    }

}
```

Before the array is returned to the driver and sent out, the values are put into a three-dimensional distance formula that is used to compare subsequent readings so that the temporal multiplexing with the LED's can be established and multiple users can be distinguished. The differences of data are compared to the distance of 10 cm, which serves as a basis for managing the space between tracking objects. If the distance between subsequent readings is greater than the specified distance, then the program alternates the object's second array spot [1] that corresponds to whichever object's data is being read. The first spot [0] in the array sent to Unity is left unused so that Dr. Jones could have an empty array block for specifying different cameras. After the positional data is organized and stored in the array for sending to Unity, the pointer variables are then freed with their respective values still held in the array to ensure safe memory usage and safekeeping. Finally the function returns to the connection loop, the tracking data is sent to Unity, and the process repeats until the connection is ended.

3.6. Unity Script

In the final stage of my project, I developed a client-side UDP connection within Unity that received the organized user information from my C server and applied this data into Unity's three-dimensional game objects. This coding was done in C sharp, which is a language I was completely unfamiliar with before this project. So while I worked on other areas of my project, I found time to research into coding with C sharp by watching tutorials and experimenting with some basic work in Unity. After I built my C server up to the point where it was was functionally receiving information from FaceTrackNoIR, then I began creating a socket connection within Unity. This turned out to be another rather difficult area of project,

yet again involving work with networking, socket programming, and packet decoding in a language that I was unversed in before beginning on this project. Unity works by calling a Start() function to begin the C# script, which was where I initialized the socket connection and a queue for averaging out every five readings for each object. Then, Unity calls an Update function which reads positional data and manages the queue for every frame once the script has started. At the end of the update function, my script takes the tracking data and applies it to either game object depending on the packet's user specification.

FIGURE 5 – Unity Script

```
    void Update(){
            if(user == 1) {
                if(objPos.Count == 0) {
                objPos.Enqueue(new Vector3(xPos, yPos, zPos));
                 }
                else{
                Vector3 objAvgPos = new Vector3(0.0f, 0.0f, 0.0f);
                    for (int i = 0; i< objPos.Count; i ++)
                     {
                    objAvgPos += (Vector3)(objPos.ToArray())[i];
                     }
                objAvgPos /= objPos.Count;

                Vector3 objCurPos = new Vector3(xPos, yPos, zPos);
                if(Mathf.Abs((objAvgPos - objCurPos).magnitude) <= 0.1)
                {
                    objPos.Enqueue(objCurPos);
                    if (objPos.Count > 5)
                    {
                        objPos.Dequeue();
                    }
                }
            }
            Vector3 objAvg = new Vector3(0.0f, 0.0f, 0.0f);
            for (int i = 0; i < objPos.Count; i++){
                objAvg += (Vector3)(objPos.ToArray())[i];
            }
            objAvg /= objPos.Count;
            in_obj.transform.localPosition = objAvg;
```

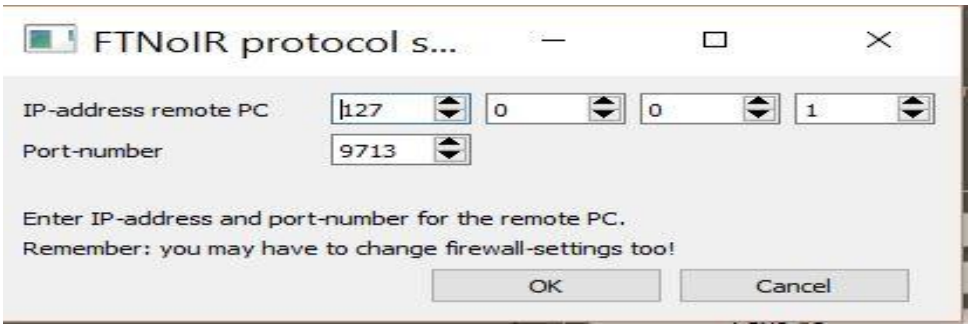
4 Use

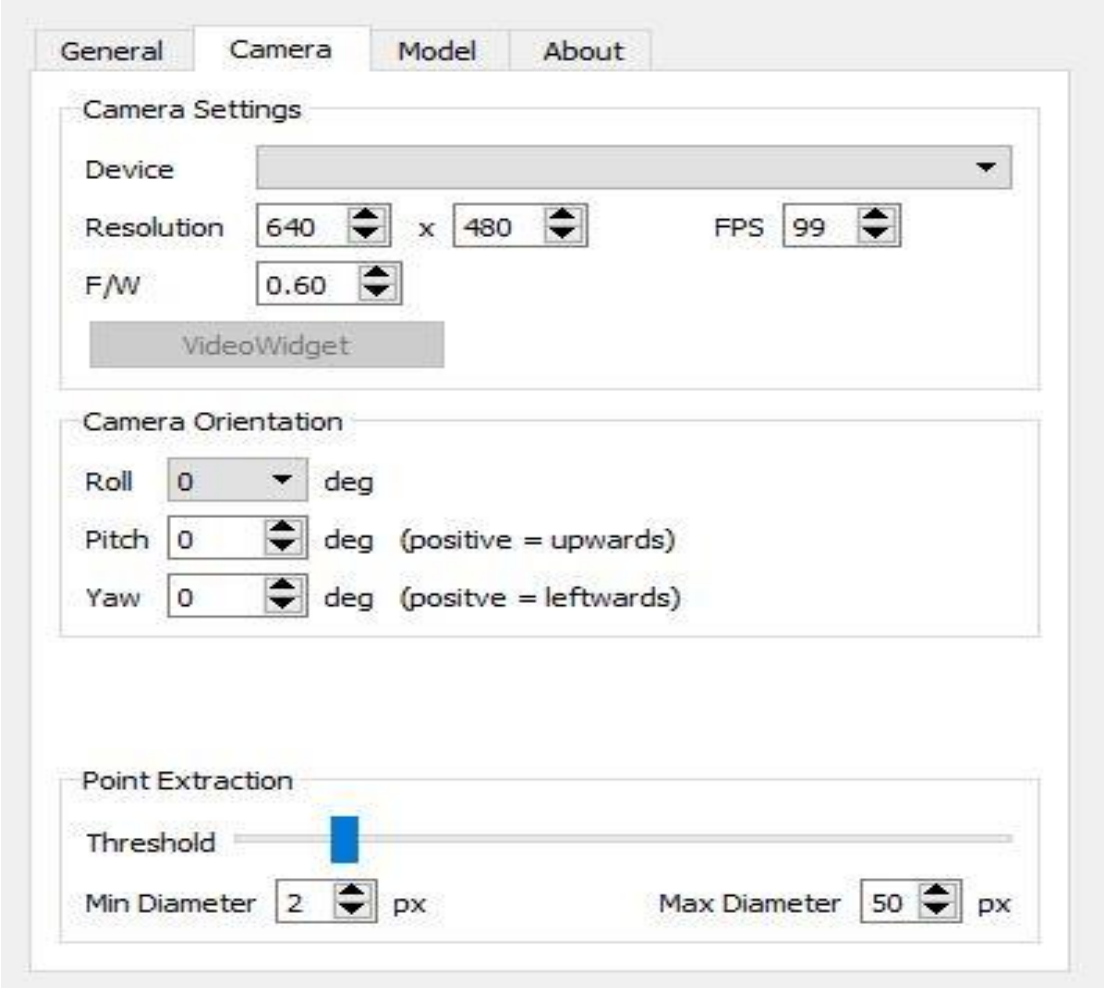
4.1. Multiple User Tracking Walk-through

My project can be used as a standalone motion tracker for detecting and tracking multiple objects, or it can be integrated within a game's tracking system to give the option for a multiple user setup. In order to successfully use my project's contents, there is a specific order of steps to follow in terms of running the code so that the UDP connections are ready for one another with aligned ports.

First, an instance of Unity's environment must be opened with the client script attached to its main camera and game objects. The Unity script is then put into action by pressing the play button so that Unity is ready before the connection is completed within FaceTrackNoIR and the C Server. Then, the C server needs to run from the command line, which brings the user to a point where both sides of the network communication link are ready for interaction with the camera and tracking application. Then, power up the LEDs with the nine volt power supplies, also making sure that the Arduinos are set up correctly within the black boxes. Remember that these Arduinos have a symbiotic relationship and rely on one another to ensure functionality. Finally, start the tracking within FaceTrackNoIR and adjust the light-capture threshold to account for room lighting. FaceTrackNoIR could also be started in the initial step so that the point systems can be checked before initiating my program. It's protocol settings need to be aligned with the IP address and port for the C server, and the camera's settings also can be adjusted. Figure 6 on the following page shows these layouts.

FIGURE 6 – FaceTrackNoIR Tracker and Protocol Settings





If the protocol is set correctly and a 'sweet spot' for the light-threshold is found, then both objects should be detected and tracked within Unity.

BIBLIOGRAPHY

"Sourceforge FaceTrackNoIR Wiki." Sourceforge., n.d. Web. 25 Feb 2016. <https://sourceforge.net/p/facetracknoir/wiki/Home/>.

"Visual C#." Visual C#. Microsoft, Inc., n.d. Web. 26 Feb. 2016. <https://msdn.microsoft.com/en-us/library/kx37x362.aspx>.