|  |  |  |
| --- | --- | --- |
| **Report Component** | **Description** | **Pts. received** |
| 1. *Letter of Transmittal*   **5 points** | Formal letter to tell client what is included in the transmittal package. This item is composed after the report has been completed. |  |
| 1. *Title page* **2 points** | First sheet in report. Include title of project, date, client name, contract/reference number if applicable, statement of confidentiality if applicable. |  |
| 1. *Executive Summary*   **5 points** | This is a brief summary of the design report describing key features such as results, design overview, cost, schedule, etc. Should be no more than a couple of pages. |  |
| 1. *Acknowledgements*   **2 points** | Here is an opportunity to thank your sponsors, technical advisor, client advisor, and supporters who have provided funds, equipment, space, or advice. |  |
| 1. *Table of Contents*   **5 points** | The Table of Contents should include the major report sections and the page on which these sections begin. For longer reports there could also be a Table of Figures and/or List of Tables. |  |
| 1. *Introduction* **10 points** | Background information, historical background, motivation, etc. |  |
| 1. *Problem/Project Definition* **10 points** | Statement of the desired outcome or work product (is it a physical product, software product, process design, building/structure design, etc.), definition of the problem or problem statement, design constraints, related work, summary of literature review, summary of patent review. |  |
| 1. *Evaluation of Alternatives*   **6 points** | Screening level of analysis used to focus the design team on the alternative most likely to succeed. Alternatives not selected can be described in the appendices. |  |
| 1. *Design Approach*   **8 points** | Description of the components of the design process that the team employed in arriving at the final design. |  |
| 1. *Design Narrative, Design Verification/ Implementation, Performance Evaluation, Testing, etc.*   **22 points** | Detailed description of the design, design verification/implementation, performance evaluation, system capabilities, documentation. Discussion of testing and verification protocols and the results of testing, whether physical testing or simulation. Details of testing and other topics are typically placed in appendices. |  |
| 1. *Professional and Societal Concerns, Cost and Economic Evaluation*   **5 points** | Safety concerns and how they will be mitigated, potential environmental impacts of design, potential health impacts (both positive and negative), other societal impacts. Summary of the cost estimation and economic evaluation of the selected design including protocols applied to the economic analysis. Additional details may be appropriate in appendices. |  |
| 1. *Discussion*   **5 points** | Summary of the entire design process. |  |
| 1. *Conclusions and Recommendations*   **5 points** | Statement of the project team's conclusion of the best course of action based on the completed design process. Detailed recommendations regarding how best to execute the design. May include recommendations for additional work. |  |
| 1. *References/ Citations*   **5 points** | Documentation of the verified sources of information that provide background information on which the design is based. Should include the clients request for work, the project team’s proposal, and references to patent literature, standard handbooks, journal articles, textbooks, industry and government standards, and internet based sources. |  |
| 1. *Appendices*   **5 points** | The appendices provide an opportunity to present additional details that are not critical to the design focus. Items to include here (as appropriate) are Bill of Materials, details of modeling and simulations, details of testing protocols, detailed test results, certification of performance, details of economic analysis, alternatives evaluated but not selected, etc. |  |
| ***TOTAL*** |  |  |

Mr. Jeffery Ruocco; Mr. Jeffrey Falberg; Mr. Getro Jean-Bapiste  
Electrical & Computer Engineering Senior Design  
UNH: Tagliatela College of Engineering  
300 Boston Post Road  
West Haven, CT 06516

April 28, 2016

Bijan Karimi, Ph.D., Professor  
Electrical & Computer Engineering and Computer Science  
300 Boston Post Road; Office: B246  
West Haven, CT 06516  
203-932-7164

Dear Dr. Karimi,

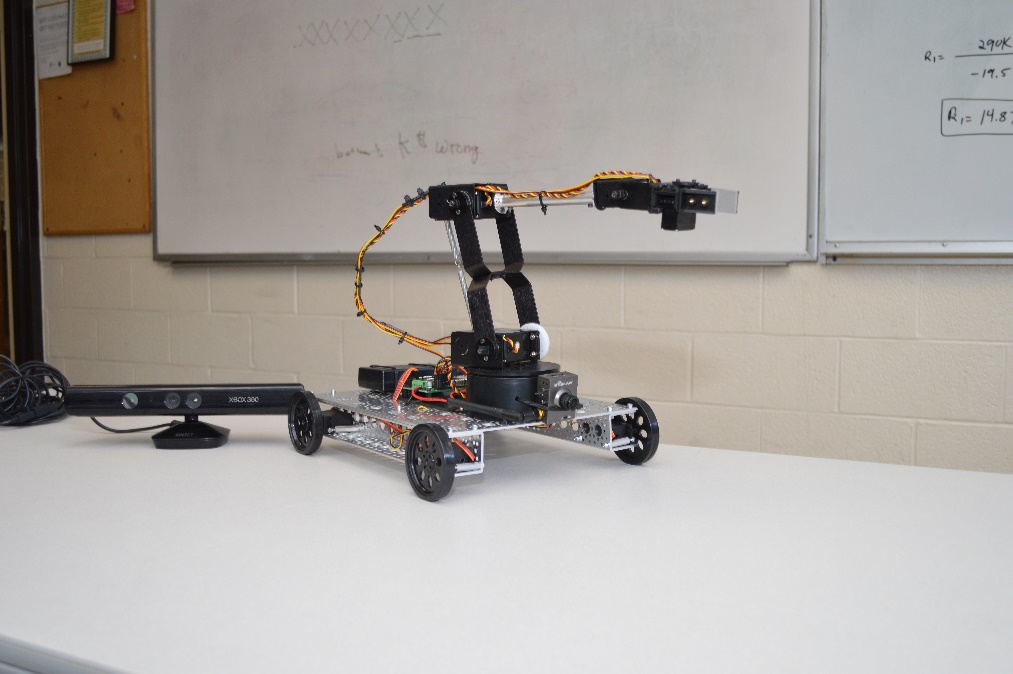
Attached is the report entitled "Mobile Motion Tracking Robot Arm". This report explains in detail the work completed at University of New Haven, by the UNH students in Electrical & Computer Engineering during the fall of 2015 and spring of 2016, for the Senior Design courses required by the school.

We would much appreciate it if you read our report carefully where our report focuses on the design and construction of the robot arm, mobile platform, and programming of the Arduino and Microsoft Kinect. This is the final submitted copy of the report. Should you have any questions or comments about the report, feel free to contact us via email any time.

Sincerely,   
Jeffery Ruocco Jeffrey Falberg Getro Jean-Bapiste

**University of New Haven** **Tagliatela College of Engineering   
Department of Electrical & Computer** **Engineering and Computer Science**

Mobile Motion Tracking Robot Arm



**Team Members:**

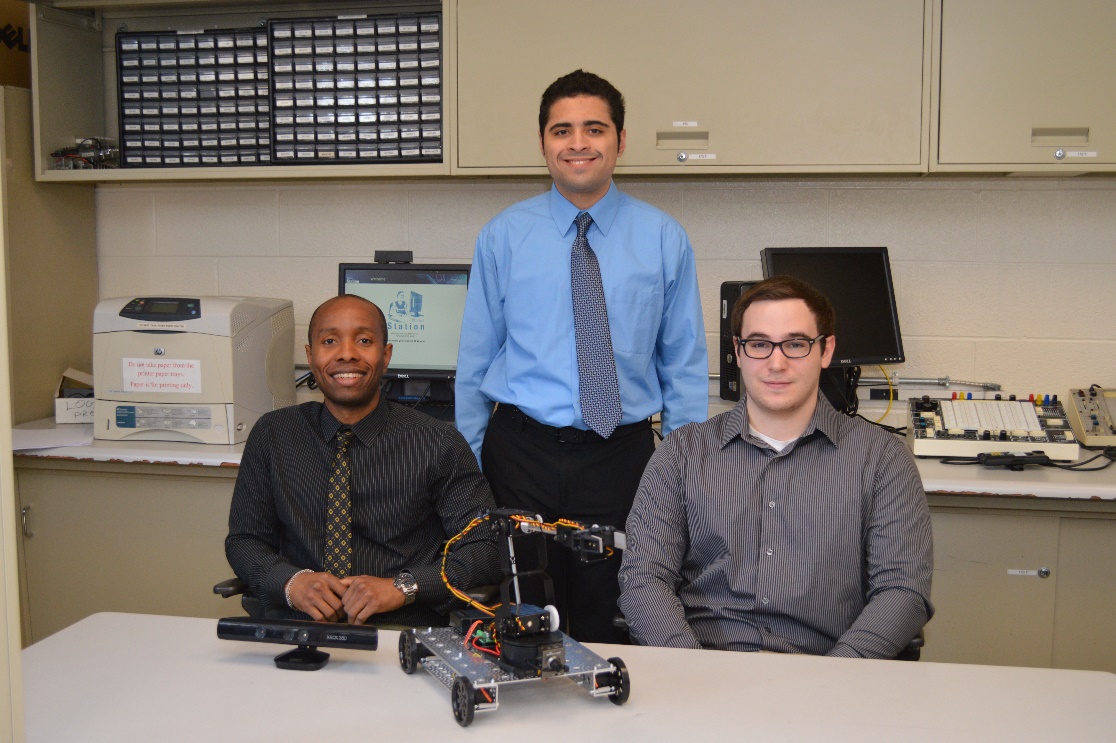
**Jeffery Ruocco, Jeffrey Falberg, Getro Jean-Baptiste**

**CMPE 4498 – Section 01**

**Professor Karimi**

**Date: April 28th, 2016**

**Group Members**



From left to right:   
**Getro Jean-Baptiste** – Responsible for constructing/designing robot arm and platform, as well as all power requirements.  
**Jeffrey Falberg** – Responsible for Arduino coding  
**Jeffery Ruocco** – Responsible for Kinect and Arduino coding

**Executive Summary**

We successfully created a mobile, motion tracking-controlled, robot arm capable of picking up small objects that are at a distance from the user. Motion tracking is handled by the Microsoft Kinect for Xbox 360. The Kinect interacts with our coordinate-mapping program written using the Kinect for Windows SDK v1.8. Coordinate information for the user’s right hand and additional commands are sent through serial to the Arduino Uno, where the coordinates are passed through inverse kinematic and linear regression calculations to be translated into servo positions. The servo positions are formed into commands and sent wirelessly over Xbee modules to the servo controller, which in turn controls the robot arm and mobile platform.

Throughout the design and implementation stages of the project, our typical schedule involved meeting on Tuesday and Thursday from 3:00PM to 6:00PM, with individual work done on weekends or before meetings. Our total project cost was approximately $650.

**Acknowledgments**

We would like to thank our technical advisor, Dr. Bijan Karimi, for advising us through the proposal phase of the fall semester, helping us through the design phase, and for ultimately approving our project.

We also thank Mark Morton for helping us obtain the required equipment and hardware for the project, and for his helpful suggestions.

Contents

[1.0 Introduction 1](#_Toc449644057)

[2.0 Project Definition 1](#_Toc449644058)

[2.1 Patent Search 1](#_Toc449644059)

[3.0 Evaluation of Alternatives 1](#_Toc449644060)

[4.0 Design Approach 2](#_Toc449644061)

[4.1 Robotic arm 2](#_Toc449644062)

[4.2 Arduino 3](#_Toc449644063)

[4.3 Xbee 3](#_Toc449644064)

[4.3 Kinect 3](#_Toc449644065)

[4.4 Platform 4](#_Toc449644066)

[4.5 Wireless Camera 4](#_Toc449644067)

[5.0 Design Narrative 5](#_Toc449644068)

[5.1 Robotic arm 5](#_Toc449644069)

[5.2 Kinect 6](#_Toc449644070)

[5.3 Arduino 7](#_Toc449644071)

[5.4 Platform 12](#_Toc449644072)

[6.0 Professional and Societal Concerns, Cost and Economic 13](#_Toc449644073)

[Evaluation 13](#_Toc449644074)

[7.0 Discussion 14](#_Toc449644075)

[8.0 Conclusion and Recommendations 15](#_Toc449644076)

[8.1 Robotic arm 15](#_Toc449644077)

[8.2 Kinect 15](#_Toc449644078)

[8.3 Arduino 16](#_Toc449644079)

[8.4 Platform 16](#_Toc449644080)

[8.5 GitHub 16](#_Toc449644081)

[10.0 References 16](#_Toc449644082)

[11.0 Appendices 17](#_Toc449644083)

# Introduction

When thinking of an idea for our senior design project, we wanted something that applied to all of our strengths. We all had an interest in robotics, and when a robot arm controlled by motion tracking was suggested, we knew that was what we wanted to build. It provided us with an opportunity to utilize the skills that we had been developing and create something that would impress anyone that passed by. We live in an age of computers and factories run by robots. The use of robots is increasing every day and we believe this project could further the development and potential use of robots.

# Project Definition

Currently, there are a vast array of various kinds of robotic arms, but most of them require a skilled operator and delicate instrumentation to control. Our idea of controlling a robot arm through motion tracking eliminates this. With our project, the user just needs to move their arm as they would if they were in place of the robot. The learning curve is small and controlling the arm is natural.

The purpose of our project was to create a mobile motion tracking controlled robot arm. The main goal was to be able to allow the user to control the arm from a distance and interact with an object. Because we had a limited budget for our project we had to design on a small scale and with lower quality components, but we wanted our project to serve as a prototype and proof of concept for future projects to expand and improve on. These future projects could be developed for the military, law enforcement, handling unsafe materials, or practically any situation that would be hazardous to humans.

## Patent Search

We were not able to find a patent on a mobile motion tracking controlled robot arm.

# Evaluation of Alternatives

Our initial idea for our project was to build a stationary robot arm and hand. The hand would have individual finger movement and be able to pick up things that were in front of it. After some discussion amongst ourselves and our advisor, Dr. Karimi, we decided our project would be more useful and more versatile if the arm was mobile. We also decided that individual movement for five fingers was not necessary for the purpose of our project.

When choosing a robot arm, we had three alternatives. The first alternative was to build the arm ourselves. This was quickly dismissed as it would require a lot of time and effort, and would detract from the goal of the project, as building a robot arm is a project in itself. Had we tried to build our own arm, we likely would not have had time to finish the project. The other alternatives were a RobotGeek Snapper Arm kit or a Lynxmotion AL5D. Both arms offer up to five degrees of freedom, but the RobotGeek arm is priced at $200 and the Lynxmotion arm is priced at $310. We decided to choose the Lynxmotion AL5D because it had a longer reach and was capable of lifting objects weighing up to 10oz, whereas the RobotGeek Snapper could only lift objects weighing up to 50 grams. The Lynxmotion AL5D is a bigger arm, and although it costs more, we decided it was worth it because the arm is the center of our project.

From the beginning of our design we had planned to use a Microsoft Kinect, however we wanted to use the Kinect 2.0. It offers higher resolution color and depth cameras, a greater field of view, a higher number of defined skeleton joints, and more accurate motion tracking. We decided to use the Kinect for Xbox 360 instead because one of the group members already owned one, meaning we could save money to spend somewhere else. We believed the Kinect for Xbox 360 would provide motion tracking that was accurate enough for our design.

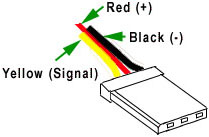
Our original plan for the mobile platform was to purchase a prebuilt platform. After looking at a few kits however, we realized that buying a prebuilt platform would be expensive and may not suit our needs. We decided that we would build our own platform so that we could have enough room for all our components to sit and save money.

# 4.0 Design Approach

## 4.1 Robotic arm (Getro)

In the process of selecting our robot arm, our main focus was budget and ease of use. We decided to use the Lynxmotion AL5D robot arm. This arm allows 5 degrees of freedom through five servos and requires 6V of power for operation. There is one servo at the base for 180 degrees rotation from left to right, one above the base for forward and backwards movement, one at the elbow for up and down rotation, one at the wrist to tilt up and down, and one to control the opening and closing of the gripper. Five degrees of freedom is more than enough for our design, as our goal was only four degrees of freedom. Also, the arm is tested to hold up to 10oz at full reach. This provides us with enough strength to fulfill our goal of being able to lift small objects easily. Full specifications for the Lynxmotion AL5D are listed in Appendix D.2.

The arm also comes with the SSC-32U servo controller (Figure 1.1). Having a servo controller makes working with the servos much easier; it allows us to send a single string command and move every servo to the desired position. The SSC-32U can control up to 32 servos at once, which are laid out in two sections of 16 pins. There are also two possible voltage inputs for the servos, VS1 and VS2, and a jumper to connect them together if you only wish to use one power source. Pins 0 to 15 correspond with VS1 and pins 16-31 correspond to VS2. In Figure 1.2 you can see the servo connector that plugs into the SSC-32U. When connected to the SSC-32U, the black wire connects to ground, the red wire connects to voltage, either VS1 or VS2 depending on the configuration of the board and the pin it is connected to, and the yellow wire provides the signal for the servo. The signal comes in the form of a pulse width modulation. A pulse of 5V is sent over the signal wire, the length of the pulse corresponds to the desired position of the servo. The servo times the pulse it receives and rotates based on the length of the pulse; the longer the pulse the bigger the degree of rotation from center. For the SSC-32U, pulses can go from 0.5ms to 2.5ms. A 1.5ms pulse corresponds to 0° rotation and acts as the center of the servos rotation, 0.5ms corresponds to -90° rotation, and 2.5ms corresponds to +90° rotation [1].

**Figure 1.1:** SSC-32U Servo Controller **Figure 1.2:** Servo connector

## 4.2 Arduino (Jeff F.)

The programming of the robotic arm's movement comes from the Adafruit METRO 328 microcontroller, which is similar to an Arduino Uno, but slightly more user-friendly. Just like an Arduino microcontroller, it is programmable with an IDE such as the Arduino IDE. The four LEDs are placed on the edge of the PCB so that they are seen easier when the METRO has a shield mounted on top of it. For easier debugging, these indicator LEDs have one green power LED, two RX/TX LEDs, and a red LED connected to pin PF5. Adafruit designed this microcontroller specifically to run the Atmega328 brain, which has 32 KB of Flash and 2 KB of RAM, running at 16 MHz and preloaded with the Optiboot bootloader. The METRO has an FTDI USB-to-Serial converter for the purpose of sending and receiving data to a computer. The logic level is at 5-V, but could convert to 3.3-V logic if required [2]. Essentially, this is an Arduino Uno, and it will be referred to as an Arduino throughout this report.



**Figure 2.1:** Adafruit METRO 328

## 4.3 Xbee (Jeff F.)

The Digi International XBee 802.15.4 module manages the wireless data communication from the computer to the robotic arm. It operates on a 2.4GHz frequency to transmit data to a receiver. To connect the Xbee transmitter to the Arduino, it must be attached to an Xbee adapter, and then connected to an Arduino shield stacked on top of the Arduino Uno microcontroller. The receiver mounts on the SSC-32U microcontroller to communicate with the Arduino without needing to wire the two boards together.



**Figure 3.1:** XBee 802.15.4 module

## 4.3 Kinect (Jeff R.)

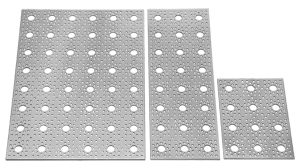
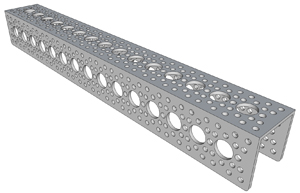
The Microsoft Kinect for Xbox 360 utilizes two sensor to perform motion tracking, an IR emitter and an IR depth sensor (Figure 4.1), and operates in two stages. In the first stage, the Kinect uses a technique known as structured light. The IR emitter emits a pattern of infrared light into the room, and as this pattern of light hits objects in the room, it becomes distorted. The IR depth sensor then reads this distortion and uses it to create a depth map of the room, including any human bodies in the room. In stage 2, the depth map is passed through a machine learning algorithm, where it is compared to over one million training examples of body positions. The algorithm determines if there are any body parts in the scene and where they are located. Once body parts are inferred, skeletal joints are assigned to the each body part [3]. The result is motion tracked skeletal joints. Using the Kinect for Windows SDK, we can retrieve the position of each joint applied to the body. This allows us to gather information about the position of the user’s hand at any time and meets our design requirements.



**Figure 4.1:** Microsoft Kinect for Xbox 360 Sensors

## 4.4 Platform (Getro)

We decided to build our own mobile platform instead of buying a pre-designed kit. Building our own platform was cheaper and allowed us to design it specifically for our purpose. We built it out of an aluminum panel and two aluminum channels. We chose aluminum because it is lightweight and more than strong enough to hold our arm. The panel is 9”x12” and the channels are 12” long to match the panel. We chose a simple Adafruit servo wheel and a FeeTech (FS5103R) continuous rotation servo to power the wheels. Additional information on the continuous rotation servos can be found in Appendix D.1. Both are affordable and fulfill their needs. Also, the servos have 2.8 pounds of torque, which is enough to move our arm, and operate under 6V, meaning we can control them from our servo controller.

 ****

**Figure 5.1:** 9”x12” Aluminum panel **Figure 5.2:** 12” Aluminum channel

## 4.5 Wireless Camera (Getro)

We needed our camera to provide clear vision of the robot arm movement form a distance. We decided on a Boscam 5.8GHz all-in-one camera and a Boscam RC305 5.8GHz receiver. The camera provides 1440x1080 30fps video with a 130 degree field of view. This gives us clear vision with a good view of the arm. We wanted a 5.8GHz transmitter and receiver to avoid any possible interference with 2.4GHz devices, like our wireless Xbee modules. The receiver we purchased only comes with an AV output, so we also bought a small video converter that converts AV input to VGA output. This allows us to connect the receiver to nearly any computer monitor to view our robot arm. The camera requires a 12V DC power supply.



**Figure 6.1:** Boscam 5.8 GHz transmitter + camera

# 5.0 Design Narrative

## 5.1 Robotic arm (Getro)

The arm was put together within a week, and required some drilling to put together as some of the parts did not seem compatible with each other. Once all the servos were wired, Jeff F. made some sample code to test the arm. We worked together to test each servo and record initial position values for the servos. These initial positions are used in the final design to set the arm in a safe position when the code begins running. In Figure 7.1 you can see the arm during construction.



**Figure 7.1:** AL5D construction

## 5.2 Kinect (Jeff R.)

The first step to getting motion tracking was to use the Kinect for Windows SDK v1.8 to create a program that would assign joints to a body. With reference to some publicly available tutorials [4], we were able to create a GUI that would draw the position of joints to the screen. With this GUI, we were able to test the motion tracking and confirm that it accurately track the user’s joints. Our next step was to retrieve the position of the user’s right hand so we could send that information the Arduino.

To do our motion tracking, we first need to assign a KinectSensor object to our Kinect [5]. Next, we enable the ColorStream, DepthStream, and SkeletonStream, and assign an event handler that is called at the end of every frame. A frame is the current image that has been processed by the Kinect, and the Kinect processes multiple frames in a second. This allows the Kinect to pass information from the color camera, depth sensor, and skeletal positions. At this point, our event handler is being called every frame. In the event handler, we open each of the enabled streams for the frame and pull information from them. From the ColorStream, we pull the image of the room that the user will see on the GUI. From the DepthStream we pull the depth map of the room. From the SkeletonStream we pull the skeletal joint information in the form of an enum. With the joint information, we look for the “HandRight” joint by saying:

*if (joint.JointType != JointType.HandRight) continue;*

This line of code will filter out any joint that is not the user’s right hand. Once we have the right hand joint, we read the coordinates from it by saying *joint.X*, *joint.Y*,and *joint.Z*. We compare those values to the values of the previous frame to see if they have changed before sending them to the Arduino. We compare the current position to the previous position because we do not need to send an update if the arm is in the same position as the previous frame. This prevents updates being sent every frame unnecessarily.

We also need to be able to open and close the gripper on our robot arm. To do this, we use an InteractionStream [6]. The Kinect is capable of recognizing gestures that the user performs. To open and close the gripper, we use a “grip” gesture. If the user closes his/her hand, the “Grip” event triggers, and if the user opens his/her hand, the “GripReleased” event triggers. We enable and assign an event handler to the InteractionStream the same way we did for the other streams. Once in the event handler, we open the InteractionStream and copy the data from it. We pass this data through a switch case that looks for a Grip or GripRelease interaction event. If we find an event, we tell the Arduino to open or close the gripper.

To send information between the Kinect and the Arduino we use serial communication. Before we can start communicating with the Arduino, we must find which COM port it is connected to. First, we get a list of all COM ports on the computer using *SerialPort.GetPortNames()*. Then we send a handshake to each COM port and wait for a response. The Arduino contains code that waits for this handshake and sends a handshake back. If the Kinect code receives the returning handshake, it saves the COM port as *currentPort*. We can then send information over the COM port using *currentPort.WriteLine(“HandRight,xPos,yPos,zPos”);*, where xPos refers to the x position as a floating point number, yPos to the y position, and zPos to the z position. To tell the Arduino to open or close the gripper, we send the string *“HandOpened”* or *“HandClosed”*.

Once we were able to send joint positions to the Arduino, we did our first motion test. We mapped the hand’s y coordinate to the elbow servo on the arm and converted the coordinate to a servo position using a linear regression calculation.

The Kinect code is also responsible for getting input for the mobile platform. We detect if the keys W, A, S, or D have been pressed on the keyboard, and send the appropriate command to the Arduino so it can move the required servos. Key W is for forward, S is backward, A is left, R is right, and space is used as an emergency brake. The commands for these controls are sent through the COM port as “*forward”*, *“backward”*, *“left”*, *“right”*, and *“stop”*. When the Arduino sees these commands, it turns the appropriate servos on or off. To go forward, the Arduino sends the wheel servos a forward position; to go backwards, it sends a backward position. To turn the platform left, the left side servo goes backward and the right side servos go forward. The opposite is true for turning right.

The program for the Kinect can be found in our public GitHub, the address for which can be found in the conclusion of this report under the [GitHub](#_8.5_GitHub) section.

## 5.3 Arduino (Jeff R. & Jeff F.)

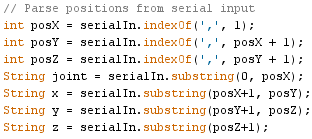
To connect the Arduino to the Kinect, as discussed in section 5.2, we need to be able to read and write over serial with the Arduino. To do that, we need to use the Serial library. Using the line: *if (Serial.available() > 0)*, we can detect if there is data in the serial buffer. If there is data and the Arduino is not connected to the Kinect, it compares the data to the known Kinect handshake format. If it matches, the Arduino sends back its handshake and the two are connected.

Once connected, the Arduino again waits to detect data in the serial buffer. Once it detects data, it read it in as *string serialIn* and looks for known commands (Figure 8.1).



**Figure 8.1:** Look for Kinect commands

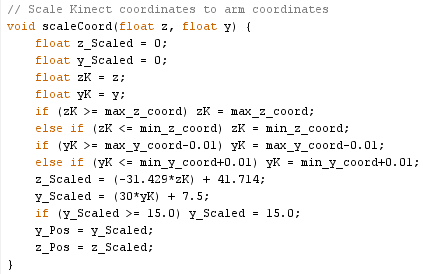
If no commands are found, it parses the data it got from the buffer into *joint*, *x*, *y*, and *z* (Figure 8.2). These are the coordinates of the right hand that are sent from the Kinect.



**Figure 8.2:** Parse serial data into coordinates

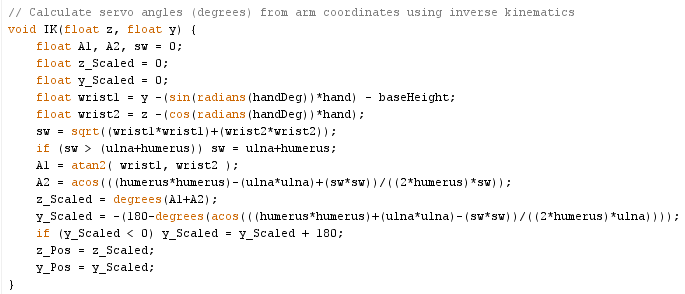
Once we have the Kinect coordinates, we can begin translating them to servo positions for our arm. Initially we used only linear regression calculations to do this. We found the maximum and minimum positions we wanted to allow for our Kinect coordinates and our servo positions, and then found the slope and intercept of the two sets. These calculations can be found in Appendix B. However, this results in mapping each coordinate to one servo. Coordinate X was mapped to the base servo for left to right rotation, Y was mapped to the elbow for up and down rotation, and Z was mapped to the shoulder servo for backwards and forwards movement. This meant that if you wanted to reach down and forward into a full reach with the arm, the user would actually have to move their arm up and forward in order to get the elbow servo to extend for a full reach, because a higher Y coordinate means the elbow servo extends to go “up”. After testing the arm ourselves, and having other people try the arm at an open house, we found this method of control unnatural and difficult to master. This goes against the purpose of our project, so we began researching other ways to calculate servo positions from coordinates.

In our final design, we use inverse kinematic calculations for the Y and Z coordinates (the X coordinate does not affect the Y or Z positions, therefore we were able to continue using linear regression for the X coordinate). Inverse kinematics uses kinematic equations to determine joint angles desired to put the end of an arm at a specific position. It is used specifically for robot arms. In Appendix C you can see the spreadsheet we used to work out the calculations and find the right values for our arm [7] [8]. This spreadsheet also shows the resulting position of the arm, allowing us to test the calculations quickly. The inverse kinematic calculations are implemented in three steps. The first step is to translate the coordinates of the user’s hand, which we get from the Kinect, into coordinates for the arm’s gripper (Figure 8.3).



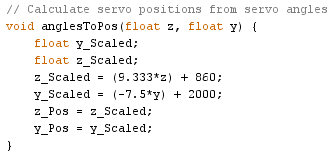
**Figure 8.3:** Scale Kinect coordinates into robot arm coordinates

The Y and Z coordinates are passed through a linear regression calculation to get the required coordinates for the robot’s gripper. The values in these calculations come from a spreadsheet that can be found in Appendix C.2. In the second step, the calculated arm coordinates are passed through inverse kinematic equations in order to get the required servo angles to move the gripper to the specified coordinates (Figure 8.4).



**Figure 8.4:** Inverse kinematic equations to calculate servo angles from arm’s gripper coordinates

These equations come from the spreadsheet in Appendix C. This function will take in the coordinates that we want for the gripper and return the required angles that the shoulder and elbow servos need to be at in order to match that coordinate. The servo angles are used in the final step to calculate servo positions (Figure 8.5).

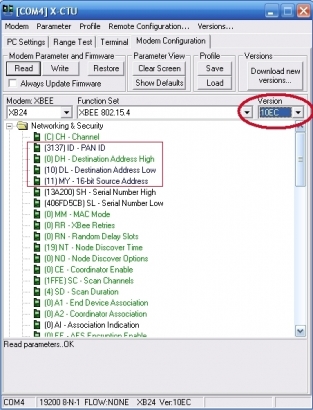


**Figure 8.5:** Calculate servo positions from servo angles

In the third and final step, the servo angles are used to calculate the required servo positions; the position will be sent to the SSC-32U servo controller as pulse widths.

The servo positions must be formed into SSC-32U commands. Commands are sent to the SSC-32U as strings in the following format: *#<ch> P <pw> S <spd> T <time><cr>*. In this format, *#<ch>* specifies the pin/channel connected to the servo that we want to move, *P <pw>* sets the desired pulse width, *S <spd>* sets the maximum servo movement speed (in microseconds per second), *T <time>* specifies the time in microseconds you want it to take the servo to move from its current position to the new desired position, and *<cr>* is a terminating carriage return. The calculated servo positions are formed into string commands and sent to the SSC-32U. To test this, we wired the Tx and Rx pins on the Arduino to the Rx and Tx pins on the SSC-32U and did a *Serial.println()*. The Arduino successfully sent movement commands to the SSC-32U. The next step was to make this communication wireless.

Commands sent from the Arduino to the SSC-32U must be done wirelessly if the arm is to be mobile. To achieve this, we use two Xbee 802.15.4 2.4GHz modules. One Xbee module is connected to the Arduino, and the other is connected to the SSC-32U. We first had to configure the Xbee modules to communicate with each other. To configure the modules, you must connect 5Vdc from the Arduino to 5Vdc on the Xbee, ground on the Arduino to ground on the Xbee, Tx to Tx, and Rx to Rx. Then, using the X-CTU software, you can set the ID and destination address for the Xbees [9] (Figure 8.6).



**Figure 8.6:** Xbee configuration using X-CTU

For the first Xbee, we set PAN ID to 3137, DH to 0, DL to 10, and MY to 11. For the second Xbee, we set PAN ID to 3137, DH to 0, DL to 11, and MY to 10. You’ll notice that DL of the first Xbee matches MY of the second, and vice versa. This is what links the two Xbees together. Once the two Xbees are configured, they will automatically send and receive data between each other.

Initially, we used the Xbee shield for the Arudino Uno, however, because the shield used the same Rx and Tx pins as the USB serial (pins 0 and 1), there was a conflict. We had to create our own shield, which used different pins for Rx and Tx (pins 2 and 3). With this shield, 5Vdc and ground from the Arduino are connected to their respective pins on the Xbee. Rx on the Arduino is connected to Tx on the Xbee, and Tx on the Arduino is connected to Rx on the Xbee.

We use pins 2 and 3 on the Xbee to act as serial Rx and Tx pins, so we had to use the SoftwareSerial library on our Arduino to send serial data to these pins instead of the default pins (pins 0 and 1). This is done easily by creating a SoftwareSerial object: *SoftwareSerial Xbee(2, 3);*. We use the *Xbee* object as you would a normal Serial object: *Xbee.println(data);*. Commands are sent from the Arduino through the SoftwareSerial pins to the Xbee. They Xbee then sends those commands to the connected Xbee on the SSC-32U, where the commands are read by the servo controller and executed by the servos.

The program for the Arduino can be found in our public GitHub, the address for which can be found in the conclusion of this report under the [GitHub](#_8.5_GitHub) section.

## 5.4 Platform (Getro)

The two aluminum channels were attached to the panel with screws, and the four continuous rotation servos were attached to the inside of the channels. From there, the wheels were attached to the servos and the platform was complete. The arm was attached to the panel using zip ties and the servos were wired to the SSC-32U. Holes were drilled into the base of the arm to attach the camera with zip ties, the placement can be seen above in Figure 6.1.

Once the arm was mounted on the platform, we needed a wireless power source for the arm and the camera. A 12V DC 3000mAh output lithium ion battery pack was used for the camera, and a 6V DC 2000mAh NiMH RX battery pack was used for the arm. The 6V battery pack for the arm did not come with the right connector to connect it to the SSC-32U, so we had to solder on a 2.1x5.5mm DC male power pigtail. This meant we also had to solder on a female connector to the charger to be able to charge the battery pack. The battery pack for the arm can be seen in Figure 9.1 below.



**Figure 9.1:** 6V DC 2000mAh NiMH RX battery pack

To connect the camera transmitter and receiver, we simply set both to the same channel. We then connected the AV out of the receiver to the AV in on our AV to VGA converter, and the VGA out of the converter to a monitor. That provided us with a clear picture of our arm from a distance.

# 6.0 Professional and Societal Concerns, Cost and Economic

# Evaluation

As the robot will move wirelessly on the mobile platform, it may be vulnerable to safety concerns. Pedestrians may step on it by accident, so we could prevent such a scenario by attaching a motion sensor in front of the robot that could sound an alarm if a person was too close to it. Despite how much we optimize the robot's movements, anyone who operates it will need to familiarize himself or herself with the Kinect's motion sensor before using it for any purpose, whether it is for a recreational, business, or emergency use. The operator of the robot will need to be efficient enough to interact with objects properly.

The robot uses a 6-volt DC 2000mAh NiMH RX Battery Packs rechargeable battery for power, and therefore does not produce any harmful byproduct to the environment. For a purpose regarding cleaning pollution, the robot could transport radioactive material away from a hazardous location.

The most intended purpose of the robotic arm is to interact with objects from a distance and without human touch. The robotic arm works well in environments hazardous to humans. The operator can move the robot arm to travel through scenarios including dangerous gases, ditches too small for people, and to interact with unsafe objects such as bombs or radioactive material with the flexibility of human hand motion and without the risk of harm to a person.

As the robotic arm uses a camera for the user to see where it is going from a distance, it may face similar legal concerns to that of commercial drone products due to privacy. However, unlike drones, the robotic arm is not capable of flight as of yet and would therefore have limited capability to spy illegally on other people. There may also be societal complaints that the use of the mobile robotic arm may encourage people to distance their human interaction more from their work when it approaches a more reliable state. Alternatively, people who use the robotic arm for recreational purpose may injure themselves or someone else unintentionally if not operated responsibly, especially if using a product for an unintended purpose.

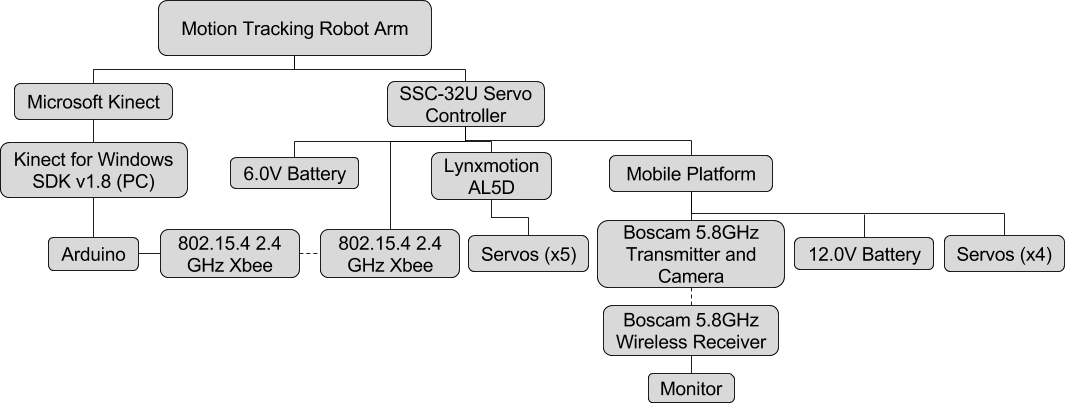
The Mobile Motion Tracking Robot Arm is a project that we believe is open to innovation. Our programming with the Arduino and the Microsoft Kinect SDK can provide the groundwork for any other group to expand onto other kinds of robotic arms besides the Lynxmotion used for this project. With the vast possibilities available to many engineering companies, there will be different variants of the robotic arm for different purposes. When the product becomes affordable enough for recreational purpose, it will be feasible for there to be customizable parts for the arm itself, such as a softer material for the gripper. We believe that this vast amount of options would benefit the economy. A full cost breakdown of the project can be seen in Appendix A.

# 7.0 Discussion

The first step to our project design was to determine which component we would use. After careful research, we began designing our system. The first subsystem we design was the Microsoft Kinect; it handles motion tracking and provides the coordinates for the user’s hand. We decided to use an Arduino Uno to do the calculations required to convert the Kinect coordinates into servo positions, and to communicate with the SSC-32U Servo controller. We liked the idea of keeping the motion tracking and the calculations and servo commands separate. After getting basic motion tracking done, we began writing the code for the Arduino to receive data from the Kinect and turn it into commands for the SSC-32U using linear regression calculations. Once the initial test code was done and the Lynxmotion AL5D was built, we began performing tests to verify our design. We started by linking the X coordinate of the Kinect to the base rotation servo of the robot arm, and we were able to move the arm using the Kinect. When the test was completed successfully, we linked each coordinate from the Kinect to a servo of the arm and we were able to achieve three degrees of freedom with the arm. At this point we decided to move on to the next subsystem and make the arm mobile. Making the arm mobile would mean that we did not have to wire the Arduino to the SSC-32U every time we wanted to do a test, and we could use batteries to power the arm instead of having to plug it in to an outlet. Also, without a platform the arm was not stable, and we were in constant fear of it falling off the table.

The next subsystem to design was the mobile platform. We originally wanted to buy a prebuilt mobile platform, however, we could not find one that suited our needs. We decided the best route would be to construct our own mobile platform. We used aluminum, because it was strong and lightweight, and continuous rotation servos, because we could control them using the SSC-32U servo controller that controls our arm. The arm was mounted to the platform, and batteries were purchased and modified for the arm and camera. Heavier testing could now be done, and the arm was found to be too sensitive. We were able to resolve this by applying smoothing values to the motion tracking that are defined in the Kinect SDK, as well as decreasing the window in which the arm can move that is created by the maximum and minimum values in our linear regression equations.

We performed extensive testing, including having other people use the arm, and concluded that there were some flaws in our design. The platform performed well, but the arm was difficult to use. Linking each coordinate to a servo resulted in an unnatural and difficult control method. To fix this, we implemented inverse kinematic equations. After many failed tests involving the arm moving to incorrect positions, we were able to find the right values and put limits in place on the servos to keep them from moving the arm into positions that might break it. More tests found the arm easier and more natural to control. The rest of our design process included fine tuning the smoothing values for the Kinect and refining the platform movement. At the end of our project, we fulfilled our final goal of being able to drive to a distant object, pick it up, and bring it back. In Figure 10.1 you can see the hierarchy for our system.



**Figure 10.1:** System Hierarchy

# 8.0 Conclusion and Recommendations

## 8.1 Robotic arm (Getro)

We are satisfied with the Lynxmotion AL5D and would recommend it for a future version of this project. However, future improvements may want to invest in a larger and higher quality arm, similar to one currently in use in the military. This would allow much larger objects to be lifted, along with quicker and more precise arm movement.

## 8.2 Kinect (Jeff R.)

The Microsoft Kinect for Xbox 360 works for our project, however I would recommend that in any future work on this design, use a better form of motion tracking. I believe even using the Kinect 2.0 would have benefited us in the form of more accurate and consistent motion tracking. If the budget is available, the best method would be 3D motion tracking done with multiple cameras from multiple angles.

When programming the motion tracking, it is recommended to use smoothing or data filtering on the joint information. This provides much more consistent data and smoother arm movement. Before we applied smoothing to our design, the robot arm was very shaky and it was hard to move the gripper to an exact location to pick up an object. It would also be beneficial to have as little delay as possible between the user’s movement and the motion tracking. When increasing smoothing and filtering on the Kinect, it added delay to the motion tracking, and in turn to the robot arm. The delay was slight, but still noticeable. Reducing this delay should be an objective of any future designs.

## 8.3 Arduino (Jeff R.)

The Arduino and calculations worked well for us. I would recommend inverse kinematic equations over linear regression calculations when appropriate. If we were to redesign the project, I would conduct the calculations more carefully. When going through the calculations for the first time, we made many mistakes, and even at the end of our project I believe we could have better arm movement we were to recalculate the values used in our program. Many of the max and min values used for the linear regression calculations are approximations and depend on the location of the Kinect in relation to the user. It would be wise to set up the Kinect the same position and stand in the same location for each test to ensure repeatability and maximize accuracy.

## 8.4 Platform (Getro)

Over all the servos and tires we use to move the platform get the job done, but we would recommend using all terrain tires and servos with more torque for more power and speed. As it is, the robot performs well on flat surfaces, but struggles when driven on carpet or other rough types of terrains. Another option would be to use tracks instead of wheels. Tracks provide excellent traction and work well on many surfaces. Our platform has one panel on top of tires to support the arm because we wanted to save money, but if you add another plate at the bottom there should be room to hide all the wires, power supplies and the servo controller for a cleaner look and to protect vital components. It would also be easier to control the platform if you used a handheld keypad instead of the computer’s keyboard. Often times, the setup for the computer and Kinect makes it awkward to drive the platform and control the arm at the same time.

## 8.5 GitHub

All of our code for the Kinect and Arduino, as well as documents used for calculations, is publically available on GitHub. You can find the repository at:

<https://github.com/StormWulf/Mobile-Motion-Tracking-Robot-Arm>

# 10.0 References

[1] Lynxmotion. "Lynxmotion - SSC-32U USB Servo Controller." Lynxmotion. N.p., n.d. Web. <<http://www.lynxmotion.com/p-1032-ssc-32u-usb-servo-controller.aspx>>.

[2] "METRO Development Boards with ATmega328." *Digikey*. N.p., n.d. Web. <https://www.digikey.com/en/product-highlight/a/adafruit/metro-development-boards-with-atmega328>.

[3] MacCormick, John, Dr. "How Does the Kinect Work?" (n.d.): n. pag. 6 Sept. 2011. Web. <<http://users.dickinson.edu/~jmac/selected-talks/kinect.pdf>>.

[4] Pterneas, Vangos. "Vangos/kinect-coordinate-mapping." GitHub. N.p., 4 May 2014. Web. <<https://github.com/Vangos/kinect-coordinate-mapping>>.

[5] Smith, Tammy R. "Kinect for Windows Programming Guide." Microsoft Developer Network. Microsoft, 15 Jan. 2015. Web. <https://msdn.microsoft.com/en-us/library/hh855348.aspx>.

[6] Velvárt, András. "Kinect Interactions With(out) WPF – Part III: Demystifying the Interaction Stream." Dotneteers. N.p., 3 May 2013. Web. <<http://dotneteers.net/blogs/vbandi/archive/2013/05/03/kinect-interactions-with-wpf-part-iii-demystifying-the-interaction-stream.aspx>>.

[7] Keesling, Mike. "Lynxmotion Arms and Inverse Kinematics." Lynxmotion. N.p., n.d. Web. <http://www.lynxmotion.com/images/html/proj057.htm>.

[8] Micromega Corporation. "Application Note 44: Controlling a Lynx6 Robotic Arm." (n.d.): n. pag. 9 Feb. 2009. Web. <http://www.micromegacorp.com/downloads/documentation/AN044-Robotic%20Arm.pdf>.

[9] Kawal. "How to Use Xbees with Arduino." Community of Robots. N.p., 4 Dec. 2011. Web. <http://communityofrobots.com/tutorial/kawal/xbee-and-arduino>.

# 11.0 Appendices

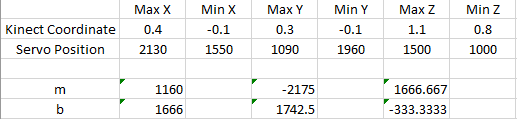
**Appendix A**

|  |  |
| --- | --- |
| **Part** | **Cost** |
| 6V Battery Charger | $16.00 |
| 6V Battery | $8.60 |
| 12V Battery + Charger | $24.00 |
| Lynxmotion AL5D + SSC-32U | $310.00 |
| Platform Components (1x aluminum plate, 2x aluminum channel) | $37.00 |
| Platform Hardware | $11.00 |
| FeeTech Continuous Rotation Servo (x4) | $47.80 |
| Adafruit Servo Wheel (x4) | $11.80 |
| Arduino Uno | $25.00 |
| 2x 2.4ghz xbee module | $50.00 |
| Xbee adapter | $10.00 |
| Kinect | N/A |
| Boscam All-In-One Camera & Recevier | $59.99 |
| Boscam 5.8GHz Receiver | $17.98 |
| AV to VGA Converter | $18.00 |
| **Total** | **$647.17** |

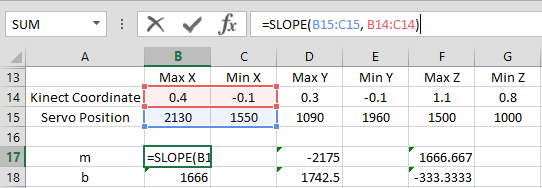
**Figure A.1:** Cost Breakdown

\* Kinect was already owned by a group member

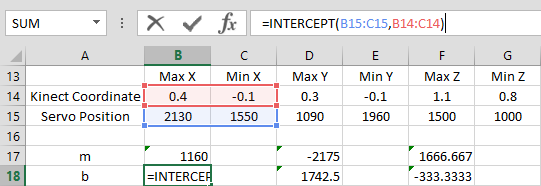
**Appendix B**



**Figure B.1:** Linear Regression Calculations

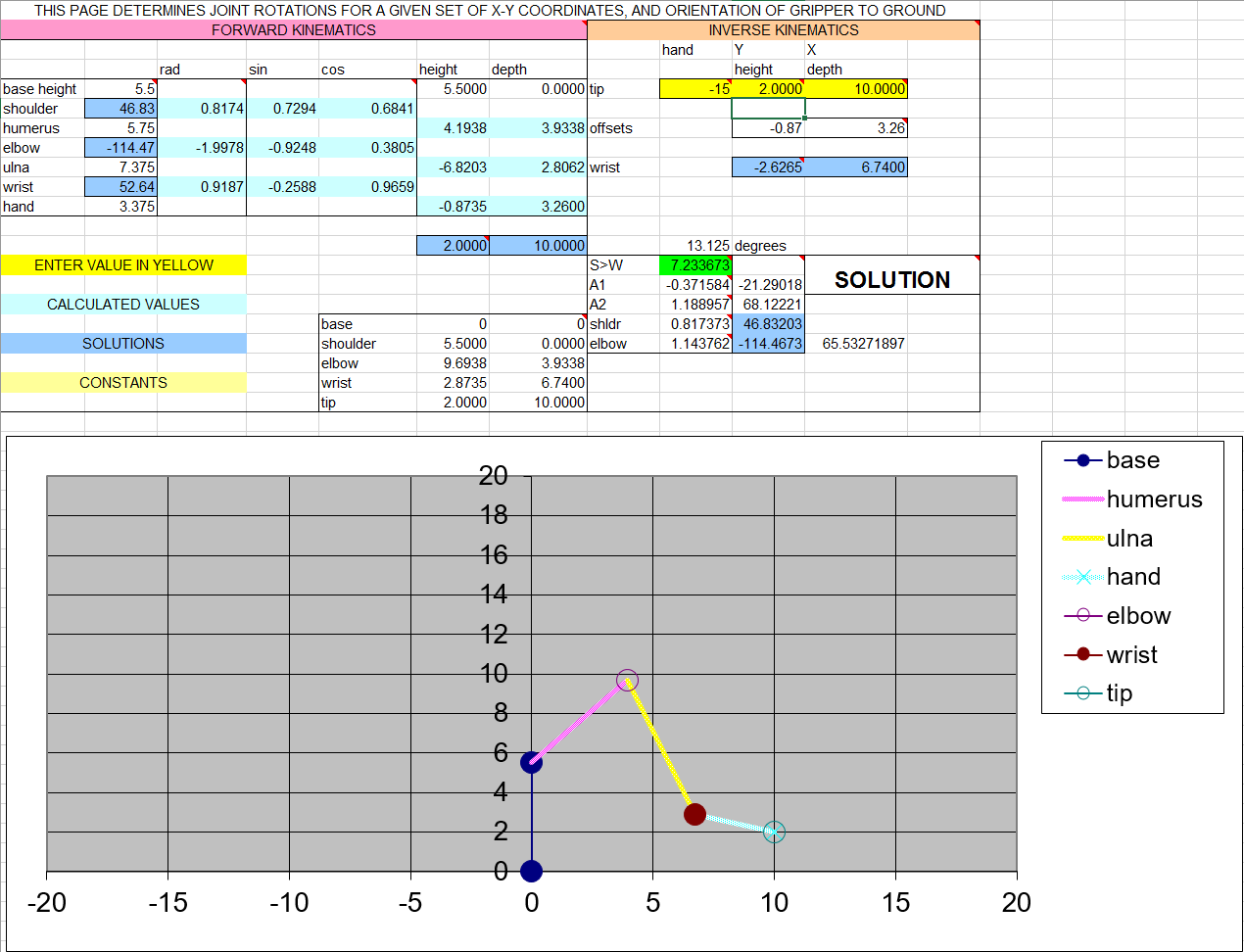


**Figure B.2:** Linear Regression Calculations

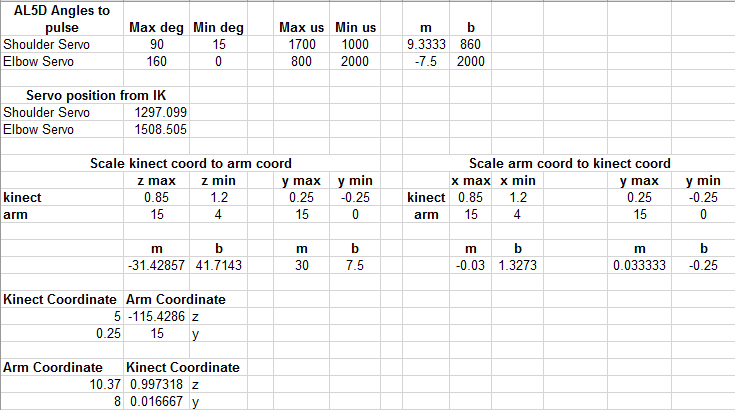


**Figure B.3:** Linear Regression Calculations

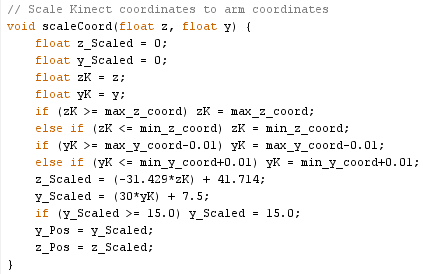
**Appendix C**



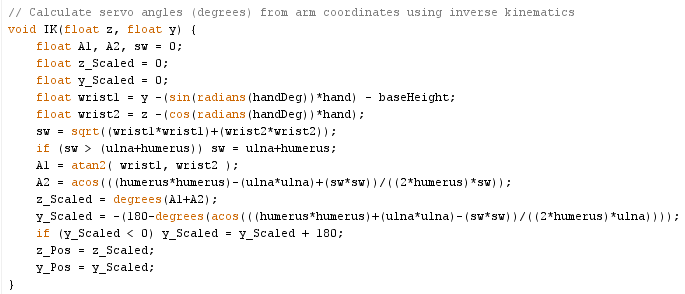
**Figure C.1:** Inverse Kinematic Spreadsheet [6]



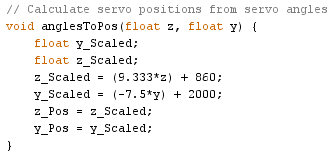
**Figure C.2:** Inverse Kinematic Calculations



**Figure C.3:** Arduino – Calculate robot arm coordinates from Kinect coordinates using linear regression.



**Figure C.4:** Arduino – IK calculations. Calculate servo angles from arm coordinates.



**Figure C.5:** Arduino – Calculate servo positions from servo angles using linear regression.

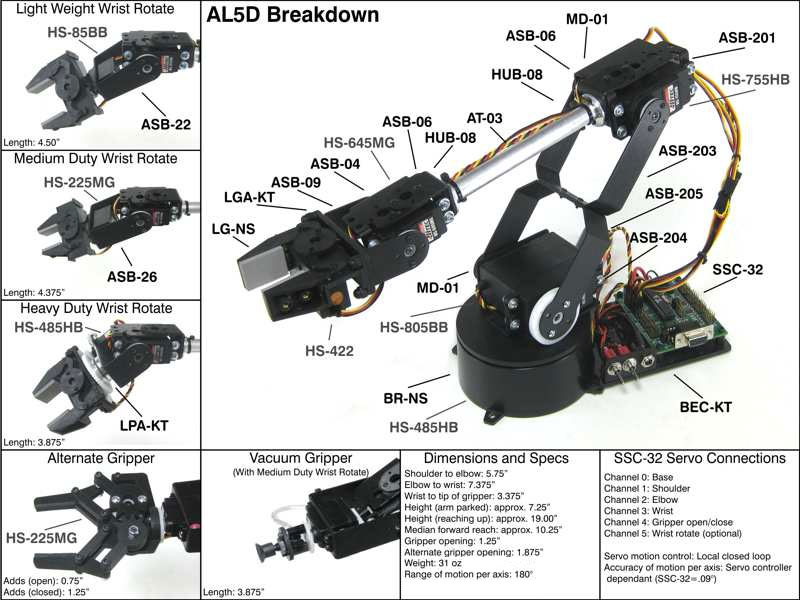
**Appendix D**



**Figure D.1:** FeeTech Continous Rotation Servo (FS503R)

Details:

* Continuous Rotation Servo Free Tech (FS503R)
* Operating Voltage: 4.8V~6V (5V works best)
* Average Speed: ~0.18sec/60°
* Stall Torque (4.8V): 3kg.cm/41.74oz.in
* Stall Torque (6V): 3.2kg.com.in
* Required Pulse: 500us-2500us
* Connector Wire Length: 30cm / 11.8"
* Dimensions: 37mm x 54mm x 20mm / 1.5" x 2.1" x 0.8"
* Weight (no horns): 40g
* Spline Count: 25



**Figure D.2:** Lynxmotion AL5D Specifications