



Moving Human Electromagnetic Scattering Simulator

ECE480 Final Report

December 05, 2012

Design Team 2: Will Juszczyk, Ryan Lattrel, Michael Lazar, Camden Smith, Mark Birdsall

Sponsored by:

Air Force Research Laboratory

Sensors Directorate WPAFB, OH

With the assistance of Dr. Analee Miranda and Mr. Kenneth Schafer

Faculty Facilitator: Dr. Wen Li

Acknowledgements

We would like to thank the Air Force Research Laboratory for providing us with the opportunity and financial support to complete this project, and thanks to Dr. Analee Miranda for her help in working us through the project details. Also, we would like to thank our facilitator Dr. Wen Li for advising and helping us throughout the semester.

Executive Summary

The objective of our design is to create a portable system capable of collecting and processing data on persons detected by an attached sensor. The system needs to be accessible via Ethernet or Wi-Fi and should be able to transmit data to a computer connected to the same network. To accomplish this task a microcontroller is connected to a depth and motion sensing imaging device. The data collected by the imaging device is processed by the microcontroller and is forwarded to client devices connected over an embedded Wi-Fi chip. The sensor needs to be read by open source software being utilized by the microcontroller, and should be able to determine the sizes of different parts of a target's body. The sizes of the target should then be submitted to an AFRL provided program that will complete an electromagnetic scattering simulation algorithm.

Contents

Chapter 1 - Introduction and Background.....	2
Chapter 2 – Selecting an Approach.....	3
System Specifications	3
Subsystem Design and Project Management	3
Chapter 3 – Technical Work.....	6
Hardware Requirements.....	6
Hardware Implementation.....	6
Software Requirements	10
Software Implementation	11
Chapter 4 – Testing	19
Testing Overview.....	19
Testing the Xbox Kinect.....	19
Testing the ASUS Xtion.....	20
Testing the PandaBoard Server Program	21
Testing the Xtion with a PandaBoard.....	21
Testing the PC application.....	22
Testing the Battery	23
Testing Conclusion.....	23
Chapter 5 – Conclusions.....	24
Appendix I – Technical Roles.....	26
Will Juszczuk	26
Ryan Lattrel	27
Camden Smith.....	28
Michael Lazar	29
Appendix II – References.....	30
Appendix III – Technical Attachments	31

Chapter 1 - Introduction and Background

Radar systems are important pieces of equipment in monitoring and tracking humans and other objects. Radar systems are extremely effective because of their ability to be able to track in many different conditions including night, fog, and smoke or dust. In environments with these conditions visible sensors and cameras do not work. This is why it is important to create a visible sensor that can use radar technologies.

Electromagnetic scattering simulations are readily available software technologies that model the radar scattering phenomenon. These pieces of software can model complex radar scattering events related to humans including a person walking or running. Currently, these programs require a person to manually input human parameters or features into a program. The goal of this project is to complete this task automatically, which will eliminate the problems that exist with the current methodology, such as these electromagnetic scattering simulations not being able to produce results in real time. This means that current implementations of electromagnetic scattering simulation are limited.

The Air Force Research Laboratory (AFRL) has requested a lightweight, portable system that will capture human movement and send parameters that characterize identified humans to a PC application that will turn these parameters into a radar scattering model. After communication with our sponsors at the AFRL we learned that their electromagnetic scattering software is still a work in progress and the main objective of our system is to create a device that can automatically identify a human and calculate parameters such as head size, arm and leg length, and torso size. The goal is then to transmit this data to a PC so their future software can be run with the parameters received from this system.

The system is to be implemented using a microcontroller and an imaging device such as an Xbox Kinect. An Xbox Kinect is a cheap and powerful tool that can be used to accomplish the needs of the project in the area of automatically detecting and parsing persons. The Xbox Kinect is a result of recent developments in the gaming industry. New camera technologies have been developed at a fraction of the price of those that existed before 2010 when the Xbox Kinect was released.

The Kinect combines an infrared camera with an RGB camera. Microsoft combined forces with an Israeli company, PrimeSense, to develop a device that can identify multiple humans at once and recognize gestures made by the humans. Due to readily available software released to developers by PrimeSense and the decision by Microsoft to unlock the USB port for use by all who wish to create a use for it, it is possible for anyone to use an Xbox Kinect or similar sensor for their own purposes, and easily detect and track people.

Chapter 2 – Selecting an Approach

System Specifications

The design requirements for the Moving Electromagnetic Human Scattering Simulator were very open-ended. There were numerous ways that the prototype could be implemented. A specific microcontroller was not required or recommended, and any imaging device could be used. In addition to an open-ended hardware requirement, the software requirement was much of the same. The software was required to automatically detect the size of targets found by the sensor on the microcontroller, send the processed information to a PC, and run software provided by the AFRL using the received data.

The specific requirements provided by the Sponsor are as follows:

1. A small, portable, lightweight system
2. Read an imaging device using a microcontroller
3. Take advantage of open-source software to read the imaging device
4. Accurately Identify humans based off of human body characteristics
5. Connect to a PC over Ethernet or Wi-Fi
6. Run the AFRL provided software from a PC using data received from the microcontroller

These requirements are not the same as the requirements given in the initial project description, and changed slightly over the course of the project. The changes were to the PC software. The AFRL software took care of all of the advanced imaging algorithms such as the FFT and back-projection imaging techniques, allowing us to simplify the program for the PC client to only collecting data from the microcontroller and designing a way to send that data to the AFRL provided software.

Subsystem Design and Project Management

To design a system to meet the needs of the project sponsor, the project was first analyzed using several different methods, and from there broken down into several different subsystems that would be needed to meet the project goals.

The project was divided into three subsections: the hardware, the microcontroller software, and the PC software. Using a FAST diagram (Figure A3.3 in Appendix III) the team was able to determine what functionality would be needed to accomplish the overall goal of

tracking people and processing their parameters with scattering simulator software. The design of the three subsections was then focused on meeting these needs. To manage the project, each subsection was determined to require the work of two or three team members depending upon the complexity of the section. The team resources would also be dynamically allocated to different sections as needed.

The team developed a Gantt chart (Figure A3.4) to initially plan how to best allocate our time and team member's talents. The Gantt chart was designed to be extremely flexible while still allowing ample time to complete our overall task. During the work on our project, there were some changes that had to be accounted for, one of these being that when our camera was incompatible with our microcontroller the team began to fall slightly behind. A new camera was ordered and a new Gantt chart (Figure A3.5) was developed. The new Gantt chart was similar to the old in that it was flexible and dynamically allocated resources; however, it had a stricter schedule that the team agreed had to be completed on time.

The decision for the microcontroller needed for the project was based on a number of different required features. The most important feature of the controller was that it needed to be able to run a Linux operating system. This is due to the requirement of utilizing open source software to assist in the implementation of the design. Several design matrices were used to help come to a decision about the most appropriate controller to use. These matrices can be seen in Figures A3.6 and Figure A3.7 in Appendix III. Initial ideas included using an Arduino due to the simplicity of the device. It is easy to program and extremely cheap. It quickly became apparent however that not only would the device not be powerful enough to process all the data from the imaging device, it also cannot run Linux or take advantage of any common open source libraries. Eventually the decision came down to two different ARM based boards: the BeagleBoard and the PandaBoard. Comparing the features and the price the PandaBoard was chosen. This was mostly due to the Wi-Fi capabilities built into the board that were not available on the other option. Based on research of other projects using the PandaBoard, it was found that it is possible to install Linux on it, connect it to an imaging device and then read data from the device using the open source OpenNI library, along with an additional library called NITE.

The software decisions did not require any decision matrices, as the process was straightforward. In order to fulfill the requirements of the sponsor the PandaBoard software simply had to collect and process data on targets found with the imaging device, and transmit them over a network to a PC. The only option available for us to do this was to use C++ with the OpenNI and NITE libraries that were found during the research stages of the project. To transmit data between computers a socket would be needed and this could be done by using standard Linux C++ libraries. The PC software had more options, however, due to the team's

familiarity with Microsoft's .NET framework, and no strict guidelines from the sponsor, C# was determined to be the best way to connect and receive data from the PandaBoard, and from there launch any software provided by the AFRL.

The initial cost of the system is purely based on the cost of the hardware. All of the software is custom built and uses open sources libraries or development packages freely available to students. The estimate for building the prototype was just under the budget provided. The microcontroller budget was about \$200, and the imaging device was \$100. It was decided to include a battery with ours system as well to help meet the sponsor's goal of having a portable system, and the complete battery and power system was estimated to be about \$100. The final budget can be seen in the appendix in Figure A3.8.

Chapter 3 – Technical Work

Hardware Requirements

The hardware requirements for our project can be split into 4 major components: the sensor, the microcontroller, the power system and the enclosure.

- 1 The sensor is the project's source of information. It must have the ability to identify human targets with precision and reliably. It must also work in a variety of environments with different lighting and no manual calibration. We needed a sensor capable of providing a continuous stream of data to a microcontroller through a standard connector, preferably USB. It needed to be compatible with open-source or free image processing software. Finally, because of our project's budget constraints we did not want to spend over ~\$150 for a sensor.
- 2 The microcontroller facilitates the flow of data from the sensor to an external PC. This means accepting the output from the sensor, parsing the information into human parameters and streaming this information. In order to work with the sensor's drivers, we were restricted to microprocessors capable of running a Linux kernel. In order to parse the information in real-time, we needed to select a microcontroller with sufficient processing power and RAM. Our microcontroller needed to be capable of Ethernet or wireless communication either built-in or through an attachable shield.
- 3 The power system must provide a stable and reliable source of power to both the microcontroller and the sensor. This could be as simple as a pair of wall adapters running from a wall outlet to the device. A system that includes a battery is desirable for portability, but not required. Also, the user should be able to easily turn the system on and off in a reasonable amount of time, and have the system run for extended periods of time.
- 4 The project required an enclosure in order to protect the internal components and allow the user to move the sensor around. Our goal was to build an enclosure that could be entirely operated through built-in ports and switches, eliminating the need for the user to open up the enclosure during normal use. The sensor needed to be firmly attached to the enclosure and the entire design needed to be lightweight and portable.

Hardware Implementation

The first step in hardware implementation was getting our PandaBoard to function properly on its own, and this meant providing power. The PandaBoard will accept a 5V input from its dedicated barrel connector, USB, or POE (Power Over Ethernet). After compiling a basic estimate of the current that we expected the board to draw, we decided that the barrel

connector was the most appropriate choice. Over the course of development, we used a standard wall adapter to develop and test the PandaBoard. After we were able to power the microcontroller we were able to connect both our input sensor to the PandaBoard USB port and an antenna in order to extend its signal and range.

One of our project goals was to design a portable enclosure for our sensor and microcontroller. The main technical challenge associated with this was to find an effective way for the user to provide power to the enclosure and both devices contained within it. Our goal was to allow the enclosure to switch from being plugged-in (for extended use) to running on battery power (for shorter, portable sessions). An operational flowchart for our system's main components can be seen below (Figure 3.1). The team underwent multiple design iterations in order to accomplish this task.

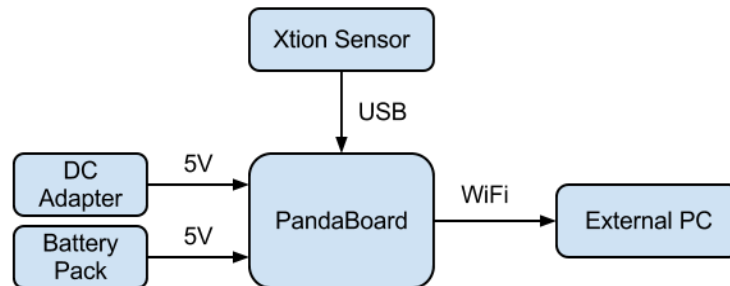


FIGURE 3.1

The first system was designed to function with a Kinect sensor. A description of the Kinect sensor can be found in the appendix (A3.1). This proved to be a challenge because the Kinect requires a substantial amount of power to operate, and as a result it has a separate 12V cord in addition to its USB cord. We decided that the most elegant approach would be to design a system with a single power supply and use regulators to drop the voltage to appropriate levels. The envisioned solution would behave similarly to a laptop, where it could be plugged and unplugged without any interruption in operation. The decision was made to use lithium-based batteries due to their exceptional capacity to size ratio. Lithium batteries have several special charging requirements which can be dangerous if implemented incorrectly. For this reason, we selected a pre-built UPS device that was capable of automatically charging and power-path switching between our battery and a wall adapter. The two modes of power-path operation are depicted in the figures below (Figure 3.2, 3.3). The UPS was expensive, but saved us the trouble of dealing with complex surface mounted battery management ICs.

Between the UPS, battery and regulators, we would have been almost exactly at our project's budget. This system was never implemented, as switching from the Kinect sensor made it no longer compatible with our new requirements and reduced budget.

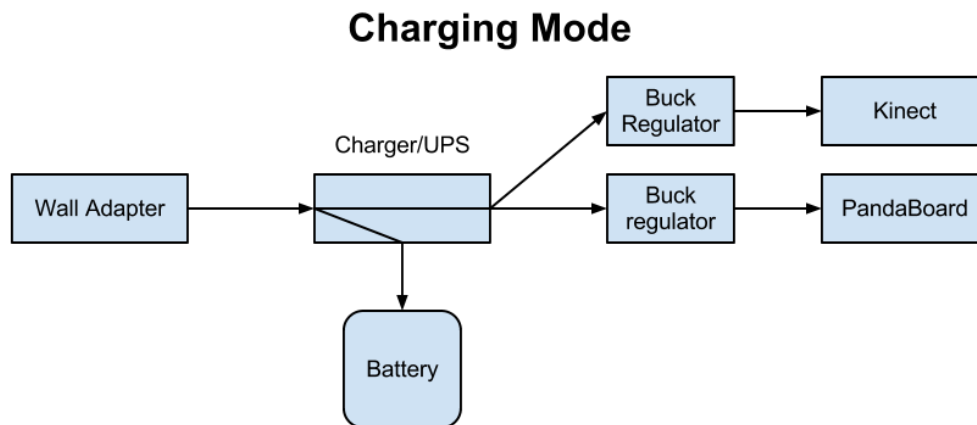


FIGURE 3.2

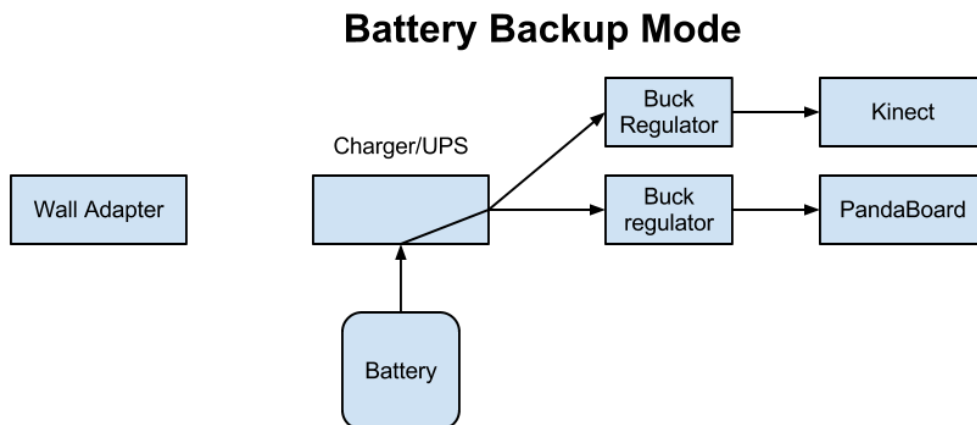


FIGURE 3.3

After our team switched to the ASUS Xtion sensor, we were left with a limited timeframe to design and implement a new power system. A description of the Xtion sensor can be found in the appendix (Figure A3.2). Fortunately, the Xtion sensor does not require dedicated power in addition to USB, and this simplified the design problem. It was decided that the best option would be to use an external power pack. These packs are marketed as portable

chargers. They contain built-in lithium batteries with charging circuitry and can deliver high currents with a regulated output. Some of our main reasons for selecting this option are its simplicity, reliability, and low-cost. However, it also means that our final design is not capable of automatically switch between battery and wall power without resetting the PandaBoard. The battery must be taken out and charged independently. A pair of switches was added to the design in order to allow the user to manually turn power on/off and select between the wall adapter and battery power. The diagram of the final design is shown below (Figure 3.4).

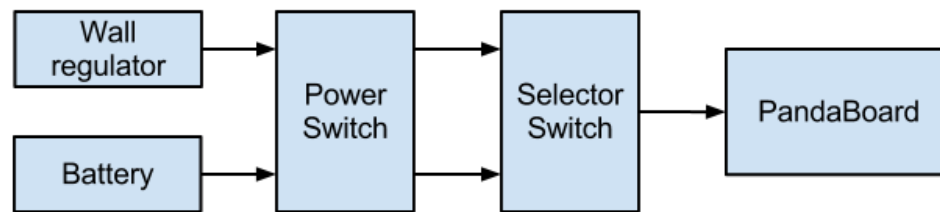


FIGURE 3.4

Once the design was finalized, all of the parts were ordered. After we received the enclosure and the necessary components to control the internal hardware, we began measuring the PandaBoard and battery to see how they would fit into the enclosure. Most importantly, we wanted to make sure we could still reach the button on the battery when the PandaBoard was mounted in place. We realized the PandaBoard could only fit in the enclosure at an angle. To solve this problem, we cut a hole in the enclosure for the USB Mini port and Serial port. The holes were cut just above the height of the battery so the ports on the PandaBoard could slide into the holes in the enclosure and the board could rest on top of the battery. Holes were then cut on the side of the enclosure for the power switch, battery/AC control switch, and the port for the AC adapter. On the exterior of the enclosure, the Xtion was mounted on top of the lid and the component switches and plug for AC adapter were labeled. A hole also had to be cut so the cord coming from the Xtion could be tucked into the box and could access the USB port on PandaBoard; making every aspect of our final design completely locked in or on our enclosure. While wiring these connections together, we made care not to directly modify any of the components. This way, the board, sensor and battery can all be unplugged and taken out of the enclosure if desired, as seen in Figure 3.5 below.

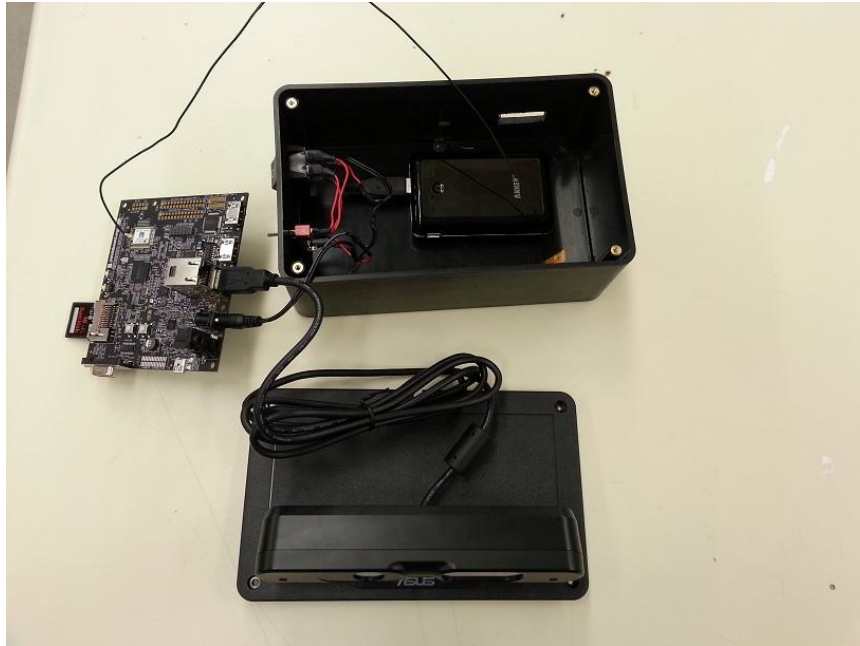


FIGURE 3.5

Software Requirements

There were two separate programs that were required in order to complete the prototype. One program needed to be run on our microcontroller and another needed to communicate with the microcontroller program and run on a client PC. In order to develop an advanced program for the PandaBoard, we also had to install Linux on the board. Linux allows the program to take advantage of built in operating system functionality, such as access to Ethernet and Wi-Fi, as well as access to libraries that can access and read the sensor device needed to detect and analyze humans.

The program developed for the PandaBoard has several different requirements. The first requirement was that it needs to read and parse data from the sensor connected to it. This task was accomplished by using the libraries OpenNI and NITE. These libraries have been developed in order to allow developers to take advantage of the detection abilities of the sensor without having to write their own software to track targets with the device's depth sensor data. In addition to reading the sensor, the software also needed to transmit data over an Ethernet device, ideally over Wi-Fi to allow better portability of the prototype. Using Linux made it simple to develop code which uses standard sockets to read and write over Ethernet.

For the PC client program the only requirements were to read data from the microcontroller program and execute Matlab code provided by the AFRL. In addition to this, a tracking system was developed in order to identify different targets. This gives the ability to

choose which target to examine with the AFRL software, instead of only allowing the first person found to be plotted.

Software Implementation

The first step of the software implementation was to successfully install Linux on the PandaBoard and set up the OpenNI and NITE libraries. There were a lot of difficulties in getting the correct version of Linux installed which would be compatible with the required libraries. The main issue encountered was that the PandaBoard utilizes an ARM core processor, while the latest versions of the libraries are developed for use with a standard X86 processor architecture. This made finding a version of the libraries that would work difficult. Initially Ubuntu 12.04 was installed on the PandaBoard as this version is the most stable version of Linux for the PandaBoard that could be found. Ubuntu 12.04, however, did not work with the NITE library binaries that were located. The NITE library is not open source, and therefore could not be compiled for this version of Linux.

After doing a bit of research, it was found that the NITE libraries would be compatible with Ubuntu 11.10. There was a bug in the installer for the 11.10 version which led to some issues getting it to run, but ultimately installing it ended in success. This however only led to further problems, as the Xbox Kinect drivers were not compatible with Ubuntu 11.10, but OpenNI and NITE were. In order to resolve this issue, another sensor was located - the ASUS Xtion. Conveniently, packaged with the NITE binaries found, were the driver binaries for the ASUS Xtion. This led us to swapping the Xbox Kinect for the ASUS Xtion in our design. The devices are nearly identical and so there weren't any issues regarding functionality due to this change.

To simplify the connectivity between the PandaBoard and the PC, the PandaBoard was turned into a Wi-Fi access point. A representation of this is in Figure 3.6 below. The way this simplifies the process is that by creating an access point a permanent IP address can be assigned to the PandaBoard. It also creates a unique network that only contains the devices needed for our system. This was done using open source software available for Ubuntu. First an access point is setup, with an SSID and an encryption key. An IP address is then set for the PandaBoard. After this DHCP software also needs to be enabled so the PandaBoard can give clients connecting to it their own addresses. SSH was also configured on the PandaBoard so full control of the system can be enabled through a PC terminal. Updates can also be pushed to the board wirelessly without the need to ever remove the PandaBoard from its enclosure.



FIGURE 3.6

The PandaBoard program was developed in C++ and designed to run on the Linux operating system. As mentioned, using Linux allows both the ability to access the sensor as well as the Ethernet devices. The application was designed using object oriented programming design methods. The influence of these objects and the design was directly from the requirements. Two main functions were needed by the program: the ability to receive data from the sensor, and the ability to forward that data to a PC. The second requirement meant that a protocol would need to be developed in order to reliably transmit information to a PC, as well as allow give the ability to the PC to control what the PandaBoard program was doing, and when it should do it. The PandaBoard server is designed to serve any number of connected clients, meaning that more than one computer can run the PC client software in real time, simultaneously. Figure 3.7 below shows the overall design of the objects used by the PandaBoard application.

The sensor portion of the code was developed using the OpenNI libraries. There are many examples that can be found online that show how to start detecting with the sensor, and how to find the location of the found user's joints. The data received from the libraries skeleton detection code is what was utilized in order to determine the parameters needed by the AFRL. These parameters include the user's arm length, leg length, and head diameter. In order to calculate this data, several different skeleton joint positions would be found. For example, the arm length was calculated as the length from the hand to the elbow, added to the length from the elbow to the shoulder. The data received about the skeleton is in XYZ coordinates; therefore, a standard equation was used in order to determine the length:

$$\text{SQRT}((X2 - X1)^2 + (Y2 - Y1)^2 + (Z2 - Z1)^2)$$

Unfortunately, the sizes calculated by the PandaBoard software are questionable. The data changes rapidly and produces unrealistic values in many instances. The cause of this has been isolated, however, and is from the fact that the closed source NITE software gives the incorrect values of joint locations in many situations. For unknown reasons the software reports that it has confidently found the joint of a target (due to the library providing a specific ‘confidence’ value) and then gives an incorrect reading. This is likely due to the ARM binaries for this software being outdated. The newest version cannot be tested on this board because the new binaries do not exist yet, and they cannot be compiled because the code is closed source and unavailable to the public. When a new version of the software is released, it is likely that the accuracy of the program will greatly increase. When the sensor is connected to a computer running windows with the latest NITE release, there is no problem quickly and accurately finding a user’s skeleton and movement.

In order to send data to the PC application, a TCP server has to be established on the PandaBoard. This was done using standard UNIX sockets, along with a custom application protocol written on top of it. Two separate C++ classes were written in order to accomplish this goal. A lower level socket class was designed using standard C++ libraries to, one, initiate the TCP Server on the desired IP address at a specified port, and two, read and write data buffers of a given size. A network controller class was then written to use this lower level code. The network controller consisted of functions that would receive and sent application specific data. This data included messages to be relayed between the PC client and the server, along with the actual parameter data detected and calculated through the sensor.

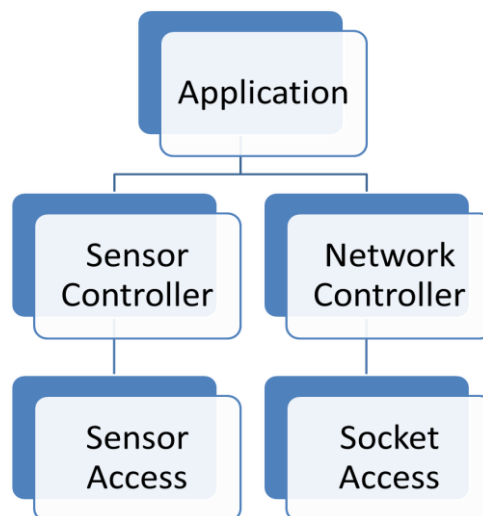


FIGURE 3.7

The PC application was developed using Microsoft's C# and .NET platform. The reason C# was chosen was due to the simplicity of integrating a GUI along with backend code, and for the numerous libraries Microsoft includes in .NET, such as a graphics plot to draw found targets, as well as socket libraries needed to communicate with the PandaBoard server. The code for the client was also developed with an object oriented design in mind, but it also utilizes a concept known as MVC (model, view, and controller). In this format the model is the raw data, which is controlled by a separate controller class, and then the view is the window, which receives data from the controller rather than the model. Similar to the PandaBoard server, there are several different functions of the PC client as well. The only requirement was that it would record the parameters of found persons for the AFRL Matlab code to use. On top of this a center of mass tracking system was developed in order to differentiate different targets, as well as determine their positions relative to one another. This provides the ability to launch the AFRL software with different parameters based on which user is chosen to analyze. A general design of the program is in Figure 3.8.

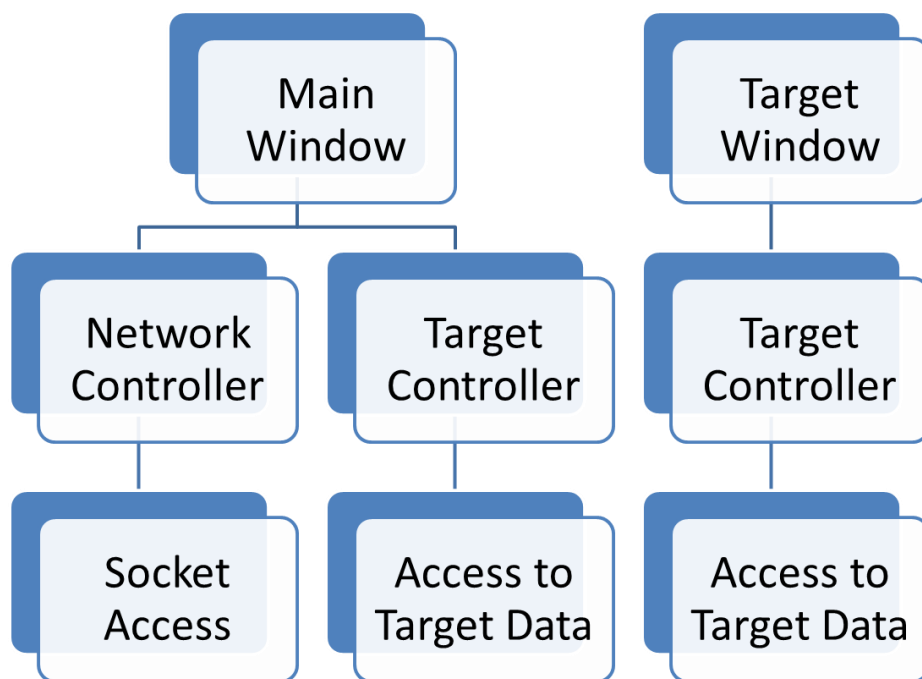


FIGURE 3.8

Two different GUI windows were created for the software. The first window is simply defined as the main window. The PandaBoard server is connected through this window's interface, and basic commands can be sent. The window is seen below in Figure 3.9. Each

number on the radar image represents a target, and the relative positioning between the targets can be identified. To connect to the server, the IP address of the PandaBoard is entered, as well as the port. The 'Connect' button is then pressed. On success a message will be displayed in the log. After this, the command 'Track' will tell the PandaBoard to start looking for targets. When a target is found it will be revealed on the graph. These points will continue to be updated until the 'Stop' command is issued. At this point the PandaBoard will quit its target finding routine and turn idle.

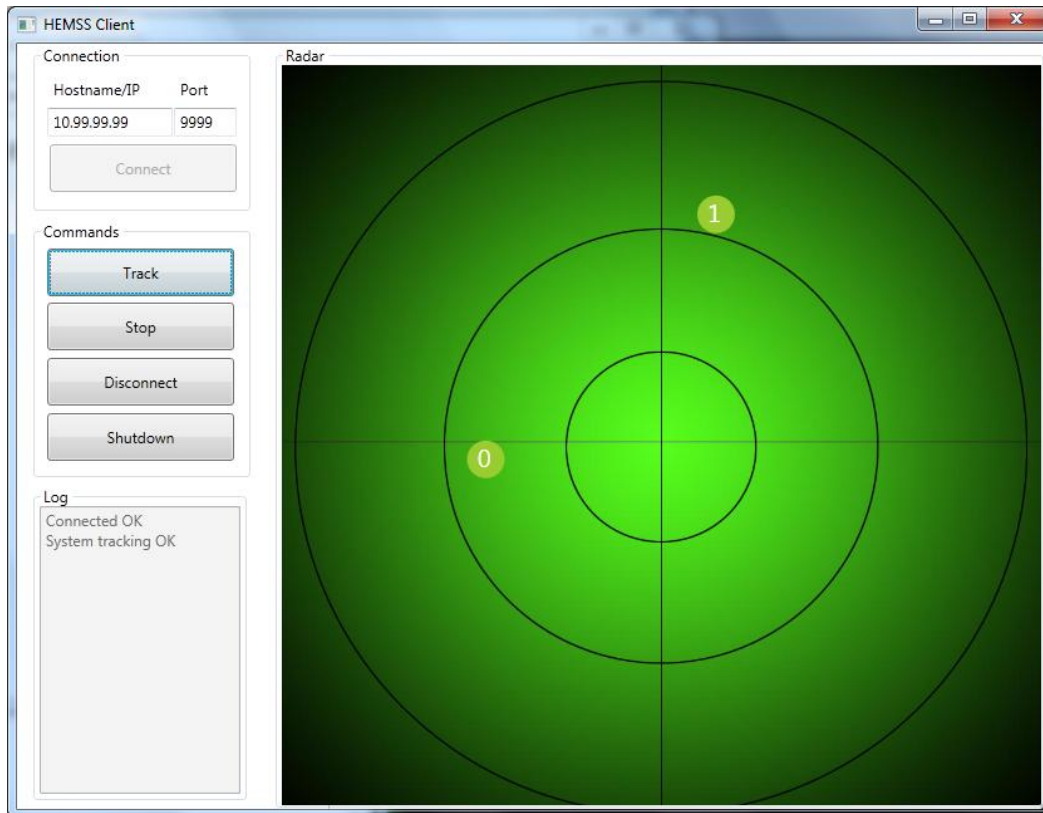


FIGURE 3.9

Along with the main window, there is also a target window - shown in Figure 3.10. This window is what contains the data to be used by the AFRL software. A target window will appear for each target person tracked by the system. At the same time that a person appears on the main window radar grid, a window will pop up with the new found user's parameters. This window will display the current calculated information about the target, as well as an option to both save these parameters to a file for later use, as well as an option to launch Matlab automatically from the interface and run the AFRL software with the displayed values.

Target 1 Parameters	
Head diameter	3.263617
Torso width	0
Torso height	4.849802
Left arm width	0
Left arm length	421.0428
Left leg width	0
Left leg length	571.3506
Right arm width	0
Right arm length	470.9273
Right leg width	0
Right leg length	588.8524

Call Matlab Save Parameters

FIGURE 3.10

The flow of the programs is observed in Figure 3.11. The programs were designed to rely on each other, and the protocol, described in Figures A3.9 and A3.10 in Appendix 3, was created specifically for this system. The system is started when a client connects to the server. The server then sends a command verifying that the client has connected without any issue. To begin the detection, the track command is to be issued from any client connected. This command will cause the server to begin utilizing the sensor and the OpenNI libraries. Once this is enabled, the server will be able to send data about every user it has detected to every connected client. A client can disconnect and reconnect without issuing the track command again, as the client will immediately begin receiving data if the system is already tracking. Once the stop command is sent this functionality will cease, and the server will idle and await new commands. A shutdown command can also be sent from any client, which will cause the PandaBoard to stop tracking turn off. This feature is implemented in order for any user to turn

off the board without an SSH terminal. The shutdown process will only be initiated if the PandaBoard server is running as a root user process, allowing for security if this feature is needed to be disabled.

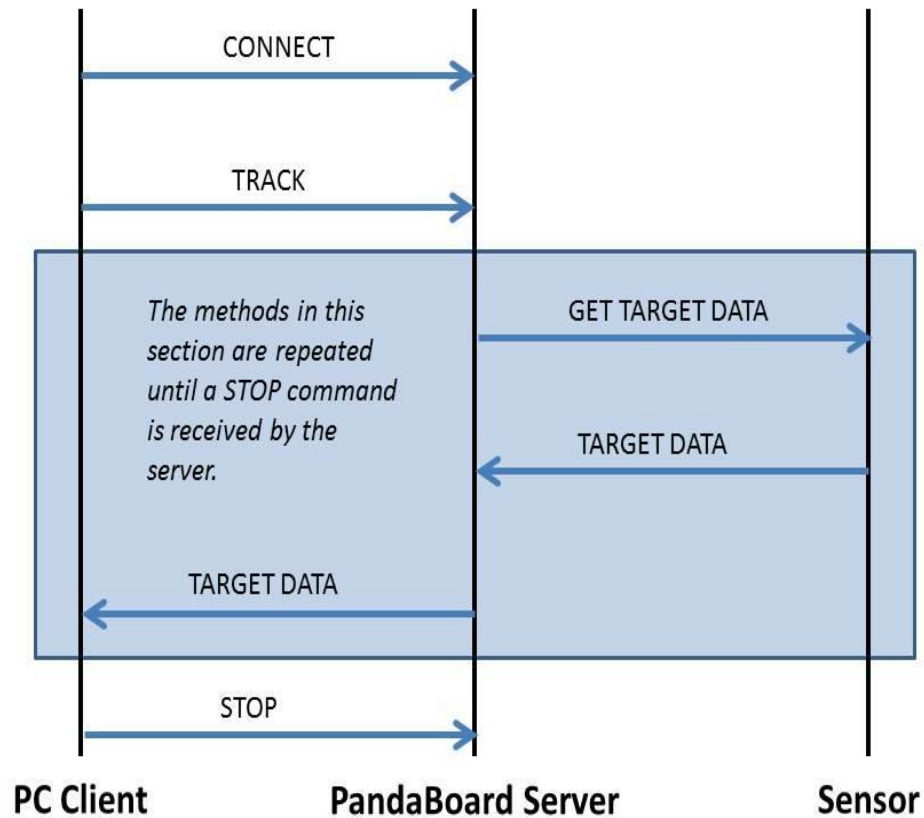


FIGURE 3.11

The electromagnetic simulation software is a series of Matlab scripts and functions that are provided by the AFRL. At the current time this is a work in progress for the AFRL and is not complete. The code is designed to read in a text file that is generated by our prototype and process the numbers it finds. The text file contains parameters for “human parameters” or body part sizes. Each body part will have its own Matlab script file that will perform calculations that will then get input into a back-projection imaging script. This will then simulate a radar system. The planned outline is described in Figure 3.12 by a simple flowchart.

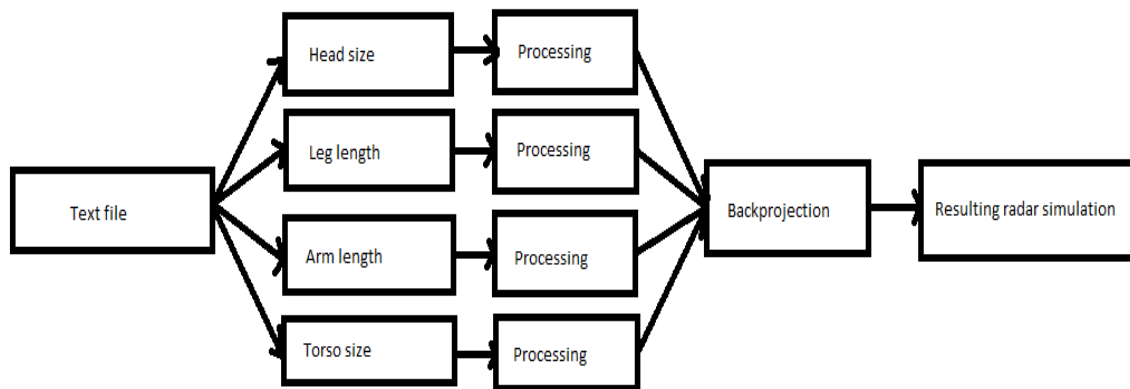


FIGURE 3.12

The software developed for this prototype is new in several ways. In general, the ASUS Xtion and Xbox Kinect are meant to be used directly by the computer connected to the device. This software not only allows the wireless transmission of the data captured by the device, but also allows simultaneous use of this data by many devices at the same time. Not a lot of work has been done with connecting these sensors to microcontroller boards or boards with ARM processors, which while a newer concept, also lead to the many difficulties encountered in the development and implementation of this project. This project has been designed to be extendable beyond the AFRL's initial requirements. Not only are user's parameters sent by the PandaBoard server, but the user's center of mass, as well as the position of each joint is distributed over the network.

Chapter 4 – Testing

Testing Overview

Testing our project went through several different phases. As a team we choose to divide the project into three different main tasks at the beginning of our project and we formed teams of two to three people to work on these tasks. This was successful because it allowed us to isolate problems to one of the three systems. There can be small problems with this approach, however. A problem that can arise is that pieces may not connect as seamlessly as planned, which may increase the time it takes to get a task fully working. The goal of our testing was to validate each of the three subsystems, as well as individual components of these systems. Validation was verified by setting several milestones for each system so the progress could be tracked. The milestones for each system are listed in Figure 4.1 below.

Milestones		
Hardware Setup	PandaBoard Server	PC Client
Setup Linux	Setup Linux	Connect to server
Connect sensor	Run TCP server	Send and receive data
Run off battery	Connect to sensor	Plot data
Enclose PandaBoard	Send and receive data	Launch AFRL software

FIGURE 4.1

Testing the Xbox Kinect

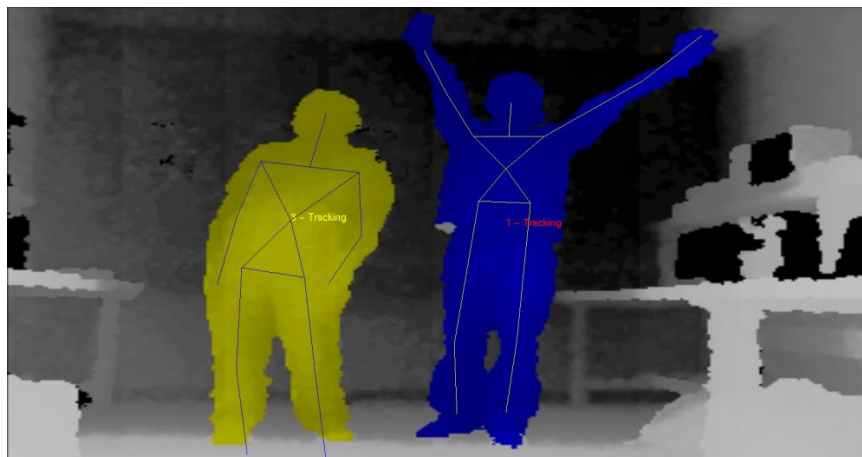
We began the testing by running tests on the Kinect. Some of the advantages we knew the Kinect had was the free, stable open source APIs that PrimeSense developed called OpenNI. We decided to first download some open source projects to our Microsoft Windows 7 PCs that performed tasks that could be useful for our future tasks on the PandaBoard. After we downloaded the OpenNI APIs to our computers we started to test the open sources projects provided by the API. When we tried to compile these programs we discovered that we were missing some libraries that were not included by default in the OpenNI APIs provided. After these were installed we successfully ran a skeleton tracing program. The code that was written for this did not help us in completing our project much, but did prove that the Kinect could do what we wanted it to do.

Our next test was to see if we could get the Kinect and PandaBoard to interact with each other. The goal of this test was for the PandaBoard to run a simple “Hello, world!” program that was supposed to be able to read data sent by the Kinect on the PandaBoard. This proved to be extremely difficult. It required many tests with different versions of Ubuntu and different libraries and APIs that were supposed to work with the Kinect. When these tests failed we had to do more research to try to find alternatives. At first our research was to find older versions of the OpenNI API and NITE middleware. Then we began to try these different versions with different versions of Ubuntu. Everything we tried failed and we could not get the Xbox 360 Kinect to interact with our PandaBoard in the way we needed. This led to research about Xbox 360 alternatives. We found a very similar device called the ASUS Xtion. The Xtion we discovered used most of the same technologies as the Kinect, but it was more open than the Kinect. This research eventually led to somebody who actually created his own personal project using the Xtion and a PandaBoard. We then returned the Kinect and bought an Xtion.

Testing the ASUS Xtion

When we received our Xtion we had to work a little harder for a few reasons. One reason that we had fallen behind was because we were waiting for it to arrive. A second reason is that we still wanted to do small test driven development. The good part about this, however, is that we gained a lot of knowledge while we were troubleshooting our Kinect and because of this our first tests ran without much of a hitch. The Xtion was compatible with an ARM processor, and all of the libraries needed to be installed to make use of it had already been found, as the ones that failed with the Xbox Kinect worked perfectly with the Xtion.

This led to our first major test whether we were going to be successful with this project. We began to see if the Xtion could identify a human being. This was something we were not sure of because the Xtion is a much newer device than the Kinect and has not been tested by as many people. We ran the same tests with Xtion as the Kinect on a Microsoft Windows 7 PC and thankfully it worked.



This meant that as long as the Xtion could interact with the APIs and libraries provided by OpenNI we would be able to similarly recognize humans when the Xtion was connected to the PandaBoard.

Testing the PandaBoard Server Program

One of the most important parts of the project is getting a PC to reliably connect to and receive and send data from the PandaBoard program. This was the first part of the code that needed to be developed. It was able to be started without any hardware, as it was just code for Linux, and any code developed on one Linux machine could, usually without much issue, be ported to another; therefore, this was initially written for a laptop computer. To test this code both a TCP server program was created, to be run eventually on the PandaBoard, as well as an elementary TCP client test program. The client created was used solely to test the TCP server program, and so this part of the project could be developed independently of the final PC client software. These initial tests gave us a foundation to begin coding the Xtion tracking code into. With the test client we were able to verify that both the TCP server could be connected to and that it could receive and decode commands sent from the test client. This also aided in the initial development of the application protocol.

Testing the Xtion with a PandaBoard

Our first tests with the PandaBoard and the Xtion were almost successful for a very early test of our code. It would recognize a human; however it was extremely prone to mistakes and tended to send more garbage numbers than good numbers. The team anticipated that this test would not necessarily go as smoothly. The main reason was because although PrimeSense had developed software that would compile on the ARM processor it was not well supported or recently updated. The first thing the team tried to resolve was to make the Xtion more reliable at detecting humans. It was determined that some values calculated by the PrimeSense's API were the problem, and this problem could not be easily resolved. The API was found to be excellent at finding humans, and tracking their relative location, but the skeleton tracking was very unreliable.

Regardless of the data received from the API not being perfect, we continued working on the project as if the data was accurate. In our code the way parameters are computed is from joint locations. For instance, if we wanted to get the length of a subject's arm we had the hand joint, elbow joint and shoulder joint. The joint locations are XYZ coordinates that used the metric system. We could then compute the length of these joints finding the differences between theses joint location's XYZ coordinates. When we ran this we found that lengths and radii we found were fluctuating quite a bit and never really stabilized. Once again our first solution was to look through the API to see if there was a function already written or one that could be slightly altered to give us better results. A function that seemed perfect is available

that assigns a confidence rating for every joint position that it calculates. When we tested the code to only calculate the joint size when an extremely high confidence rating was assigned the data improved slightly, but not as much as we had hoped. The system continues to send some bad parameters, and the only way to fix this problem is to wait for an update from PrimeSense.

On many tests, either at the very beginning of running our PandaBoard application's sensing routine or sometimes after running for a while correctly, the application would have a segmentation fault and the program would terminate. The segmentation fault we were getting is most likely caused either by something in our program accessing memory that it isn't allowed to, or something that does not exist. What we quickly realized was that the segmentation faults were not happening in any code that we had written. This could be determined since some algorithms were written to generate data without using the sensing device, and the program ran without issue when using the generated data. It was likely happening in the NITE middleware binaries that are an integral part of tracking a skeleton. This makes the problem unsolvable by us because our options are to either write our own middleware, which would be extremely time consuming and require years more of knowledge that our team does not have or to wait for PrimeSense to develop a better version of the NITE middleware for an Arm processor, which is unlikely as it seems that PrimeSense does not have much support for Arm systems.

We have been able to limit the amount of segmentation faults in our most recent version. Our final test ran for more than ten minutes each time we tried it and did not have a segmentation fault. Although at this time we still believe they can happen, in recent test they have not. We solved this problem by doing some more research regarding skeleton detection code using OpenNI and NITE. Several different example programs were found, and it was noticed that a few lines of code that were copied from the first sample program might have been the issue. These lines were modified to match a different example, appearing to resolve most of our stability issues.

Testing the PC application

The PC application part of project is a fairly simple application. It uses a TCP protocol that connects the PandaBoard to multiple PC clients. The PandaBoard is a Wi-Fi hotspot so when a PC connects to the PandaBoard the PC can use the IP address and a port number that we have set gather information from it. The PC application has several features. The first thing the application does is tell the Xtion when to start and stop tracking. After the application connects to the PandaBoard it will have several options including a start tracking option, a stop tracking option and a disconnect option. After the tracking command is sent, the Xtion finds a human and starts sending parameters. When the PC Client receives a new human target a window with options including save parameters and run Matlab for the AFRL will appear. The

first thing we worked on was the track and stop tracking commands. This was important and was developed using basic software development techniques. The main way this was tested was by making it independent from the PandaBoard program. This was done by generating fake data in order to make sure that this data was being displayed correctly, and was not reliant on the other parts of the system. After this was proven successful it was then made to import data from the TCP connection. We then needed to create a button that saved the parameters to a file and a button that would call the run the Matlab code provided to us. The Matlab code was written based off of internet searches that provided ways to make command line calls using C# code. This was also developed quickly without many problems.

Testing the Battery

Before connecting the battery system to the PandaBoard, we hooked it up to an oscilloscope to make verify that with was producing correct voltage levels. The battery was observed to produce a clean, DC output of around 5.3V. While this is a little higher than what the PandaBoard specification requests, it does not appear to affect operation and is certainly not high enough to threaten damaging the board. All switches were observed to operate correctly and without voltage spikes. While running off of the battery, the system has a life of between 5 and 10 hours. We are very pleased with this value, as we were expecting the life to be closer to 3 hours. The battery can be fully charged with a mini-USB wall adapter overnight.

Testing Conclusion

Our prototype consists of two main parts. The first part is our radar system. The radar system is enclosed by a case. On the top of our case is our camera, the ASUS Xtion, which is mounted by screws so it is secure, but also easy to take off. Inside the case is the PandaBoard which is running Linux and is reading in the data from the Xtion. The Xtion is powered by the USB in the PandaBoard. Also inside the case is a battery, which can power the PandaBoard. There are also ports on the case to power the PandaBoard from the wall and switches to turn the PandaBoard on or off and to choose to run the PandaBoard off of the battery or the wall outlet. The second part of our project is the PC application. The PC application communicates with the PandaBoard and displays the results sent to the PandaBoard. It also has the capability to save the parameters requested by the AFRL and to run the Matlab code with those parameters. Our final prototype was mostly successful. The reason we hesitate to call it totally successful is because it is also a work in progress for the AFRL who is working on the simulation software. The other minor problem we were unable to solve was a segmentation fault that was caused by PrimeSense's NITE middleware binaries, as well as data that is not always entirely accurate.

Chapter 5 – Conclusions

The final prototype meets the general needs of the sponsor. The PandaBoard is successful in both detecting people, and transmitting found data to a connected client PC. As required, the PC client program can then launch and input parameters into a separate program created by the AFRL. All the requirements listed in Chapter 2 regarding the design specifications have been at least partially, if not completely met. The end result is completely portable, and is completely wireless. No cables need to be run to the box. The PandaBoard is able to read data from the ASUS Xtion on command and takes advantage of open source APIs to simplify the programming required to complete the required tasks.

There are several aspects of our project that need to be improved in order for the prototype to be completely reliable and functional. Most of these aspects are, however, are beyond our control, and are due to the limitations of the current software available that needed to be utilized. The first problem is with Linux for the PandaBoard. Linux for the PandaBoard and low-powered ARM system boards is still in development and is not as stable as a version of Linux for the standard desktop. This means that sometimes there are errors on startup, especially with regards to auto-starting the Wi-Fi access point, and often the system will fail to shut down or reboot gracefully. This issue is very minor though, and more of an inconvenience than a real problem.

A more serious problem exists in the sensor detection APIs: OpenNI and NITE. There is no up-to-date version of these libraries for an ARM processor. The NITE version we are currently using frequently fails to report the correct coordinates of a target's skeleton and joints. This means that the sizing data passed to the PC program can be inaccurate and highly variable. The only consistently correct value picked up by the sensor appears to be the center of mass location. It can be concluded that this is not due to the sensor, as this data is accurate when the sensor is connected to a Windows PC. It is quite possible that the situation regarding inaccurate data can be auto-corrected with an updated API, which is likely to be released at some point in the future since NITE is still in development for ARM processors.

Our schedule proved to be successful overall. The initial schedule the team planned to follow worked well early on. The PandaBoard had Linux installed early on even with some install problems. We then ran into problems when our Xbox 360 Kinect camera failed to be compatible with our PandaBoard. During our troubleshooting phase the team fell slightly behind and had to wait for a new camera, the ASUS Xtion, to be delivered. The team decided it would be best to develop an updated Gantt chart with a revised set of plans. The revised Gantt chart was successful and the team even got slightly ahead of our planned schedule.

The team was able to stay within our original budget allocation of \$500. A majority of the budget went towards purchasing our prototype's two major components, the PandaBoard and the Xtion sensor. Luckily, we were able to return the Kinect for a refund once we found out that it would not work in our system. If we had not been able to return the Kinect, we would not have had enough of a cushion to purchase the parts related to the battery. If the enclosure and battery were ignored, a functioning prototype could be built with only the major components for around \$350. Our final project cost was a total of \$455.51 for all parts used to both build and test the prototype.

Appendix I – Technical Roles



Will Juszczyk

The major technical issues I dealt with were on the PC application side. One of the most important parts of our project was to be able to communicate with AFRL software. Without being able to communicate with the software we would not be able to simulate the electromagnetic radar system.

The first step in doing this was decide what the packet will look like that would needed to be parsed. In order to do this I had to talk to Ryan who was working with the PandaBoard. He decided that one packet would be 260 bytes. Each packet would look like a string of floating point numbers. Each number represented a different “human parameter” or body part size. From there I output these numbers to a text file. This text file was then to be read in by a Matlab script that was provided to us by the AFRL. In order to do this the software sent to us by the AFRL had to be modified. This was, however, a simple activity.

The PC application attempted to make writing the parameters to a text file and running the Matlab code as seamless as possible. This required creating a window for each human identified by the camera. In this window we displayed all the current parameters. This is important because on occasion numbers may not be as predicted and you will not want to write those to file. The other button will automatically run the Matlab code. In order to accomplish this, research had to be done to make command line calls from C# code within the .NET framework.

As a team we all worked together to help others out in areas where we may be needed. The big part of our project was working with identifying humans and calculating their parameters. When troubles arose working on this I would help debug.



Ryan Lattrel

I was the main architect of both the PandaBoard software and the PC client. I also set up and configured the PandaBoard. The PandaBoard was the first part of the project I worked on. This involved finding the correct version of Linux to install, as well as locating the correct OpenNI, NITE, and Sensor libraries and drivers needed for the ASUS Xtion device to be able to detect target skeletons. I also setup the PandaBoard as a Wi-Fi access point in order to allow a PC to directly connect to the board and access it with a predetermined IP address, with both SSH and the PC software.

I designed the PandaBoard software from scratch. The software utilized UNIX sockets to create a TCP server. The server was designed to accept connections from any number of clients. When a client connected to the server, a new thread was spawned, this allowed each client to send commands to the server while it awaited even more connections. The server was also able to send data to each connected client, and was made thread safe allowing clients to abruptly connect and disconnect with no detected issues. I also set up the initial reading of the ASUS Xtion by utilizing demo skeleton detection programs that were included with the libraries. These methods were embedded into the software to be called whenever a client computer send a specific command. I created an application layer protocol to allow the ability to send multiple commands seamlessly between the PC client application and the PandaBoard server application. This protocol also accounted for the data that needed to be sent from the server to the client. The protocol allows for expansion and the sending of more data than is required by this specific prototype.

The PC client was also designed by me. I created the GUI for the both the main radar window and the tracking window. I also wrote the network code required to be able to connect to, send data to, and receive data from the PandaBoard server. I wrote the client to allow plug-ins by other team members to utilize the data received from the server and to integrate the client with the AFRL software.



Camden Smith

As Lab Coordinator for our team, my responsibility was to order the required parts from vendors as necessary. One of my first major technical contributions was researching the hardware needed to build our project. I studied the hardware and software requirements on multiple websites detailing the Microsoft Xbox Kinect. This was to be sure we were ordering the most powerful Microcontroller to support our Motion Sensing Input Device (MSID). After studying and comparing more than five microcontrollers, the team and I made a final decision on the PandaBoard. I began the research of the software and libraries required to program the Kinect and PandaBoard combination. I discovered multiple tutorials that we used to (initially) program and test the Kinect and PandaBoard (specifically with the board installed with Ubuntu 12.04). During my research, I found a detailed tutorial explaining the steps to successfully program the board with another MSID called a Xtion made by ASUS. So when the team and I made the switch to the Xtion, I had already documented several tutorials on the installation of the Xtion and PandaBoard, which made the switch less painful.

As the project started coming together with the basic hardware (the battery, PandaBoard and Xtion), I began making drawings of different design ideas for an enclosure. Some of my first designs were good but too complicated and out of reach due to time constraints. I created a simplified design that was more in reach with our timeline and budget. We decided on a simple polyurethane box with dimensions 8.25" x 5.0" x 3.0". The calculations of the internal dimensions of the box were just right to fit the board and battery. The length of the box was a little more than the length of the Xtion, so we decided it would be a proportional fit. When we received enclosure, I installed the board and battery to see how they fit in the box together. When installing the board it was important we could still reach the button on the battery. I first cut a hole in the box for the USB Mini port and the Serial port just above the height of the battery so the ports on the board could slide into the holes in the box and the board could rest above the battery. Then I measured the dimensions of each component switch and drilled a hole for the power switch, battery/AC control switch, and port for the AC adapter. After that Mike was going to solder the switches to the board and then connect them to the battery for power.



Michael Lazar

As the only pure Electrical Engineering major in the group, it was decided from the start that my main contributions to the project would be on the hardware side. I gladly embraced this role, and it gave me the opportunity to research several topics including infrared sensors, microcontrollers, voltage regulators, battery specifications, battery chargers, and more. I suggested that we try to implement a battery into our system, and because the PandaBoard and Sensor were non-negotiable, I took responsibility for allocating the rest of the team's budget. I compiled an estimate of our system's power consumption, and weighed different options. Because the team needed to switch sensors midway through the project, I ended up designing two completely different power systems. The first one utilized buck converters and a mini UPS in order to supply both 12 and 5 volts. The second one consisted of a pair of switches and a preassembled battery pack. I selected and ordered all of our projects components to be within spec. Upon arrival, I built the system and tested all outputs manually to ensure that a stable, regulated voltage was correctly being output. Finally, I took great care in soldering all switches and ports to the enclosure, following the circuit that I had designed. The final system was observed over a period of time and determined to provide stable operation for the PandaBoard.

Another one of my technical roles included building an infrared camera in order to observe the Xtion and act as a nice supplemental visual on design day. I selected and purchased a webcam, modified it by replacing the infrared filter with a visual light filter, and set it up on my laptop. The video from the webcam works very well in low-light and is an insightful demonstration of how the sensor operates.

One of my smaller contributions included getting the sensors working with Window's drivers on a laptop. This gave the team a chance to interact with the PrimeSense and NITE SDK and demos far in advance of when we finally got them working with the PandaBoard. As a result, we were able to test the sensor capabilities and confirm that skeleton tracking would be adequate for our project needs.

Appendix II – References

"Anker Astro2 8400mAh Dual USB Output 5V 1A / 2A External Battery With Flashlight."

ANKER, <<http://www.ianker.com/>>.

"ASUS Xtion Pro." ASUS, <http://www.asus.com/Multimedia/Motion_Sensor/Xtion_PRO/>.

"Create A WiFi Router." *Sawo Design*. <<http://sawodesign.blogspot.com/2012/01/create-wifi-router.html>>.

"Haptic Belt for the Blind." *Kinecthesia*. <<http://www.kinecthesia.com/>>.

"Introducing Kinect for Xbox 360." *Kinect*. Microsoft, <<http://www.xbox.com/en-US/KINECT>>.

" OmapDesktopInstall ." Ubuntu Wiki. Canonical Ltd.,
<<https://wiki.ubuntu.com/ARM/OmapDesktopInstall>>.

"OpenNI." OpenNI, <<http://www.openni.org/>>.

"PandaBoard." PandaBoard, <<http://pandaboard.org>>.

"TreasureHuntingRobot." Ed. Hirotaka Niisato. SIProp,
<<http://www.siprop.org/en/2.0/index.php?product%2FTreasureHuntingRobot>>.

Appendix III – Technical Attachments

Figure A3.1

Kinect Description

Microsoft's Kinect is an impressive human detection and motion sensor developed for the gaming industry in conjunction with Microsoft's Xbox 360. It operates by projecting a pseudo-random pattern of infrared light onto a room, and observing the structure of this light with a corresponding infrared camera. This method is advantageous over time-of-flight cameras, which require sensitive instrumentation and can cost thousands of dollars. Because the Kinect uses infrared light, it works exceptionally well in low-light areas and is able to distinguish objects regardless of color and texture. However, it does not work well in outdoor setting as natural light contains infrared wavelengths. The structured light approach also inhibits the use of more than one sensor in a room unless special measures are taken to prevent interference. The Kinect is capable of outputting a three-dimensional depth map with resolution of 320x240 at 30 frames per second. It has an effective field of view of around 60 degrees, and a range of between 0.6 meters and 4-5 meters. The unit also contains 4 microphones which are used for voice recognition, and motor in the base that can be used to adjust the yaw of the camera. The original Kinect required both a USB and a 12V regulated power input provided from an AC/DC adapter. The newer Kinect model contains only a single modified-USB port designed to connect to Xbox 360 systems. Enthusiasts who wish to develop applications for PC generally use the older model, and as of this time it is still easily obtainable from the Microsoft store. The Kinect has substantial power requirements, and the official adapter rating is 12V at 1.1A max. The Kinect contains a peltier chip in order to self-regulate its infrared laser's temperature, and this can draw quite a bit of power while in use. During normal use, the Kinect will draw around 0.35A. The Kinect can be purchased bundled with an Xbox 360, or as a standalone unit for \$110 USD.

Figure A3.2

Xtion Pro Live Description

The Asus Xtion Pro Live is a PC-exclusive motion sensor aimed primarily towards developers. The unit can be thought of as a stripped down version of the Kinect. The unit itself is smaller and lighter than the Kinect. The hardware inside of the Xtion operates with the same design principles. Its laser is not quite as powerful and as a result the range of the device is slightly less. Its base does not contain a motor and is instead manually adjustable. The only input on the Xtion is a standard USB port. The system draws all power from the USB port, up to a maximum of 500mA at 5V. This makes it simple to implement an Xtion sensor into a variety of systems. All PrimeSense software, including NITE middleware which provides skeleton tracking, is compatible with the Xtion Pro Live. The device is currently available as a standalone unit for \$160 USD.

Figure A3.3

Fast Diagram

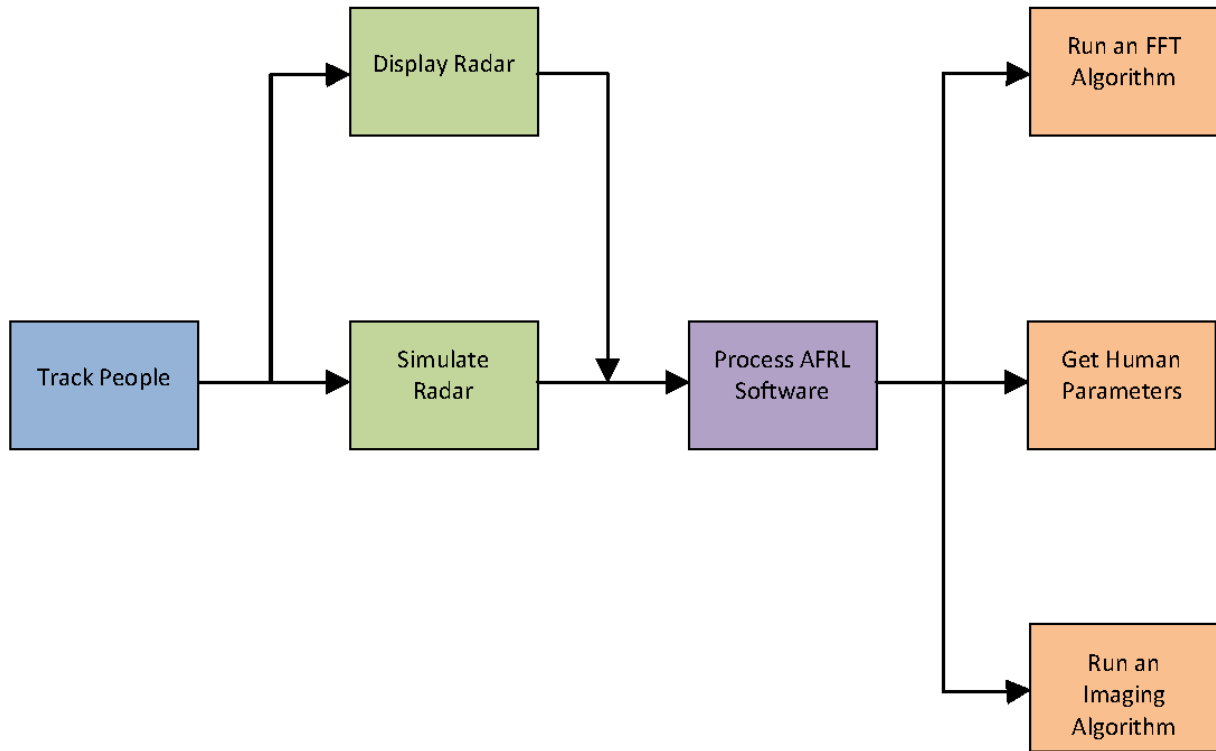


Figure A3.4-1

INITIAL GANTT CHART

Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names
	Order Items	10 days	Mon 9/17/12	Fri 9/28/12		
	Get Pandaboard set up with Linux	5 days	Mon 10/1/12	Fri 10/5/12	2	
	Start working with Kinect and Pandaboard	5 days	Mon 10/8/12	Fri 10/12/12		2-3 team members
	Set up Wifi module on Pandaboard	5 days	Mon 10/1/12	Fri 10/5/12	2	2-3 team members
	Testing and catch-up week	5 days	Mon 10/15/12	Fri 10/19/12		All team members
	We should have something presentable, maybe a bit buggy	0 days	Fri 10/26/12	Fri 10/26/12		
	Testing (project will hopefully start working)	10 days	Mon 10/22/12	Fri 11/2/12		All team members
	PC application (communicating with Pandaboard)	5 days	Mon 10/22/12	Fri 10/26/12		2-3 team members
	Test parameter with AFRL software	5 days	Mon 10/29/12	Fri 11/2/12	9	2-3 team members
	Project should work. Pandaboard communicating with PC and sending parameters	0 days	Mon 10/29/12	Mon 10/29/12	9	All team members
	Get a power supply working	5 days	Mon 10/1/12	Fri 10/5/12	2	2-3 team members
	Design an enclosure	5 days	Mon 11/5/12	Fri 11/9/12		2-3 team members
	Late round testing (hopefully it mostly works by now)	7 days	Mon 11/12/12	Tue 11/20/12		All team members
	Get the enclosure done and place components permanently inside	5 days	Mon 11/12/12	Fri 11/16/12	13	2-3 team members
	Final Testing	5 days	Mon 12/3/12	Fri 12/7/12		All team members

Figure A3.4-2

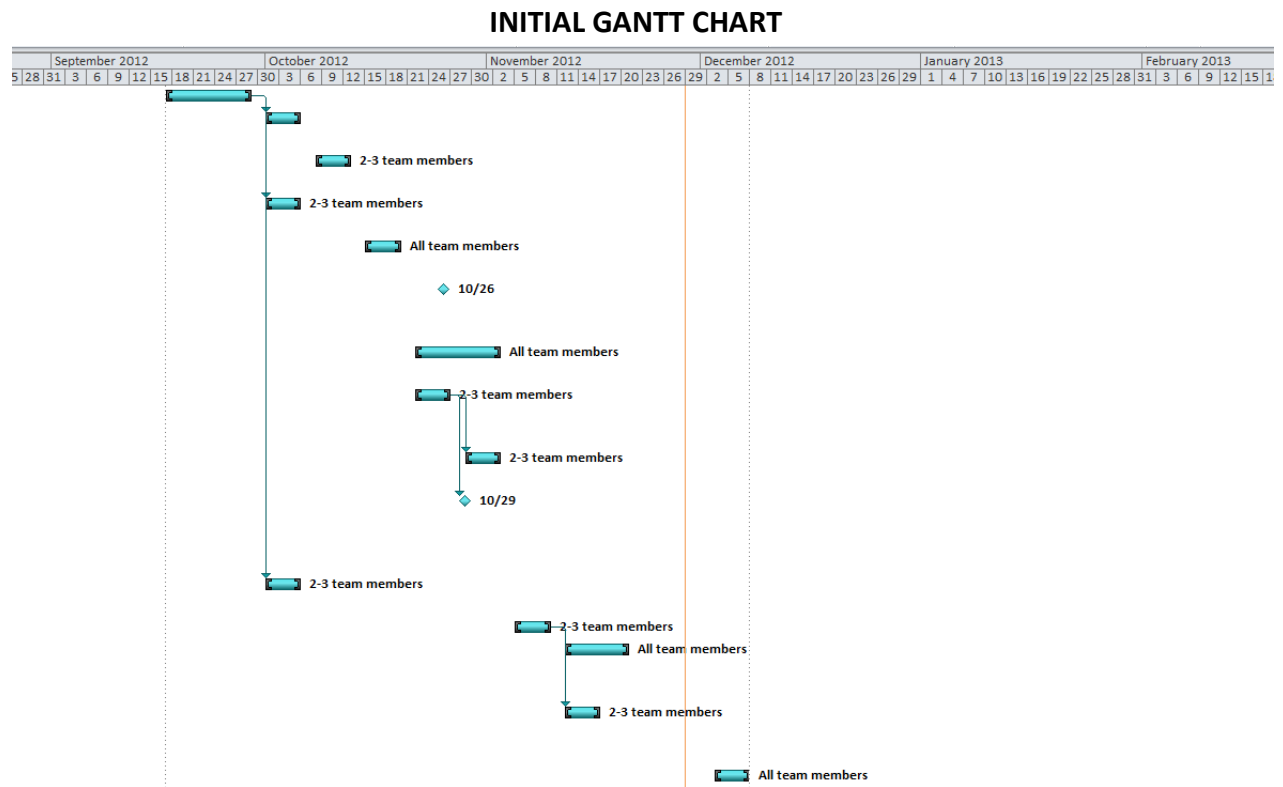


Figure A3.5

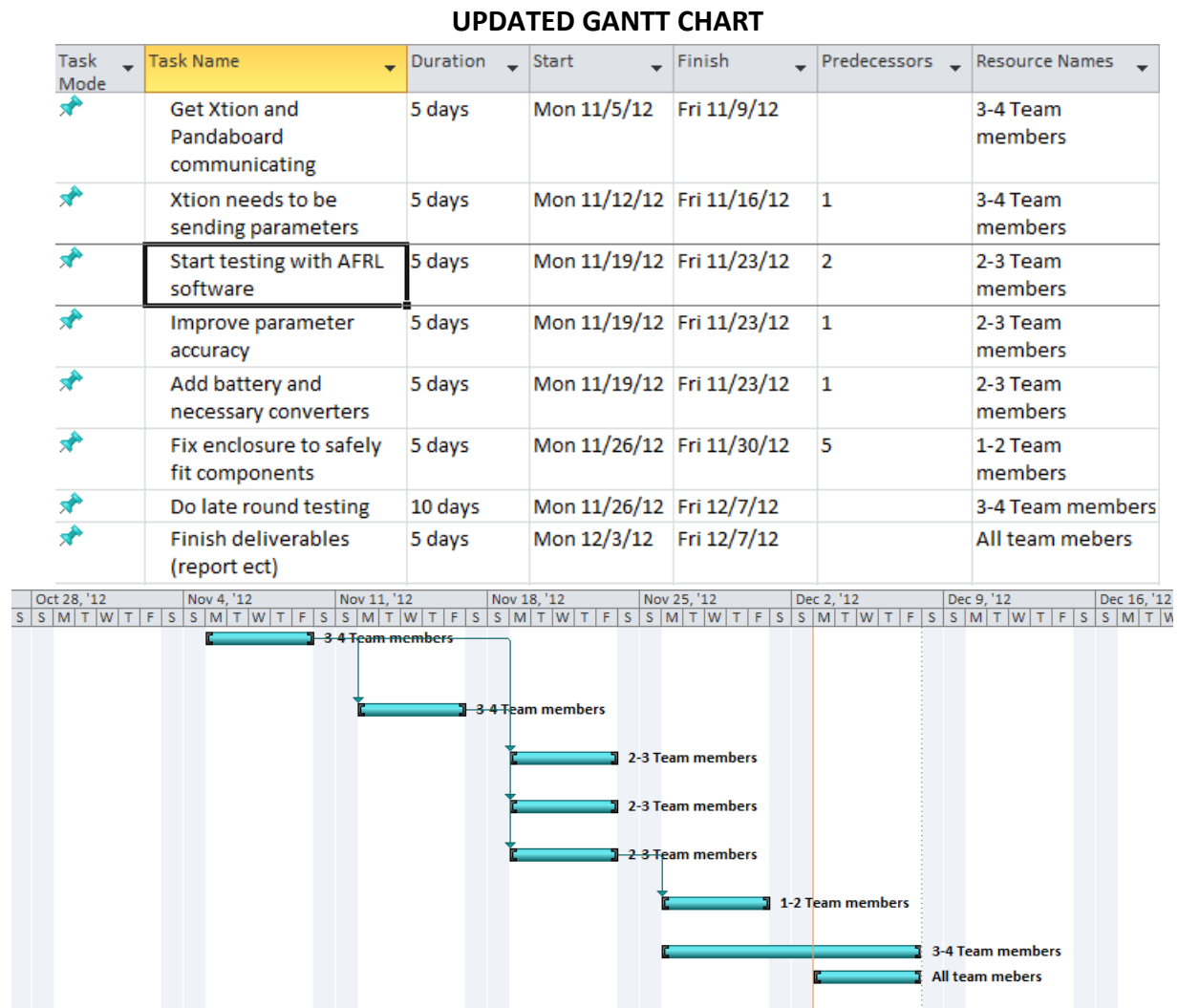


Figure A3.6

Solution Selection Matrix

Criteria	Importance	Design Concepts		
		Arduino	BeagleBoard	PandaBoard
Portable	5	5	4	4
Ethernet	5	3	5	5
USB	5	3	5	5
Runs Linux	4	0	5	5
		55	90	90

Figure A3.7

Feasible Design Matrix

Criteria	Importance	Design Concepts	
		BeagleBoard	PandaBoard
Portable	5	4	4
Lightweight	4	4	4
Wifi	2	0	5
Ethernet	5	5	5
USB	5	5	5
Runs Linux	4	5	5
Powerful	4	3	5
Power usage	1	5	3
Cost Effective	3	4	3
		135	148

Figure A3.8

Final Project Budget

Item	Supplier	Base Price (USD)	Shipping (USD)	Final Price (USD)
PandaBoard ES	DigiKey	\$161.64	\$7.40	\$169.04
Asus Xtion Pro Live	Newegg	\$159.99	-	\$159.99
Anker Battery	Amazon	\$39.99	-	\$39.99
SD Card	Amazon	\$17.72	-	\$17.72
Enclosure	PolyCase	\$11.93	\$9.52	\$21.45
Webcam	Amazon	\$19.99	-	\$19.99
USB-DC Cable (2)	DigiKey	\$9.10	\$1.68	\$10.78
Switch - Rocker	DigiKey	\$2.68	\$1.68	\$3.36
Switch - Toggle	DigiKey	\$2.11	\$1.68	\$3.79
Power Jack	DigiKey	\$1.18	\$1.68	\$2.86
AC/DC adapter	-	\$0.00	-	\$0.00
Kinect*	Microsoft Store	\$6.54	-	\$6.54
Total				\$455.51

**The Kinect was ordered, and then later returned. The refunded value did not include initial sales tax.*

Part Description and Justification

- PandaBoard ES - The PandaBoard is a low cost, open mobile software development platform. The new ES generation adds a faster processor and Bluetooth capability for a modest price increase. The board supports Linux-based operating systems via its SD expansion slot, and features a dual-core ARM A9 processor. The PandaBoard is the largest part of our designed system and provides us with all of the tools we need including USB, wireless connectivity, processing power and debugging capabilities.
- Asus Xtion Pro Live - The Xtion sensor detects and tracks human using the next-generation infrared technology that is also found in the Kinect. Xtion software is compatible with ARM-based systems, which makes it ideal for our design solution.
- Anker Battery - The Anker Astro2 8400mAh battery provides dual-USB 2A outputs intended for charging portable devices. It features a built in Lithium battery with charging circuitry and a regulated 5V output. The Astro2's regulated output and compact design makes it an ideal and affordable solution for providing an alternative source of power to our system.
- SD Card - The SanDisk Extreme Pro 16 GB memory card features a 45MB/s read and write speed. The memory card is required to boot a Linux based OS onto the PandaBoard.
- Enclosure - The PolyCase enclosure is made from black, Flame Retardant ABS plastic and measures 8.25" x 5.00" x 3.00". The package includes a base, cover, and 6 enclosure screws. Its size allows space for our battery and microcontroller.
- Webcam - The Logitech C300 webcam is a basic function webcam with a manual focus and up to a resolution of 1280x1024. With a simple modification it can be turned into a infrared camera. This webcam is not a part of the project's final design, but is useful in testing the sensor's functionality.
- Cables and switches - All cables and switches were selected to contain appropriate voltage and current ratings. These are used to connect the various components in our design.
- AC/DC adapter - A 5V AC/DC adapter is needed to connect the PandaBoard to a wall outlet. An adapter with at least a 2.5A rating is recommended. Our team obtained an adapter free-of-charge from the ECE lab.

Figure A3.9

PROTOCOL DEFINITION

Client to Server Commands	Server to Client Commands
TRACK message: This message is sent from the client to tell the server to start its tracking routine. STOP message: Sending this message will tell the server to end the tracking routine. QUIT message: This message will tell the server to disconnect the client. SHUTDOWN message: Sent by the client to turn-off the server box. This command only works in the server program is running in the root user space.	TRACK CONFIRM message: Sent after a track command has been successfully received. CONNECT OK message: Sent after a client has connected successfully. STOP CONFIRM message: Sent after a stop command has been successfully received. SUCCESS message: This message is sent after a user has been found by the imaging device. Attached to this message is both the parameter data and position data of the detected user. FAILURE message: Sent when a user previously detected can no longer be found. Attached is a user ID.

Figure A3.10

Data Message Structure

Message Structure	Parameter Structure	Position Structure
1. Message type 2. Data buffer 1. Target ID 2. Parameter structure 3. Position structure	1. Torso width 2. Torso height 3. Left arm width 4. Left arm length 5. Left leg width 6. Left leg length 7. Right arm width 8. Right arm length 9. Right leg width 10. Right leg length 11. Head diameter	1. Target center of mass coordinates 2. Left hand coordinates 3. Right hand coordinates 4. Left elbow coordinates 5. Right elbow coordinates 6. Left shoulder coordinates 7. Right shoulder coordinates 8. Head coordinates 9. Neck coordinates 10. Left hip coordinates 11. Right hip coordinates 12. Left Knee coordinates 13. Right knee coordinates 14. Left foot coordinates 15. Right foot coordinates

Figure A3.11

PANDABOARD CONFIGURATION

