Utah State University

Lighthouse Positional Tracking for Mobile VR

Design Documentation

Brady Riddle & Samuel Jungert
12-16-2016

Table of Contents

List of Figures 2
1 Introduction 3
2 Scope 3
3 Design Overview 3
3.1 Requirements 3
3.2 Dependencies 3
3.3 Theory of Operation 4
4 Design Details 5
4.1 Hardware Design 5
4.2 Software Design 6
4.2.1 TM4C123GH6PM Setup 8
4.2.2 ESP8266 Setup 8
4.2.3 Unity Engine and The Tracking Algorithm 8
5 Testing 11
5.1 Test 1 – Sensors 11
5.2 Test 2 – Interrupts 11
5.3 Test 3 – Serial Transmission 12
5.4 Test 4 – WiFi 13
5.5 Test 5 – Tracking Algorithm 14
6 Conclusion 14
References 15
Appendix A 16
Appendix B 17

List of Figures

Figure 1 – Theory of Operation 4
Figure 2 - Hardware 6
Figure 3 - Flow chart of positional tracking including all partitions 7
Figure 4 – Azimuth and Elevation 9
Figure 5 – Angle Between Two Sensors 10
Figure 6 – System of Non-Linear Equations 10
Figure 7 – Jacobian Matrix 10
Figure 8 - Sync Flash 11
Figure 9 – Sensor Timings 12
Figure 10 – Console Output, Putty 13
Figure 11 – UDP Packet Viewer 13

1 Introduction

This document describes the design for a virtual reality tracking system based on the Lighthouse tracking system developed by Valve. The Lighthouse tracking system uses a base station that fires infrared light in three distinct ways. The base station starts with a flash that hits all the infrared sensors simultaneously and follows with a vertical or horizontal sweep. Sensors mounted onto the Samsung Gear VR headset provide positional tracking data based off of the base stations infrared firing pattern. Timing data, for the infrared sensors, is collected using a microcontroller and sent to a computer via UDP packet over Wi-Fi. Unity, a gaming engine, uses this data to update the user position within the virtually generated environment.

2 Scope

This document discusses, in detail, the hardware requirements and software design for our infrared tracking system. It includes the requirements, dependencies, and theory of operation. A Schematic and example code are included in the appendices of this document. Testing procedures to verify the functionality of each of the requirement is explained.

The infrared base station, Wi-Fi adapter, and Samsung Gear VR headset are not discussed in detail in this document.

3 Design Overview

3.1 Requirements

The following are the given requirements for the infrared tracking system.

1. The system shall run on 5 volt USB supplied power.
2. The system shall use three infrared sensors.
3. The system shall use a TM4C123GH6PM microcontroller.
4. The system shall use a Wi-Fi enabled microcontroller.
5. A Valve infrared base station should be used.
6. A Samsung Gear VR headset should be used.
7. A separate interrupt controller should be configured for each infrared sensor.
8. The System clock should be configured to run at 66.67 Mhz.
9. Timing data should be transmitted to the Wi-Fi enabled microcontroller via UART at 216,000 baud.
10. The Wi-Fi enabled microcontroller should send the timing data to a PC via UDP packets.
11. The tracking algorithm should use Newtonian Root finding to calculate the coordinates.

3.2 Dependencies

The following are the dependencies for the infrared tracking system.

- A 5 volt USB power source.

- A Valve Lighthouse base station
- Wi-Fi capable microcontroller
- TM4C123GH6PM or similar microcontroller
- Samsung Gear VR Headset or Google Cardboard

3.3 Theory of Operation

The basic operation of the infrared tracking system is as follows. Upon powering on the TM4C123GH6PM microcontroller, the microcontroller will wait for a sync flash from the base station. By design, once the base station is powered on, it will start by sending out a sync flash. After the sync flash, the base station performs either a vertical or horizontal sweep. The base station operates this way indefinitely. Once a sync flash is received a timer on the TM4C is started.

If an infrared sensor is hit with infrared light during a sweep, the TM4C triggers an interrupt, and stores the current timer value. After a vertical or horizontal sweep has taken place, the timing information for each sensor is sent to the Wi-Fi enabled microcontroller. The process repeats itself every 8.3 milliseconds. The PC, upon receiving the sensors timings, sends the information to the Unity engine. There, the timing information is used, in conjunction with Newtonian root finding, to calculate and update the objects position.

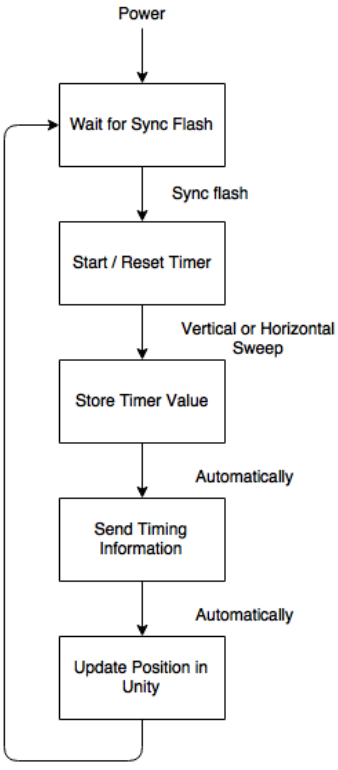


Figure 1 – Theory of Operation

4 Design Details

4.1 Hardware Design

The Lighthouse base station sends three infrared patterns. The first infrared pattern is a sync flash. The sync flash is a wide area saturation of infrared light. The sync flash is followed up by a vertical sweep. The vertical sweep is a vertical band of infrared light which sweeps the room from left to right. Another sync flash occurs and is followed by a horizontal sweep. The horizontal sweep is a horizontal band of infrared light which sweeps from the bottom of the room to the top. The three patterns of infrared light continue to repeat in this fashion indefinitely.

Three infrared sensitive sensors are mounted to the front of a Samsung gear VR headset. The sensors are active high. When a sensor is hit with a sync flash, vertical sweep, or horizontal sweep, it goes low. The TM4C123GH6PM microcontroller is used to capture the sensors transition from a high to low state.

The TM4C123GH6PM has the three infrared sensors connected to three different GPIO ports. When a low is detected on any of these ports, an interrupt is triggered. The interrupt stores the timing data. Upon the next sync flash, another GPIO port transmits the timing data at 216,000 baud, via UART, to the ESP8266 wireless enabled microcontroller.

The ESP8266 Wireless enabled microcontroller collects the timing data from its RX serial input. Then the ESP8266 puts the serial data in a UDP packet and transmits the data to a connected PC. The UDP packet is opened in Unity, where the positon of the Samsung Gear VR headset is updated.

See Appendix A for a detailed schematic.

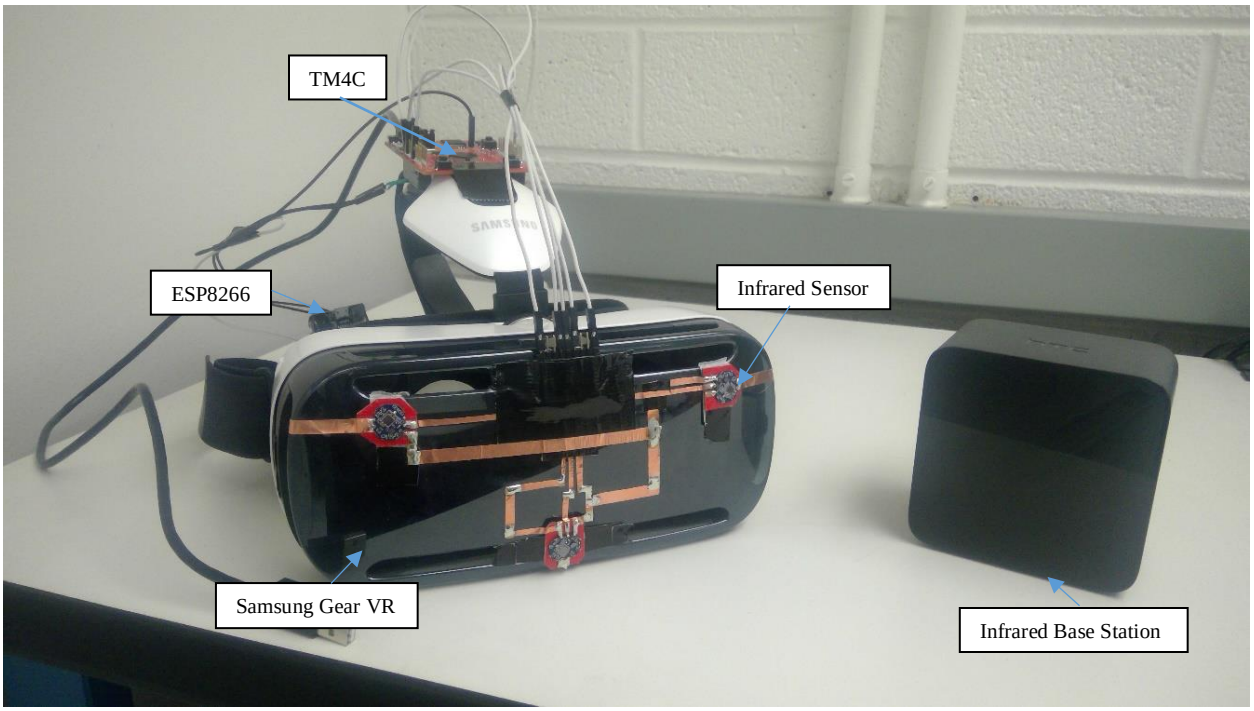


Figure 2 - Hardware

4.2 Software Design

The software design is broken into three functional partitions: TM4C123GH6PM, ESP8266, and Unity Engine and Tracking Algorithm. Each heading will focus on the software implementation required for that specific microcontroller or application. The TM4C and ESP8266 were both written in the C language. The code for Unity was written in C#. Figure 3 below shows all three functional partitions together.

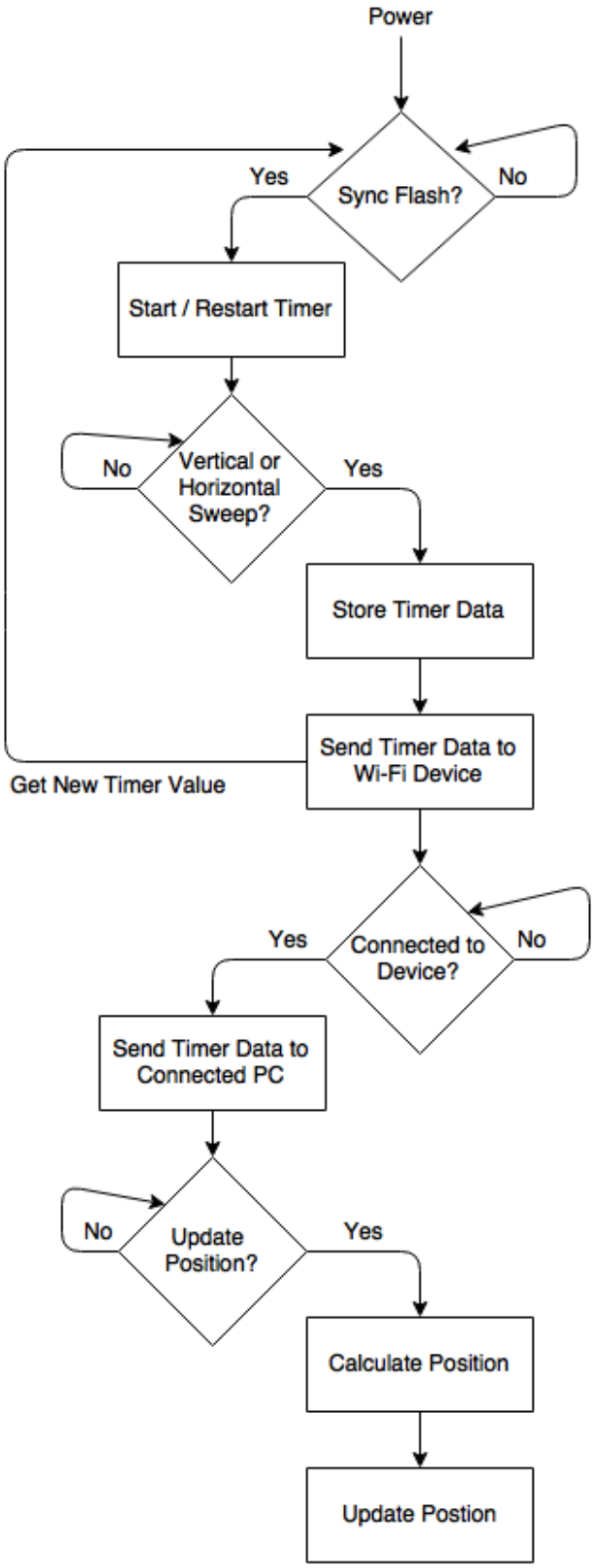


Figure 3 - Flow chart of positional tracking including all partitions

4.2.1 TM4C123GH6PM Setup

As discussed earlier the TM4C microcontroller is used for timing, interrupting on a sensor low, and transmitting the timing data to the Wi-Fi enabled ESP8266 microcontroller.

The code logic of the TM4C is based off of a sync flash event. A sync flash event is when all three infrared sensor go low simultaneously. For the first sync flash event a timer should be stared and three interrupts enabled. For every following sync flash event, the timer needs to be reset and the interrupts, again, enabled.

Three individual interrupt need to be configured for each of the three infrared sensors. One note, it is important to keep each sensor on a separate NVIC controller, to prevent potential conflict. Each interrupt handler needs to store the value of the timer at the instant the interrupt is called. The respective interrupt needs to be disabled at the end of the handler call.

Sensor timings need to be transmitted to the Wi-Fi device through UART. To ensure the timing data is being transmitted fast enough, it is recommended you don't go below 216,000 baud. To achieve this transmission rate the system clock, on the TM4C, was configured to run at 66.67 MHz. The sensor timing data is transmitted with each sync flash event, excluding the first.

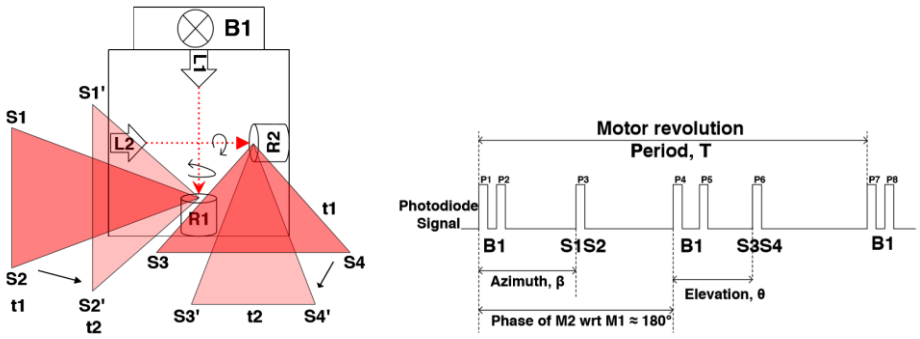
4.2.2 ESP8266 Setup

The ESP8266 has one purpose, to transmit the timing data to a PC or smartphone. To achieve this the ESP8266 starts off by making a connection to a computer or smartphone. The IP address is hard coded when writing the program. Once a connection is established, the ESP8266 listens on its seral RX for incoming data. When it RX buffer is full, or it has stopped receiving data, the buffer's contents is packaged into a UDP packet. The UDP packet is then sent to the hardcoded IP address.

4.2.3 Unity Engine and the Tracking Algorithm

The computations to compute position are done in Unity. The logic behind our lighthouse tracking algorithm comes from Shahidual Islam, Bogdan Ionescu, Cristain Gadea, and Dan Ionescu who authored, "Indoor Positional Tracking Using Dual-Axis Rotating Laser Sweeps." Figure 4 below was pulled from their research article to help with comprehension. To better understand the lighthouse tracking algorithm, you must understand how the lighthouse base station works. Tracking is done in spherical coordinates using azthmus, elevation, and range. When the infrared base station is first turned on, it sends out a "sync flash." This illuminates all the infrared sensors. When an infrared sensor is illuminated, a logic low signal is asserted. The sync flash is used to start a timer. Immediately after a sync flash occurs, a laser starts sweeping through the room, from left to right. The TM4C captures the timing data. The timing it captures is the period between a sync flash, to when a sensor is hit by a laser sweep. This timing gives us

our azthmus angle. Immediately after the horizontal laser sweep ends, another sync flash occurs; the laser then sweeps vertically. By measuring the timing deltas here, we get our elevation angle.



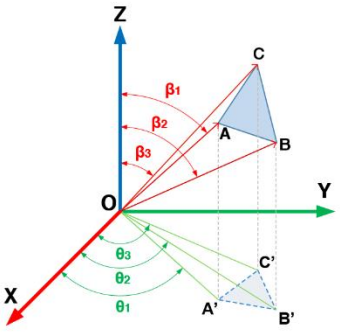
$$Azimuth, \beta = \left(\frac{time(P_3) - time(P_1)}{T}\right) \times 360°$$

$$Elevation, \theta = \left(\frac{time(P_6) - time(P_4)}{T}\right) \times 360°$$

Figure 4 – Azimuth and Elevation: Architecture for the lighthouse base station**.** [Top Left] Pulse train for light house sensors [Top Right] and the formula for elevation and azimuth angles. [Bottom] [1;3]

Our infrared tracking system assumes that you have three sensors in some coplanar configuration. Once we have the azthmus and elevation angles for all three sensors, we set up a system of equations, and solve for the range. Figure 5 helps to describe this process.

Assume that you have three sensors labeled A, B, and C. These sensors are at a certain elevation, azthmus, and range, away from an origin point. If we know the elevation and azthmus angles, we can use the relationships between polar and Cartesian coordinates, to obtain the angle between any two sensors.



$$cos\alpha_{BC} = sin\beta_2 cos\theta_2 sin\beta_3 cos\theta_3 + sin\beta_2 sin\theta_2 sin\beta_3 sin\theta_3 + cos\beta_2 cos\beta_3$$

$$cos\alpha_{AB} = sin\beta_1 cos\theta_1 sin\beta_2 cos\theta_2 + sin\beta_1 sin\theta_1 sin\beta_2 sin\theta_2 + cos\beta_1 cos\beta_2$$

$$cos\alpha_{AC} = sin\beta_1 cos\theta_1 sin\beta_3 cos\theta_3 + sin\beta_1 sin\theta_1 sin\beta_3 sin\theta_3 + cos\beta_1 cos\beta_3$$

Figure 5 – Angle between Two Sensors: Tracked object in a 3d graph. [Top] Derived equations for angles between the sensors. [Bottom] [1; 4]

Solving for these angles allow us to draw a triangle from the origin to any two points (OAB, OAC, and OBC). Once we establish the distance between our sensors AB, BC, and AC, we can apply the law of cosines and get the following systems of equations:

$$f(R_A, R_B) = R_A^2 + R_B^2 - 2R_A R_B \, cos\alpha_{AB} - AB^2 = 0$$

$$f(R_B, R_C) = R_B^2 + R_C^2 - 2R_B R_C \, cos\alpha_{BC} - BC^2 = 0$$

$$f(R_A, R_C) = R_A^2 + R_C^2 - 2R_A R_C \, cos\alpha_{AC} - AC^2 = 0$$

Figure 6 – System of Non-Linear Equations [1; 4]

To get the range for each of the sensors, Ra, Rb, and Rc, we solve the system of non-linear equations in Figure 6.

There are several numerical methods that can be used for solving a system of nonlinear equations. The most common is Newton's root finding method. Using Newton's method, in conjunction with the Jacobian method, we numerically solved the equation by guessing an answer and iteratively solving the systems of equations. For convenience, we include the derived Jacobian matrix in the Figure 7.

$$\begin{bmatrix} 2R_A - 2R_B \, cos\alpha_{AB} & 2R_B - 2R_A \, cos\alpha_{AB} & 0 \\ 0 & 2R_B - 2R_C \, cos\alpha_{BC} & 2R_C - 2R_B \, cos\alpha_{BC} \\ 2R_A - 2R_C \, cos\alpha_{AC} & 0 & 2R_C - 2R_A \, cos\alpha_{AC} \end{bmatrix}$$

Figure 7 – Jacobian Matrix [1; 5]

Once a certain number of iterations are hit, or solution becomes, "good enough" and our third and final coordinate, range, is solved. Then, using our range values, we can calculate the headsets position in either spherical or Cartesian coordinates.

5 Testing

5.1 Test 1 – Sensors

We expect the infrared sensors to remain high without infrared light, then go low when hit with infrared light. To verify this, we point the base station at the infrared sensors mounted on the Samsung Gear VR Headset. With the sensors connected to an oscilloscope, we confirm that with each sync flash and sweep, the sensors go low. See oscilloscope capture below in Figure 8. This fulfills requirement two, five, and six.

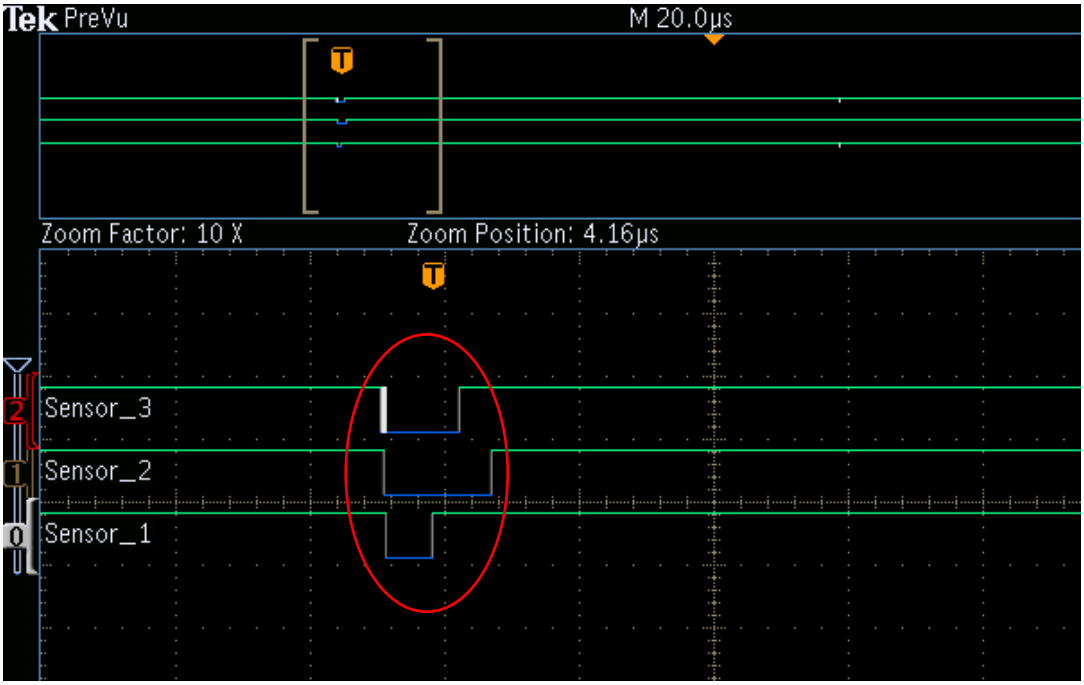


Figure 8 - Sync Flash

5.2 Test 2 – Interrupts

The interrupts are used to capture the exact time the infrared sensors are hit with a vertical or horizontal sweep. To ensure the interrupts are working as expected, we look at our timing values, to see if they are reasonable. To determine what is, "reasonable," we start by observing that a sync flash occurs every 8.3 milliseconds. If a sweep occurs in between two sync flashes, we would expect a time of about less than 8.3 milliseconds to be stored in our sensor timings. In Figure 9 you can see our timing values in processor cycles. If the processor is running at 66.67 MHz we know that equates to 66,670,000 cycles per second. If we divide 66,670,000 by 1,000 we get 66,670 cycles per millisecond. If we pull sesnor1Time, 240,432 cycles, and divide it by 66,670 cycles, we get 3.6 milliseconds. This data is consistent with what we would expect and confirms our interrupts are capturing accurate sweep values. This fulfills requirements one, three, seven, and eight.

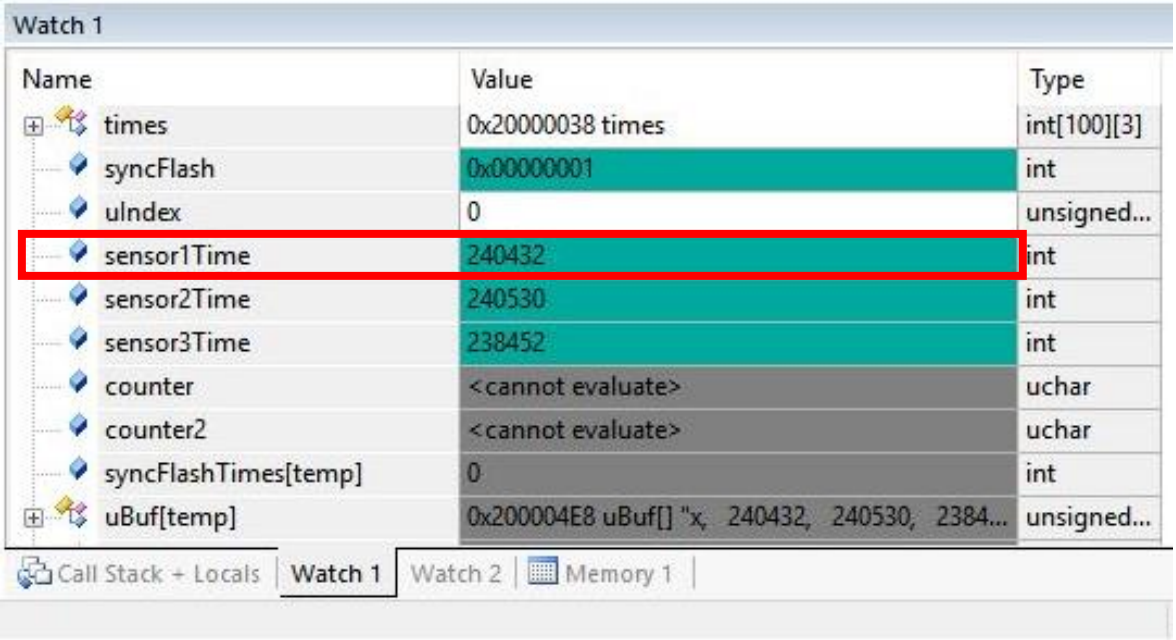


Figure 9 – Sensor Timings

5.3 Test 3 – Serial Transmission

We are using UART, on the TM4C, to transmit the timer data to our ESP8266 Wi-Fi capable microcontroller. To verify the functionality of our serial transmission we will use a putty console. If UART is properly configured, we expect to see an x or a y, followed by three sensor timings. It is important to note, again, that we are using a 216,000 baud rate. It is almost important to note that to achieve this high baud rate, we had to ramp up the system clock to 66.67 MHz. In putty we input the appropriate baud rate and select the appropriate com port. The results are shown in Figure 10. As you can see, the data is being successfully transmitted confirming the functionality of UART. This fulfills requirement one, eight, and nine. With the data being transferred successfully over UART, we can now test our ESP8266.

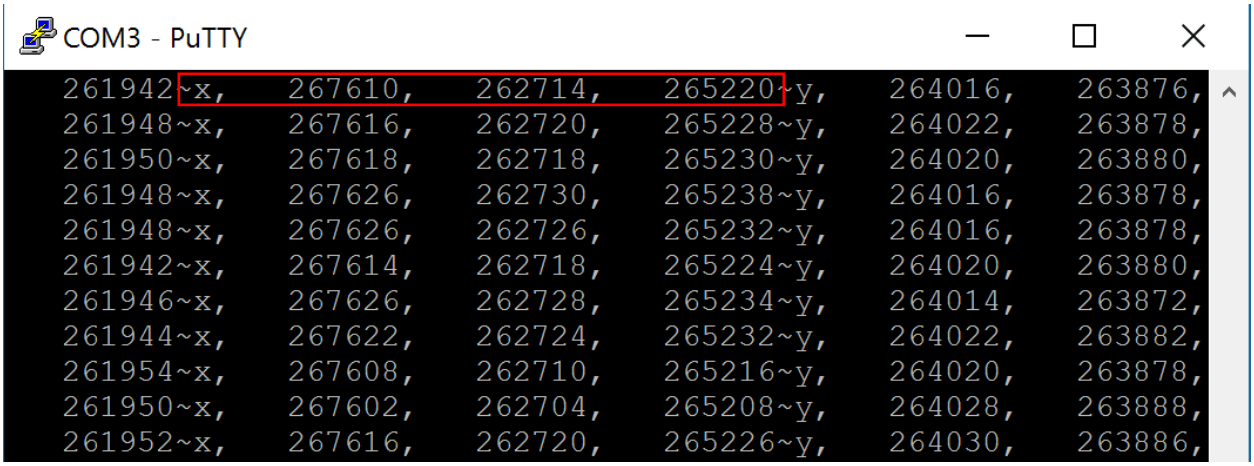


Figure 10 – Console Output, Putty

5.4 Test 4 – WiFi

The ESP8266 is expected to convert the serial data from the TM4C into UDP packets. These UDP packets will then be transmitted to a PC or Smartphone over Wi-Fi connection. The first step is to hard code the IP address of the PC or Smartphone you are going to send the UDP packets to. Then, to verify correct functionality, download a UDP packet viewer, and look for an output like that from our above serial transmission. As you can see in the Figure 11, the data is successfully being sent over UDP packet to our PC. This fulfills requirements nine, and ten.

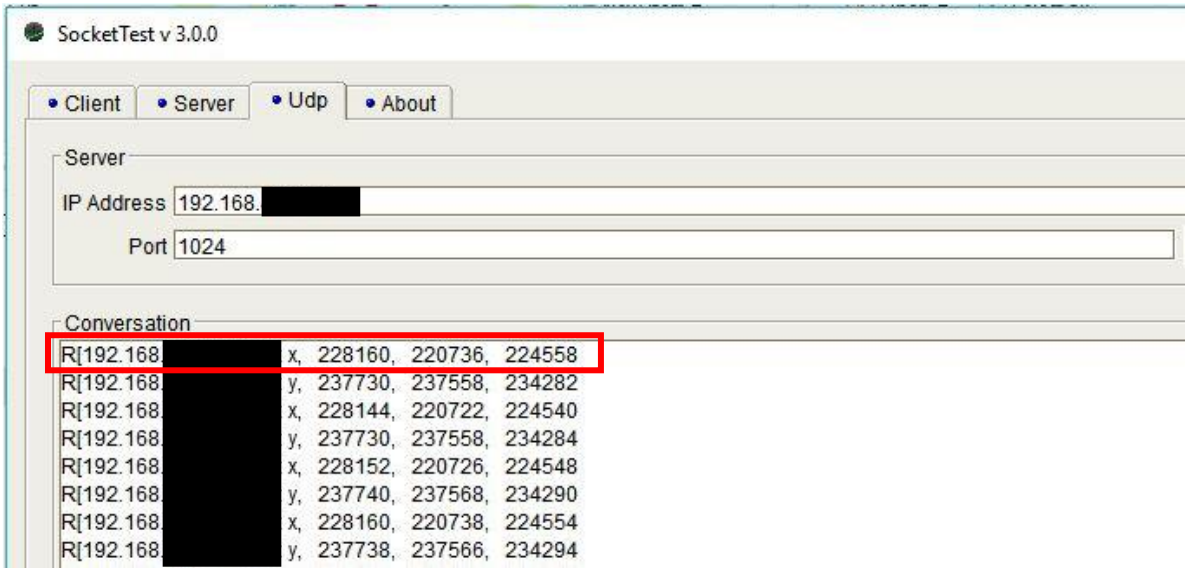


Figure 11 – UDP Packet Viewer

5.5 Test 5 – Tracking Algorithm

Testing the tracking algorithm is straightforward. With the lighthouse base station on, and the tracking system connected, start the Unity application. In the Unity application, have the main camera set up to update its position based on the data received from the tracking system. This means that when you move the headset around, it will move the camera.

Initial testing found that the X,Y,Z coordinates we obtained from our tracking system did not line up with Unity's tracking system. When we obtained our first position, from the headset, the coordinates would look something like (60,7,100), when the desired result was (0,0,0). To fix this, we remapped the axis from our tracking system to line up with Unity's. When we calculated the first position we stored an offset, to correct the deviation; then, we subtracted that offset from every position calculation, moving forward.

Implementing the above suggestions in your project, will help you ensure your tracking algorithm is working accurately.

6 Conclusion

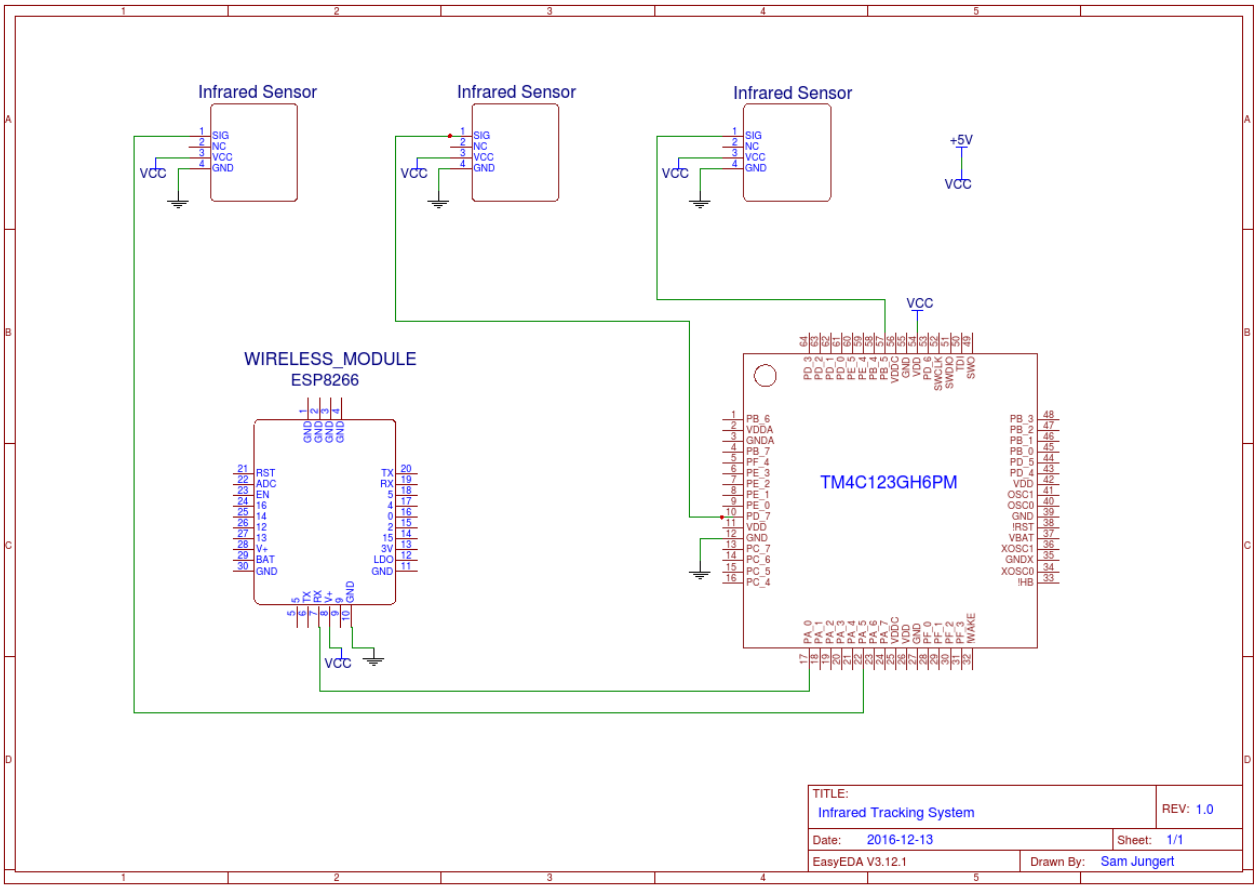
This document outlines the process to make an infrared tracking system for a device like the Samsung Gear VR or Google Cardboard. An infrared base station and three infrared sensors were used to get positional data. The TM4C microcontroller collected timing data, which it relayed via UART, to the ESP8266 Wi-Fi microcontroller. The ESP8266 packaged this data into a UDP packet. The packets are then sent to a PC or Smartphone. The PC or Smartphone decode the UDP packets and forward the data to Unity. In the C# programming language code was written for Unity, which used our timing data and Newtonian root finding, to calculate the position of the headset. As a result, the headsets position is updated in a virtual environment, allowing full 3-D moment, in a virtual world.

References

[1] S. Islam, B. Ionescu, C. Gadea, and D. Ionescu, “Indoor Positional Tracking Using Dual-Axis Rotating Laser Sweeps,” *School of Electrical Engineering and Computer Science, University of Otawa*, Ottawa, Ontario, Canada.

Appendix A

Hardware schematic.



Appendix B

C code for TM4C

--------Functions.h

```
#include "TM4C123.h"
void timer0Config(void);

void timer0Enable(void);
void timer0Disable(void);

void setSysTick(void);

void systemClock(void); //Sets system clock to 66.67 MHz

void enableInterruptA(void);
void enableInterruptB(void);
void enableInterruptD(void);
void enableInterruptE(void);

void disableInterruptA(void);
void disableInterruptB(void);
void disableInterruptD(void);
void disableInterruptE(void);

void clearIntAFlag(void);
void clearIntBFlag(void);
void clearIntDFlag(void);

void enableGPIOPortA(void); //Sensor 1 Interrupt PA5
void enableGPIOPortB(void); //Sensor 2 Interrupt PB5
void enableGPIOPortD(void); //Sensor 3 Interrupt PD7
void enableGPIOPortE(void);
void enableGPIOPortF(void); //Testing Purposes

void initializeUART(void);
void iToA(int32_t const num, char *string);

void enableAllInterrupts(void);
void startAllInterrupts(void);
void clearAllICR(void);
```

--------Functions.c

```
#include "C:\Keil_v5\ARM\INC\TI\TM4C123\TM4C123GH6PM.h"
#include "functions.h"

//Variables
volatile unsigned int* PRIRegister = (unsigned int *) 0xE000E000; //Data
sheet page 154
volatile unsigned int* NVIC_Enable = (unsigned int *) 0xE000E000; //Data
sheet page 142
```

```
void timer0Config(void){
	SYSCTL->RCGCTIMER |= (1<<0); //Enable Timer 0 Clock
	TIMER0->CTL &= ~(1<<0);                              //Disable the
Timer
	TIMER0->CFG = 0x0;
	//Configure 16/32-bit timer to 32-bit timer mode
	TIMER0->TAMR |= 0x12;                                //Periodic Timer
Mode and TACDIR(Timer Direction - Counts up)
	TIMER0->TAILR = 0xFFFFFFFF;          //Upper bound for the timeout
event (Maximum value timer can reach)
	//TIMER0->IMR |= (1<<0);      //Enable Mask for interrupts on timer
A
	//NVIC_EnableIRQ(TIMER0A_IRQn); //Enable Interrupts for Timer0A
	//TIMER0->CTL |= (1<<0);                             //Start the
Timer

}

// Call after all GPIO setup has been done;
void enableAllInterrupts(void){
	//NVIC_Enable[0x100/4] |= 0x1;
	NVIC_Enable[0x100/4] |= 0x2;
	NVIC_Enable[0x100/4] |= 0x8;
	NVIC_Enable[0x100/4] |= 0x10; // PORT E
}

void startAllInterrupts(void){
	//enableInterruptA();
	enableInterruptB();
	enableInterruptD();
	enableInterruptE();
}

void clearAllICR(void){
	//GPIOA->ICR = 0x20;
	GPIOB->ICR = 0x20;
	GPIOD->ICR = 0x80;
	GPIOE->ICR = 0x20;
}

void timer0Enable(void){
	TIMER0->TAV = 0x00000000;     //Zer
	TIMER0->CTL |= (1<<0);                               //Start the
Timer
}

void timer0Disable(void){
	TIMER0->CTL &= ~(1<<0);                              //Stop the Timer
}


void systemClock(void){
```

```
		char end=0;
	SYSCTL->RCC=0x00000800;
	SYSCTL->RCC|=0x540;
	//SYSCTL->RCC|=0x02400000;//configure to 40 MHz
	//SYSCTL->RCC|=0x01C00000;//configure to 50 MHz
	SYSCTL->RCC|=0x01400000;//configure to 66.67 MHz
	//if we use pwm, we may need to configure clock
	while(end==0)
	{
		char pllBit=SYSCTL->RIS;     //poll bit and wait for pll to
lock
		pllBit&=0x40;
		if(pllBit==0x40)end=1;
	}
	SYSCTL->RCC&=0xFFFFF7FF;//clear bypass bit


	/*
	//Step 1
	SYSCTL->RCC &= 0xFFBFF7FF; //Set USESYSDIV to 0, kept everything
else the same
	SYSCTL->RCC |= 0x00400800;
	//Step 2
	SYSCTL->RCC &= 0xFFFFD80F;
	SYSCTL->RCC |= (0x15<<6);
	//step 3
	SYSCTL->RCC &= 0xF83FFFFF;
	SYSCTL->RCC |= 0x01400000; //Sets system divider to 0x2 (66.67 MHz)
and enables the USESYS bit
	//step 4 (5 in manual)
	SYSCTL->RCC &= 0xFFFFF7FF;
*/
	}

//Sensor 1
void enableGPIOPortA(void){
	SYSCTL->RCGCGPIO |= 0x1;                    //Enable Clock for Port
A
	GPIOA->CR = 0xFF;
	//Write 1111 1111 to Commit Regsiter
	//GPIOA->DIR |= 0x20;                                    //Enable
Port PA5 to Input
	GPIOA->AFSEL = 0x0;                                 //Disable
Alternate Functionality
	GPIOA->DEN |= 0x20;
	//Digital Enable PA5
	//Interrupt Settings
	GPIOA->IS = 0x00;
	//PA5 is set to 0, edge-sensitive
	GPIOA->IBE = 0x00;                                     //PA5 is
set to 0, falling edge or low level input
	GPIOA->IEV = 0x00;                                     //PA5 is
set to 0, falling edge sensative
```

```
	//GPIOA->ICR = 0x20;                                          //Clear
Flag for PA5
	GPIOA->IM = 0x20;                                           //Arm
interrupt on PA5 (Interrupt Mask)
	//PRIRegister[0x400/4] &= 0x1F; //Priority Register 0, offset
0x400, set to GPIOA Handler to Priority 0
	disableInterruptA();
}

void enableGPIOPortE(void){
	SYSCTL->RCGCGPIO |= 0x10;                   //Enable Clock for Port
E
	//GPIOA->CR = 0xFF;
	//Write 1111 1111 to Commit Regsiter
	//GPIOA->DIR |= 0x20;                                         //Enable
Port PA5 to Input
	GPIOE->AFSEL = 0x0;                                 //Disable
Alternate Functionality
	GPIOE->DEN |= 0x20;
	//Digital Enable PA5
	//Interrupt Settings
	GPIOE->IS = 0x00;
	//PA5 is set to 0, edge-sensitive
	GPIOE->IBE = 0x00;                                          //PA5 is
set to 0, falling edge or low level input
	GPIOE->IEV = 0x00;                                          //PA5 is
set to 0, falling edge sensative
	//GPIOA->ICR = 0x20;                                          //Clear
Flag for PA5
	GPIOE->IM = 0x20;                                           //Arm
interrupt on PA5 (Interrupt Mask)
	//PRIRegister[0x400/4] &= 0x1F; //Priority Register 0, offset
0x400, set to GPIOA Handler to Priority 0
	disableInterruptE();
}

//Sensor 2
void enableGPIOPortB(void){
	SYSCTL->RCGCGPIO |= 0x2;                    //Enable Clock for Port
B
	GPIOB->CR = 0xFF;
	//Write 1111 1111 to Commit Regsiter
	//GPIOB->DIR |= 0x20;                                         //Enable
Port PB5 to Input
	GPIOB->AFSEL = 0x0;                                 //Disable
Alternate Functionality
	GPIOB->DEN |= 0x20;
	//Digital Enable PB5
	//Interrupt Settings
	GPIOB->IS = 0x0;                                            //PB5 is
set to 0, edge-sensitive
	GPIOB->IBE = 0x0;                                           //PB5 is
set to 0, falling edge or low level input
```

```
	GPIOB->IEV = 0x0;                                               //PB5 is
set to 0, falling edge sensative
	//GPIOB->ICR = 0x20;                                            //Clear
Flag for PB5
	GPIOB->IM = 0x20;                                               //Arm
interrupt on PB5 (Interrupt Mask)
	//PRIRegister[0x400/4] |= 0x2000; //Priority Register 0, offset
0x400, set to GPIOB Handler to Priority 1
	disableInterruptB();
}

//Sensor 3
void enableGPIOPortD(void){
	SYSCTL->RCGCGPIO |= 0x8;                    //Enable Clock for Port
D
	GPIOD->LOCK = 0x4C4F434B;                   //Unlock Port D
	GPIOD->CR = 0xFF;
	//Write 1111 1111 to Commit Regsiter
	//GPIOD->DIR |= 0x80;                                           //Enable
Port PD7 to Input
	GPIOD->AFSEL = 0x0;                                 //Disable
Alternate Functionality
	GPIOD->DEN |= 0x80;
	//Digital Enable PD7
	//Interrupt Settings
	GPIOD->IS = 0x0;                                                //PB5 is
set to 0, edge-sensitive
	GPIOD->IBE = 0x0;                                               //PB5 is
set to 0, falling edge or low level input
	GPIOD->IEV = 0x0;                                               //PB5 is
set to 0, falling edge sensative
	//GPIOD->ICR = 0x80;                                            //Clear
Flag for PB5
	GPIOD->IM = 0x80;                                               //Arm
interrupt on PB5 (Interrupt Mask)
	//PRIRegister[0x400/4] |= 0x40000000; //Priority Register 0, offset
0x400, set to GPIOD Handler to Priority 2
	disableInterruptD();
}

//Testing Purposes
void enableGPIOPortF(void){
	SYSCTL->RCGCGPIO |= 0x20;                   //Enable Clock for Port
F
	GPIOF->LOCK = 0x4C4F434B;                   //Unlock Port F
	GPIOF->CR = 0x1F;
	//Write 0011 1111 to Commit Register
	GPIOF->PUR = 0xFF;                                              //Enable
Pull Up Configuration on Port PF4 and PF0
	GPIOF->DIR = 0xFF;                                              //Enable
Port PF4 and PF0 to Input
	GPIOF->DEN = 0xFF;
	//Digital Enable all ports on F
}
```

```
void enableInterruptA(void){
	GPIOA->IM = 0x20;
}
void enableInterruptB(void){
	GPIOB->IM = 0x20;
}
void enableInterruptD(void){
	GPIOD->IM = 0x80;
}

void enableInterruptE(void){
	GPIOE->IM = 0x20;
}

void disableInterruptA(void){
	GPIOA->IM = 0x0;
}
void disableInterruptB(void){
	GPIOB->IM = 0x0;

}
void disableInterruptD(void){
	GPIOD->IM = 0x0;
}

void disableInterruptE(void){
	GPIOE->IM = 0x0;
}

void clearIntAFlag(void){
	GPIOA->ICR = 0x20;                                          //Clear
Flag for PA5
}

void clearIntBFlag(void){
	GPIOB->ICR = 0x20;                                          //Clear
Flag for PB5
}

void clearIntDFlag(void){
	GPIOD->ICR = 0x80;                                          //Clear
Flag for PB5
}
// WARNING: THIS MAY NOT BE RESETTING THE CURRENT VALUE OF SYSTICK (TRY
WRITING TO SYSTICK->VAL)
void setSysTick(void)
{
	SysTick->CTRL = 0x0;
	__nop();
	__nop();
	SysTick->LOAD = 0xFFFFFF;
	__nop();
	__nop();
```

```
        SysTick->VAL = 0x0;
        __nop();
        __nop();
        SysTick->CTRL = 0x5;
        __nop();
        __nop();       // use system clock, enable sysTick;
}

void initializeUART(void)
{
        // We might have to adjust this for our current system clock;
        SYSCTL->RCGCUART |= (1<<0);
        SYSCTL->RCGCGPIO |= (1<<0);
        GPIOA->AFSEL = (1<<1)|(1>>0);
        GPIOA->PCTL = (1<<0) | (1<<4);
        GPIOA->DEN= (1<<0) | (1<<1);

        UART0->CTL &= ~(1<<0);
        //Baud Rate (256000 @ 66.67 mhz sys clock)
        UART0->IBRD =  0x10; // 16
        UART0->FBRD = 0x12;   // 18
        //256000 Baudrate @ 16 mhz
        //UART0->IBRD = 0x3;
        //UART0->FBRD = 0x3A;
        UART0->LCRH = (0x3<<5);
        UART0->CTL = (1<<0)|(1<<8)|(1<<9);
}

void iToA (int32_t const num, char *string)
{
        int32_t n;
        int32_t temp = 0;

        // Clear the string
        for(temp = 0; temp < 10; temp++)
        string[temp] = '0';

  if(num<0){
    n = -num;
  } else{
    n = num;
  }

        if(n>9999999){      // 10000000 to 99999999
        if(num<0){
      string[0] = '-';
    } else {
      string[0] = ' ';
    }
              string[1] = '0'+n/10000000;
              n = n%10000000;
    string[2] = '0'+n/1000000;
              n = n%1000000;
    string[3] = '0'+n/100000;
```

```
  n = n%100000;
  string[4] = '0'+n/10000;
  n = n%10000;
  string[5] = '0'+n/1000;
  n = n%1000;
  string[6] = '0'+n/100;
  n = n%100;
            string[7] = '0'+n/10;
  n = n%10;
  string[8] = '0'+n;
  return;
}

      if(n>999999){      // 1000000 to 9999999
  string[0] = ' ';
            if(num<0){
    string[1] = '-';
  } else {
    string[1] = ' ';
  }
  string[2] = '0'+n/1000000;
            n = n%1000000;
  string[3] = '0'+n/100000;
  n = n%100000;
  string[4] = '0'+n/10000;
  n = n%10000;
  string[5] = '0'+n/1000;
  n = n%1000;
  string[6] = '0'+n/100;
  n = n%100;
            string[7] = '0'+n/10;
  n = n%10;
  string[8] = '0'+n;
  return;
}

if(n>99999){      // 100000 to 999999
  string[0] = ' ';
  string[1] = ' ';
            if(num<0){
    string[2] = '-';
  } else {
    string[2] = ' ';
  }
  string[3] = '0'+n/100000;
  n = n%100000;
  string[4] = '0'+n/10000;
  n = n%10000;
  string[5] = '0'+n/1000;
  n = n%1000;
  string[6] = '0'+n/100;
  n = n%100;
            string[7] = '0'+n/10;
  n = n%10;
```

```
  string[8] = '0'+n;
  return;
}
if(n>9999){  // 10000 to 99999
  string[0] = ' ';
  string[1] = ' ';
            string[2] = ' ';
            if(num<0){
    string[3] = '-';
  } else {
    string[3] = ' ';
  }
  string[4] = '0'+n/10000;
  n = n%10000;
  string[5] = '0'+n/1000;
  n = n%1000;
  string[6] = '0'+n/100;
  n = n%100;
  string[7] = '0'+n/10;
  n = n%10;
  string[8] = '0'+n;
  return;
}
if(n>999){   // 1000 to 9999 ERROR HERE!
  string[0] = ' ';
  string[1] = ' ';
            string[2] = ' ';
            string[3] = ' ';
            string[4] = ' ';
  if(num<0){
    string[5] = '-';
  } else {
    string[5] = ' ';
  }
  string[6] = '0'+n/1000;
  n = n%1000;
  string[7] = '0'+n/100;
  n = n%100;
  string[8] = '0'+n/10;
  n = n%10;
  string[9] = '0'+n;
  return;
}
if(n>99){   // 100 to 999
  string[0] = ' ';
  string[1] = ' ';
            string[2] = ' ';
            string[3] = ' ';
            string[4] = ' ';
            string[5] = ' ';
  if(num<0){
    string[6] = '-';
  } else {
    string[6] = ' ';
```

```
    }
    string[7] = '0'+n/100;
    n = n%100;
    string[8] = '0'+n/10;
    n = n%10;
    string[9] = '0'+n;
    return;
  }
  if(n>9){   // 10 to 99
    string[0] = ' ';
    string[1] = ' ';
                string[2] = ' ';
                string[4] = ' ';
                string[5] = ' ';
                string[6] = ' ';
    if(num<0){
      string[7] = '-';
    } else {
      string[7] = ' ';
    }
    string[8] = '0'+n/10;
    n = n%10;
    string[9] = '0'+n;
    return;
  }
  // 0 to 9
    string[0] = ' ';
    string[1] = ' ';
                string[2] = ' ';
                string[4] = ' ';
                string[5] = ' ';
                string[6] = ' ';
                string[7] = ' ';
  if(num<0){
    string[8] = '-';
  } else {
    string[8] = ' ';
  }
  string[9] = '0'+n;
}
```

--------Main.c

```
#include "TM4C123.h"
#include "functions.h"
#include "stdint.h"
volatile int sensor1Time = 0;
volatile int sensor2Time = 0;
volatile int sensor3Time = 0;
volatile int sensor1Flag = 0;
volatile int sensor2Flag = 0;
volatile int sensor3Flag = 0;
volatile int syncFlashTime = 0;
```

```
volatile int xSweep = 0;
volatile int firstSweep = 0;
volatile int times[100][3];
volatile uint8_t uBuf[300][33];
volatile uint16_t uIndex = 0;
char * tempWord;

//PA5
void GPIOA_Handler(void){
	clearIntAFlag();
	GPIOF->DATA = 0xFF;
	GPIOF->DATA = 0x0;
	sensor1Time = TIMER0->TAV;
	if (sensor1Time < 1000)
		return;
	disableInterruptA();

}

void GPIOE_Handler(void){
	//clearIntEFlag();
	GPIOE->ICR = 0x20;
	GPIOF->DATA = 0xFF;
	GPIOF->DATA = 0x0;
	sensor1Time = TIMER0->TAV;
	if (sensor1Time < 50000)
		return;
	disableInterruptE();

}

//PB5
void GPIOB_Handler(void){
	clearIntBFlag();
	GPIOF->DATA = 0xFF;
	GPIOF->DATA = 0x0;
	sensor2Time = TIMER0->TAV;
	if (sensor2Time < 50000)
	return;
	disableInterruptB();
}

//PD7
void GPIOD_Handler(void){
	clearIntDFlag();
	GPIOF->DATA = 0xF0;
	GPIOF->DATA = 0x0;
	sensor3Time = TIMER0->TAV;
	if (sensor3Time < 50000)
	return;
	disableInterruptD();
}

volatile int syncFlashTimes[100];
```

```
void writeToUart(char *string)
{
	int character = 0;

	for(character = 0; character < 33; character++)
	{
		while((UART0->FR & 0x20) == 0x20);
		UART0->DR = string[character];
	}

}

int main(void)
{
	int data = 0;
	int delay = 0;
	int temp = 0;
	int syncFlash = 0;
	int firstTime = 0;
	systemClock(); //Change system clock
	initializeUART();
	//enableGPIOPortA(); //Sensor 1 Interrupt PA5
	enableGPIOPortB(); //Sensor 2 Interrupt PB5
	enableGPIOPortD(); //Sensor 3 Interrupt PD7
  enableGPIOPortE();
	enableGPIOPortF(); //Testing Purposes
	enableAllInterrupts();
	timer0Config();

	/*
	while(1)
	{
		UART0->DR = 0x44;
	}
	*/

	while(1)
	{
		syncFlash = 0;
		while (GPIOE->DATA != 0x0 || GPIOB->DATA != 0x0 || GPIOD-
>DATA != 0x0)
		{};

		while (GPIOE->DATA == 0x0 && GPIOB->DATA == 0x0 && GPIOD-
>DATA == 0x0){

		if(syncFlash == 0)
			{
			syncFlash++;
			timer0Disable();
			timer0Enable();
			// Don't try and store anything on the first run
through...
```

```
                        if (firstTime == 0)
                    {
                        firstTime++;
                    }
                    else
                    {
                    times[temp][0] = sensor1Time;
                    times[temp][1] = sensor2Time;
                    times[temp][2] = sensor3Time;

                    if (xSweep == 0)
                    {
                    uBuf[temp][0] = 'x';
                    uBuf[temp][1] = ',';
                    xSweep++;
                    }
                    else
                    {
                    uBuf[temp][0] = 'y';
                    uBuf[temp][1] = ',';
                    xSweep = 0;
                    }
                    data = sensor1Time;
                    iToA(data,(char*)&uBuf[temp][2]); // 9 digit number
                    uBuf[temp][11] = ',';
                    data = sensor2Time;
                    iToA(data,(char*)&uBuf[temp][12]); // 9 digit number
                    uBuf[temp][21] = ',';
                    data = sensor3Time;
                    iToA(data,(char*)&uBuf[temp][22]); // 9 digit number
                    uBuf[temp][31] = '~';
                    //uBuf[temp][31] = 0x0D;
                    //uBuf[temp][31] = ' ';
                    //uBuf[temp][32] = 0x0A;
                    // TODO: Add serial write here!
                    //writeToUart((char *)&uBuf[temp][0]);
                    }
                    if (temp > 300)
                        temp = 0;

                    while (GPIOE->DATA == 0x0 || GPIOB->DATA == 0x0 ||
GPIOD->DATA == 0x0){};
                    clearAllICR();
                    startAllInterrupts();
                    if(firstTime == 1)
                        firstTime++;
                    else{
                        writeToUart((char *)&uBuf[temp][0]);
                        //temp++;
                    }
                    }
                };
        }
}
```

C code for ESP8266

--------***Main.c***

```
#include <ESP8266WiFi.h>
#include <WiFiUDP.h>
extern "C" {
#include<user_interface.h>
}

const char* ssid     = "loading...";
const char* password = "password";
//xString incomingData;
// A UDP instance to let us send and receive packets over UDP
WiFiUDP Udp;
int messageSize;

void setup() {
  Serial.begin(256000);
  delay(10);
  messageSize = 32;
  // We start by connecting to a WiFi network

  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);
```

```
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  Udp.beginPacket("192.168.43.1", 1024);
}
void loop() {
  String incomingData  =  Serial.readStringUntil('~');
  //if (incomingData.length() != messageSize)
    //return;
  Serial.println(incomingData);
  //Udp.beginPacket("192.168.43.209", 1024);
  Serial.println("sending Packet");
  //Udp.print(incomingData);
  //Udp.endPacket();

  Udp.beginPacket("192.168.43.1", 1024);
  Udp.print(incomingData);
  Udp.endPacket();

  Serial.println("Packet Sent");
}
```

C# code for Unity

--------Main.cs

```
using UnityEngine;
using System.Collections;
using System.IO.Ports;
using System;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.Threading;


public class ThreadedUDP : MonoBehaviour {

    Thread receiveThread;
    Vector3 newPos;
    Vector3 startPos;
    SerialPort stream;
    private int port = 8080; // define > init
    private IPEndPoint anyIP;
    UdpClient client;
    Boolean firstTime = false;
    int globalCameraRefresh = 0;
    int lineNumber = 1;

    public string lastReceivedUDPPacket = "";
    float m_B1 = 0, m_B2 = 0, m_B3 = 0;
    float m_theta1 = 0, m_theta2 = 0, m_theta3 = 0;
    float m_cos12, m_cos13, m_cos23;
    public float m_distAB, m_distAC, m_distBC;
    float m_RA = 1, m_RB = 1, m_RC = 1;
    float syncFlashTime = 133280; // SyncFlash time in cycles for a 16mhz
clock
    float period = 1111122;
    public bool anglesReady = false;
    bool altitudeReady = false;
    bool azthmusReady = false;
    bool firstPosition = false;
    int numSweeps = 0;
    public string IP;
    float A_offsetX, A_offsetY, A_offsetZ;

    // Use this for initialization
    void Start () {
        anyIP = new IPEndPoint(IPAddress.Any, 1024);
        client = new UdpClient(1024);
        startPos = this.transform.position;
        //test();
        print("Client Started");
        receiveThread = new Thread(
        new ThreadStart(ReceiveData));
        receiveThread.IsBackground = true;
```

```
        receiveThread.Start();

    }

    private void ReceiveData()
    {

        while (true)
        {
            try
            {
                byte[] data = client.Receive(ref anyIP);
                string text = "";
                text = Encoding.ASCII.GetString(data);
                string[] splitArray = parseData(text);
                if (splitArray == null)
                    continue;
                lastReceivedUDPPacket = text;
                //print("UNVALIDATED: " + splitArray[0] + splitArray[1] +
"" + splitArray[2] + splitArray[3]);
                inputValidation(splitArray);
                extractAngles();
                if (altitudeReady == true && azthmusReady == true)
                {
                    NewtonsNewMethod();
                    updatePosition();
                }

            }
            catch (Exception err)
            {
                print(err.ToString());
            }
        }
    }

    void inputValidation(string[] splitArray)
    {
        if (splitArray[0] == "x")
            validateX(splitArray);
        else if (splitArray[0] == "y")
            validateY(splitArray);
    }


    void validateX(string[] splitArray)
    {
        float tempB1, tempB2, tempB3;

        tempB1 = (float.Parse(splitArray[1]) / period) * 360;
        tempB2 = (float.Parse(splitArray[2]) / period) * 360;
        tempB3 = (float.Parse(splitArray[3]) / period) * 360;

        if (tempB1 > 180 || tempB2 > 180 || tempB3 > 180)
```

```
        {
            print("ERROR: TRACKING LOST");
            print(splitArray[0] + splitArray[1] + "" + splitArray[2] +
splitArray[3]);
            return;
        }
        else
        {
            altitudeReady = true;
            print("M_B1: " + tempB1 + " M_B2: " + tempB2 + "M_B3: " +
tempB3);
            m_B1 = tempB1 * Mathf.Deg2Rad;
            m_B2 = tempB2 * Mathf.Deg2Rad;
            m_B3 = tempB3 * Mathf.Deg2Rad;
        }
    }

    void validateY(string[] splitArray)
    {
        float tempTheta1, tempTheta2, tempTheta3;

        tempTheta1 = (float.Parse(splitArray[1]) / period) * 360;
        tempTheta2 = (float.Parse(splitArray[2]) / period) * 360;
        tempTheta3 = (float.Parse(splitArray[3]) / period) * 360;
        // Something went wrong print everything
        if (tempTheta1 > 180 || tempTheta2 > 180 || tempTheta3 > 180)
        {
            print("ERROR: TRACKING LOST");
            print(splitArray[0] + splitArray[1] + "" + splitArray[2] +
splitArray[3]);
            return;
        }
        else
        {
            azthmusReady = true;
            print("M_Theta1: " + tempTheta1 + " M_Theta2: " + tempTheta2 +
"M_Theta3: " + tempTheta3);
            m_theta1 = tempTheta1 * Mathf.Deg2Rad;
            m_theta2 = tempTheta2 * Mathf.Deg2Rad;
            m_theta3 = tempTheta3 * Mathf.Deg2Rad;
        }
    }

    string[] parseData(string text)
    {
        string[] array = new string[4];
        for (int c = 0; c < 4; c++)
            array[c] = "";
        if (text.Length == 31)
            text += '0';

        if (text.Length != 32)
        {
            print("ERROR: MESSAGE NOT 32 BYTES!");
```

```
            print("Length of Message: " + text.Length);
            for (int k = 0; k < text.Length; k++)
                print(k+"." + text[k].ToString());
            return null;
        }
        array[0] = text[1].ToString();
        for (int i = 2; i < 32; i++)
        {
            // Skip the commas in the data
            if (i > 2 && i < 12)
                array[1] += text[i].ToString();
            if (i > 12 && i < 22)
                array[2] += text[i].ToString();
            if (i > 22 && i < 32)
                array[3] += text[i].ToString();

        }
        return array;
    }

    private void extractAngles()
    {

        //Make sure to include the syncFlashPeriod here
        // Is this returning in degrees or radians? Probably radians... Is
that important?
        m_cos12 = (Mathf.Sin(m_B1) * Mathf.Cos(m_theta1) * Mathf.Sin(m_B2)
* Mathf.Cos(m_theta2)
            + (Mathf.Sin(m_B1) * Mathf.Sin(m_theta1) * Mathf.Sin(m_B2) *
Mathf.Sin(m_theta2)
            + Mathf.Cos(m_B1) * Mathf.Cos(m_B2)));
        //print("CosAB: " + m_cos12);
        m_cos23 = (Mathf.Sin(m_B2) * Mathf.Cos(m_theta2) * Mathf.Sin(m_B3)
* Mathf.Cos(m_theta3)
            + (Mathf.Sin(m_B2) * Mathf.Sin(m_theta2) * Mathf.Sin(m_B3) *
Mathf.Sin(m_theta3)
            + Mathf.Cos(m_B2) * Mathf.Cos(m_B3)));
        //print("CosBC: " + m_cos23);
        m_cos13 = (Mathf.Sin(m_B1) * Mathf.Cos(m_theta1) * Mathf.Sin(m_B3)
* Mathf.Cos(m_theta3)
            + (Mathf.Sin(m_B1) * Mathf.Sin(m_theta1) * Mathf.Sin(m_B3) *
Mathf.Sin(m_theta3)
            + Mathf.Cos(m_B1) * Mathf.Cos(m_B3)));
        //print("CosAC: "+ m_cos13);
        anglesReady = true;
    }

    public void NewtonsNewMethod()
    {


        int iterations = 0;

        //Initial Point to start with
```

```
        float tempRA = 1, tempRB = 1, tempRC = 1;
        // Calculating results of initial point substituted into original
system of non-linear equations
        float gRA, gRB, gRC;
        for (int i = 0; i < 50; i++)
        {
            // TODO: Make sure these are correct (Double Check)
            gRA = -(Mathf.Pow(tempRA, 2) + Mathf.Pow(tempRB, 2) - (2 *
tempRA * tempRB * m_cos12) - Mathf.Pow(m_distAB, 2));
            gRB = -(Mathf.Pow(tempRB, 2) + Mathf.Pow(tempRC, 2) - (2 *
tempRB * tempRC * m_cos23) - Mathf.Pow(m_distBC, 2));
            gRC = -(Mathf.Pow(tempRA, 2) + Mathf.Pow(tempRC, 2) - (2 *
tempRA * tempRC * m_cos13) - Mathf.Pow(m_distAC, 2));

            float p1_RA, p1_RB, p1_RC;
            float p2_RA, p2_RB, p2_RC;
            float p3_RA, p3_RB, p3_RC;

            p1_RA = ((2 * tempRA) - (2 * tempRB * m_cos12));
            p1_RB = ((2 * tempRB) - (2 * tempRA * m_cos12));
            p1_RC = 0;

            p2_RA = 0;
            p2_RB = ((2 * tempRB) - (2 * tempRC * m_cos23));
            p2_RC = ((2 * tempRC) - (2 * tempRB * m_cos23));

            p3_RA = ((2 * tempRA) - (2 * tempRC * m_cos13));
            p3_RB = 0;
            p3_RC = ((2 * tempRC) - (2 * tempRA * m_cos13));

            // Now it's time to create the submatrix and solve it with the
jacobian method.
            float sm1_const, sm2_const, sm3_const;
            float sm1_RA, sm1_RB, sm1_RC;
            sm1_RA = p1_RA;
            sm1_RB = p1_RB;
            sm1_RC = p1_RC;
            sm1_const = (-tempRA * p1_RA) + (-tempRB * p1_RB) + (-tempRC *
p1_RC);
            // sm1_RA + sm1_RB + sm1_RC + sm1_const = gRA
            float sm2_RA, sm2_RB, sm2_RC;
            sm2_RA = p2_RA;
            sm2_RB = p2_RB;
            sm2_RC = p2_RC;
            sm2_const = (-tempRA * p2_RA) + (-tempRB * p2_RB) + (-tempRC *
p2_RC);
            // sm2_RA + sm2_RB + sm2_RC + sm2_const = gRB
            float sm3_RA, sm3_RB, sm3_RC;
            sm3_RA = p3_RA;
            sm3_RB = p3_RB;
            sm3_RC = p3_RC;
            sm3_const = (-tempRA * p3_RA) + (-tempRB * p3_RB) + (-tempRC *
p3_RC);
            // sm3_RA + sm3_RB + sm3_RC + sm3_const = gRC
```

```
            //The matrix to solve will have the following form...Do a
jacobian solve on it...
            //sm1_RA + sm1_RB + sm1_RC + sm1_const = gRA
            //sm2_RA + sm2_RB + sm2_RC + sm2_const = gRB
            //sm3_RA + sm3_RB + sm3_RC + sm3_const = gRC
            Matrix4x4 matrixA = Matrix4x4.identity;
            matrixA[0, 0] = sm1_RA;
            matrixA[0, 1] = sm1_RB;
            matrixA[0, 2] = sm1_RC;

            matrixA[1, 0] = sm2_RA;
            matrixA[1, 1] = sm2_RB;
            matrixA[1, 2] = sm2_RC;

            matrixA[2, 0] = sm3_RA;
            matrixA[2, 1] = sm3_RB;
            matrixA[2, 2] = sm3_RC;
            //Debug.Log("Matrix:\n" + matrixA);
            //Debug.Log("Matrix Inverse:\n" + matrixA.inverse);

            Matrix4x4 matrixB = Matrix4x4.zero;

            // setting the final column vector of the matrix to be the
constant vector
            matrixB[0, 3] = (gRA - sm1_const);
            matrixB[1, 3] = (gRB - sm2_const);
            matrixB[2, 3] = (gRC - sm3_const);

            Matrix4x4 solution = matrixA.inverse * matrixB;
            //Debug.Log("Solution?\n" + solution);
            if (tempRA == solution[0, 3] && tempRB == solution[1, 3] &&
tempRC == solution[2, 3])
            {
                print("Problem Solved!");
                break;
            }
            else
            {
                tempRA = solution[0, 3];
                tempRB = solution[1, 3];
                tempRC = solution[2, 3];
            }
            iterations++;
            //if (gRA < .001f && gRB < .001f && gRB < .001f)
            //    break;

            if (i == 29)
            {
                if (gRA > .001f || gRB > .001f || gRB > .001f)
                    i--;
            }

        }
```

```
        // How close to zero did we get?
        gRA = (Mathf.Pow(tempRA, 2) + Mathf.Pow(tempRB, 2) - (2 * tempRA *
tempRB * m_cos12) - Mathf.Pow(m_distAB, 2));
        gRB = (Mathf.Pow(tempRB, 2) + Mathf.Pow(tempRC, 2) - (2 * tempRB *
tempRC * m_cos23) - Mathf.Pow(m_distBC, 2));
        gRC = (Mathf.Pow(tempRA, 2) + Mathf.Pow(tempRC, 2) - (2 * tempRA *
tempRC * m_cos13) - Mathf.Pow(m_distAC, 2));
        // print("Iterations: " + iterations);
        //print("RA got this close to zero: " + gRA);
        //print("RB got this close to zero: " + gRB);
        //print("RC got this close to zero: " + gRC);
        //if (gRA > .001f || gRB > .001f || gRB > .001f)
        //    return;
        m_RA = tempRA;
        m_RB = tempRB;
        m_RC = tempRC;
        //print("RA: " + m_RA + " At " + iterations);

        if (firstPosition == false)
        {
            firstPosition = true;
            //A_offsetX = m_theta1 * Mathf.Rad2Deg;
            //A_offsetY = m_B1 * Mathf.Rad2Deg;
            //A_offsetZ = m_RA;

            A_offsetX = Mathf.Abs(m_RA) * Mathf.Sin(m_B1) *
Mathf.Cos(m_theta1) - startPos.x;
            A_offsetY = Mathf.Abs(m_RA) * Mathf.Sin(m_B1) *
Mathf.Sin(m_theta1) - startPos.y;
            A_offsetZ = Mathf.Abs(m_RA) * Mathf.Cos(m_B1) - startPos.z;
            print("OFFSETX: " + A_offsetX + " OFFSETY: " + A_offsetY +
"OFFSETZ: " + A_offsetZ);


        }
    }

    public void OnApplicationQuit()
    {
        receiveThread.Abort();
        if (stream != null)
        {
            //stream.Close();
            client.Close();
        }
    }

    public void NewtonsTestMethod()
    {
        int iterations = 0;

        //Initial Point to start with
        float tempRA = 1, tempRB = 1, tempRC = 1;
```

```
        // Calculating results of initial point substituted into original
system of non-linear equations
        float gRA, gRB, gRC;
        for (int i = 0; i < 30; i++)
        {
            // TODO: Make sure these are correct (Double Check)
            gRA = -(Mathf.Pow(tempRA, 2) + Mathf.Pow(tempRB, 2) - (2 *
tempRA * tempRB * m_cos12) - Mathf.Pow(m_distAB, 2));
            gRB = -(Mathf.Pow(tempRB, 2) + Mathf.Pow(tempRC, 2) - (2 *
tempRB * tempRC * m_cos23) - Mathf.Pow(m_distBC, 2));
            gRC = -(Mathf.Pow(tempRA, 2) + Mathf.Pow(tempRC, 2) - (2 *
tempRA * tempRC * m_cos13) - Mathf.Pow(m_distAC, 2));
            print("gRA: " + gRA);
            print("gRB: " + gRB);
            print("gRC: " + gRC);

            float p1_RA, p1_RB, p1_RC;
            float p2_RA, p2_RB, p2_RC;
            float p3_RA, p3_RB, p3_RC;

            p1_RA = ((2 * tempRA) - (2 * tempRB * m_cos12));
            p1_RB = ((2 * tempRB) - (2 * tempRA * m_cos12));
            p1_RC = 0;

            print(lineNumber + " Jacobian1: " + p1_RA + " + " + p1_RB + "
+ " +p1_RC);
            lineNumber++;
            p2_RA = 0;
            p2_RB = ((2 * tempRB) - (2 * tempRC * m_cos23));
            p2_RC = ((2 * tempRC) - (2 * tempRB * m_cos23));

            print(lineNumber+ " Jacobian2: " + p2_RA + " + " + p2_RB + "
" + p2_RC);
            lineNumber++;
            p3_RA = ((2 * tempRA) - (2 * tempRC * m_cos13));
            p3_RB = 0;
            p3_RC = ((2 * tempRC) - (2 * tempRA * m_cos13));

            print(lineNumber + " Jacobian3: " + p3_RA + " + " + p3_RB + "
+ " + p3_RC);
            lineNumber++;
            // Now it's time to create the submatrix and solve it with the
jacobian method.
            float sm1_const, sm2_const, sm3_const;
            float sm1_RA, sm1_RB, sm1_RC;
            sm1_RA = p1_RA;
            sm1_RB = p1_RB;
            sm1_RC = p1_RC;
            sm1_const = (-tempRA * p1_RA) + (-tempRB * p1_RB) + (-tempRC *
p1_RC);
            // sm1_RA + sm1_RB + sm1_RC + sm1_const = gRA
            float sm2_RA, sm2_RB, sm2_RC;
            sm2_RA = p2_RA;
            sm2_RB = p2_RB;
```

```
            sm2_RC = p2_RC;
            sm2_const = (-tempRA * p2_RA) + (-tempRB * p2_RB) + (-tempRC *
p2_RC);
            // sm2_RA + sm2_RB + sm2_RC + sm2_const = gRB
            float sm3_RA, sm3_RB, sm3_RC;
            sm3_RA = p3_RA;
            sm3_RB = p3_RB;
            sm3_RC = p3_RC;
            sm3_const = (-tempRA * p3_RA) + (-tempRB * p3_RB) + (-tempRC *
p3_RC);
            // sm3_RA + sm3_RB + sm3_RC + sm3_const = gRC

            //The matrix to solve will have the following form...Do a
jacobian solve on it...
            //sm1_RA + sm1_RB + sm1_RC + sm1_const = gRA
            //sm2_RA + sm2_RB + sm2_RC + sm2_const = gRB
            //sm3_RA + sm3_RB + sm3_RC + sm3_const = gRC
            Matrix4x4 matrixA = Matrix4x4.identity;
            matrixA[0, 0] = sm1_RA;
            matrixA[0, 1] = sm1_RB;
            matrixA[0, 2] = sm1_RC;

            matrixA[1, 0] = sm2_RA;
            matrixA[1, 1] = sm2_RB;
            matrixA[1, 2] = sm2_RC;

            matrixA[2, 0] = sm3_RA;
            matrixA[2, 1] = sm3_RB;
            matrixA[2, 2] = sm3_RC;
            //Debug.Log("Matrix:\n" + matrixA);
            //Debug.Log("Matrix Inverse:\n" + matrixA.inverse);

            Matrix4x4 matrixB = Matrix4x4.zero;

            // setting the final column vector of the matrix to be the
constant vector
            matrixB[0, 3] = (gRA - sm1_const);
            matrixB[1, 3] = (gRB - sm2_const);
            matrixB[2, 3] = (gRC - sm3_const);
            print("SM1_Const: " + sm1_const);
            print("SM2_Const: " + sm2_const);
            print("SM3_Const: " + sm3_const);

            print(lineNumber + " Equals: " + (gRA - sm1_const) + ", " +
(gRB - sm2_const) + ", " + (gRC - sm3_const));
            lineNumber++;

            Matrix4x4 solution = matrixA.inverse * matrixB;
            //Debug.Log("Solution?\n" + solution);
            if (tempRA == solution[0, 3] && tempRB == solution[1, 3] &&
tempRC == solution[2, 3])
            {
                print("Problem Solved!");
                break;
```

```
            }
            else
            {
                tempRA = solution[0, 3];
                tempRB = solution[1, 3];
                tempRC = solution[2, 3];
                print(lineNumber + "Solution: " + tempRA + ", " + tempRB +
", " + tempRC);
                lineNumber++;
            }
            iterations++;
            //if (gRA < .001f && gRB < .001f && gRB < .001f)
            //    break;

            if (i == 29)
            {
                if (gRA > .001f || gRB > .001f || gRB > .001f)
                    i--;
            }

        }

        // How close to zero did we get?
        gRA = (Mathf.Pow(tempRA, 2) + Mathf.Pow(tempRB, 2) - (2 * tempRA *
tempRB * m_cos12) - Mathf.Pow(m_distAB, 2));
        gRB = (Mathf.Pow(tempRB, 2) + Mathf.Pow(tempRC, 2) - (2 * tempRB *
tempRC * m_cos23) - Mathf.Pow(m_distBC, 2));
        gRC = (Mathf.Pow(tempRA, 2) + Mathf.Pow(tempRC, 2) - (2 * tempRA *
tempRC * m_cos13) - Mathf.Pow(m_distAC, 2));
        // print("Iterations: " + iterations);
        print(lineNumber+ " RA got this close to zero: " + gRA);
        lineNumber++;
        print(lineNumber + " RB got this close to zero: " + gRB);
        lineNumber++;
        print(lineNumber + " RC got this close to zero: " + gRC);
        lineNumber++;
        //if (gRA > .001f || gRB > .001f || gRB > .001f)
        //    return;
        m_RA = tempRA;
        m_RB = tempRB;
        m_RC = tempRC;
        print(lineNumber + " RA: " + m_RA + " At " + iterations);
        lineNumber++;
        anglesReady = false;
    }

    private void updatePosition()
    {
        float x = Mathf.Abs(m_RA) * Mathf.Sin(m_B1) * Mathf.Cos(m_theta1);
        float y = Mathf.Abs(m_RA) * Mathf.Sin(m_B1) * Mathf.Sin(m_theta1);
        float z = Mathf.Abs(m_RA) * Mathf.Cos(m_B1);

        //print(lineNumber + ". X: " + x + " Y: " + y + " Z: " + z);
        //lineNumber++;
```

```
        //newPos = new Vector3((x), (y), (z));
        //float theta1Deg = m_theta1 * Mathf.Rad2Deg;
        //float b1Deg = m_B1 * Mathf.Rad2Deg;
        //newPos = new Vector3((theta1Deg - A_offsetX), (b1Deg -
A_offsetY), (m_RA - A_offsetZ));
        newPos = new Vector3(-(x - A_offsetX), -(z - A_offsetZ), -(y -
A_offsetY));
    }

    // Update is called once per frame
    void Update () {
        //Vector3 newpos = new Vector3(m_theta1, m_B1, 0);
        this.transform.position = newPos;
        //NewtonsNewMethod();
       //transform.position = Vector3.Lerp(this.transform.position,
newPos, Time.deltaTime);
    }



    public void test()
    {
        try
        {
            byte[] data = client.Receive(ref anyIP);
            string text = "";
            text = Encoding.ASCII.GetString(data);
            string[] splitArray = parseData(text);
            if (splitArray == null)
                return;
            print("UNVALIDATED: " + splitArray[0] + splitArray[1] + "" +
splitArray[2] + splitArray[3]);
            lastReceivedUDPPacket = text;
            inputValidation(splitArray);
        }
        catch (Exception err)
        {
            print(err.ToString());
        }

        try
        {
            byte[] data = client.Receive(ref anyIP);
            string text = "";
            text = Encoding.ASCII.GetString(data);
            string[] splitArray = parseData(text);
            if (splitArray == null)
                return;
            print("UNVALIDATED: " + splitArray[0] + splitArray[1] + "" +
splitArray[2] + splitArray[3]);
            lastReceivedUDPPacket = text;
            inputValidation(splitArray);
            print("b1 = " + m_B1 + " b2 = " + m_B2 + " b3 = " + m_B3);
```

```
            print("theta1 = " + m_theta1 + " theta2 = " + m_theta2 + "
theta3 = " + m_theta3);
            extractAngles();
            NewtonsTestMethod();
            //updatePosition();
        }
        catch (Exception err)
        {
            print(err.ToString());
        }

    }


}
```