Stony Brook University

IMPROVED USABILITY FOR A CONTROL SYSTEM USING HUMAN BODY IMAGING AS THE CONTROL METHOD FOR ARM BASED ROBOTICS IN ASSISTIVE TECHNOLOGIES

A THESIS PRESENTED

by

TAUREAN LINDSAY DYER

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Master of Science

in

Mechanical Engineering

Stony Brook University

August 2013

Stony Brook University

The Graduate School

Taurean Lindsay Dyer

We, the thesis committee for the above candidate for the

Master of Science degree, hereby recommend

acceptance of this thesis.

Professor Yu Zhou

Associate Professor in Dept. of Mechanical Engineering

Professor Qing Chang

Associate Professor in Dept. of Mechanical Engineering

Professor Chad Korach

Associate Professor in Dept. of Mechanical Engineering

This thesis is accepted by the Graduate School

Charles Taber

Interim Dean of the Graduate School

Abstract of Thesis

Improved Usability for a Control System using Human Body Imaging as the Control Method for Arm Based Robotics in Assistive Technologies Mobile Robotic Arm

By
Taurean Dyer
Master of Science
in
Mechanical Engineering
Stony Brook University
2013

The overall goal of this project is to move the assistive control system and test bed from "proof of concept" to "functional prototype" based upon the novel approach presented by Dyer, et al for the Stony Brook 2012 Group 14's Senior Design, "Human Position Imaging as a Control Method for Robotics and Assistive Technologies". The novel approach was a proof of concept that of a cross compiling, computer assisted method that relied on cheap, off the shelf parts to produce easy, natural , powerful, yet affordable control over a mobile 5 degree of freedom robotic arm. This project further improves on that initial proof of concept system and applies focus on address system usability issues. After parametrically analyzing existing joint tracking input software, it adds a smoother motion profile output, visual feedback for the operator, and elements were introduced to make operating the control system easier and more informative. It also finished off areas of implementation not fully fleshed out by the original proof of concept. The addressed areas of research and design in this research were successful in improving the overall operator experience of the control system, while holding dear to the intended design philosophy.

Table of Contents

Table of Figures

List of Tables

Acknowledgements

Thanks to

Richard Anger, Anthony Hannigan, and David Umlas of my Senior Design Team, who all assisted me greatly with design ideas, construction, testing, and tedious report writing of the proof of concept control system.

Paul St. Denis of the TLT Media Labs, who introduced me to the Kinect, hacking it, Python, and computer systems programming during my time at the Labs. He was also a wealth of knowledge and an ever giving resource and helped debug my code or guide me through rough programming spots.

Will Thompson, who worked with me over the summer of 2012 cleaning up my code, helping build troubleshooting tools, improving the smoothness of the arm, working with PID, and helped implement and test the control system with DC and stepper motors for future work.

Mark Krieger, who encouraged and taught practical electronics and microcontroller philosophy and implementation to a crazy kid from New York whom he had never met.

Professors Wang, Kincaid, Testa, and Sesay for their teachings, encouragement, assistance. A special mention to Mayra Santiago for all the good times during my Master's degree.

A Special Thanks to

My mom, dad, both grandmothers, my fiancé Afiya Walters, and my extended family, like Dr. Denise Thompson, Dr. Edghill Messiah and Donna Dove, who actively taught, encouraged, helped, cheerlead, distracted when necessary, elevated me when they could, and just pushed me through life's milestones, ideas and progress on life, school, and thesis.

A Very Special Thanks to

Professor Yu Zhuo for his kindness, guidance, and foresight. He was my advisor for Senior Design, was the one who encouraged me to pursue my Masters and beyond, and guided me throughout the entirety of these 2.25 years. He brought out my best, gave me the chance I needed, and has my eternal respect and gratitude. Truly, all of this was because of him

Chapter 1

Introduction and Background

1.1 Motivation

There is an increasing demand of using robots and computer systems to assist the consumer human population in almost every facet of life. People today walk around with a new generation of hybrid personal digital assistants and mobile communication devices that have the processing power of 10 year old desktop computers, drive in cars with 30+ ECUs that can monitor and control almost every internal component in the car, and have automatic, self-powered vacuum cleaners and lawn movers. Unmanned aerial vehicles (UAVs) are now being in used in fields from home security to fighting fires, to monitoring and dusting crops. For the disabled population, there bionic body parts are being surgically implanted into human bodies to assist those who have been injured or were born disabled. Electronic wheel chairs have been developed to give those who cannot walk under their own power mobility. However, for this demographic, if not in the chair or not a candidate for surgery, and unable to easily move under their own power, there is little in existence that will allow you the ability to be self-reliant to perform simple tasks. The reason for that is not due to the mechanical constraints of technology, but the deficiency in human interface between person and machine. Current technologies are difficult to use, hard to learn or master, are cumbersome, or are expensive. In this project, I will build upon my previous work in the concept of using human body mapping sensors to provide a natural and intuitive control system for a robotic arm and increase the feasibility of utilizing this concept by making the system more stable, user friendly in operation, and easier to set up and run. The ease of use of this system, the extremely low learning curve due to the natural control scheme, can potentially open the door to a new breed of powerful assistive robots that operators of all ages can used to better their life.

With America's aging, yet increasingly tech savvy population, this control system maybe the final piece in the puzzle to increase mass production and bring robots once only intended for research or industry cheaply into the home.

The overall goal of this project is to move the assistive control system and test bed from "proof of concept" to "functional prototype".

1.2 Background on Tele-operated Mobile Robotics

Nikola Tesla was experimenting with possible applications for his new radio communications system when he theorized and implemented a way of sending operator commands to a remote device using this radio technology. This "radio control" scheme, patented in 1898 (Tesla), became the foundation of almost all our current radio, or remote, controlled devices. Its uses in tele-operated machinery became a growing field it and started to merge with the growing field of computer science

Developed initially by Nikola Tesla during his work in radio communications, this system of using electromagnetic signals to send a user's intended motion commands to a controller or processing plant has grown to myriad applications. Since its inception, tele-operated mobile robotics has been an ever growing field with research, industrial, military, and some civilian applications. In the research field, these robots are being pushed into sensor integration and greater processing power to assist in greater autonomy. On the industrial side and military, building upon research work In military systems, In the civilian market, hobbies such as remote controlled vehicles,

1.3 Background on Hardware and Software Control Systems for Robotic Arms

1.3.1 Hardware

Robotic arms require control across each degree of freedom (DOF). Initially, the human machine interface for this control came in the form of displacement mapping methods, using input devices such

as control sticks or sliders, or increase-decrease methods using levers or buttons. These forward kinematic methods were satisfactory for a 1DOF, 2 DOF, or even 3 DOF arm for a single operator, yet as the arms complexity increase, so did the number of axis inputs, and the system usability suffered, often requiring multiple operators working together. As embedded systems and computer science progressed, new, more single user friendly methods were also devised. The added processing power allowed the user to input something simple which causes the robot to calculate the more complex joint solution of how to get there through a mathematical process called inverse kinematics.

1.3.2 Software

Inverse kinematics (IK) takes the starting point of the robot's joints and the end effector point of the position you want the tip of the robot's joints to be located, and uses that date in conjunction with the lengths and angle range of each joint in order to solve the necessary angle for each joint will have to be at to enable the end effector to achieve its desired position in space. The IK solutions are either solved on the controlling computer and the individual joint angles are sent to the robot or the end effector position is sent to the robot directly for processing. There are several general methods of solving an IK equation, depending on complexity, such as a trigonometric solution for 2D analysis, the Cyclic Coordinate Descent (CCD), or the Jacobian method. The trigonometric method is a light IK solution requires a low complexity (2-3 DOF) arm with all link translations constrained along a 2D plane. CCD is an iterative process that optimizes the kinematic chain parameters in order to always get you a solution, if one exists, at the cost of inorganic seeming movements. The Jacobian method is a highly versatile and regarded method that accommodates for any number of DOF or motion ranges, but it may not always yield a solution, even if one exists or might give you multiple solutions, and then the software must pick the best one.

Using IK and computers to quickly work through the computational burden, new control methods have been devices that allow for greater single operator control that function with greater precision while achieving faster, smoother response times. These control systems work on tactile feedback from a human operator, joint or skeletal mapping, or highly complex relative or absolute positioning devices.

1.4 Back ground of Proof of Concept Control System

The proposed system of this thesis uses a hybrid input control system comprised of skeletal mapping, absolute positioning, and joystick control to map the 5DOF it is to control as well as mobile locomotion. Employing various off the shelf input sensors, signal conditioning and algorithms microcontrollers, it allows any person with control of their upper body easy control of a mobile 5 degree of freedom (DOF) robotic arm. Emphasis was placed on ease of use, cost, and portability. The input system does not depart from a person's natural movement and is programed to respond almost puppet-like to the orientation of the operators arm. The implementation of this human machine control system opens the door for the average person controlling these complex types of arms due to its low relative cost. The system is universal and modular in both hardware and software architecture due to its underlying foundation being in the open source community, and caters for a multi- operating system, multi-motor, and multi communication type environment. Based on its applicability, this system can be integrated quickly and cheaply into applicable preexisting or purpose built industrial, military or civilian robotics. It will reduce training times in all markets, reduce costs and complexity in all markets, and increase usability and user satisfaction, especially in the civilian market, without sacrificing too much control or precision.

1.5 Thesis Structure

Chapter 2 will discuss the control system design. It will conceptually detail the design of the robot arm system as implemented in the Senior Design, including the components, how they work together, costs, communication, and process flowcharts.

Chapter 3 presents the research done into mitigating the control system output instability, the visual feedback, and other methods of making the users' life easier while using the system. It will include comparison of the two main software development kits (SDKs) for a Primesense enabled microcontroller, the methods used to smooth the output of the servos, the implementation of the visual feedback, and steps taken to have a more informative and portable system.

Chapter 4 gives a detail of the changes made the revised system's process map and communications based on the findings of Chapter 3. It will focus on places where improvements and additions were made.

Chapter 5 demonstrates the effectiveness of the proposed changes made in Chapter 4 to the control system as a whole. Tests will seek to find improvements in the areas of usability, control, capability, and stability.

Chapter 6 concludes the work and imagines areas of potential further study. These areas include the fields of computer vision, voice control, multiple arms, and haptic feedback. Software schemes developed to implement and test this control system make it applicable to work in a wide area of devices and fields, and further research could be done to enable this to happen.

Chapter 2

Proof-of-Concept Control System Design

2.1 Introduction

The proof of concept system, which will be the test bed, was designed using the following core design philosophies:

Easy to use

The main philosophy behind the process by which the operator controls the arm by must be as initiative and natural as possible. This system is supposed to be an assistive system, and should minimize the burden of learning and get right to its function of assisting as soon as possible. Also, based on market analysis, the learning curve must be extremely low as the intended operator's ages, mental capacity, or desire to learn a more complex system may preclude them from enjoying the use and assistive benefits of a more complex system. A more complex system could potentially frustrate the operator, or the operator may not be able to physically or mentally perform the more complex tasks required to properly control the system and allow it to perform its assistive functions.

Predictable, effective control

This control system is intended to augment the need for hired help and boost self-reliance. It is to help them perform some useful, normal daily tasks that a more able-bodied could. Therefore, the arm must move and perform as the operator intends it to on a consistent basis. If the system is unable to perform, is too frustrating to use, or is not very consistent, it starts becoming less of an assistance and more of a burden, thus failing its goal.

Unobtrusive control

This control system must use as few “on operator” devices as possible as well as have as little “set up” and “take down” time as possible. The intended operator already requires assistance and this device is supposed to enable them to be self-reliant. Complex set up, or set up that an operator is unable to perform themselves, should be avoided as it defeats that goal of self-reliance. The system must be versatile to work in the chaotic environment of a home environment and should be mobile enough to be redeployed at a moment’s notice. The system should be easy to use and easy to work with.

Inexpensive

The system must be affordable to the average person. Those who fit the target demographic of this system usually are receiving assistance of some kind, be it financial or personal. They have to purchase specialty items, possibly have large medical bills due to numerous doctor visits, and the local government most likely has a salary tiered limitation on disability assistance that they may receive, if they receive any at all. This means that their budgets are a little tighter than those of a fully abled person. Thus, this system has to be cheap. To accomplish this, instead of using special purpose devices, common, off the shelf components comprise the system. The end result was a sub $300 control system with exception versatility in terms of growth, cross platform usage, and multiple motor types.

2.2 Control System Design Concept

2.2.1 Overview

There are several outputs that need to be controlled when the program begins: the 5 DOF arm kinematic output, the gripper output, and the locomotion output. The design revolved around assessing the optimum input sensor to get the best possible measurement and system “ease of use” that produces the intended output. For this stage, the systems are considered open loop.

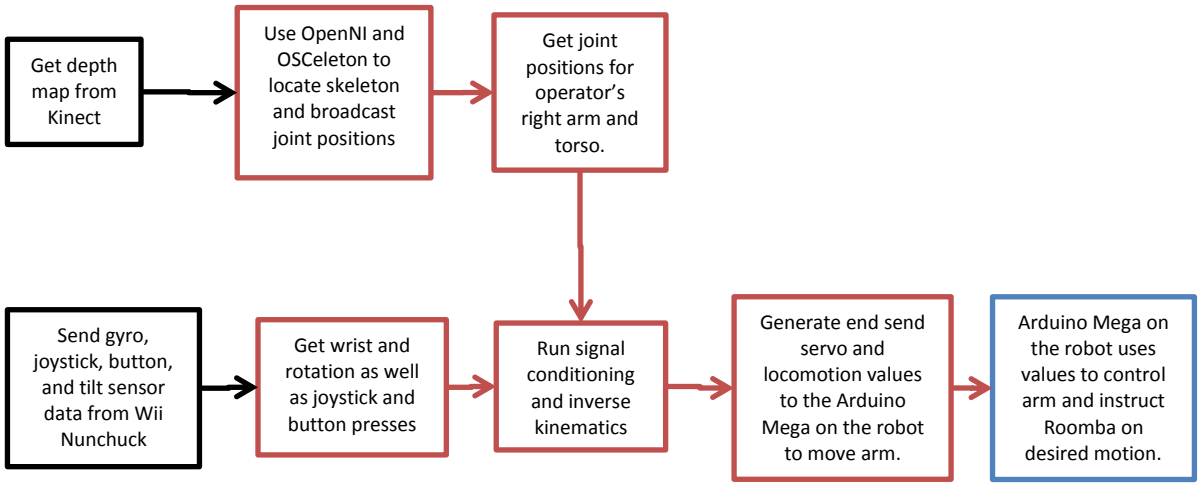


Figure 2-1: Process Flow Diagram for the Assistive Mobile Robot Arm Control System

As illustrated in Figure 2-1, there are two main input devices that incorporate a variety of sensors. The first is the Microsoft Kinect, which attaché directly to the computer via USB. Its depth data is then processed and joint data is returned to the main program for use. The second is the Nintendo Wii Nunchuck. It is attached to an Arduino Uno, who gathers the necessary data and constructs a control string to be sent to the computer also via USB. The post processed Kinect data and the Nunchuck control string are processed by the main program using an inverse kinematics algorithm and the angle results conditions and then sent to the main robot's Arduino Mega as servo angles. The Arduino Mega then controls the Roomba 530 mobile base and the servos.

2.2.2 Communication

The system uses a mostly serial based interconnect for communication between controllers. As shown in Figure 2-2, which is the map of the communication implementation of the test bed, the Kinect uses USB 2.0 to connect to the computer. For the Wii Nunchuck to the Arduino Uno, an I2C protocol is used. 115.2kbps UART is used both the Arduino Uno and Mega to computer due to the speed. This bitrate is used as it is the fasted bitrate for a 16Mhz Arduino that has a <1% error [1]. In order to match, the

communication between the Roomba and Arduino Mega is also at 115.2 kbps, using the iRobot's proprietary SCI protocol. The Arduino Mega commands the servos over PWM to manipulate arm position. Though not implemented a conceptualized video system transmits video from the camera to the video Tx via RCA in the NTSC format, then it is sent to the Video receiver via 900MHz FM, then from the receiver to the video grabber via RCA, and finally from the grabber to the computer over USB.

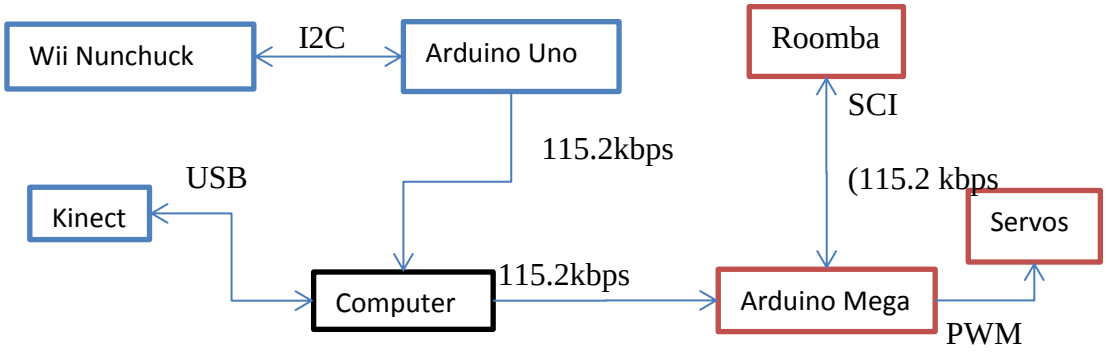


Figure 2-2: Test Platform Communications Schematic

2.2.3 Input

2.2.3.1 Kinect Input

The Microsoft Kinect is a 15-30 frame per second, 640x480, USB 2.0, Primesense enabled depth sensor. Originally created for the Xbox 360, using the Primesense chip and some interpreting software, this sensor can extrapolate the body shape and the joint positions of a person in its depth field. The found joints can be arranged to create a skeleton of the user. By using OSCeleton and OpenNI, PrimeSense's open source SDK, the control system can get the XYZ positional data of the right shoulder, right elbow, right hand, and torso joints, disseminate the kinematic chain of the human arm, and find the start point and end effector.

The Kinect uses the method of structured light to get the depth of an object. This makes it a very noisy sensor and increases the instances and propagation of errors in its skeleton tracking. Optimum positioning is important when using this device. Even so, it is not reliable enough to accurately track the hands and sometimes loses accurate tracking of some joints. Therefore, additional sensors are required to complete the missing DOF. Together with the Wii Nunchuck's array of sensors to get some more precision values, the inverse kinematics can be solved, mapped into robot arm space, and implemented in forward kinematics. It cost $90 for the refurbished Kinect.

2.2.3.2 Wii Nunchuck

2.2.3.2.1 Overview

The Wii Nunchuck is vital to the success of the control system, as the Kinect cannot track hands or wrist angles or rotations well at all. The collection of these necessary parameters had to be off loaded. The Wii Nunchuck is a sensor array of buttons, joysticks, gyroscopes, and accelerometers. Using the gyroscopes, we can get the wrist angle and rotation which are the 4^{th} and 5^{th} DOF of the robotic arm.

Using the joysticks, we can control the 2D motion of the robotic base. Using the buttons, we can control gripper open/close and the robot arm safety and control feature of enabling or disabling Kinect capture and the inverse kinematics engine on the main program side. It cost $15 for the unit and $3.95 for the Adafruit Wii Nunchuck breakout board.

2.2.3.2.2 Arduino Uno

The Wii Nunchuck moves communicates over the I2C bus. Using the open source Adafruit Wii Nunchuck Library, the Arduino acts as the I2C master and requests the raw data from the microcontroller in the Wii Nunchuck. As shown in Figure 2-3, using the supplied Wii library, the data stream that is sent from the Nunchuck's microcontroller is parsed and the raw values of the joystick, button states, and the gyroscope are collected then retransmitted to the computer at 115200 bps. Also, the button states are stored in a software enabled "press and lock" state, so that the operator does not have to hold down the button to continuously issue their intended command. It cost $30 for this Arduino Uno.

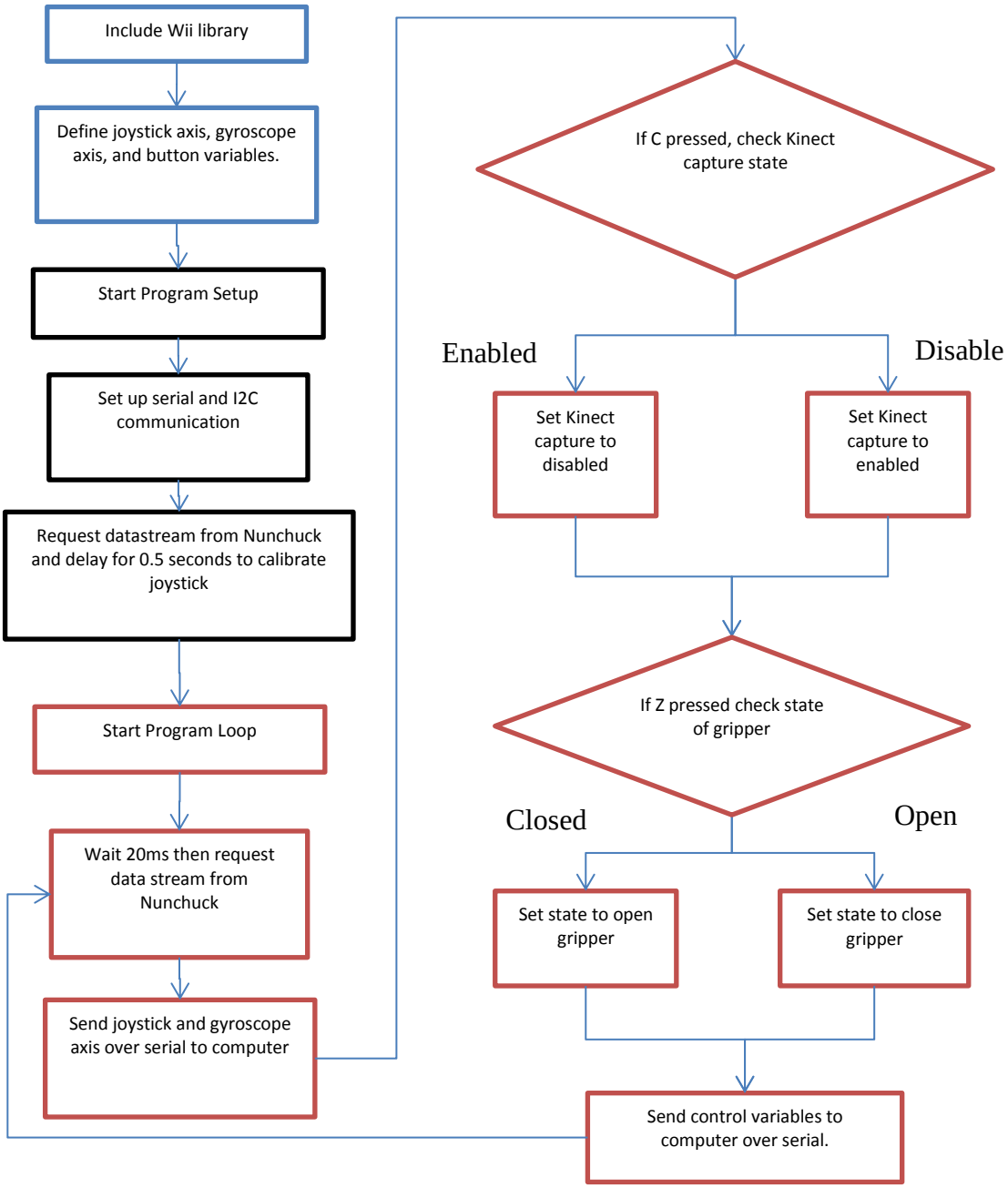


Figure 2-3 Flowchart of Code for the Wii Nunchuck Interpreter (Arduino Uno)

2.2.4 Processing

2.2.4.1 Overview

The signal processing and output of the system was written in Python. The following section will briefly go over the necessary components of how the post processing of the input signals into the servo value output is performed. All software used was free and open source

2.2.4.2 Necessary Programs

1. **OpenNI SDK**- This software development kit is the open source kit made by Primesense, the makers of the chip that enables the Kinect depth mapping capabilities to work. Standing for "Open Natural Interface", this powerful SDK lets developers build gesture reactive programs that work with Primesense enabled depth sensors, like the Kinect.
2. **NiTE Middleware**- This middleware program interprets the Kinect depth data from OpenNI and returns any users and their joint data that it finds.
3. **OSCeleton**- This server binds to OpenNI's NiTE middleware, gets the user data from found players, and broadcasts it as a local server in the popular Open Sound Control (OSC) messages.

2.2.4.3 Libraries

The libraries are program modules built to enhance the functionality of a main code. There are several libraries implemented:

1. **OSCeleton** (pyOSCeleton)- this library enables the receiving of the Kinect player data OSC messages from the OSCeleton server. This player data contains a user id and all the player's joint XYZ values. It is this data that is parsed to work the inverse kinematics engine.
2. **sys**- this library enables access to system functions. It is used to cleanly exit python when the operator chooses to end the program.
3. **time**- this library is used to keep timing.

4. **numpy**- this mathematical library enables python to work on high level mathematical problems. In this program, it is used sporadically to assist with necessary elements such as such as arrays, absolute value, square roots, radian and degree conversions, and other mathematical requirements not usually available in Python.
5. **serial**- this library enables serial communication between the computer and the Arduino Mega and Uno.
6. **re**- the regular expressions library is used to parse the incoming Wii values string from its attached Arduino Uno.

2.2.4.4 Process

The main processing program is illustrated in Figure 2-4. With OSCeleton running, the program import the libraries listed in Section 2.2.4.3, initialized the global variables for the servos, which are the servo values, link lengths, frame counters and other important system values, initialized the serial communications, and started the OSCeleton server. Once this is done, the main while loop begins, which calls getJoints(). getjoints() waits for a message from the OSC server, which would contain player data. If no player data is received, it exits the function, but the while loop calls it again on the next iteration to check for messages. If player data is found, the player data, which includes the joint data, is added to the player class and moveArm() is called. moveArm() reads the waiting serial string from the Arduino Uno (detailed in Section 2.2.3.2) and parses and conditions the data using getWii() and its helper functions. The pared Wii command string includes the controlling variable for whether to actively process the inverse kinematics. If the condition is false, it simple returns the last values of all the servos, motor controls, and gripper states. If the condition is true, the program uses getKinect() to parse the player data for the necessary joint data and then start conditioning it for the inverse kinematics by getting the rotate angle, getting the humerus and ulna link lengths using trigonometry, getting the length and magnitude of the arm, position of the start and end effector, and then maps end

result values from Kinect space to robot arm space. The new mapped values are sent to IK(), where inverse kinematics finds the servo values and returns them. After a few checks, the values are sent to the Arduino Mega in the moveArd() function. The new servo values are saved to the global values using angles() and the program ends the loop and returns to the main while loop.

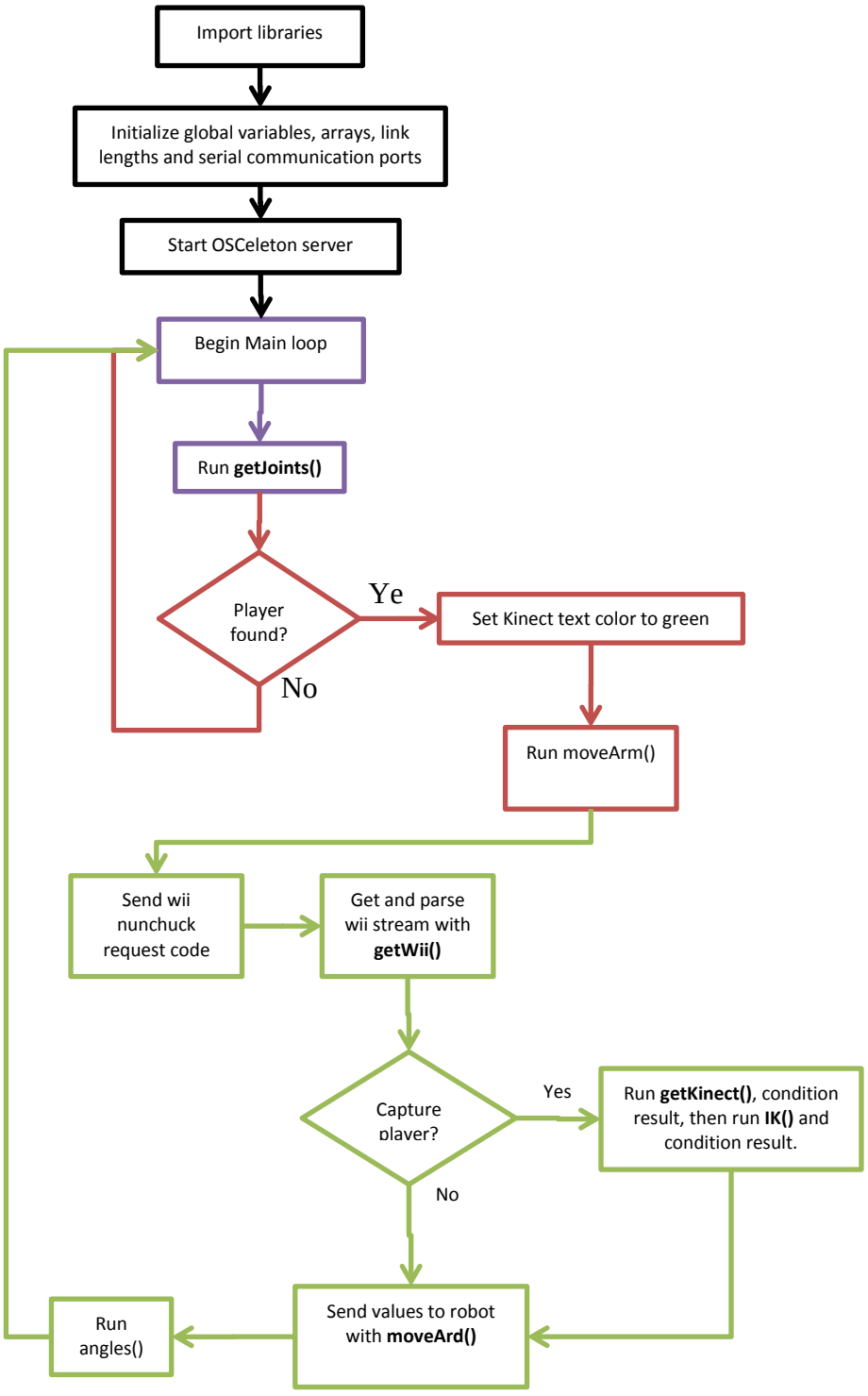


Figure 2-4: Flowchart of Control System Program

2.2.5 Output

2.2.5.1 Overview

The output of the control system is based around the Arduino Mega prototyping board. This microcontroller was selected due to its larger array of PWM and AnalogIn ports, multiple hardware UART serial ports, and larger memory size. The control system utilizes 2 serial ports (Roomba and computer communications), 6 servo ports for the 5DOF and gripper, a digital pin for the Roomba, and up to 6 AnalogIn ports for the future implementation of feedback. It was 3 high resolution times to drive all of these functions. It cost $65.

2.2.5.2 Hardware

2.2.5.3 Firmware

The firmware (Figure 2-5) for the Arduino Mega imports 2 libraries, the modified Roomba500.h, which enables communication with the Roomba robots over the SCI protocol, and Servo.h, which is a container for sending appropriate PWM signals for a given angle to the servo motor. There servo global variables, serial global variables, and the input array are initialized. The firmware enters setup(), where there servos are attached, pins are declared as inputs and outputs, serial communication parameters are set, and communication between the Roomba and the Arduino and the Arduino Mega and the computer are established. Once the setup is complete, the program enters the main loop, void loop(). The loop starts waiting for serial input. If no input is found, it simple instructs each servo and the Roomba motors to move to the last value of their respective globals. If there is Serial input, the program finds the start byte, which equals 255, and then parses the serial string and organizes the values. After conditioning the motors to be value appropriate for the Roomba, it sets the gripper servo based on the gripper state

and then sends all 6 servo values each servo. After a delay of 15ms, the Arduino checks the serial input again.

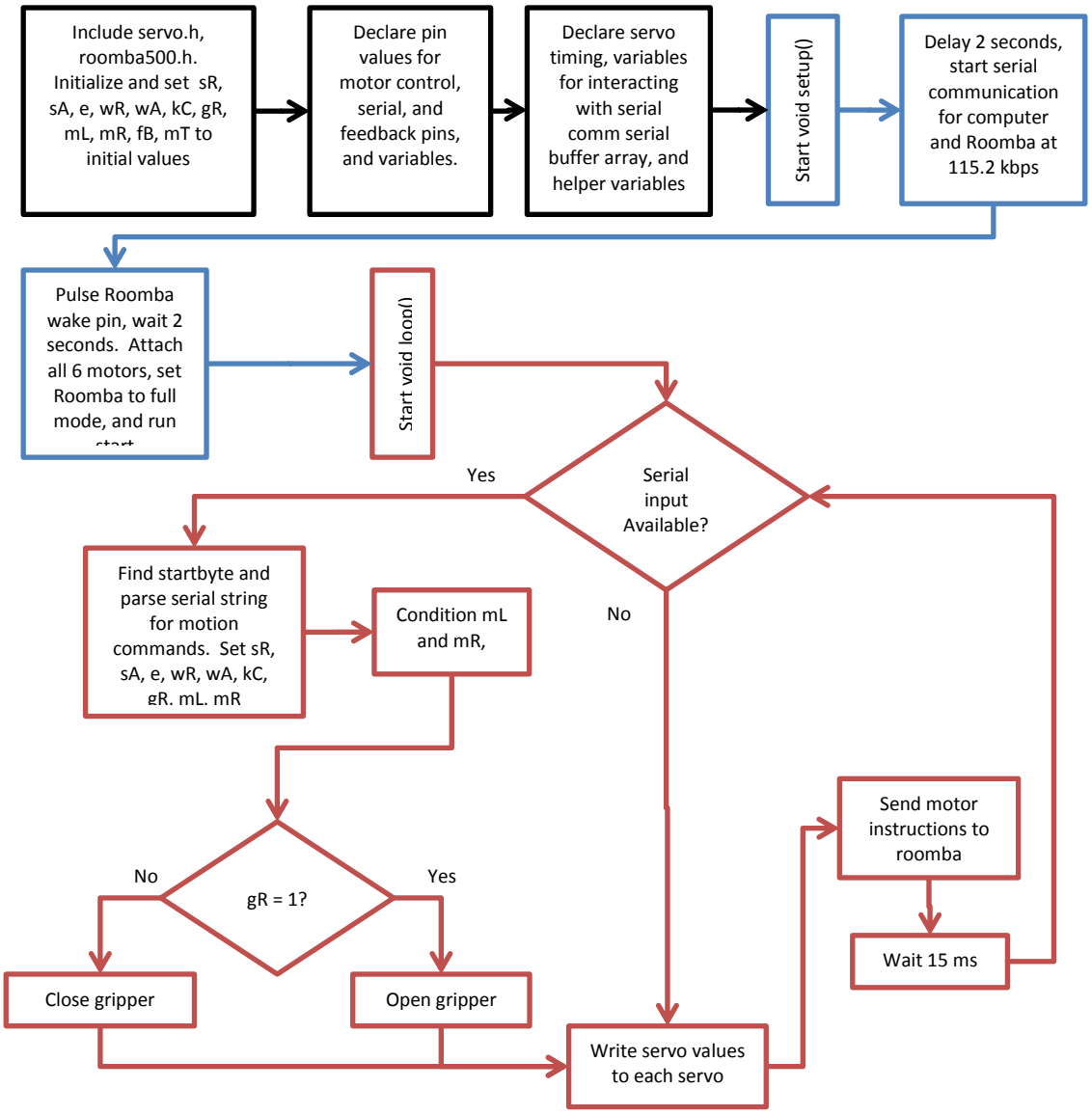


Figure 2-5: Flowchart for Arduino Mega Firmware (Robot Output)

.

Chapter 3

This Project

3.1 Introduction

3.1.1 Overview

The proof of concept system achieved the goal of showing the viability of such as system design, but was also successful in showing us the limitations of the hardware, software, and aspects of our implementation. There are several problems with the Using the proof of concept test system itself.

- The robot arm was incredibly jerky and the momentum of the arm caused disturbances. This quality was also exhibited in another servo-based robot arm using direct servo control, This problem is the primary focus of this research.
- The control concept, while demonstrating the ability to control a robot arm, was not tested for those sitting or in bed
- No remote viewing of robot arm, despite being a product specification
- No way of giving feedback to the operator or important system parameters.
- Was completely code based with no user interface.

3.1.2 Intended Improvements of the Control System

Using the selected elements of the final concept design, the control system created. The research will envelope the key areas that was not addressed in the proof of concept design. The research will include:

1. Comparing the OpenNI SDK versus the Microsoft's Official SDK
    a. Various compatibility and Performance checks
    b. Assessing Usefulness of Microsoft Kinect SDk's "Seated Mode" and "Near Mode".

2. Addressing bed's servo jerkiness issues
    a. Mitigating noise in for better control of the robot arm
    b. Changing motion profile of drive systems
3. A visual feedback implementation and possible GUI design for system ease of use
4. A more robust and coding free way of configuring the control system to the user

3.2 System Specifications

The design and test computer system to be used with the test bed platform is a Lenovo Thinkpad x200t.

3.2.1 Hardware Specifications

Lenovo Thinkpad x200t

- SL9400(1.86GHz)
- 8GB RAM
- 120GB OCZ Vertex 2 SSD/240GB PNY XLR8/750GB WD Black HDD
- 12.1in 1280x800 LCD
- Intel X4500HD
- Intel 802.11agn wireless
- Bluetooth 3.0
- Logitech C310 Webcam

3.2.2 Software Specifications

OS: Windows 8 (On OCZ and PNY SSDs)
SDK

- Microsoft Kinect SDK Modules
    - Visual Studio 2012 Ultimate
    - Kinect SDK 1.6/1.7

- OpenNI SDK +
  - OpenNI 2.1
  - Nite2.2
  - Delicode NI Mate (for OSC)

Other Software

- Arduino IDE 0023
- Arduino IDE 1.0

OS: Windows XP (On 750GB WD)
SDK

- OpenNI 1.5.1
- NiTE1.5
- OSCeleton (for OSC)

Other Software

- Arduino IDE 0023
- Arduino IDE 1.0
- VLC

Python Modules used with OpenNI x.x for each OS

- Python 2.7.3
- Numpy
- Pyserial2.6
- pyOSC
- OpenCV 2.2
- pyVLC

3.3 Comparing the OpenNI SDK versus the Microsoft's Official Kinect SDK

3.3.1 Overview

The Kinect is a notoriously noisy sensor and is by no means a precision instrument without some interpolation. Skeletal tracking values, depending on the tracking mode, are in millimeters or an arbitrary joint space value that is uniform about the domain. Experimental data has shown a random error range of a few millimeters to 4 cm at varying ranges for the depth data [2]. This causes joint positions to float round an approximation of where there joint actually is, so it rarely stays steady. It

also changes the length of the tracked arm by up to 10-cm, leading us to use the relative length of the arm for tracking instead of a standard length[1].

Due to this fact, it became imperative to compare the previously unused Official Microsoft Kinect SDK with the SDK currently being used. Each SDK has a different method of finding and tracking skeletons. If the method of the current system is inferior to the Official SDK's, then the main system should be ported that to that there. Otherwise, if the current system is superior, then no changes should be made.

Another important reason to explore using the official SDK is that it has the ability to perform no-calibration tracking of a seated person and has a specialized mode called "near mode". This would benefit the target demographic as they would normally be seated when using the control system and a better tracking system catered to that pose would be only positive. The near mode would enhance portability of the system, potentially allowing it to be attached to a wheel chair and be a mobile control system that the operator can take with them.

Finally, usability and versatility of the systems to cater to target goals and needs of the project will be evaluated, such as potential system integration, visual feedback, dynamic variable changing, and implementing a GUI.

Once the final system is decided upon, all further research will be based on using that system.

The Microsoft Kinect SDK will be analyzed first, followed by the OpenNI SDK

[1] In later experiments testing if it was more possible to use an average length, the arm was much more noisy due to unexpected shifts when the arm was outstretched facing directly the Kinect sensor.

3.3.2 Microsoft Kinect SDK

3.3.2.1 System Requirements

Kinect for Windows has the following system requirements:

- Windows 7, Windows 8, Windows Embedded Standard 7, or Windows Embedded POSReady 7.
- 32 bit (x86) or 64 bit (x64) processor
- Dual-core 2.66-GHz or faster processor
- Dedicated USB 2.0 bus
- 2 GB RAM

[3]

It is important to note that the processor requirement exceeds the specification of the test computer's processor. Despite this flaw, the test system kept up with the rigor of the tests due to it greatly exceeding the required RAM.

In order to program for the Kinect SDK, Visual Studio 2010 or higher is required. To use Seated Mode, a Kinect for Windows device must be purchased [4] and the Kinect SDK must be version 1.0 and higher. Similarly, to use Near Mode, a Kinect for Windows must be purchased, but to enable Skeletal Tracking, Kinect SDK version 1.5 and higher must be used [5].

3.3.2.2 Overview of Microsoft Kinect SDK Methodology

The Microsoft Kinect SDK methodology is one of gesture based computing. It uses a predictive body mapping algorithm based on stored images in a decision tree to find skeletons. It takes your skeleton and compares it to one of about 500,000 images of orientations, and assumes that this is the position and gesture that you are making [6]. The general method of mitigating the noise in NUI skeletal tracking

is having a confidence rating for each joint, jitter tuning, and sensitivity adjustments. The jitter parameter basically averages the samples of each joint's positions over time and keeps the joint position steady until the joint moves outside of the given jitter radius. The sensitivity parameter attenuates the Kinect's response to small changes in the skeletal positioning. A low sensitivity will require greater movements to occur before they are registered by the software. The computation requirements are higher than that of the OpenNI method.

The Kinect, using this predictive method, has the ability to take advantage of the recently released "Seated Mode" (Figure 3-2) and "Near mode" (Figure 3-1),

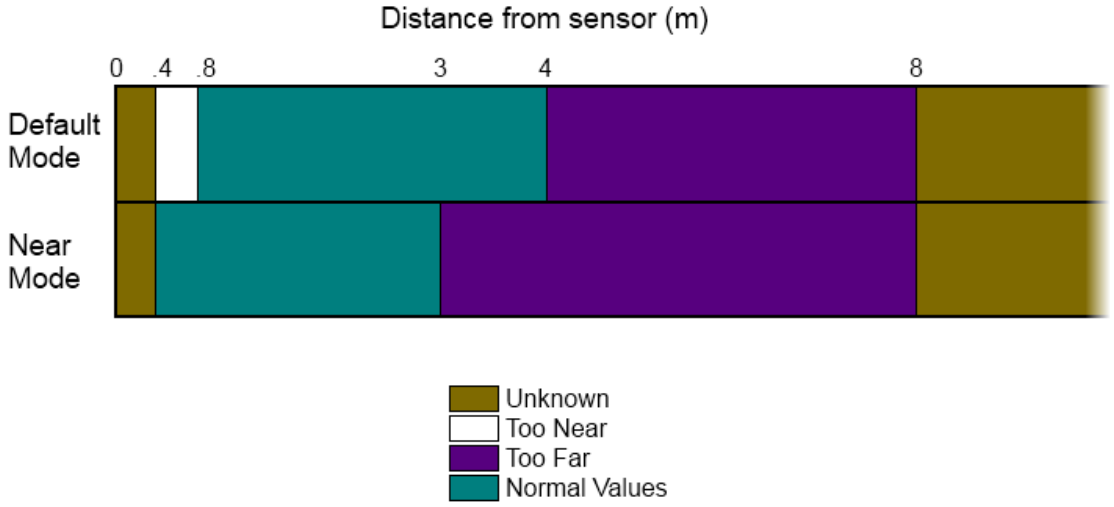


Figure 3-1: Near Mode versus Default Mode

as specialized mode with the Kinect can find your upper body even if you are in a chair, couch, or a wheelchair. This caters directly to the assistive living scope of the project and was the primary reason for testing out this SDK, as they operators, being unable to stand and properly move around, would likely be seated or sitting up, and reducing the distance

3.3.2.3 Seated Mode Tests

3.3.2.3.1 Overview

As most intended operators would be sitting, this Seated Mode feature was the reason of attraction to use the Microsoft Kinect SDK. Seated Mode option that was touted by Microsoft Labs allows those in wheel chairs to harness the power of the Kinect. If the claims are true, and this mode can be used well with the control system, it may be a viable option for starting to use the Kinect SDK for future developments, as OpenNI requires a standing calibration pose and has no "Seated Mode".

3.3.2.3.2 Testing Procedure

A small program was used to verify the usefulness of Microsoft's Current seated mode with the control system. For this test, Seated Mode was enabled programmatically and the capture mode was set to Default. The skeleton, superimposed on an image of the body, would visually show us any tracking errors or inaccuracy that may occur. Then, if the tests passed reasonably, the system would be tested using the robot arm.

The Seated Mode test parameters were as follows:

- Assess Ability to continuously track skeleton from 2 meters from tip of outstretched arm while in a chair
- View errors in capture

3.3.2.3.3 Parameters and Grading

The test motions were:

- Slow full arm motion from left to right
- Slow full arm motion from right to left
- Fast full arm motion from left to right

- Fast full arm motion from right to left
- Slow full arm motion from down to up
- Slow full arm motion from up to down
- Fast full arm motion from down to up
- Fast full arm motion from up to down
- Slow, medium, and fast speed “reach out and touch target”

The test’s grading parameters are as follows

- Good- the Kinect and the software must retain an accurate skeleton tracking of the arm throughout the motion, with allowances for one or two separate instances of accurate inferred points.
- Poor- the skeleton must retain reasonable accuracy throughout the test or have a couple frames where inferred points were used. This is considered poor as such a loss would adversely affect operation of the control system, but passible control could be attained.
- Fail- the skeleton must be inaccurate and/or an extended period of time of inferred points. This is considered a failure as the system cannot and should not be operated with such results.

3.3.2.3.4 Seated Mode Results

Motion Type	**Good/Poor/Fail**	**Comments**
Slow full arm motion from left to right	Poor	There was a loss of tracking when the arm was straight on, and across the body on,
Slow full arm motion from right to left	Poor	Did better than the previous test
Fast full arm motion from left to right	Good	

Fast full arm motion from right to left	Good	
Slow full arm motion from down to up	Fail	Loss of initial tracking from above the lap, but regained tracking once that are was raised above chest height.
Slow full arm motion from up to down	Poor	When the arm is outstretched, if the joints overlap, or are over the lap, the skeleton loses tracking for a few frames, them gets reestablished in better orientation
Fast full arm motion from down to up	Poor	Loss of initial tracking, but as regained after moving up to chest height.
Fast full arm motion from up to down	Good	
Slow speed "reach out and touch target"	Good	While a frame was erroneous, the tracking held well
Medium speed "reach out and touch target"	Poor	Loss of accurate tracking if joints overlapped
Fast speed "reach out and touch target"	Fail	Complete loss of tracking during action.

Table 3-1: Seated Mode Skeletal Tracking Test Results

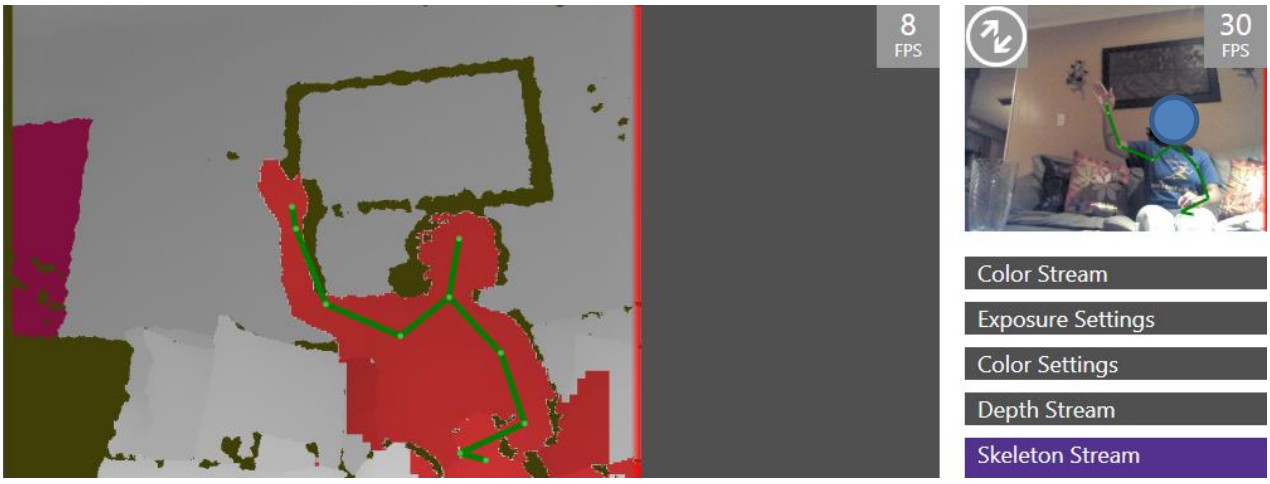


Figure 3-2: Seated Mode (Good Capture)

3.3.2.4 Near Mode +Seated Mode Tests

3.3.2.4.1 Overview

Wheel chair bound operators may not have a flat, raised surface to place the Kinect on when they intend to use it. An option is to mount the Kinect to the wheel chair and control it from there. However, this requires extremely close proximity to the Kinect. The usability of this control system would be increased by allowing operators to have this option.

3.3.2.4.2 Testing Procedure

Using the same program as in Section 3.3.2.3.2, Near Mode was enabled programmatically along with Seated Mode. Since Near Mode would be used only in Seated Mode for the target demographic, this was the only skeletal capture mode tested.

The Near Mode tests were done to determine the following:

- Assess ability to continuously track skeleton from about0.5 meters away from tip of outstretched arm, while in a chair
- View errors in capture

The test parameters were the same as in Section 3.3.2.3.3 .

3.3.2.4.3 Near Mode + Seated Mode Results

Motion Type	Good/Poor/Fail	Comments
Slow full arm motion from left to right	Fail	Failure to track across most of testing span
Slow full arm motion from right to left	Fail	Failure to track across most of testing span
Fast full arm motion from left to right	Poor	Only test that had better consistent tracking
Fast full arm motion from right to left	Fail	Failure to track across most of testing span
Slow full arm motion from down to up	Poor	Loss of tracking at a point below the chest
Slow full arm motion from up to down	Poor	Loss of tracking at a point below the chest
Fast full arm motion from down to up	Fail	Loss of tracking and inaccurate tracking throughout testing. Moe bad frames than good.
Fast full arm motion from up to down	Poor	Loss of tracking at a point below the

		chest
Slow speed "reach out and touch target"	Poor	It would fail to track at times during the motion, especially if reaching out
Medium speed "reach out and touch target"	Poor	Depending on starting orientation, it would fail to track at times during the motion, especially when reaching out
Fast speed "reach out and touch target"	Fail	Failure to track this fast motion

Table 3-2: Near Mode + Seated Mode Skeletal Tracking Test Results

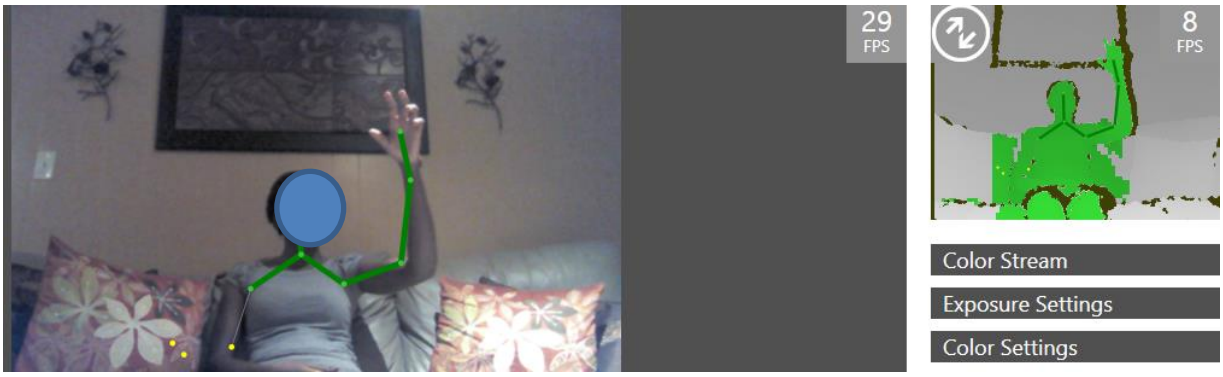


Figure 3-3: Good (Accurate Enough) Capture of Arm

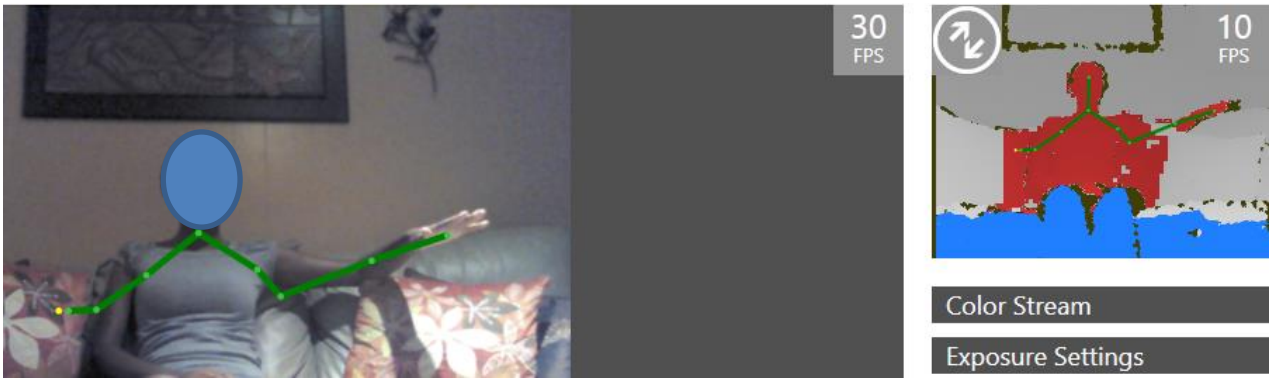


Figure 3-4: Close but Inaccurate Capture of Arm

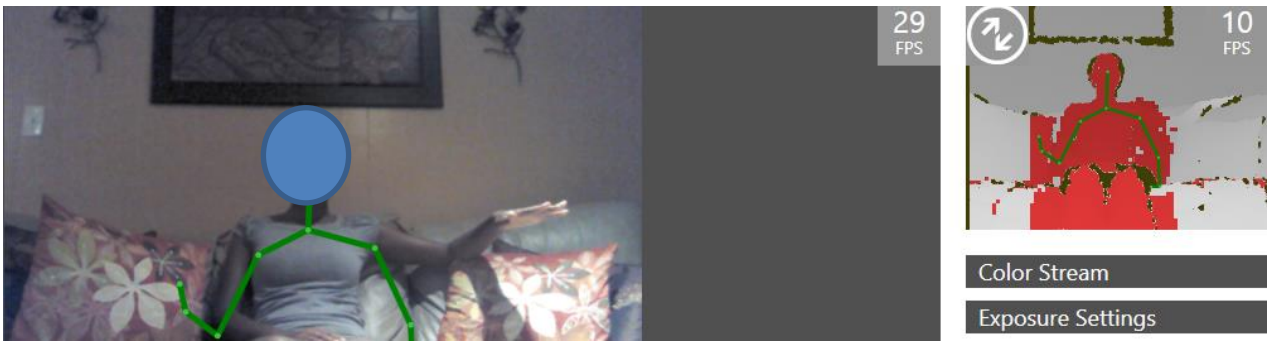


Figure 3-5: Loss of Accurate Capture during Left – Right Test

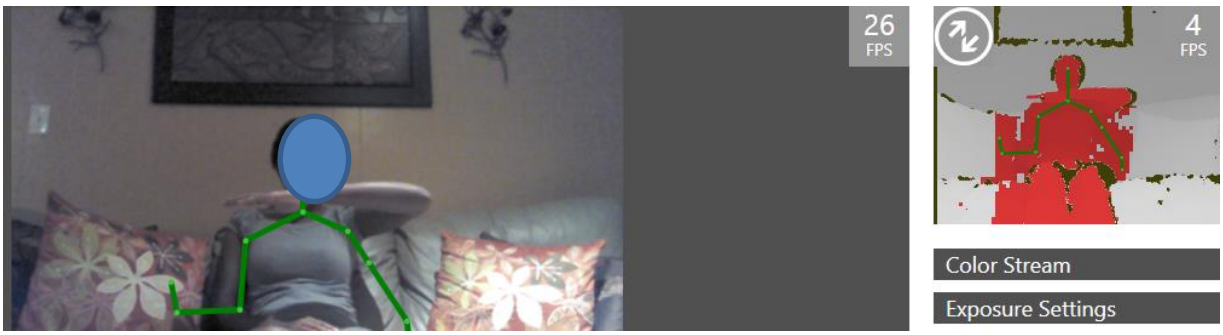


Figure 3-6: Completely Erroneous Capture of Arm Positions

3.3.2.5 Discussion of Overall Results

The Seated Mode tests performed decently well, as shown in Table 3-1, but better performance was expected. Perhaps the Kinect was too close or in less than ideal orientation, as there are blog reports of the Kinect tracking a wheelchair bound player quite accurately, such as. LeviHackWith.com's article, "Using the Kinect in a Wheel Chair", back in 2011. In truth, the operator was tracked quite well most times, like in Figure 3-2, and the only issues it had was if the arm was outstretched and directly facing the Kinect. However, this algorithmic problem exists even when standing, so it can be assumed that it was not a "mode" issue.

Near Mode + Seated Mode testing was pretty abysmal. It did not do well on any test, as shown in Table 3-2. The thought that is was simply operator orientation was removed when the depth map clearly shows that it

- Ascertained the depth values of the arm, but did not track it (Figure 3-6)
- Did not correctly ascertain the depth values of the arm (Figure 3-5) even though it was within a valid depth range, as showed in a subsequent tracking frame (Figure 3-4)
- Did not accurately track skeleton despite having ascertained the depth values of the arm (Figure 3-4)

Once again, orientation may have played a large part in the error, but even when using the tilt motor to adjust, the gain and loss of tracking was too random to ascertain where the "sweet spot" was. Also, if it has that narrow of a "sweet spot", then maybe it will not be right for an operator who would have to possibly get up and adjust it.

The use of the predictive algorithm allows for the operator to be in a more complex space and still retain a higher degree of positive readings in a complex environment. This is a wonderful positive for the Kinect SDK's methodology. However, it presently does not work well enough to be considered for a consistent control system. Using the Microsoft SDK, the arm joints were drawn overlaid on the RGB image from the Kinect, revealing that every so often, the skeleton became contorted momentarily. Apparently, for some low certainty joints, especially the elbow, the predictive algorithm shot out an erroneous assumed placement of the elbow joint. This usually happened when the arm is outstretched and facing directly to the Kinect sensor. While it is usually a momentary error, there are times that it can last until the subject moves into a more readable position. This is a potentially disastrous problem, as our code relies on the position of the shoulder, elbow, hand, and torso joints to be correctly extrapolated within a reasonable margin of error. Also, to use Seated Mode and the new Kinect SDK, a

specialized, more expensive device, the Kinect for Windows, is recommended to be purchased and the software can only be run in Windows 7 and 8. Reports that an Xbox 360 Kinect can be used are verified, but the operator cannot run it in a deployed, built executable. It can only run in the development stage with the source code in Kinect SDK. Also, Near Mode is not an available feature for an Xbox 360 Kinect. [7]

3.3.3 OpenNI SDK+NiTE 2.2 SDK

3.3.3.1 System Requirements

While official minimum system requirements have been posted, the NiTE middleware which works on top of OpenNI, touts the ability to work on x86 and ARM processors. Users have successfully had OpenNI+NiTE run on the Raspberry Pi, a 700Mhz ARM Cortex CPU embedded board. The official system requirements are as follows:

Supported Computer Hardware

- X86 based computers: Pentium 4, 1.4 GHz and above or AMD Athlon 64/FX 1 GHz and above

Supported Operating Systems

- Windows XP (32/64) with SP2 and above, Windows 7 (32/64)
- Ubuntu 12.04 (32/64) and above

Supported Development Environments

- Microsoft Visual Studio 2008 and 2010. The compiler can be an MSVC compiler or an Intel Compiler 11 and above
- GCC 4.x

[8]

NiTE 2.2 Middleware

- Multiplatform – Windows, Linux, Mac OS and Android
- Minimal CPU load, optimized for x86 and ARM architecture

[9]

3.3.3.2 Overview of OpenNI+NiTE SDK(with python bindings)Methodology

The OpenNI+NiTE SDK does not have built in routines for predictive body mapping or for jitter mitigation. Built upon the Point Cloud Library, it relies on the operator assuming a standardized calibration pose. Once calibrated, NiTE then just tracks the joint location in the depth space. This method has lower computational requirements. It also has an inferred joint position algorithm, based upon measurements taken in your calibration pose and the current orientation of your body.

This method is not very effective in a cluttered environment. If an appendage gets too close to an object, the software assumes that the object is part of your body, and will track it as such until you move away.

OSCeleton and the Kinect SDK don't really give much in the way of mitigating either of those tracking errors other than doing jitter, averaging, and sensitivity adjustments. It turns out, usually, when they occur, the game developers just reject those frames and assumes the gesture you intended rather than take based on the rest of your movements. In our case, we are attempting direct puppeteering control, so another method had to be devised.

3.3.3.3 Ease of Changing Variables

Algorithms are not an end all solution to proper tuning of this implementation. Operators of varying challenges may need to adjust and tune the parameters of the program to suit their particular abilities. Thus, this program requires a degree of adaptability in order cater to each assisted needs operator.

The Microsoft Kinect SDK using Visual Studio, with its tie in to the Visual C# WPF forms, allows for a very easy point-and-click design of the GUI and allows variables to be quickly assigned. There are placeholders and indicators for buttons, sliders, textboxes, labels, and methods for a more graphical way of displaying feedback and validation for the information input. These values can change dynamically.

Changing the variables programmatically seems to be the best and easiest way to work with OpenNI. Later on, there was exploration and implementation of a text based configuration file, which also worked for allowing the user to make changes at the start of the program, yet was difficult to implement for dynamic changes while the control system was running. Such a dynamic configuration system will be for left further work. The OpenNI SDK is open to any GUI that you want, however the Python GUI was found to be quite difficult to get working on the test machine. First, libraries for the GUI had to be installed along with a GUI designer to make things easier. wxPython and GUI2Py were chosen due to recommendations, but they were unable to be set up and configured properly with even moderate effort. A work around using OpenCV (discussed later) eventually was used.

3.3.3.4 Skeletal Tracking

3.3.3.4.1 Overview

Skeletal tracking with OpenNI requires the user to hold a calibration pose. After this, the skeleton tracking is really quite good. The data from the calibration pose can be saved and used later, thus requiring no further calibration poses (this function was not implemented in this research). Due to an absence of a marketed “Near Mode”, OpenNI was tested range tested with both a Nyko wide angle lens for the Microsoft Kinect and the regular Kinect lens to see which will allow an operator to get closest while still performing a high level of tracking quality. The Nyko Wide Angle lens is a 3rd party non-permanent attachment to the Kinect which is purported to widen the field of view of the Kinect to allow for playing in confined spaced.

3.3.3.4.1.1 Metrics

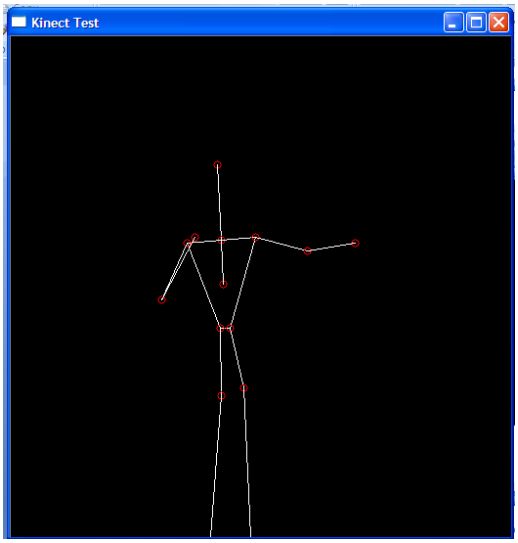
There are three metrics used to test each distance.

- Is there a skeleton. Without a skeleton, the control system will not work at all. This field is one of the most important in determining if a distance should be even considered. At 1m, it is the equivalent of “Near Mode” in the Microsoft Kinect SDK.
- Are the arms visible? If you cannot track the arms reliably, this control system will be rendered inoperable.
- Were there any errors or jitter in capture? These factors can adversely affect the control systems effectiveness, so placement and user distance must be such that it is mitigated or avoided.

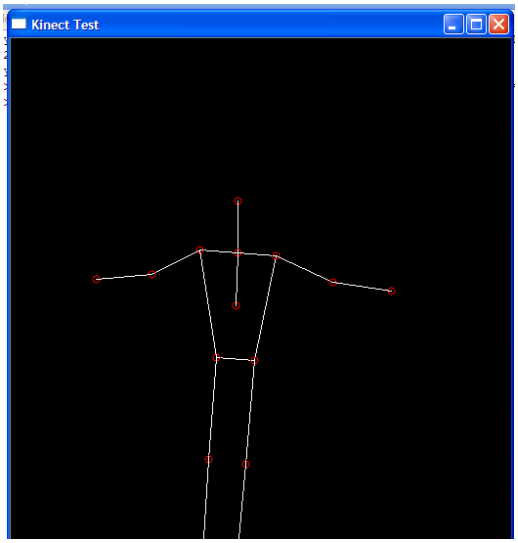
3.3.3.4.2 Procedure

The subject was first calibrated at 2, then moved into position and outstretched their arms. If a skeleton is produced, the Kinect output is observed for about 10 seconds for any aberrations, such as jitter, inferred joints, or loss of joint data. These are documented and then the operator moves to the next test point.

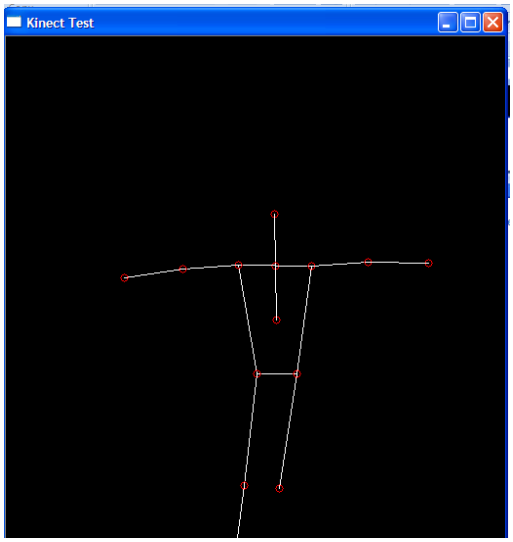
3.3.3.4.3 Results:



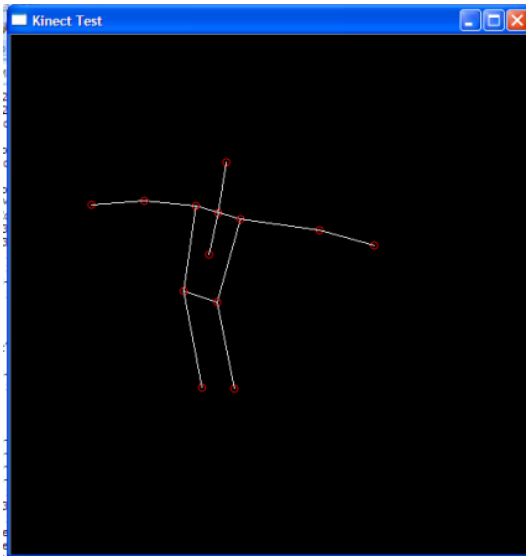
3-7: Open NI at 1 Meter



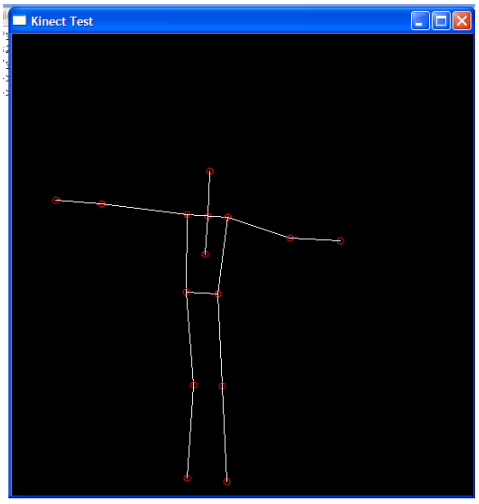
3-8: OpenNi at 2 Meters



3-9: Open NI at 3 Meters



3-10: OpenNI with Nyko Lens at 1 Meter



3-11: OpenNI with Nyko Lens at 2 Meters

Distance	Skeleton	Arms?	Errors
Normal Kinect			
1 Meter	Yes	Partial	Yes, missing limbs
2 Meter	Yes	Yes	No
3 Meter	Yes	Yes	No
Kinect with Nyko Lens			
1 Meter	Yes	Yes	Yes, Misshapen arms
2 Meter	Yes	Yes	More jittery than the Normal Kinect

3.3.3.4.4 Discussion

2 meters with the Normal Kinect seems to be the ideal distance and Kinect configuration to use with this control system.

At 1 meter, the equivalent distance of “Near Mode”, the calibration pose was difficult to capture. Moving back initially until the OpenNI tracked a good skeleton from the depth data was key to getting the calibration poser down. After the pose was captured, upon moving back up, some of the joints were out of the field of view. This low certainty joints were either inferred or discarded, such as the subject’s right forearm and waist. There were some errors, but for the most part it caught “okay”. It was surprising that it kept track of a skeleton that close to the device. For the Nyko Lens equipped Kinect, There was a marginally more solid capture of the body, but it produced joints which floated about the arm. The inability to confidently capture the arms as well as the floating nature of the joints with the Nyko Lens equipped Kinect makes this distance unacceptable for use with the control system.

At 2 meters, the capture of the outstretched arms was perfect. The FOV was wide enough to get the necessary joints for using this control system. The only issues came from improper positioning of the Kinect, which would yield bad results anyways. Once that problem was fixed, it was much better and consistent than the Microsoft SDK. With the Nyko Lens equipped Kinect, while it captured the skeleton, the floating joints were a problem again.

At 3 meters, capture was the same at 2M, but subtle motions were not captured due to the resolution of the depth sensor and the distance from the sensor. In this image the missing foot was due to an object in the way, not a capture error. The Nyko Lens was not used as its advertized benefits were that it kept your skeleton tracked while increasing the field of vow of the Kinect and decreasing your minimum range of usability the Kinect’s gesture based programs.

The OpenNI SDK has proven that it can track skeletons at a very close range as well. However the field of view of the Kinect has severely limited the useful close range to 2 meters from to sensor in order to get a full view. The Nyko Lens should not be used as part of this control system due to the joint tracking errors and the marginal improvements over the normal Kinect.

3.3.4 Final choice in SDK

3.3.4.1 Metrics

The following parameters will be used to compare the SDKs in this parametric matrix.

- Seated Mode
- Near Mode with skeleton
- Works with Robot
- Changing Variables
- GUI
- Control Parameters of Kinect Stream
- Native Speech control
- Skeletal Tracking
- Works in Windows XP
- Works in Windows Vista
- Works in Windows 7
- Works in Windows 8/8.1
- Works in OSX
- Works in Linux (Debian)

The elements of the parametric matrix will be graded as per the following:

- 1 Point - Yes, Good, or Easy
- ½ Point- No (with conditions), Okay, Moderate
- 0 Points- No, Poor, Hard

3.3.4.2 Parametric Matrix

Parameter	Kinect SDK	OpenNI
Seated Mode	Yes	Not very good, but upper body works
Near Mode with skeleton	Yes	No, but can be theoretically be programmed with some success
Works with Robot	Yes	Yes
Changing Variables	Easy	Hard, unless using openCV
GUI	Yes	No, but can be made
Control Parameters of Kinect Stream	Yes	Yes, but was not able to
Native Speech control	Yes	No, but can be implemented
Skeletal Tracking	Okay	Good
Works in Windows XP	No	Yes
Works in Windows Vista	No	Yes
Works in Windows 7	Yes	Yes
Works in Windows 8/8.1	Yes	Yes
Works in OSX	No	Yes
Works in Linux (Debian)	No	Yes
Totals	**9.5**	**10.5**

Table 3-3: Parametric Analysis of SDK Attributes

3.3.4.3 Discussion

At this point, it has become clear that the OpenNI SDK+python is superior for the needs of the system (Table 3-3). Despite the enhanced features such as seated mode, which works quite well in default mode, and the disappointing “Near mode”, for the criterion of consistent and accurate skeleton capture as the universally installable control system, the OpenNI skeleton implementation is much better for this continuously puppeteered robot. The Microsoft implementation is better for a gesture based controller, as was its main intention. The Speech recognition and tilt motor control capabilities make it a strong contender for a more closed commercial system or more consumer end product. With the Kinect 2 on the horizon [10] and the Asus Xtion Plus now including the Primesense Skeleton mapping capability and better depth resolution [11], different options and better capabilities for this implementation may be on the horizon such as much improved skeletal tracking and predictions. It is important to note that the OpenNI SDK code will work interchangeably with the Asus Xtion while the Microsoft Kinect SDK will only

work with that device and only with the Kinect for Windows. It has also been a repeated experience with Microsoft that there will be some required software upgrade to use the new hardware in order to use the Kinect 2.0, such as a required upgrade to Windows 8.1. The Kinect for Xbox or Windows only works with Windows 7 and up for the Microsoft Kinect SDK.

3.4 Mitigating Servo Jerkiness

3.4.1 Overview

The input side and the output side of the control system are the only places to see how to best reduce the noise and disturbances that affect the arm's operability. On the input side, smoothing schemes were devised, applied, and tested on the SDK code. On the output side, solutions were conceived, explored, and tested, a few with the help of William Thompson, a research assistant who worked in the lab for the Simons Fellowship and displayed some of our findings for his research requirements. Eventually it was settled upon a joint hardware/software solution that involved a new motion scheme for the firmware and altering the servo motors to generate feedback to the microcontroller.

3.4.2 Proof of Concept System

The proof of concept system was a bare bones approach to show that the concept was feasible. As shown in Figure 3-12, the concept used the real time values from the Kinect joint data and the gyroscopic and button data from the Wii Nunchuck, calculated by the inverse kinematics, and send the servo values to the Arduino Mega microcontroller, which sent those values to the servo motors directly. There was no feedback outside of the internal servos.

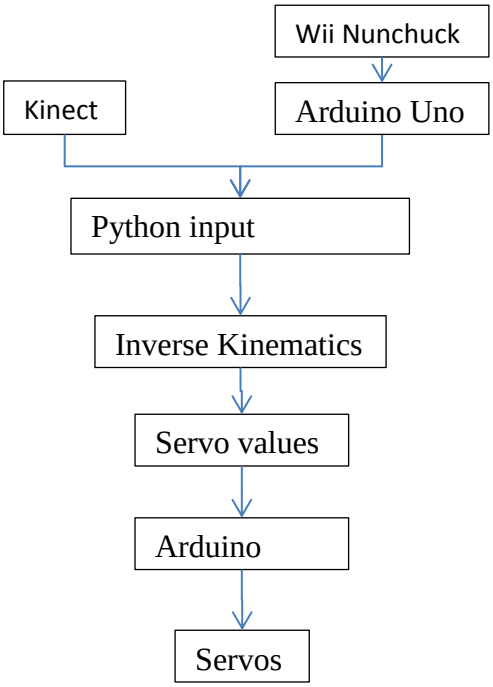


Figure 3-12: Input to Output Process Map

The resulting system's loosely controlled degrees of freedoms lead to the observed end effector sway and output error in Figure 3-13, which was measured by holding a ruler up to the gripper and viewing the displacement, yield an approximately 1.27 cm (0.5") of a 25.4 cm(10") arm, or 5%. The most offending joint was quickly discovered to be the elbow. .

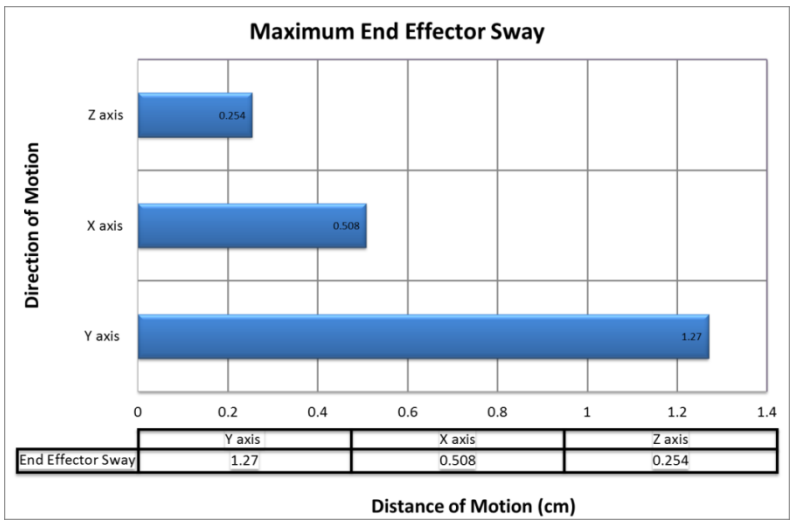


Figure 3-13: End Effector Sway of Proof of Concept [12]

Despite having a support spring, this joint had the greatest demands on it in terms of orientation changes, average dynamic moment, and average static moments. The resulting motion output was measured by attaching and calibrating a potentiometer to the arm and displaying the remapped values from 10bit analog in to degrees. These values were calculated and graphed as shown in Figure 3-14. The 180°/0 values in the beginning were capture errors and should be ignored.

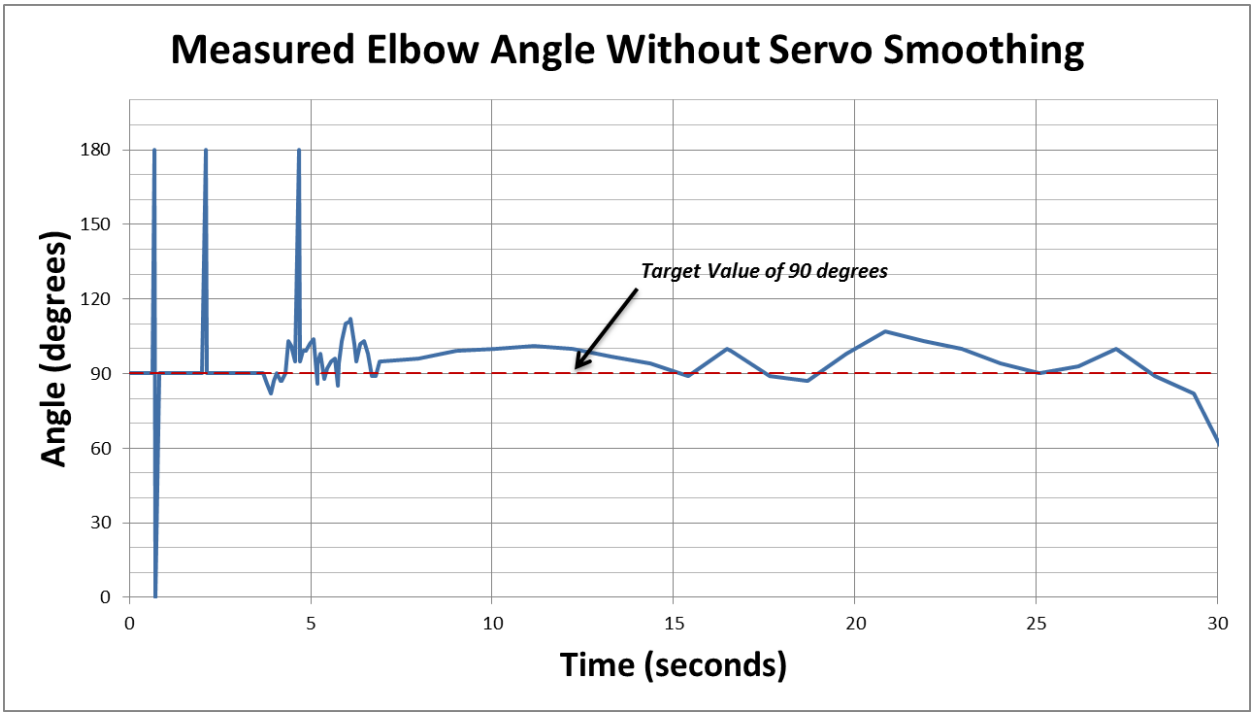


Figure 3-14: Measured Elbow Angle of Proof of Concept (used with permission) [12]

As it turns out, the servo is fighting the moment of the arm with its holding torque and PID implementation. The initial noise is from the PID settling while the remaining time is the holding torque fighting the moment. When the holding torque loses, there is a sharp correction by the PID. The angle is a bit exaggerated in the graph, but that is due to loose components in the arm and errors in the measurement process.

3.4.3 Input- Mitigating Kinect Noise

Since our control system relies on the absolute position of the joint and we do want it to have a near real-time, a solution is necessary that allows for the system to remain smooth while also responding with the lowest latency possible.

3.4.3.1 Explored Solutions

3.4.3.1.1 Overview

Four solutions were looked at.

1. Parameter (Jitter, Sensitivity, and Smoothing) tuning of Kinect Data
2. A value rejection based averaging or weighted average program of Servo calculations
3. Averaging of servos calculations
4. Weighted Averaging of servo calculations

Due to the differing methods of finding your skeleton, each solution must address either the strengths or failings of one or both SDKs. Jitter and Sensitivity plays on the strengths of internal parameters of the Microsoft SDK, though similar, but not as robust tuning is available on the OpenNI SDK. The Rejection Algorithm takes a step back and works on the servo data, rejecting sharp spike erroneous values and smoothing out the motion profile. The averaging of the servo data was initially tested as just a base calculation for reference and was done from periods 3 to a 7, while the weighted averaging was experimental, and was also done form periods 3 to 7. Of the tests, only 2, 3, and 4 were able to be done due to previous configuration issues, incomplete SDK bindings, and researcher capability boundaries. The best algorithm was found to be the 3 period weighted average and was used for the remainder the research.

3.4.3.1.2 Problems

Problems arose with proposed Test 1. First, the Microsoft SDK, which had the ability to work easily and directly with the Kinect data, was not universal and was tossed out. Second, OpenNI2's Python bindings were not completely implemented and the time of the thesis. OpenNI 1's python bindings were mostly complete, and even though the test was moved back to the original XP machine used for creating the Proof of Concept machine, they would not bind of the C. This was a configuration issue; however, in fear of the "solution" doing more harm than good. An averaging routine for the raw Kinect data that would work like smoothing was researched and explored, but varying the millimeters of change would not do as much as smoothing the final output signal, especially after the floats were converted into whole number integers. Therefore, it was theorized that maybe we could leave the raw Kinect XYZ data alone and work with the output servo values to prevent the random jumps. Test 1 was shelved.

3.4.3.1.3 Metric

As all the tests were done on the servo positions, the parameters will be uniform. The following are the three parameters used for determining the best test:

1. Response time- What is the lag from raw input action to the conditioned input reaction
2. Precision control- Ability to show small yet purposeful changes
3. Error- what is the max error and average overall error

3.4.3.1.4 Parameter tuning of Raw Kinect Data (For reference)

3.4.3.1.4.1 Introduction

The Microsoft Kinect SDK has a jitter parameter, where an algorithm basically averages the samples of the joint positions and keeps it steady until the joint moves outside of the given jitter radius. There is also another parameter in the Kinect NUI steam capture called Sensitivity, where the Kinect's response to small changes in the skeletal positioning can be attenuated. A low sensitivity will require greater

movements to occur before they are registered by the software. As stated before, the OpenNI's similar parameters, Smoothing and Sensitivity, were not able to be accessed programmatically using Python, the main code used for the system's SDK, and thus was not tested.

3.4.3.1.4.2 Implementation

This test uses the Microsoft Kinect SDK and allows for the capability to record a single motion capture and replay it. This presents the opportunity to do a truly scientific experiment, as the operator's motions and timings, and thus the raw data, are no longer variable.

A recording of an operator pretending to reach out, grab an object, and then move the arm from side to side will be made and saved as an .XED file in Kinect Studio[2]. Once the recording is done, the parameters of the Jitter, sensitivity, and smoothing control will be adjusted and the compensation will visible using the method in Figure 3-16. The default values, with all parameters set to 0, of the XYZ joint data of the shoulder, wrist, elbow, and torso will be recorded. Ellipses will be superimposed on the respective coordinates of position of the shoulder and hand joints to the RGB stream data to be used as a visual reference. Then, the values of the jitter, compensation, and sensitivity will be adjusted and the new XYZ data and recorded. After testing out the variables in a static form, towards the purposes of an easy to access GUI, slider bars to be put in place to allow the operator to dynamically readjust the jitter and sensitivity using the Visual C# WPF form. The output of this scheme would be as shown in Figure 3-15. Further algorithms were theorized, but not implemented.

[2] OpenNI SDK does apparently have this recording and playback feature, using NISample, but it was realized far too late in the research.

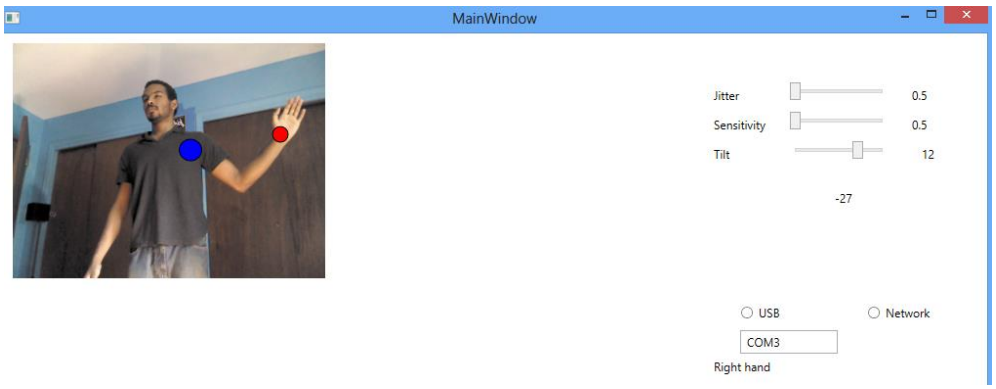


Figure 3-15: Possible Future Implementation

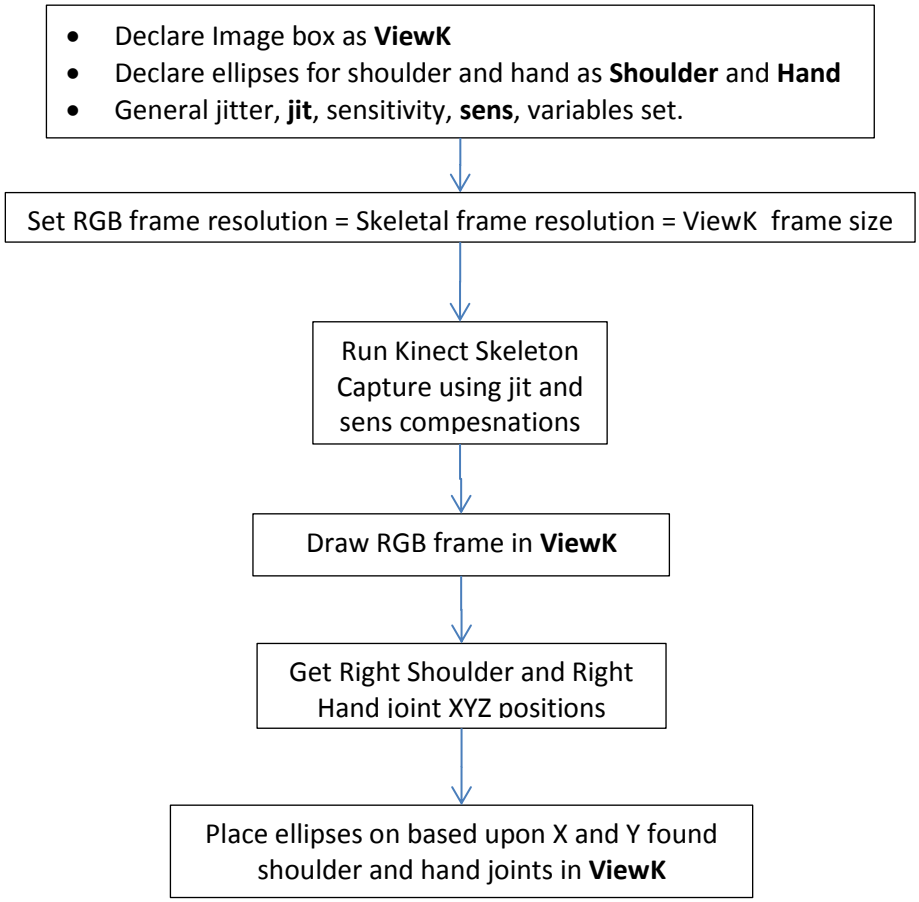


Figure 3-16: Visual Parameter Tuning Algorithm for Kinect Data Stream

3.4.3.1.4.3 Findings

In practice, a combination of the two systems will be useful, depending on the parameters given to the software by the user. The motion capture was much smoother and more fluid, and there was not play

in the registered position. However, as the jitter radius and sensitivity were increased, the latency between motion and output also increased. There was also the problem that small and gentle motions were not able to be perceived in these modes. Further algorithms, Figure 3-17, were conceived where the sensitivity and jitter radius were decreased if the operator hovered in an area for too long or upon a button press. However, as the Kinect SDK did not fit into the concept of a universally installable system, they remained simple conceived.

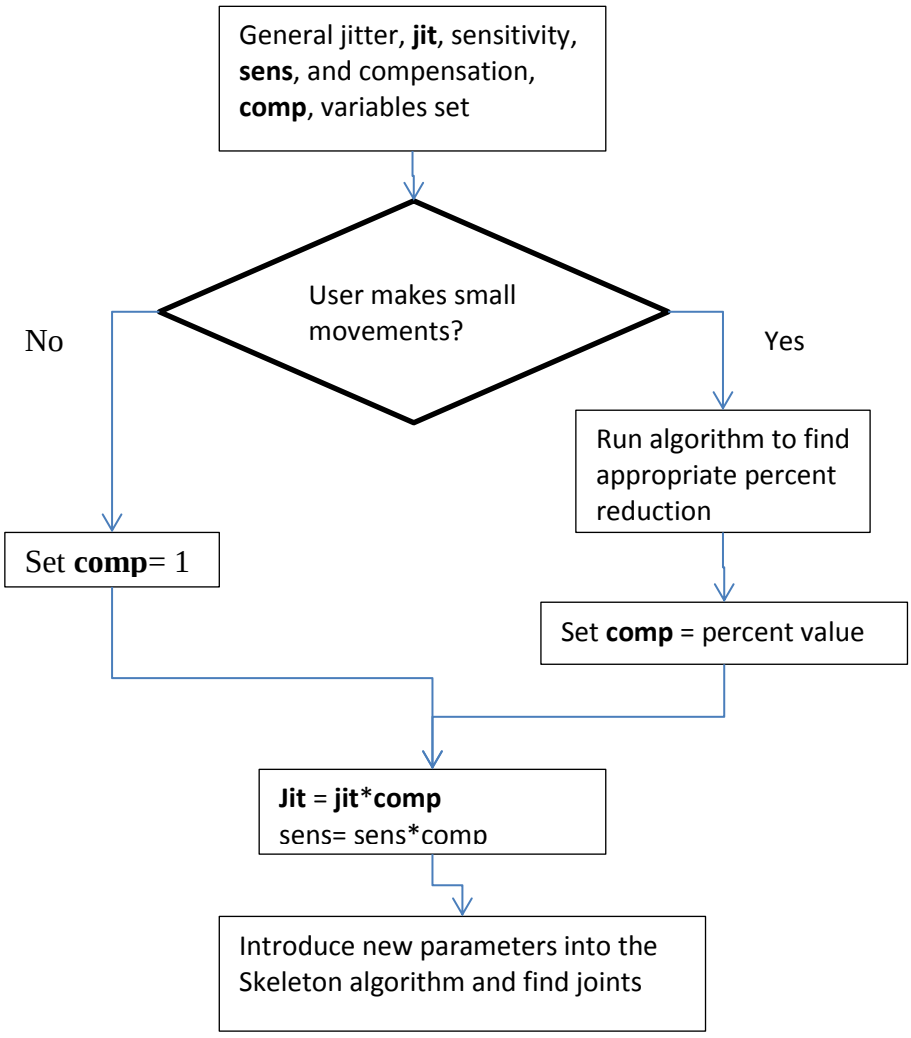


Figure 3-17: Conceptualized Enhanced Jitter, Sensitivity, and Smoothing Flowchart

3.4.3.1.5 Servo Value Smoothing Program Implementation

Each servo solution will be under a function called smoothArd. This will run in the moveArm() function and will execute right after the inverse kinematics algorithm and the sign checking subroutines. The current and last 6 previous servo output values, for a total of 7, will be stored in an array for each servo.

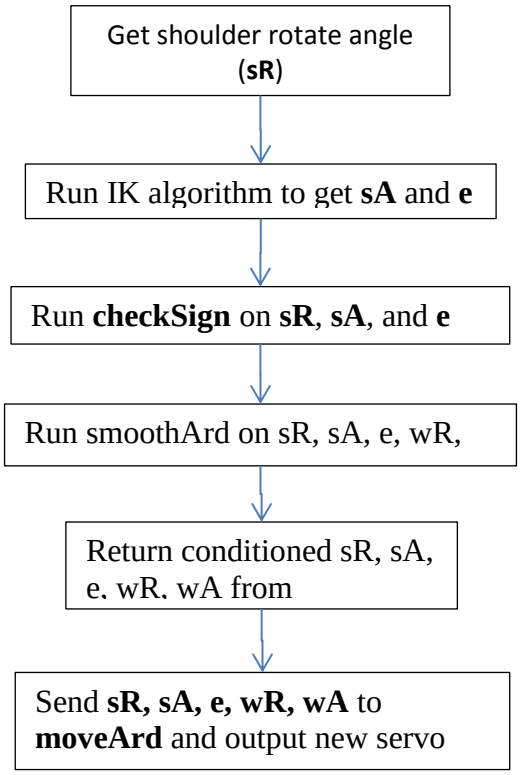


Figure 3-18: Smoothing function placement in moveArm()

The program will run the recorded out and play it out as if it was streaming live kinect data. The values of both the raw and the conditioned servo values will be stored in a CSV file for later graphing and analysis in Microsoft Excel.

3.4.3.1.6 Moving Average

3.4.3.1.6.1 Introduction

The standard way of smoothing a set of data is taking a moving average. In a previous depth-based research project with the Kinect at the TLT Media Labs, taking an average of the depth matrix values of

each point over a defined number of frames assisted in solving some of the noise problem. Thus, it is hoped that a simple averaging algorithm taking the mean of the last few historical angle values of each servo would give a smoother output for the Arduino Mega.

$$joint_{i_{ave}} = \frac{\sum_{n=0}^{j} pos[n]}{j}$$

Equation 1:Average

3.4.3.1.6.2 Data

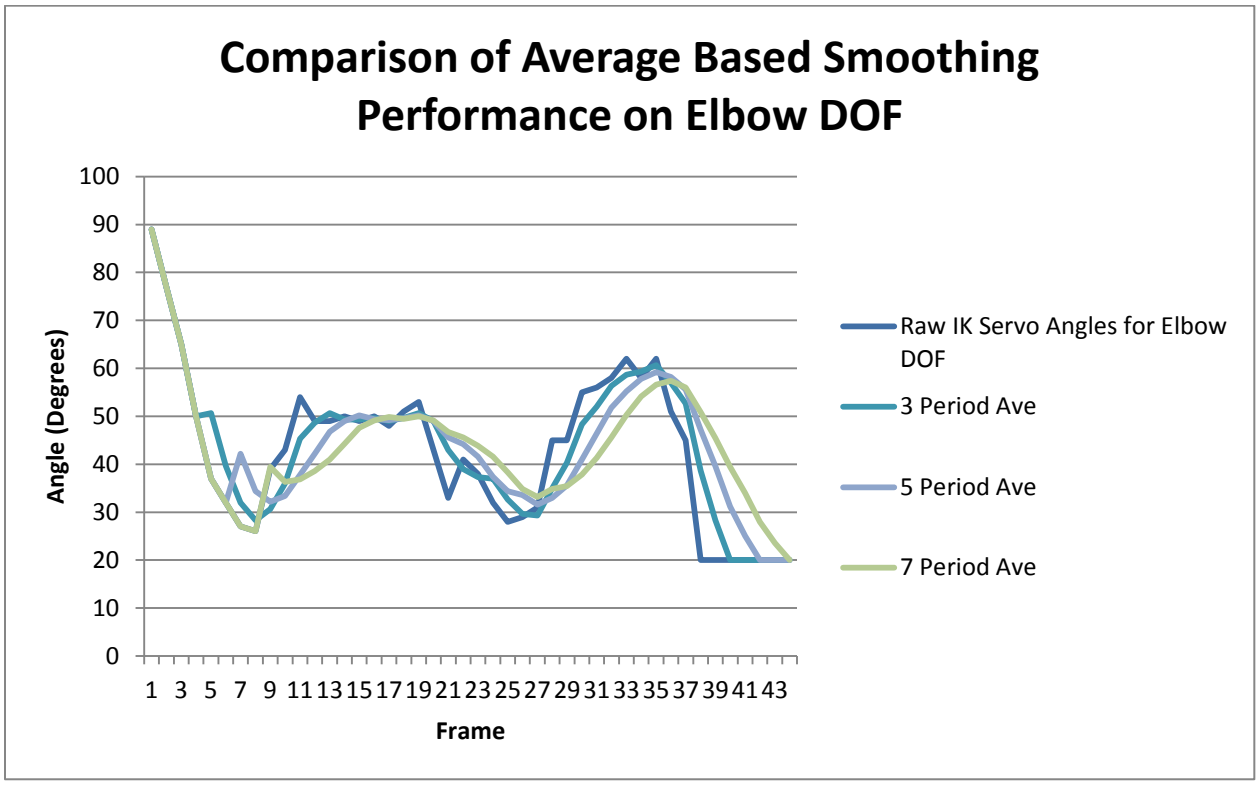


Figure 3-19: Comparison of Average Based Smoothing Performance on Elbow DOF

3.4.3.1.7 Moving Weighted Average

3.4.3.1.7.1 Introduction

While the simple average smoothed out the servo output signal, the response time started off at 3 frames and only got worse. The next algorithm to try was the weighted average, which was made more

flexible for programming with the solved Equation 2, which allows any size array of values to be evaluated. The weighted averaging scheme puts a stronger emphasis on the newest values, giving it a better response time. Imagining that there is an variable s_j in the numerator, instead of putting the same value into to s_j, as with simple averaging, the value of s_j increases the more recent the value is in the array. The denominator is the sum of the index numbers in the array.

$$\sum_{j=1}^{n} \frac{j}{\sum_{i=1}^{n} i} \times s_j$$

Equation 2: Weighted Average for any Number of Periods

3.4.3.1.7.2 Data

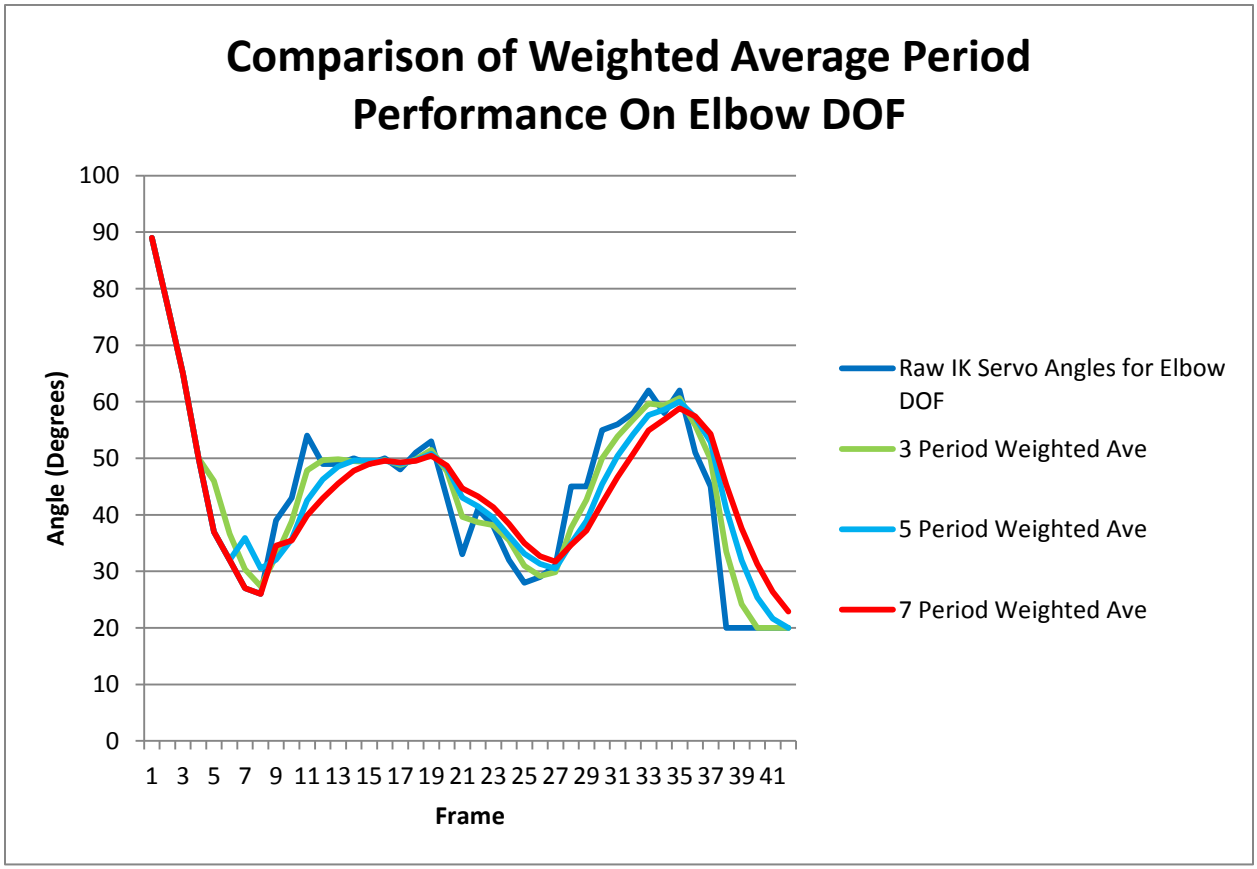


Figure 3-20: Comparison of Weighted Average Period Performance On Elbow DOF

3.4.3.1.8 Findings

It is important to note that the initial “bump” for each plot of an average on each graph is where the averaging actually starts. Previous to that point, it is simply copying the values form the raw data until the array of values is filled enough to complete the averaging period. When it starts averaging, the value will rise to the first true average, and then keep going from there.

The best scheme by far was the 3 period weighted average smoothing method. It had a response time of 1 frame in most cases and followed the trend of the servo data rather than the noise. It is precise enough to computer some nuances in the control, like the bobble between frames 10 and 20. In terms of error, the performance only misses the peaks of the raw servo motions and there is the 1 frame discrepancy. While the 4 (not pictured) and 5 period weighted average performed well, and had a smoother curve to it, the 2-3 frame delay that it sometimes had at too much of a cost in latency.

The regular averaging performed as expected. However for each incremented period, there is an increased delay in response time and was very imprecise. Thus, the error was also quite large. This smoothing method is not suitable at all.

In terms of response time, precision, and error, the weighted average is still, by far, the best performer, even on the smoothing. In future implementations, in a more mature release candidate, the weighted average scheme should be used. Compared to the 5- period average, the same period the rejection algorithm uses, it fares a bit better. There is a slightly better response time on gradual changes. However, it does not handle peaks very well, intended or noise based, and there is noticeable stepping. For a view into a future averaging technique it is interesting, but it is not ready to be an employed smoothing method.

3.4.3.1.9 Method of Choice

The chosen smoothing algorithm is the 3 period weighted average. It will be implemented in the Kinect SDK Code after the IK servo values are calculated.

3.4.4 Output- Servo Jerkiness and System feedback

3.4.4.1 Overview of the Issue and Solution

The mixed use of analog and digital servo motors presented an unusual issue where each servo had its own PID controller. These independent PID controllers create the following problems:

1. There is a high injection of disturbances into the test platform arm as the servos are programmed to move to the specified positions as fast as possible
2. There is a large visual inconsistency due to the disturbances between the natural motion of the person and the actual motion of the arm, which can throw an operator off.

A joint hardware and software solution was devised to solve this problem for both the PID and easing solutions. First, the servos each had to be modified in order to gain access, then the microcontroller to be wired to appropriated accept the inputs, then finally the method how the microcontroller's firmware processes the servo commands had to be completely revamped. This solution will be applicable for both DC and stepper motors.

3.4.4.2 Getting Hardware Feedback

The theory of operation of the servo is that the servo is a geared down DC motor that uses a potentiometer as a feedback sensor, measuring its current angle, for the PID loop the internal controller goes through. The hardware fix was to tap into the potentiometer to get the voltage of the potentiometer, and get the current angle from that using one of the Arduino Mega's Analog Input Ports, as shown in Figure 3-21.

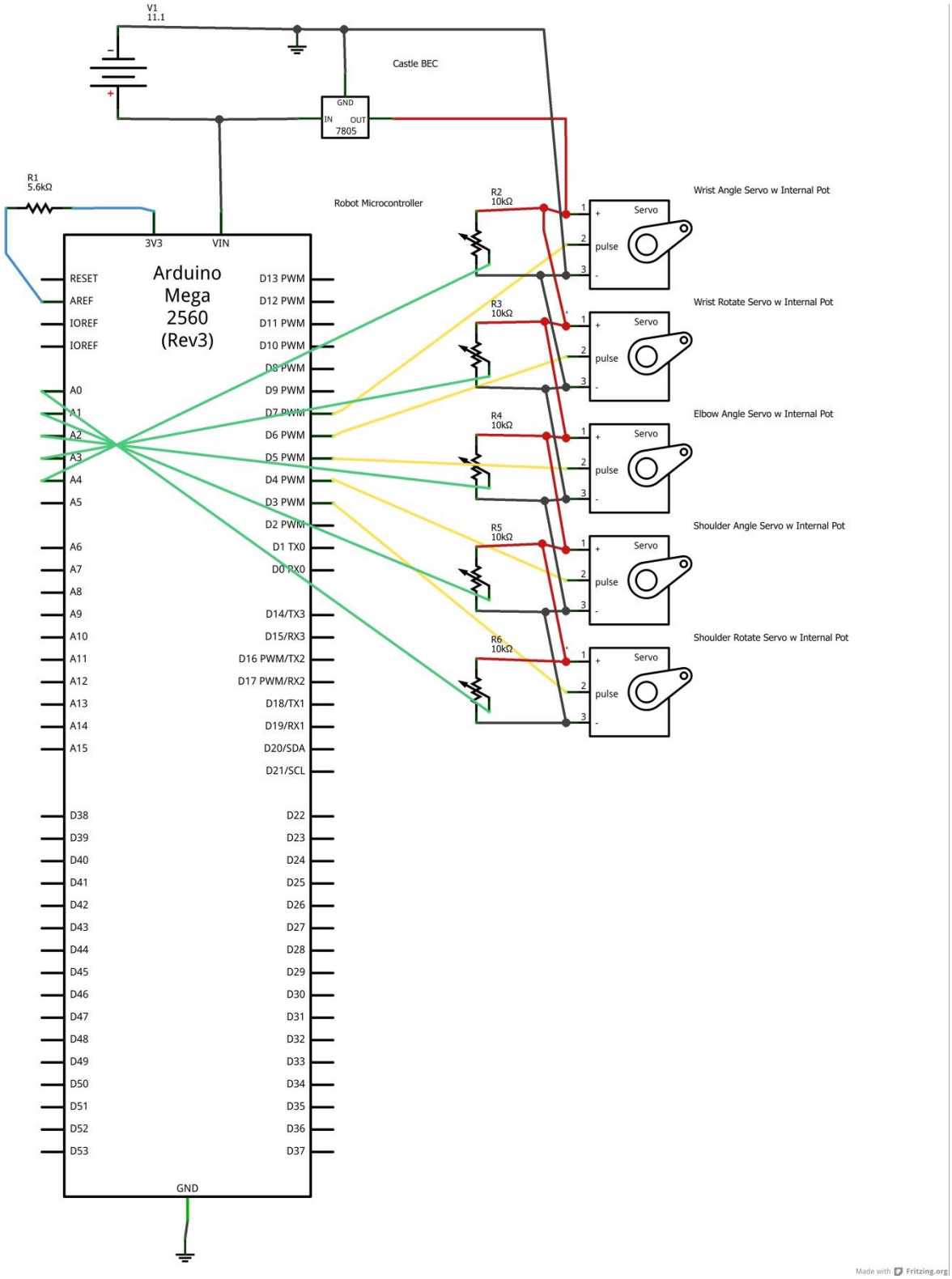


Figure 3-21: Arduino Mega With Servo Feedback

Servo	Min V	Max V
Shoulder Rotate	.31	1.88
Shoulder angle	.57	2.73
Elbow	.45	1.71

Table 3-4: Measured Voltages

The voltages of each servo's potentiometers were measured and are shown in Table 3-4. The Arduino Mega, with 16x 10 bit analog inputs (1024 divisions) would see a working span of 321-442 divisions for the range of 0V-5V. This gives a better than 180 point resolution that is being employed by the TX/RX command protocol, though it can be made better. This required the use of the analog reference voltage (AREF) pin in order to lower the reference voltage for the 1028 ADC. By using 3.3V and calculating the voltage drop across the internal 32 kΩ pullup resistor using the equation, a 5.6 kΩ

$$V_{in} \times \frac{R_1}{(R_1 + R_2)} = V_{AREF}$$

Equation 3: Voltage Divider Equation

$$3.3V \times \frac{32\ k\Omega}{(32\ k\Omega\ +\ 5.6\ k\Omega)} = 2.8085V$$

Equation 4: AREF Voltage Calculations

This new reference voltage increases the number of divisions in the working range to 572-787.

Servo	Min V	Max V	Min ADC Value	Max ADC Value	Span
Shoulder Rotate	0.31	1.88	113.028307	685.462	572.4337
Shoulder angle	0.57	2.73	207.826242	995.3783	787.5521

Elbow	0.45	1.71	164.073349	623.4787	459.4054

Table 3-5: Servo Voltage and ADC Values

With the implementation of the hardware feedback, the input/output flow map changes to include

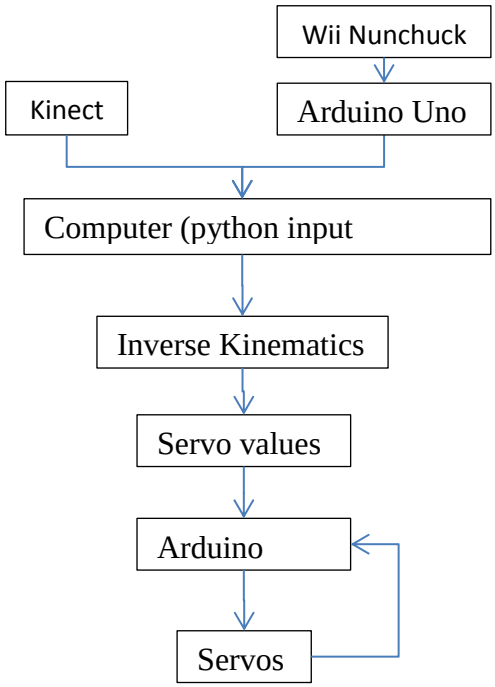


Figure 3-22: New Data Flow Map

3.4.4.3 Cascading PID

3.4.4.3.1 Overview

Proportional-Integral-Derivative algorithms work by using a closed loop system. They are regarded for their response time, ability to handle disturbances and noise, and. Early in the research, a cascading PID method, where the system post processes the servo values with a PID algorithm and then sends the processed servo values to the servos to be handled by its internal PID, as shown in Figure 3-23, was implemented for the servo and DC motors while working with William Thompson.

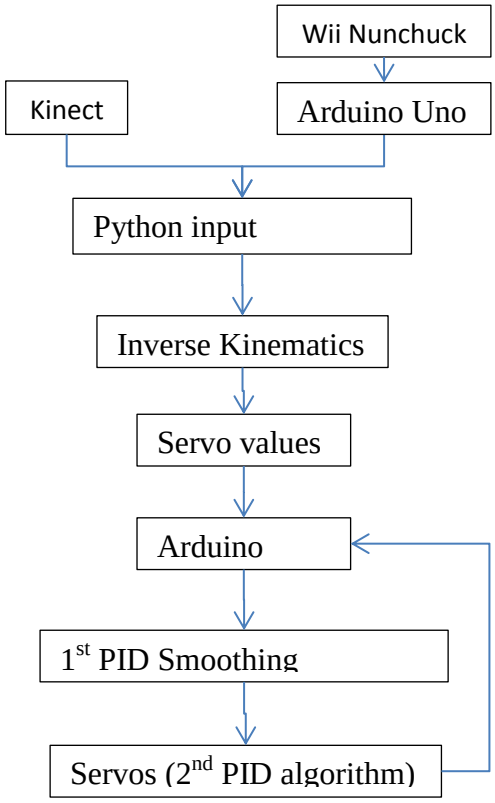


Figure 3-23: Simple 2-Stage Cascading PID Algorithm Flowchart

While it did mitigate some noise, in testing, it shed light to the root of the problem. It was found that the motion of the servos themselves introduced the most noise into the hardware system. This was due to the sudden initial jerk that the PID does and then settles easily into the setpoint. The PID algorithm was then deemed to be the inappropriate choice for a smooth and stable motion. The servo's PID were the root o some of the robot arm's instability. A step back was taken and looked at how an organic being moves versus how a robot moves. After further research into the concept of servo "easing", it seemed the viable way to go.

3.4.4.4 Easing

3.4.4.4.1 Overview

Research came to the conclusion that an "in-out easing" algorithm would be suitable to solve the problem, as it gently accelerates the motion, gets up to speed, and then gradually decelerates the

motion, based on delta between current position and end point. These equations are based on the open source Easing library for Ardiuno created by Tobias Toft of tobiastoft.dk, which are based on Robert Penner's animation easing algorithms for Adobe Flash motions. The basic equations were then changed from the static, serial, single servo code into a dynamic, parallel, multiple servo code. Simulators had to be designed to properly test, troubleshoot, and tune the easing algorithm's effectiveness before finally implementing it on the robotic arm test platform.

3.4.4.4.2 Implementation

3.4.4.4.2.1 Motion Profile and Setup

The effectiveness of four of the easing methods was tested: cubic, quadratic, quintic, and quartic.

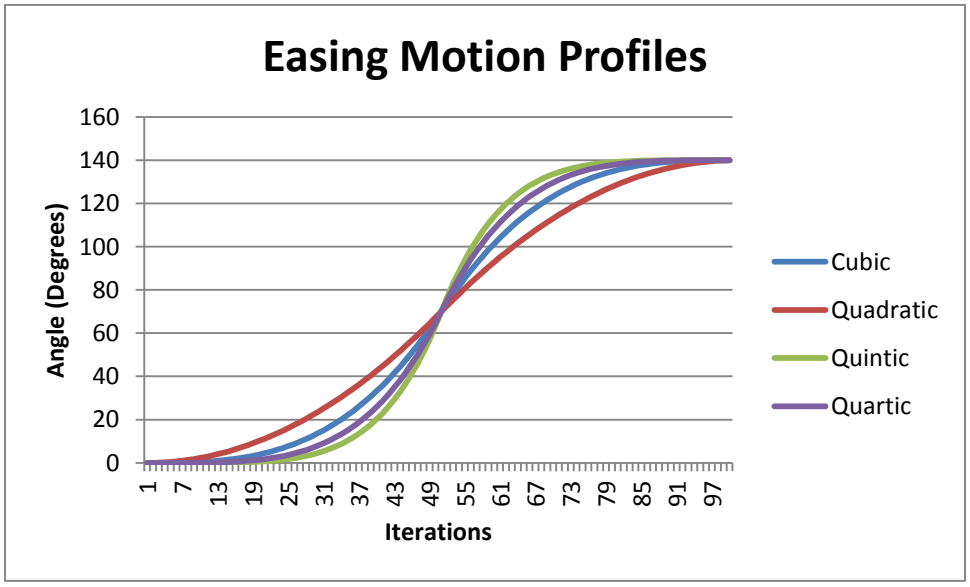


Figure 3-24: Easing Motion Profiles

Real world testing showed that quadratic easing gave the best motion profile for low disturbances, due to its shallower departure and return to stop. Quadratic In and Out (quadIO) Easing's C function is as follows,

```
float quadIO (float t, float b, float c, float d)
```

```
{
        if ((t/=d/2) < 1) return c/2*t*t + b;
        return -c/2 * ((--t)*(t-2) - 1) + b;
}
```

Quadratic's Out (quadO) Easing's C function is as follows

```
float quadO (float t, float b, float c, float d) {
        return -c *(t/=d)*(t-2) + b;
}
```

The new data flow map in Figure 3-25 shows the easing code right before the servo's internal PID implementation as the easing code will already have accommodations for it.

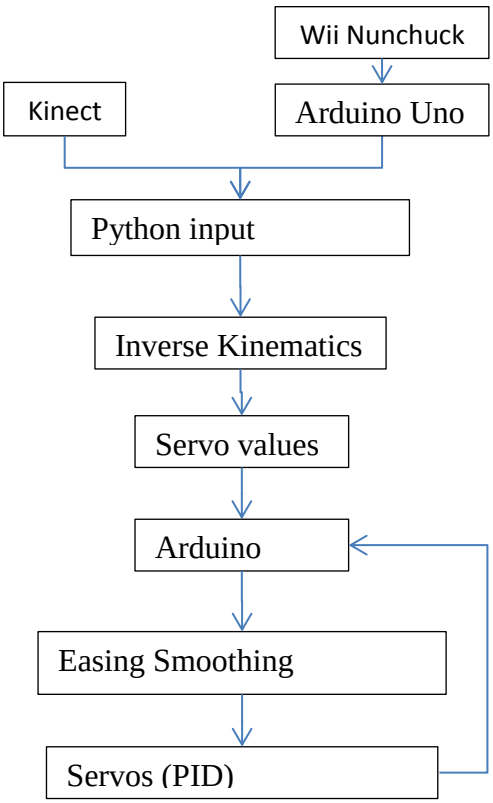


Figure 3-25: Easing Algorithm Data Flow Map

3.4.4.4.2.2 Microcontroller Code

First, the code enabling AREF was added in set up. It was enabled in a way that would allow for the user to choose whether or not to use the AREF without having to make any software changes. In order to

make a dynamic, multi-servo implementation, the iterative for loop had to be wildly revamped to be parallelized and work off a decision tree.

A 3 by 7 element array that contained each servo, Table 3-6, and the necessary variables, Table 3-7, was created for computing the easing schema.

Array Element	Joint
0	Shoulder Rotate
1	Shoulder Angle
2	Elbow

Table 3-6: Joint Array

Sub Array Element	Variable	Variable Name
0	Setpoint	goal
1	start position	start
2	difference between start position and setpoint	Δ
3	duration	dur
4	Calculated current position (last output)	cur ∢
5	Elapsed position (last output or feedback)	epos or cur ∢
6	Elapsed duration	pos
7	Moving Flag	State
8	New Angle Holder	new ∢

Table 3-7: Joint Variable sub-array

A decision tree algorithm, Figure 3-26, was devised after testing that handled the cases of moving and direction changes, moving and setpoint delta increases, moving and setpoint delta decreases, and not moving and new setpoint received. Once the decision tree is solved, the results are **quadIO**, **quadO**, and "do nothing", which all output the latest calculated servo value, **cur ∢.** Variable **pos** is updated by incrementing by 1 at the end of each active motion loop and resets to 0.

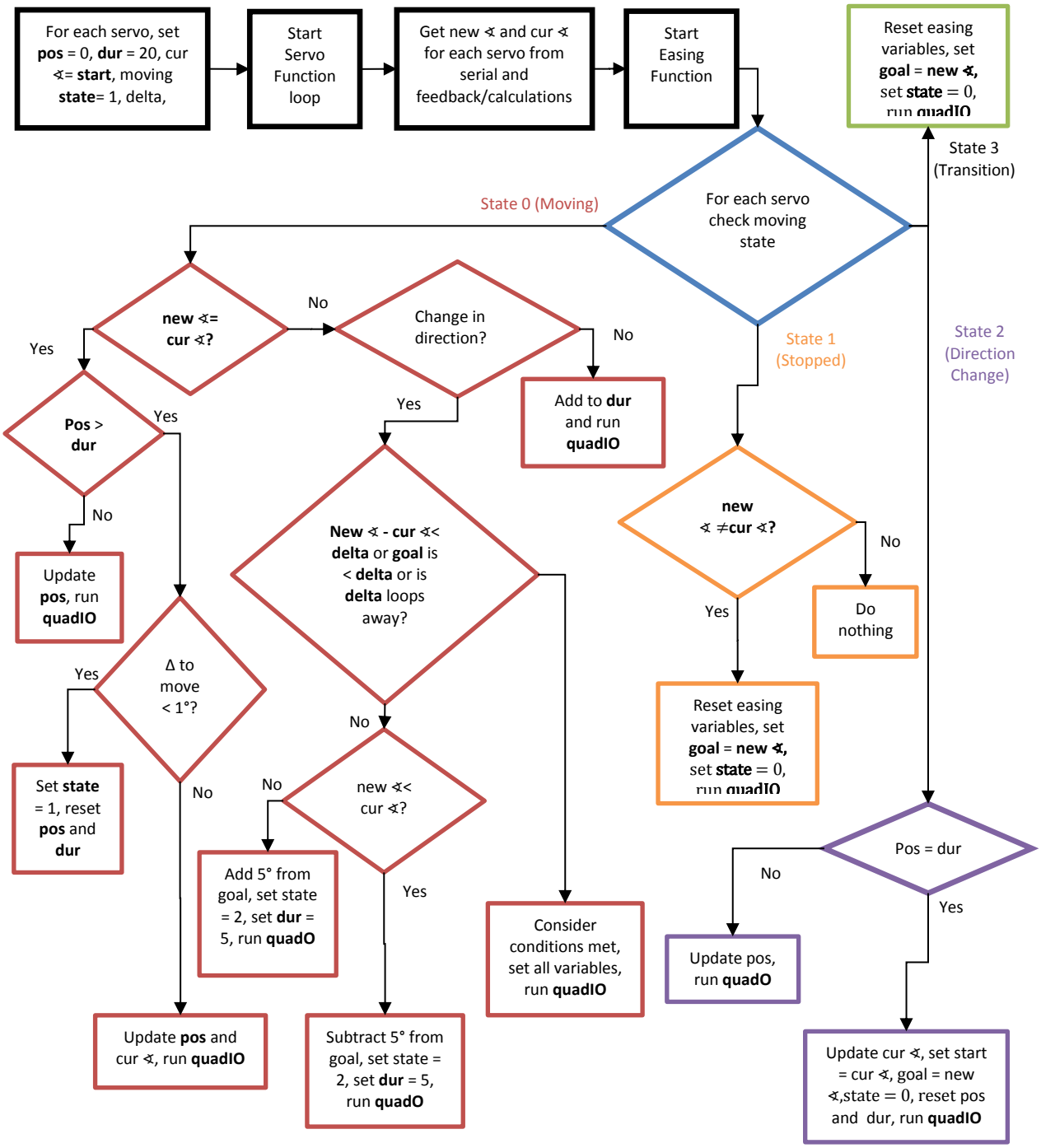


Figure 3-26: Easing Decision Tree Algorithm

3.4.4.4.3 Findings

After refining the decision tree, the final code in Figure 3-26, when simulated, created a more attractive, smoother motion profile than the no smoothing (servo PID) or Cascading PID solutions while still being close to the servo PID profile.

Figure 3-27: Robot Arm in Simulator

Testing between the raw and 3 value weighted moving average, it performed best when used in conjunction with a smoothed Kinect servo output. When used together, though, the error between the in the user input and the motion output does grow, as the latency builds, but only by a few frames due to the delay caused by the smoothing (Figure 3-28). While the average error of the motion was 1.22%, the maximum error spiked at 17% (Figure 3-29). This was due to a rapid motion in all 3 axis, each which had about 6% error at the time, and was only momentary.

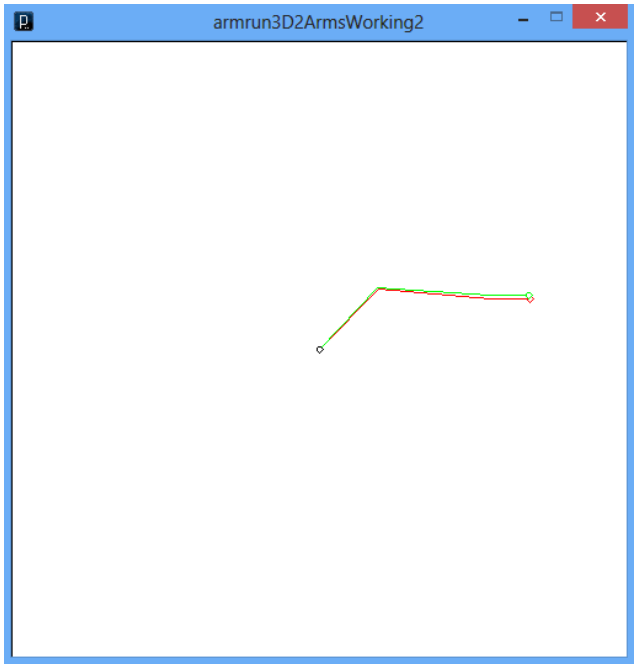


Figure 3-28: Motion Error Small but Apparent in Raw Versus Smoothed with Easing

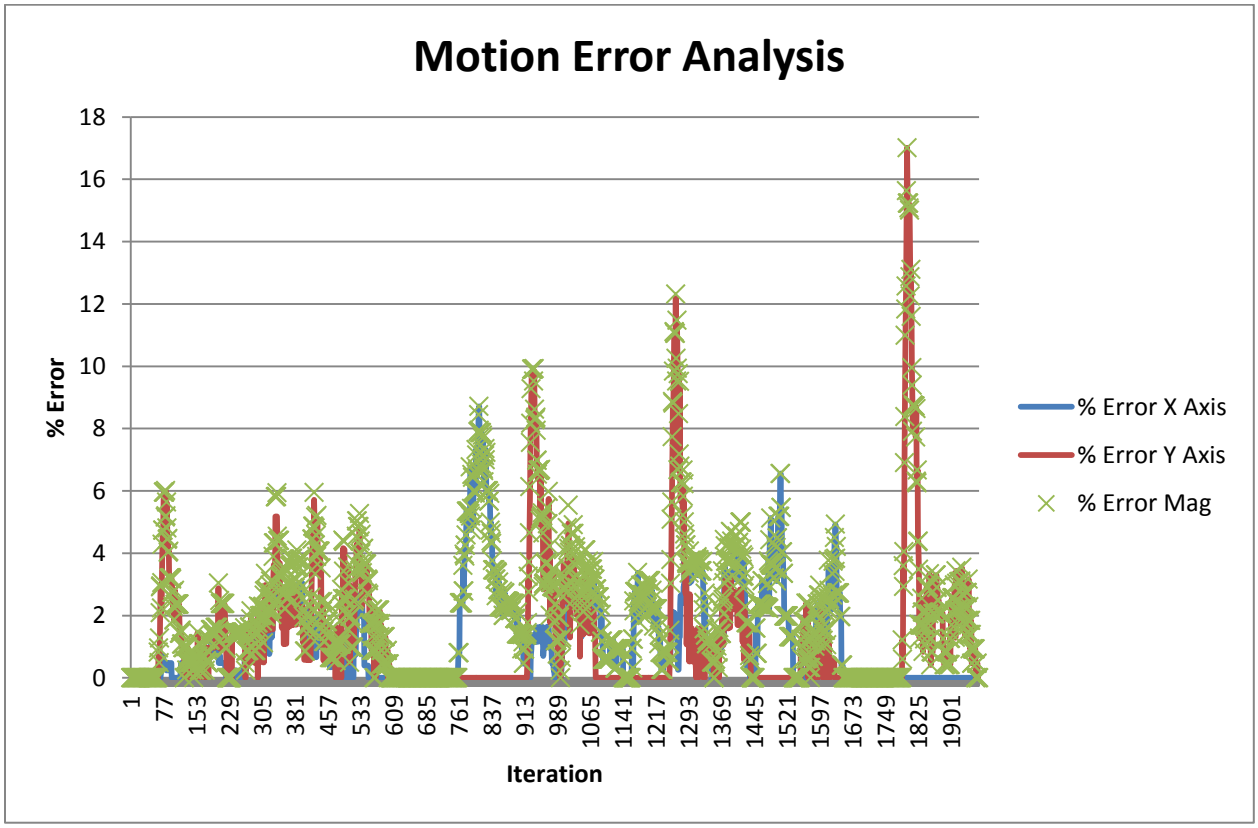


Figure 3-29: Simulation Error analysis of Raw Smoothed Values versus Easing Smoothed Values

This implementation will work for stepper and DC motors but requires feedback and may require a more dynamically tuned PID algorithm for better control.

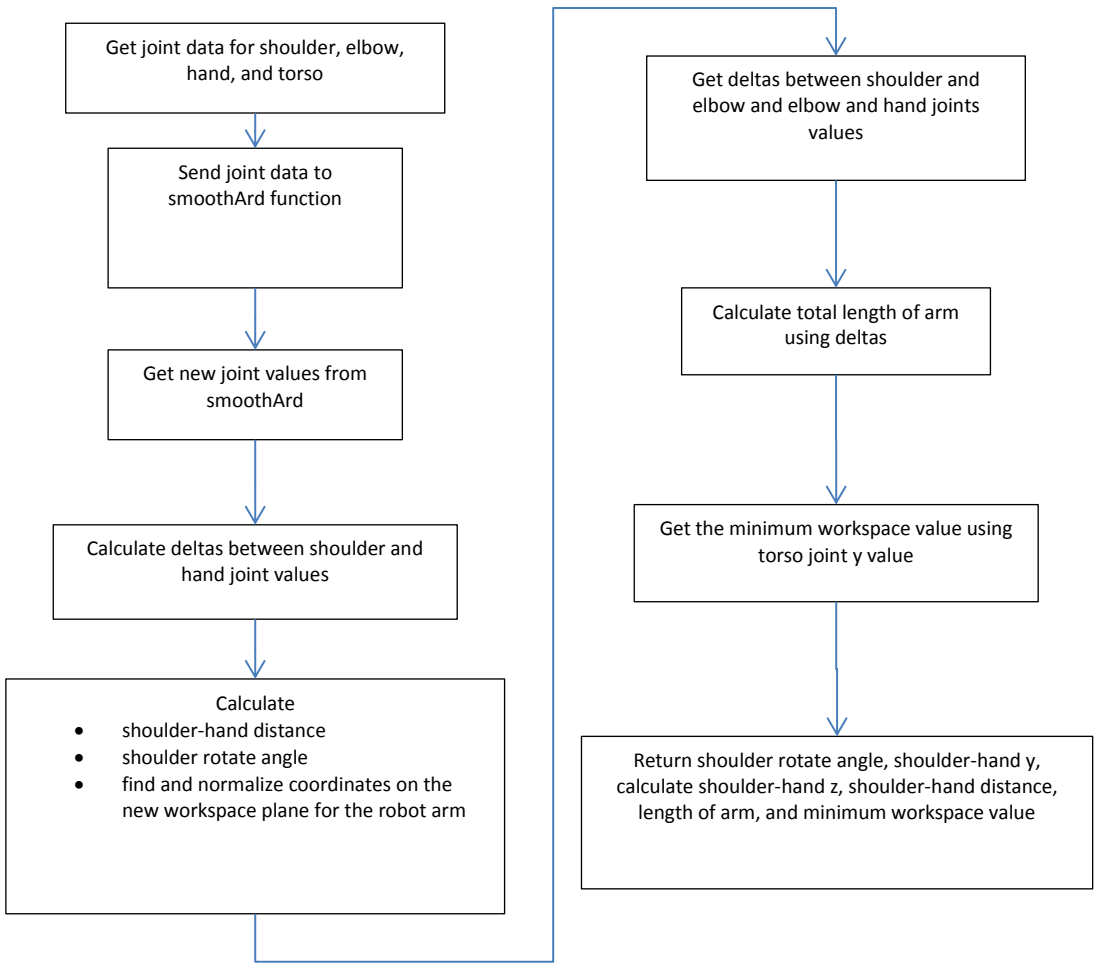


Figure 3-30: getKinect Function Flowchart

3.5 Visual feedback implementation

3.5.1 Introduction

For any mobile system that will perform its actions possibly outside of line of sight (LOS), some sort visual feedback is a necessity. As per the communication scheme of Figure 3-31, the robot is to use a 900Mhz TX/RX video transmission system to get a first person view (FPV) from the robot perspective.

However, in the concept design, there was no video output as there was no GUI. The visual feedback in important for two reasons:

- Safe locomotion of the robot from desired positions
- Verifying that the arm is manipulated as intended towards its target end position

In practice, without the visual feedback, even if the operator is in full view of the robot arm and the target object the operator wishes to retrieve, it is still incredibly difficult to quickly and efficiently manipulate the object due to the disconnect in hand-eye coordination.

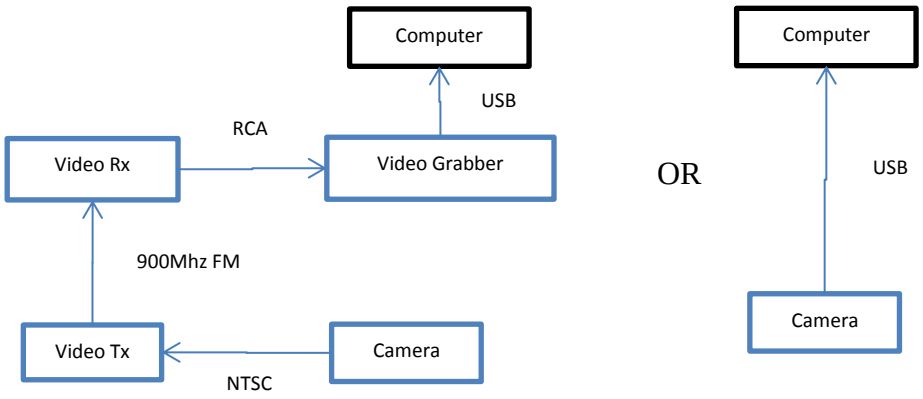


Figure 3-31: Flowchart of Conceptual Camera-to-Computer Communication Schemes

A capable yet simple GUI can be made by directly manipulating the pixel matrix, text overlays, and other various means. Instead of doing a console print, these tools can be used to show the system's working states, such as Kinect Capture on or off, whether the Kinect sees the operator's body, button states, or if the gripper is closed or open.

3.5.2 Remote Viewing Implementation

3.5.2.1 Addition packages to install

OpenCV 2.1 was installed to display the video stream from the video grabber and overlay text and graphics. It has python bindings that allow for easy integration into the code. It also has the ability to

capture and use the OpenNI Kinect stream, however, there were difficulties setting that up due to the OpenNI2 software defaulting it's installation into a folder named OpenNI2 instead of just OpenNI [13] Other packages considered were pyVLC, which are python bindings for the popular VideoLan Codec and would also allow for video and streaming, as well as wxPython, which is a GUI design based on the wx framework. pyVLC was abandoned due to not getting it to install correctly on the test system, while wxPyton was considered beyond the scope of the project for simply capturing video without pyVLC.

3.5.2.2 Programming

The implementation of the remote viewing window requires it to be integrated in the **main**, **getJoints()**, and **moveArm()** functions throughout the code, as shown in Figure 3-32. OpenCV A rudimentary window showing the view from the robot camera was initialized in **main** using openCV's **imshow**. Since the video window will be open upon program start, many states of important conditions can be shown. The Skeletal tracking state, or when **player** is found and added to **users** in **getJoints()** will be shown as a circle, where red means no skeleton found and green means a skeleton is found and is being tracked. The "Kinect capture" status, or **kC**'s state, is declared as a text overlay.

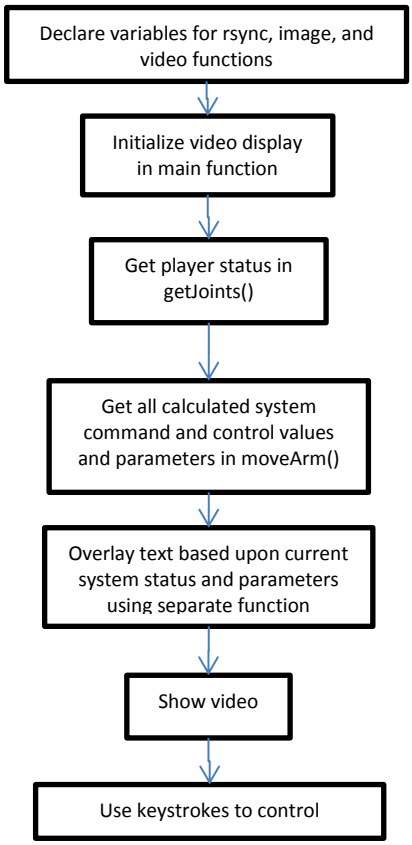


Figure 3-32: Flowchart of Conceptual Design of Video GUI

3.5.3 GUI Implementation

With the implementation of the remote viewing window, a canvas as been created to display other mission critical data in the operator's field of view. Previously, the system had no way of showing a live data feed that was not printed in a console window. The data that is required to be displayed are:

- The Kinect tracking state
- The servo angles
- The gripper state
- The active control state

This data is crucial for safety, as well as. While a proper GUI for this system should allow for changes to system variables dynamically, a graphical user interface should be simple and effective, but intelligent in

doing so as to not overwhelm the operator. Changing colors were used to show capture states and glyphs were used when possible (Figure 3-34). The concept of placing and displaying the information was straight forward, as shown in Figure 3-33. OpenCV manipulates the stored image array with the text that was gathered throughout the running iteration of the program. Text, shapes, and colors are added directly to the RGB data of the image matrix and then, when the image is being called to be shown, is displayed.

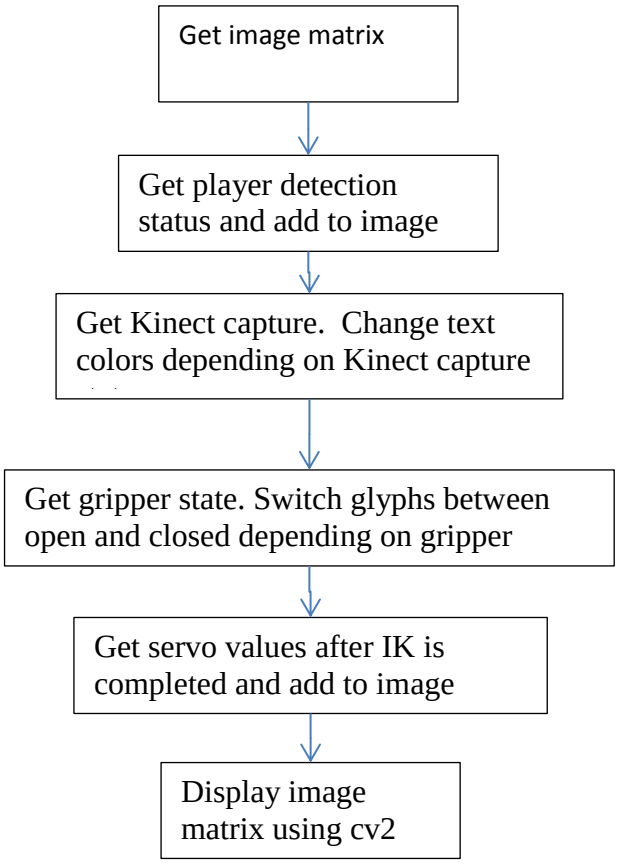


Figure 3-33: Flowchart of conceptual GUI implementation

3.5.4 Kinect View Implementation

3.5.4.1 Motivation

It is quite necessary in enabling the skeletal feedback for one primary reason about all others: if something is wrong with the Kinect capture, the operator can adjust their pose or the Kinect in order to fix the tracking error and correct the problem. Most of the observed aberrant behavior from the Kinect had to do with inappropriate positioning of the Kinect. Most of the mysterious aberrant behavior of the robot arm output was traced back to an erroneous joint position given by the Kinect. This aberrant behavior is a safety issue as the arm can move wildly out of control, breaking things, injuring animals or people, spilling liquids, or damaging the robot itself. By giving the operator the skeletal view, they can see if the Kinect is performing or there are some conditions surrounding the Kinect that needs to be addressed, and quickly stop Kinect capture or reposition their bodies back to the last good captured state.

3.5.4.2 Background on How it was Accomplished

The sample code for displaying the skeleton from the python program existed, however the libraries for doing so, namely pygame, were not included in the implementation of the control system. Will Thompson was tasked with porting the code from python to OpenCV. He successfully accomplished this task. However, the implementation for displaying video was later changed to openCV's highgui wrapper, cv2. His work had to be ported again to work with cv2, which was my task. This was not a difficult task, but features had changed and had to be researched.

3.5.5 Results

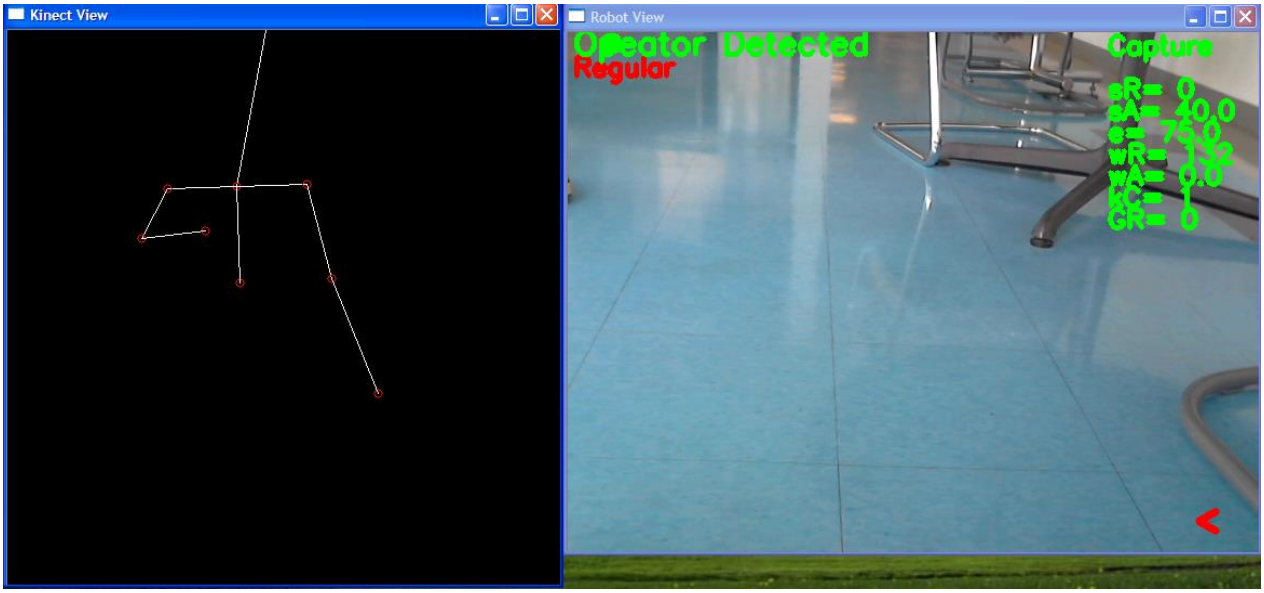


Figure 3-34: 1st Screenshot of Visual Feedback System

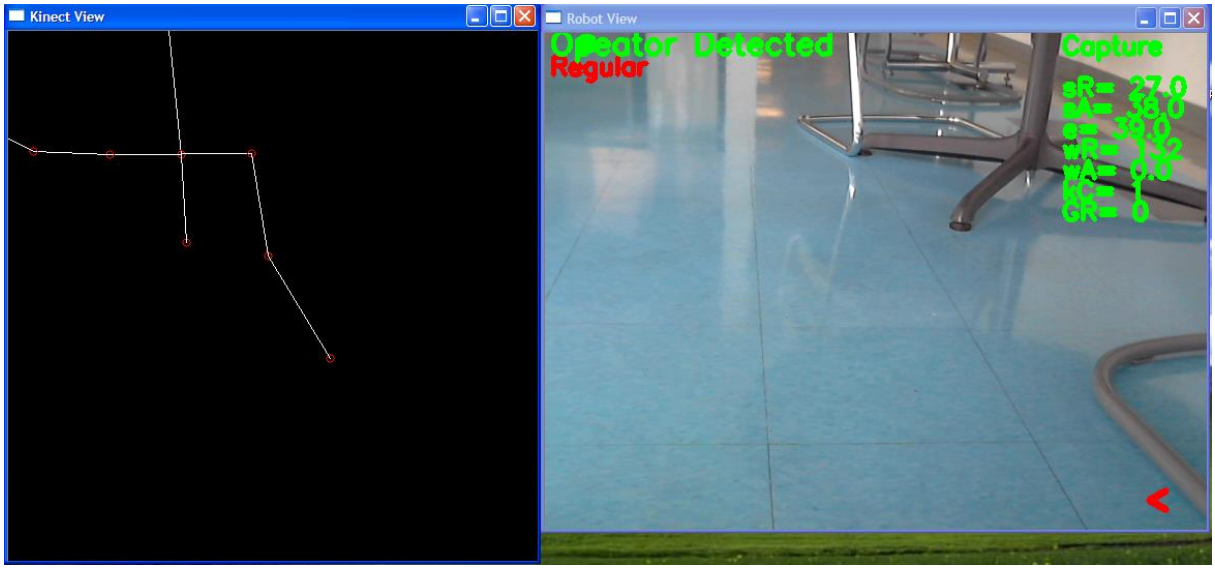


Figure 3-35: 2nd Screen Short of Visual Feedback System

3.5.6 Discussion

While the obvious practical benefits of visual feedback are that operators can see what they are picking up remotely, get better hand-eye coordination with the robot arm, and do obstacle avoidance, intangible benefits of operator security and confidence are even more important. The inclusion of the

visual feedback on the Kinect input, important system parameters, and robot arm output allow the control system to be operated inside and outside of LOS with greater safety. The operator now has a powerful troubleshooting technique with seeing the Kinect skeletal view (Figure 3-35). They can easily tell if the Kinect is seeing their bodies or not, as shown in Figure 3-35, and visually verify if the control system is actively capturing them, allowing them to freely move their arms to perform a more local task. These simple considerations are steps in the right direction for bringing safety to the potentially dangerous machine of household robotics.

3.6 Configuration Files

3.6.1 Overview

The system is touted to work with multiple robots, however, there is no user friendly way to quickly implement a new robot configuration, choose input/output COM ports, or switch back to an old configuration. Also, despite being well commented and the placement of all user variables be close to each other, even a seasoned user would have to search the program and know where the intended lines to change are, as well as how and which to change them. The solution to this was to implement a configuration file that allowed the user to navigate through the options using the SDK's text based console in lieu of a more mature and comprehensive GUI. A further benefit, using the said configuration file, a more computer savvy operator would be able to change erroneous elements of a specific machine profile, without having to recreate a whole new profile.

3.6.2 Implementation

3.6.2.1 Considerations for Config File Schema

3.6.2.1.1 Machine Identifier

The operator should be able to name each machine as they wish. They should be able choose between a list of names at the start and then have the program use the configuration variables to allow the specified robot to be properly manipulated.

3.6.2.1.2 Inverse Kinematics

The inverse kinematics system is dimensionless. The only requirement is that the units remain the same. For example, the units used to determine the link lengths of the test bed LEGO robot used with the Simon's Fellowship student were the number of studs. Studs are the knobby connecting part on the top of most LEGO bricks and are the standard unit when dealing with LEGO bricks in terms of length and depth. This allows the configuration data to also be dimensionless and assists with the user friendliness of the system.

The inverse kinematics links are named analogous to the bones of the human body. The links are:

- Humerus- the bone or link that connects the shoulder to the elbow
- Ulna- the bone of link that connects the elbow to the wrist
- Hand- the link where the gripper is attached

Other information needed for IK is the base height. This is the height above the ground to where the shoulder is attached.

3.6.2.1.3 Communications port

Although the Arduino prototype boards, from Reversion 3, have a USB VID, between different machines, operating systems, or Arduino boards, the serial communications port, or com ports, can change. Due

to this, the respective input/output comports should also not be hard coded but be user defined. Since the com ports are not static, the program should be able to see and display all of the comports and then let the user decide which to use for what.

3.6.2.1.4 Motor and Feedback Types

A robot can be manipulated using steppers, servos, or DC motors. Each robotic system can have feedback or no feedback. To comply with the universal nature of the system, a motor type and feedback identifier needs to be introduced. These variables should be passed from the SDK program to the robot controller, as that processing is handled there.

3.6.2.2 Programming

The programming of the config file had to be done outside of the main loop as all the variables changed were global variables and had to be initialized before the main loop. Two config files are created by the program, if it does not exist, and/or parsed: robot.cfg and computer.config. Two config files were necessary as their parameters are independent of each other.

Robot.cfg, the config file used in the algorithm in Figure 3-36, contains the machine profiles for the robot, such as link lengths, motor type, and feedback. The program allows the operator to enter a new profile automatically if none exists or manually make the choice to create a new one or make changes if they desire. After, the program allows the used to select an existing profile to use, using the friendly names that the operator entered, which are identified by a dynamically enumerated numbered list. The computer then parses the entry for the link lengths, motor and feedback information.

Computer.cfg, the config file used in the algorithm in Figure 3-37, contains the com ports necessary as those were machine and Arduino dependent. Like robot.cfg, the program allows the operator to enter a new profile automatically if none exists or manually make the choice to create a new one or make

changes if they desire and then pick the one that they want to use. Computer.cfg will assist the operator in selecting the com ports by scanning for available com ports in the computer.

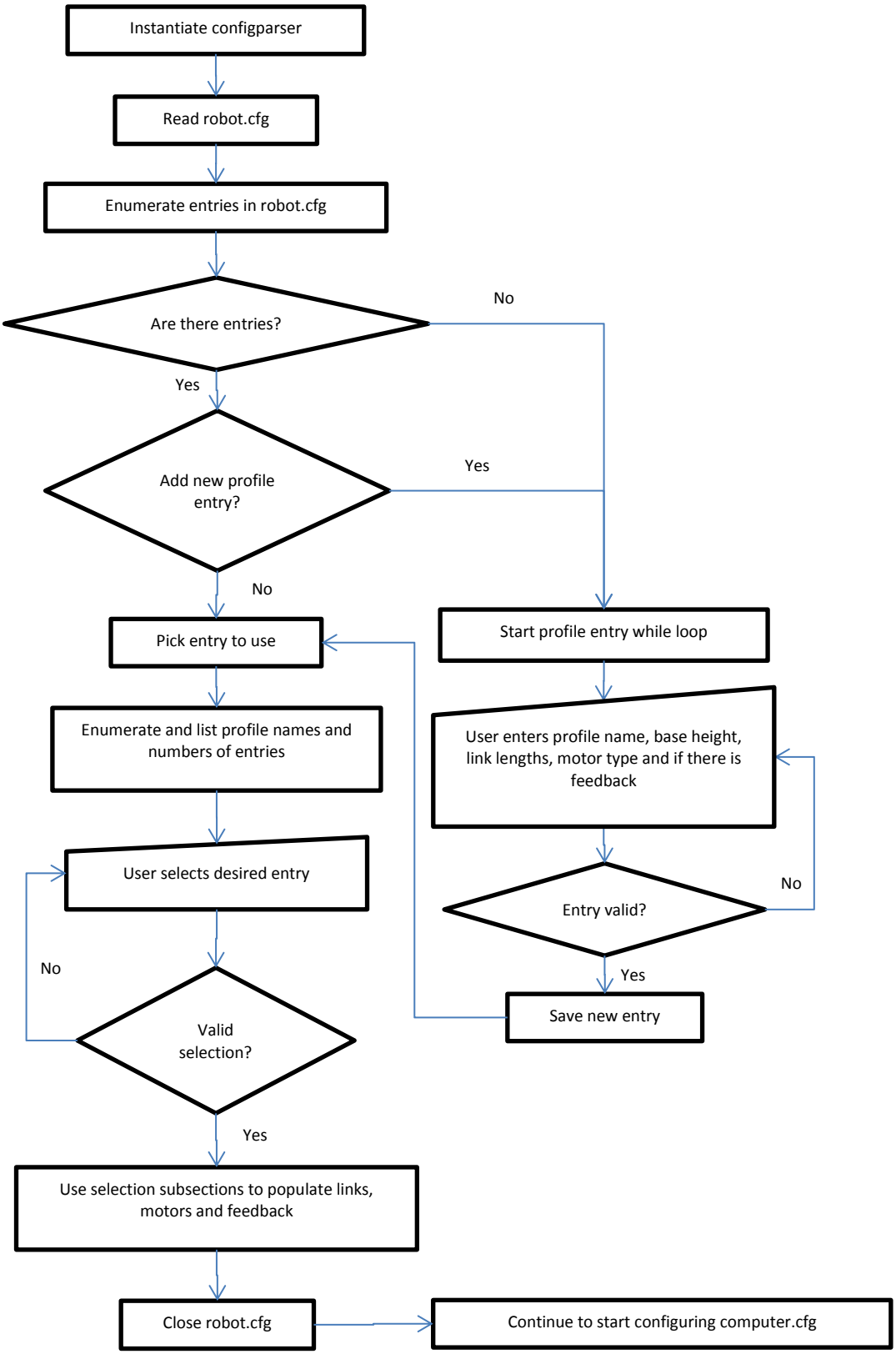


Figure 3-36: Robot.cfg Flowchart

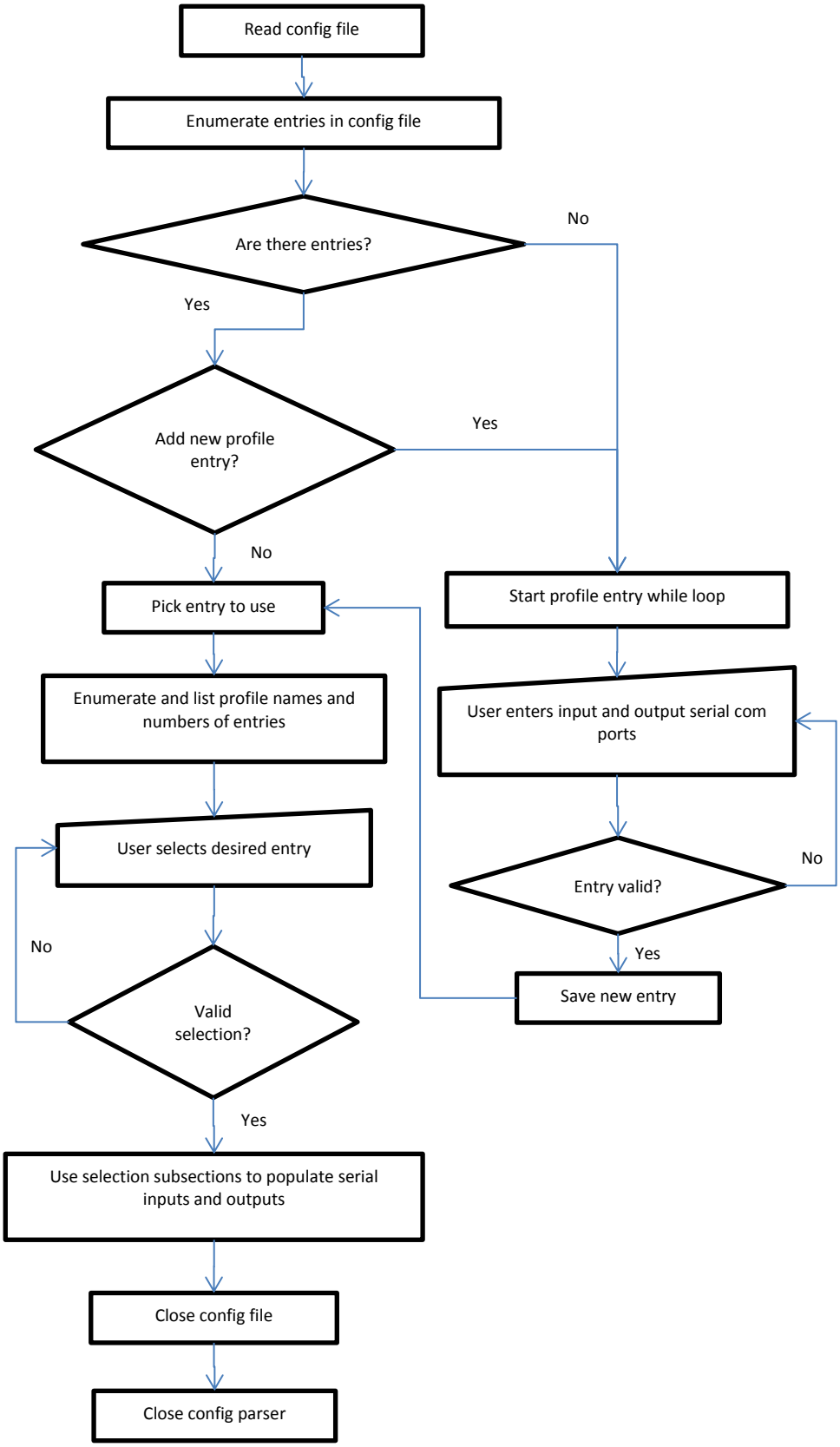


Figure 3-37: Computer.cfg Flowchart

3.6.3 Test 1

3.6.3.1 Overview

This test addresses the user friendliness of creating a new profile and accessing an old one. As previously stated, the operator would have to parse the 1000 line code and manually enter in the necessary values correctly. If the operators simply changed machines, microcontrollers, or robots, they would have to undergo this process again. The test will be a timed test, in comparison to redoing the same information using the old method. Each subject will also be given a Scoville test concerning the usability and ease of each method.

3.6.3.2 Task 1

The User will have to create a completely new robot profile given the parameters[3] that

- Profile Name = <User Choice>
- Base Height = 3.25
- Humerus = 5.75
- Ulna= 7.375
- Hand = 3.875
- Motor type = Servo
- Feedback = None

as well as a new computer profile with the parameters that

- Profile Name = <User Choice>
- Input = COM5

[3] These link length and motor parameters are from the robot arm measurements and motor type given on Lynxmotion AL5D Robot Specifications sheet found at http://www.lynxmotion.com/driver.aspx?Topic=specs04

- Output = COM8

This test will examine the time it will take to institute and use a brand new profile. Considering old method required an experienced programmer, the test is only to validate the effectiveness of the configuration file system as “easier”.

3.6.3.3 Task 2

The operator has to select their already built profile without adding any new profiles. They will repeat the process soon after to demonstrate if the selection process will speed up upon a repeat attempt

3.6.3.4 Results

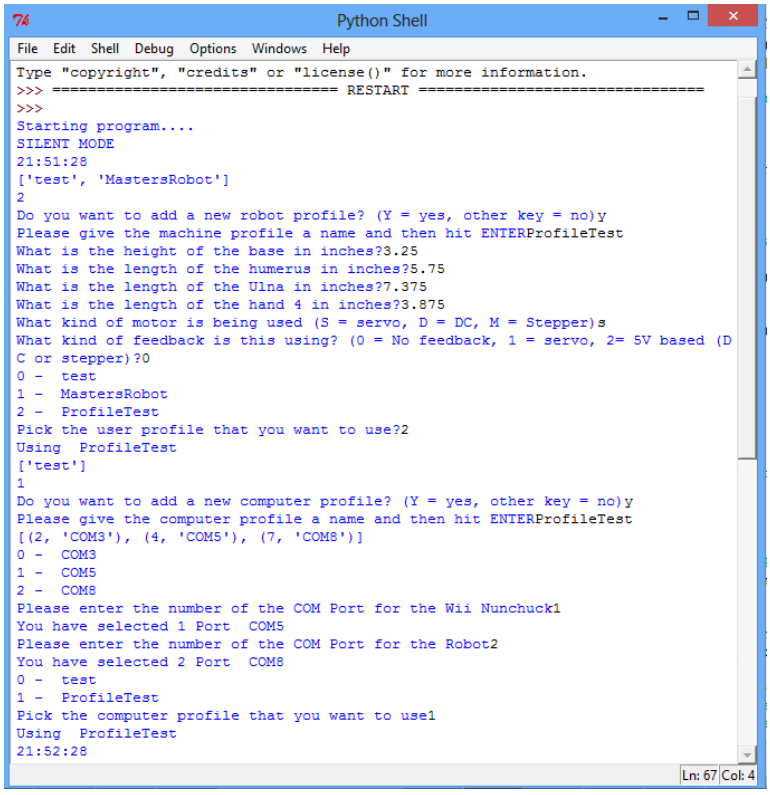


Figure 3-38: Task 1- Config File Creation Test Results

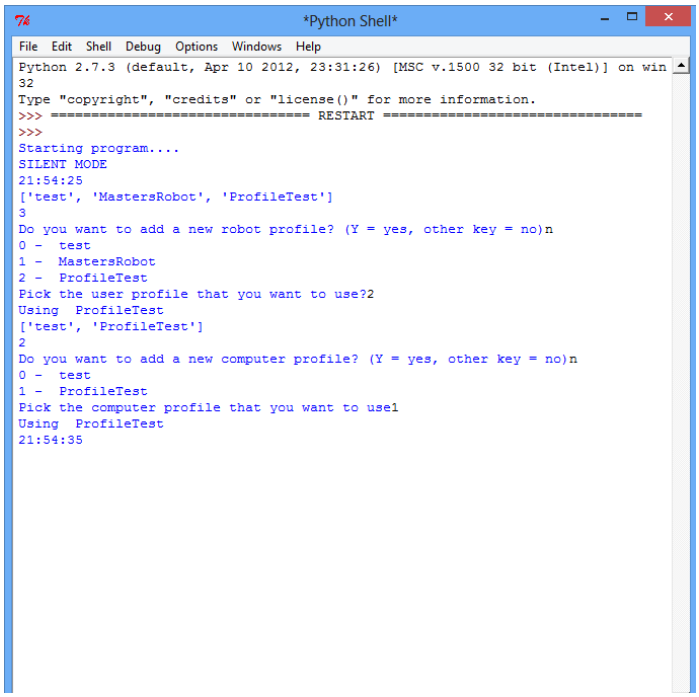


Figure 3-39: Task 2- Config File Startup Test- First Run Results

.

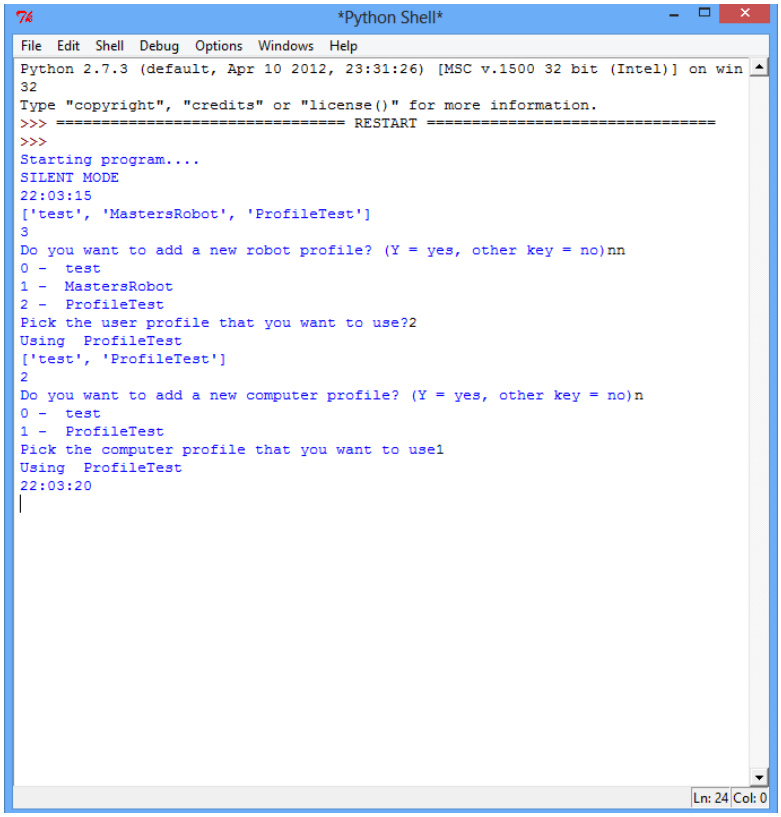


Figure 3-40: Config File Configuration Selection Speed test

	New System				Old System			
Test	Operator 1	Operator 1 Mistakes	Operator 2	Operator 2 Mistakes	Operator 1	Operator 1 Mistakes	Operator 2	Operator 2 Mistakes
Create Profile First Run (sec)	60	0	92	1	236 seconds	1	413	1
Create Profile Second	52	0	57	0	97 seconds	0	148	2

Run (sec)								
Select Profile- First Run (sec)	10	0	16	1	0			
Select Profile- Second Run (sec)	5	0	11	0	0			
Ease of Use Grade (1-10)	9		10		2		3	
Usability Grade (1-10)	9				3			

Table 3-8: Test 1 Results

3.6.3.5 Discussion

3.6.3.5.1 Profile Creation Test

This test was successfully completed in 60 seconds (Figure 3-38). To do this manually, the operator would have to go to 3 different sections of the code, identify the correct variables (the code is well commented as this was the previous method), make the correct changes in the correct format, and then hope that the system did not error out due to a mistake. This would have to occur upon each device change or operating system. When Mr. Thompson first had to configure the arm for his machine and

the LEGO test robot, the time to do this was about 4 minutes with verbal help from me. This took 25% of the time as well as is user friendly enough for anyone to do, as long as they have their machine specifications on hand, which should be provided by the machine manufacturer.

3.6.3.5.2 Profile Selection Test

The first run (in Figure 3-39) shows that with reading all the options, an operator can select an existing profile in 10 seconds. Once an operator knows their profile options, they can speed through the configuration file selection process in 5 seconds, as shown in the second run screenshot in Figure 3-40. However, as shown in Table 3-8, this test does worse than the old system, which just uses the hard coded files and doesn't ask you to interact at all. This time can be cut down by implementing a default profile, or the last used profile, making it 1 key stroke to enter instead of 4, but it still will be longer than no time at all.

3.6.3.5.3 Verdict

The program successfully improves the system by allowing the average, literate operator easy access to manipulate key parameters of the control system.

3.7 Summary of Research and Design Work

At this point, the revised system is ready for full system testing. Each of the proposed area for changes has been addressed. The SDKs were compared and parametrically analyzed, leaving us with the determination that the OpenNI SDK was the better SDK to move forward with due to the Kinect SDK poor operating system support and potentially problematic method of tracking skeletons (Section 3.3.4.3). The servo value output from the main processing was found to be extremely noisy and a variety of smoothing algorithms were used. The decidedly best smoothing method was the 3-period, moving, and weighted average (Section 3.4.3.1.9). It had a single frame of delay compared to the raw values, yet gracefully followed estimated the trend of where the raw values wanted to go, but was too

noisy to accurately reach. On the output side, a new servo output scheme on the Arduino mega was explored. After exploring a cascading PID algorithm, it was discovered that the initial burst of speed from the large initial error contributed to the system instability by disturbing the other servos. An easing algorithm was then pursed, simulated, and successfully employed in the code. This algorithm will be used in the Arduino Mega code (Section 3.4.4.4.3). Visual feedback was finally implemented using OpenCV (Section 3.5). The visual feedback scheme included remote viewing from the robot, getting text, glyph, and color based feedback of important system parameters, and viewing the skeletal representation using the joint data. This will improve user confidence, troubleshooting control system misbehavior, and safety. Finally, the addition of the configuration files allows the system to be more portable when changing robot arms, machines, or microcontrollers (Section 3.6). This replaces the old method of hard coding in values, which became increasingly difficult for the uninitiated or non-programmer to do due to the increasing complexity of the code. Other capabilities, such as recording and playback of the raw servo values, the simulator, and controlling two arms, and an a smoothing algorithm that would address the needs of those suffering from spastic muscle control maladies were implemented, partially tested, but detailed as part of this thesis. Next time.

Chapter 4

Progressed System

4.1 Overview

This section will detail the new, altered, or additional functions and methods in each program due to the findings in this research. The final implementations were

1. Control side smoothing of servo values using the 3-period moving weighted average method, as decided upon in Section 3.4.3.1.9 and showed in Section 3.4.3.1.7, pages 55 and 52 respectively. It is implemented in the Arduino Mega code as moveServos(), quadIO() and quadIO() under Section 4.2.2.2 on page 96.
2. Robot side smoothing of servo motors using the easing method of Section 3.4.4.4 on page 59. It will be implemented in the Python code as smoothard() and illustrated in Section 4.2.1.2.2 on page 90
3. The Visual feedback and GUI created in Section 3.5 using OpenCV. It will also be implemented in the python code and had to be distributed throughout several main functions, such as the main function, main while loop, getJoints(), and moveArm(). It will be used in the main process and the sub functions, as detailed in Section 4.2.1.1 on page 87 and the functions will be illustrated under Section 4.2.1.3 on page 91.
4. The configuration file and built methods of quickly switching the system variables which allows for the operator to quickly move between differing computers, robots, and Arduino devices. This was discussed in Section 0. It will be implemented in the main process of the python code and is detailed in Section 4.2.1.1 on page 87. The helper function to scan the serial ports and return the list is found in Section 4.2.1.2.1 on page 89.

5. The final communication scheme, now with the visual feedback, is documented in Section 4.3.1, on page 99.

This will be implemented and tested on the test bed robot platform for verification of system improvements. All new functions and process flows will be illustrated using flowcharts.

4.2 Code

4.2.1 Python

4.2.1.1 Main Process Loop

As illustrated in Figure 4-1, With OSCeleton running, import the program libraries listed in Section 2.2.4.3 s well as configParser, random, OpencV, and csv. Initializ the global variables for the servo arrays, link lengths, frame counters and other important system values, initialized the serial communications, and started the OSCeleton server. Open the configparser and allow the operator to choose the robot profile they want to use or to add a new one. Once they go to the next step, they are asked what computer communications port is to be used for this machine. Once selected, the video is initialized and the main while loop begins, which calls getJoints(). getjoints() waits for a message from the OSC server, which would contain player data. If no player data is received, it exits the function, but the while loop calls it again on the next iteration to check for messages. If player data is found, the player data, which includes the joint data, is added to the player class and moveArm() is called. moveArm() sends a request message to the Ardiuno Uno and then waits for the replay with the Wii control data. After it gets the data, and parses and conditions the data using getWii() and its helper functions, the parsed Wii command string value that includes the controlling variable for whether to actively process the inverse kinematics is checked. If the condition is false, it simple returns the last values of all the servos, motor controls, and gripper states. If the condition is true, the program uses getKinect() to parse the player data for the necessary joint data and then start conditioning it for the

inverse kinematics by getting the rotate angle, getting the humerus and ulna link lengths using trigonometry, getting the length and magnitude of the arm, position of the start and end effector, and then maps end result values from Kinect space to robot arm space. The new mapped values are sent to IK(), where inverse kinematics finds the servo values and returns them. After a few checks, the values are sent to the smoothArd() function to be smoothed using a 3-period weighted average. These smoothed values are saved in the servo value arrays and are sent Arduino Mega in the moveArd() function. The new servo values are saved to the global values using angles() and the program runs showVid() before ending the program loop and returns to the main while loop.

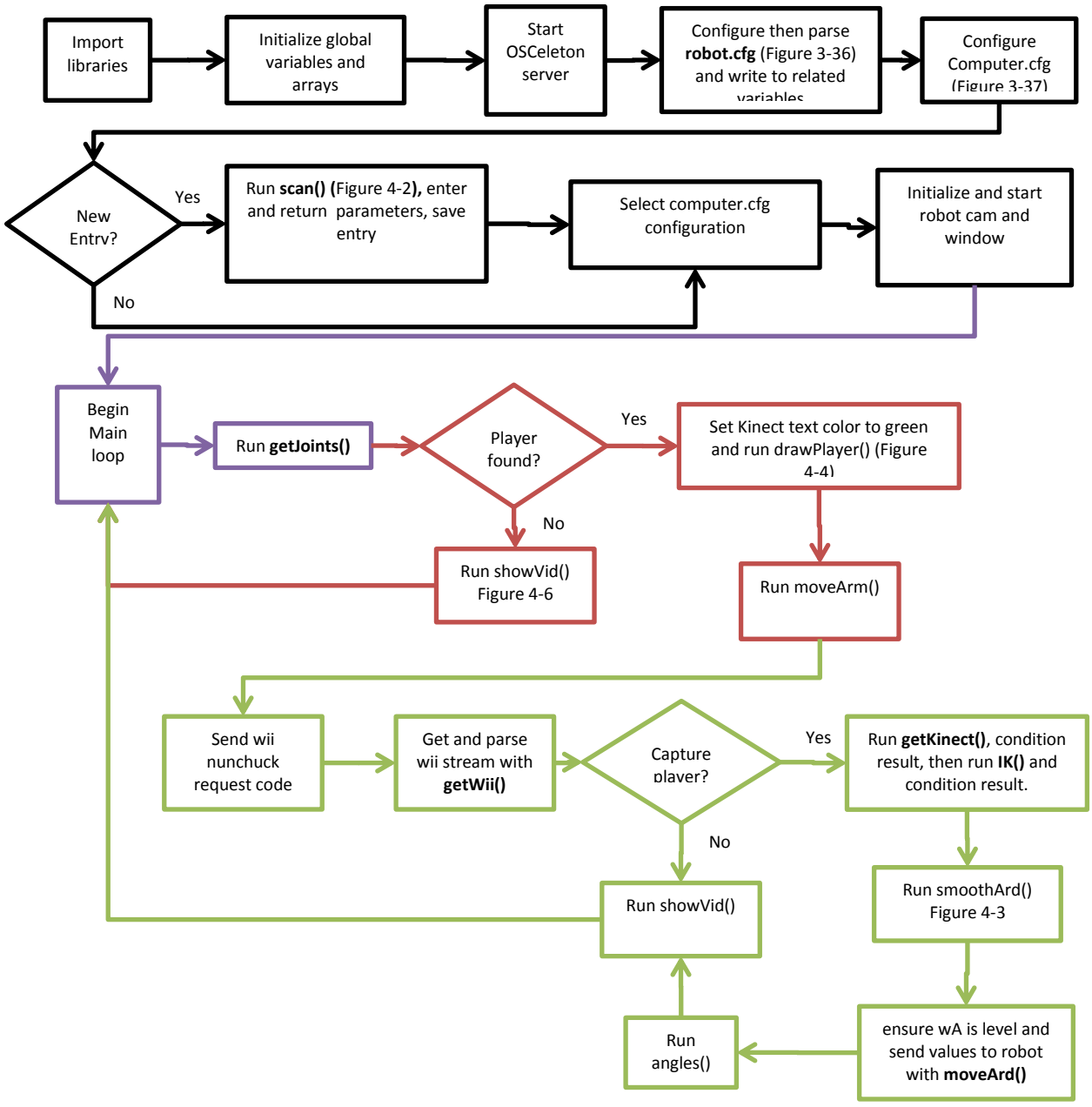


Figure 4-1: Flowchart for Main Kinect Processing

4.2.1.2 Sub functions

4.2.1.2.1 scan()

Sub function scans through available serial ports and returns list to operator.

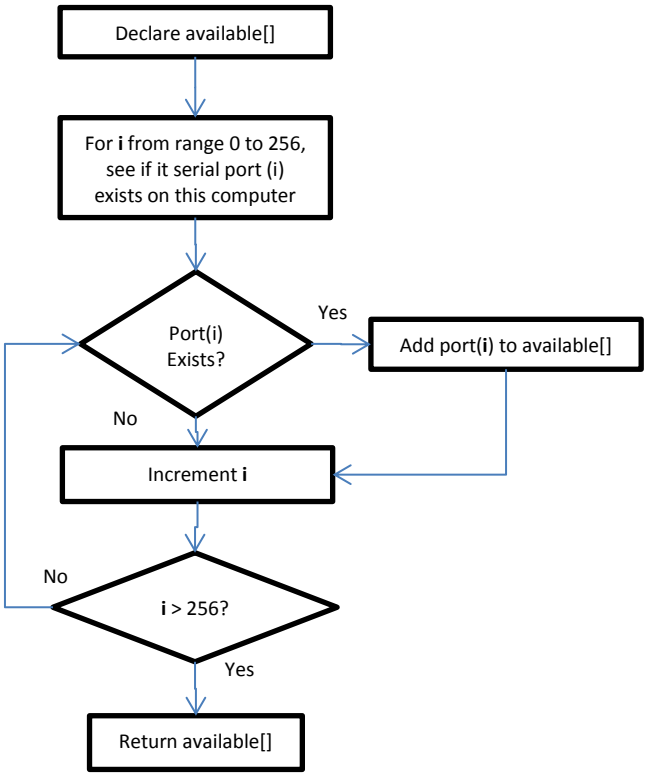


Figure 4-2: Flowchart for scan()

4.2.1.2.2 smoothArd()

This function contains the smoothing algorithm on the calculated servo values to smooth the motion profile.

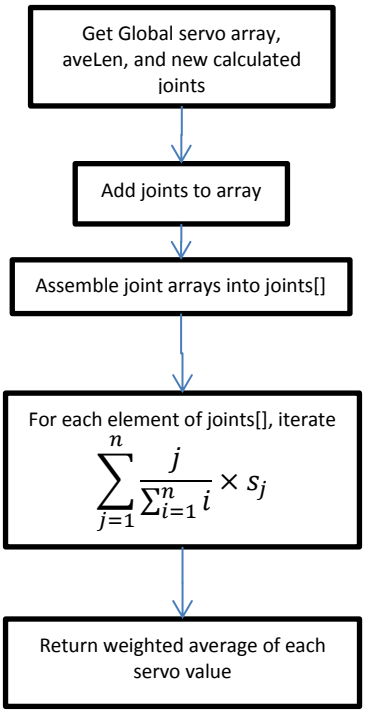


Figure 4-3: Flowchart for smoothArd()

4.2.1.3 Helper Functions

4.2.1.3.1 drawPlayer()

This function creates a new image canvas using mat, and then gets an upper body joint pair and sends it to drawLine() to be calculated and drawn.

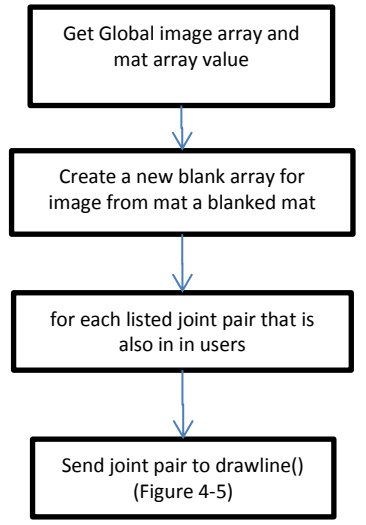


Figure 4-4: Flowchart for drawPlayer()

4.2.1.3.2 drawLine()

This function takes the player data and the two joints, then maps it on the image array and draws lines between the found z and y points, as well as circles at the found points.

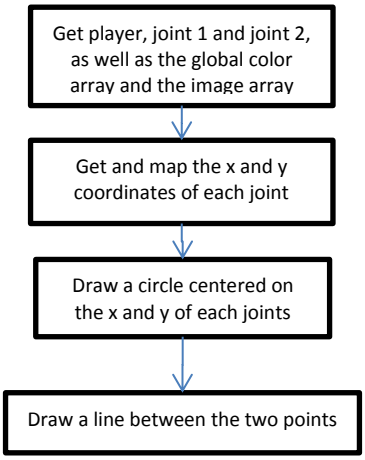


Figure 4-5: Flowchart for drawLine()

4.2.1.3.3 showVid()

This function gathers video elements and then displays frame image on the screen.

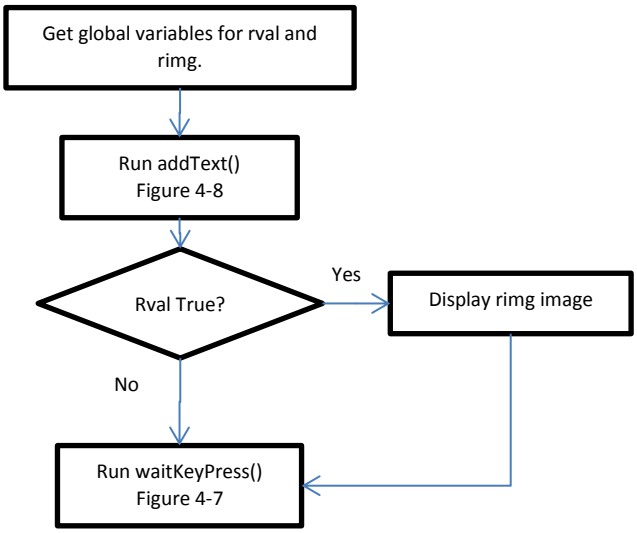


Figure 4-6: Flowchart for showVid()

4.2.1.3.4 waitKeyPressed()

This decision tree function sets flags and performs operations based on user input.

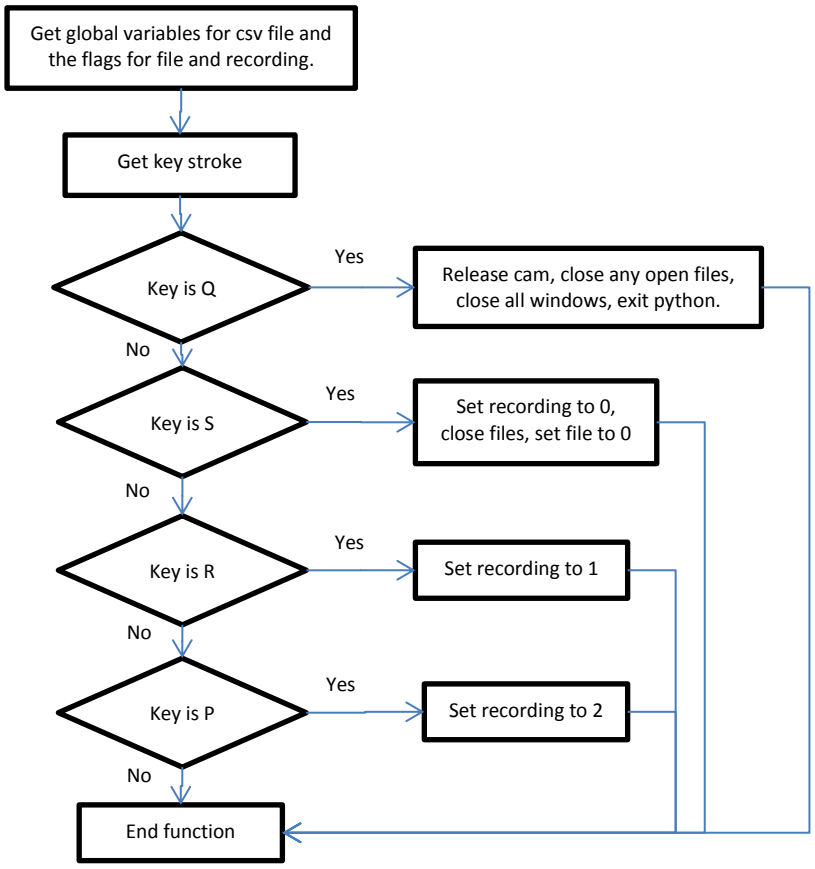


Figure 4-7: Flowchart for waitKeyPressed()

4.2.1.3.5 addText()

This function goes through system parameters and places the desired text strings in the video image.

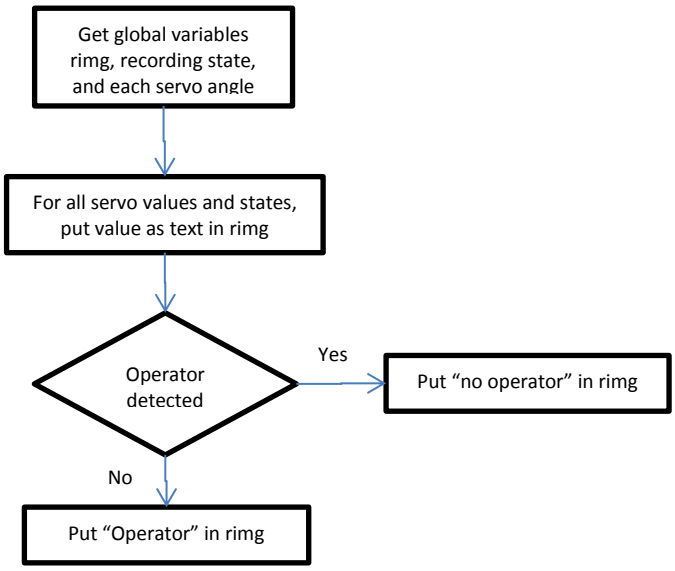


Figure 4-8: Flowchart for addText()

4.2.2 Arduino Mega

4.2.2.1 Main Processing loop

This revised version of the firmware (Figure 4-9) begins by importing the 2 libraries, the modified Roomba500.h, and Servo.h,. It creates arrays for the easing function, initializes a few pins, and the servo global variables, serial global variables, servo global variables, the input array, feedback, and a few helper variables are initialized. The firmware enters setup(), where there servos are attached, pins are declared as inputs and outputs, serial communication parameters are set, and communication between the Roomba and the Arduino and the Arduino Mega and the computer are established. Once the setup is complete, the program enters the main loop, void loop(). The loop starts waiting for serial input. If no input is found, it simple instructs each servo and the Roomba motors to move to the last value of their respective globals. If there is Serial input, the program finds the start byte, which equals 255, and then parses the serial string and organizes the values. After conditioning the motors to be value appropriate for the Roomba, it sets the gripper servo based on the gripper state and runs moveArd(), the new easing

algorithm. The results of moveArd() are either quadIO() or quadoO(), which make changes to the global servo values, or doing nothing. Send all 6 global servo values each servo and the motor vales to the roomba. After a delay of 15ms, the Arduino checks the serial input again.

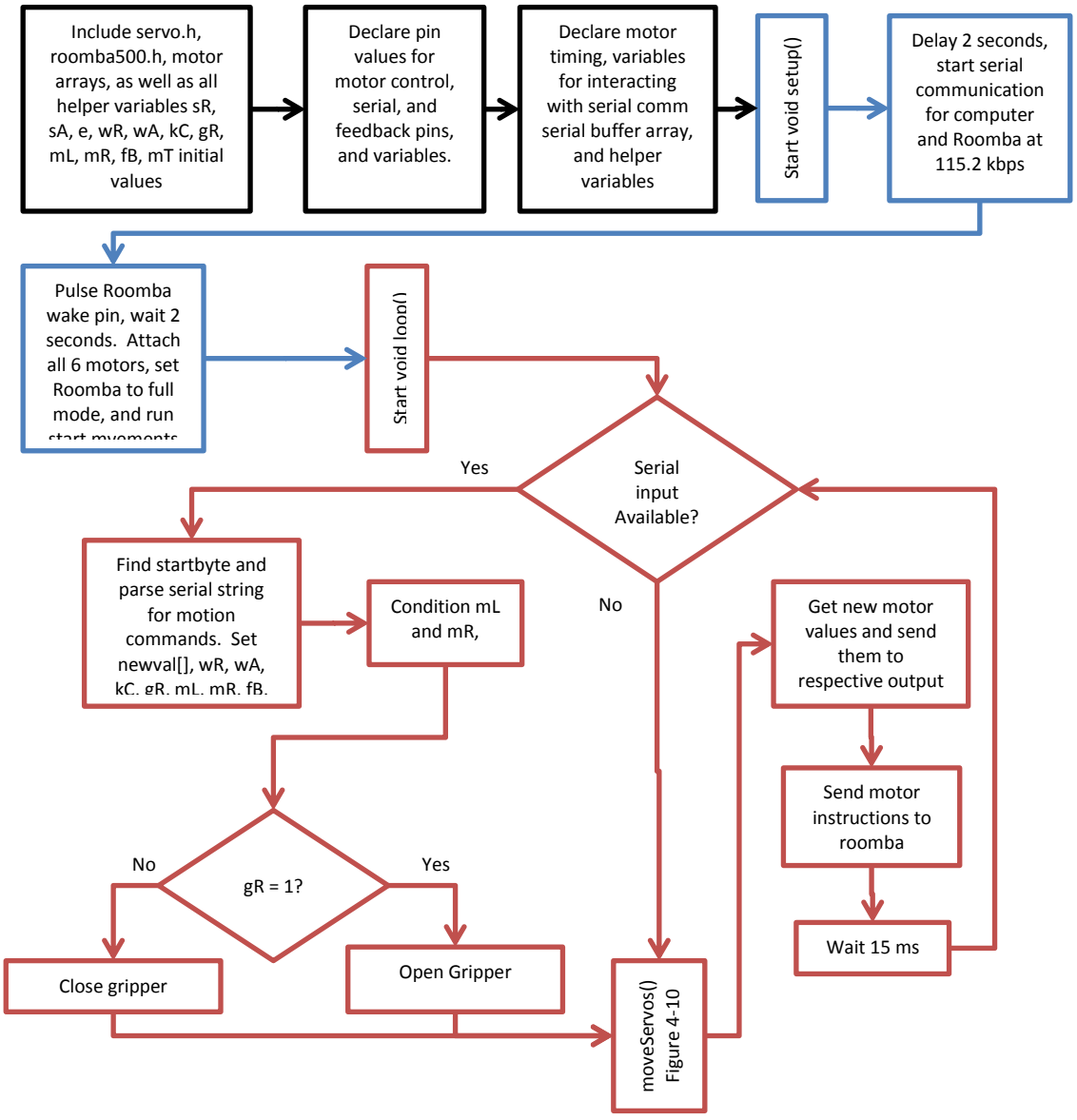


Figure 4-9: Flowchart for Arduino Mega Main Code

4.2.2.2 Sub functions

4.2.2.2.1 moveServos()

This decision tree function runs the multi-servo easing algorithm

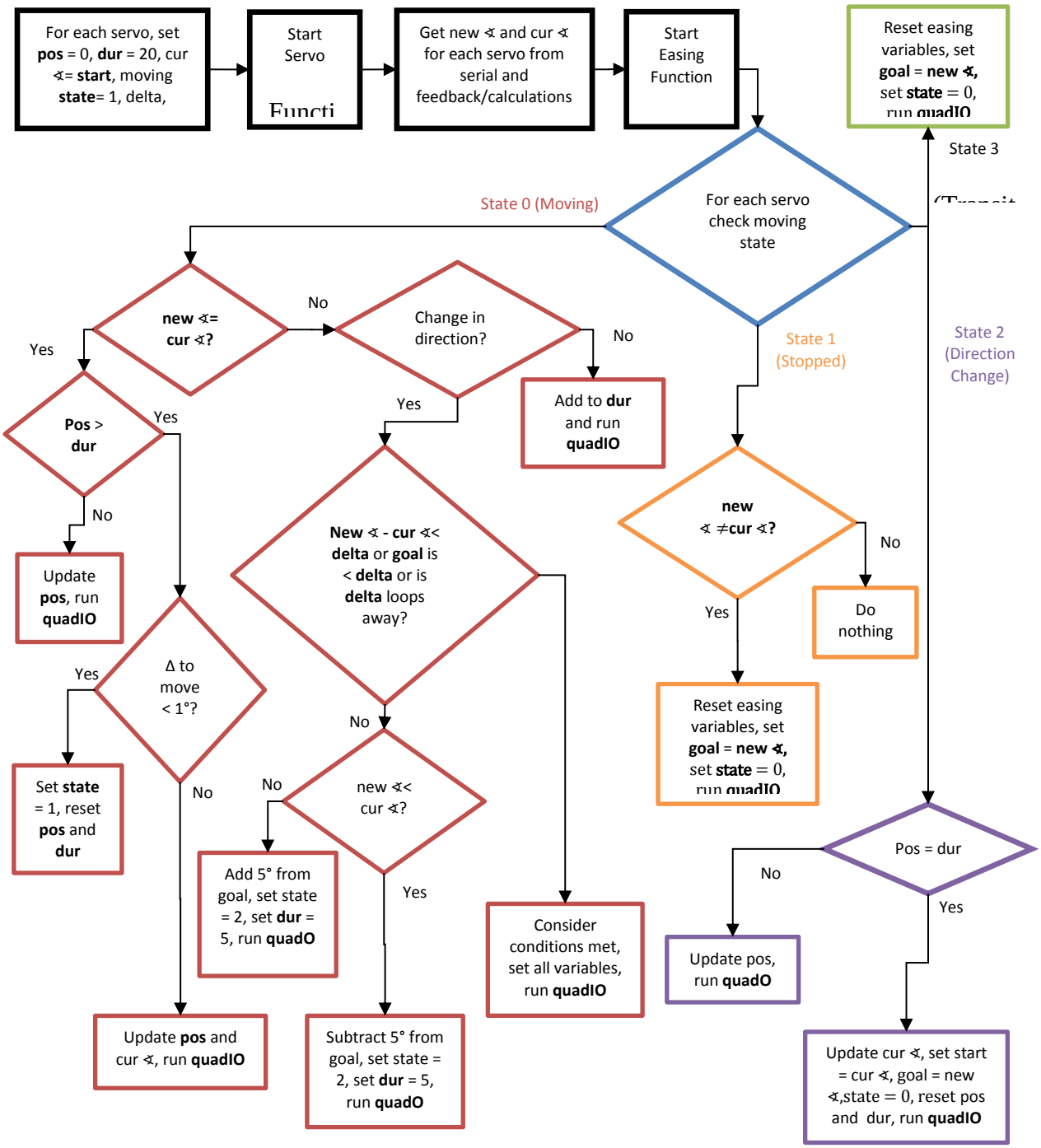


Figure 4-10: Flowchart for moveArm() for Arduino Mega

4.2.2.2.2 quadIO()

This function calculates the next value the “in-out” easing algorithm and returns it.

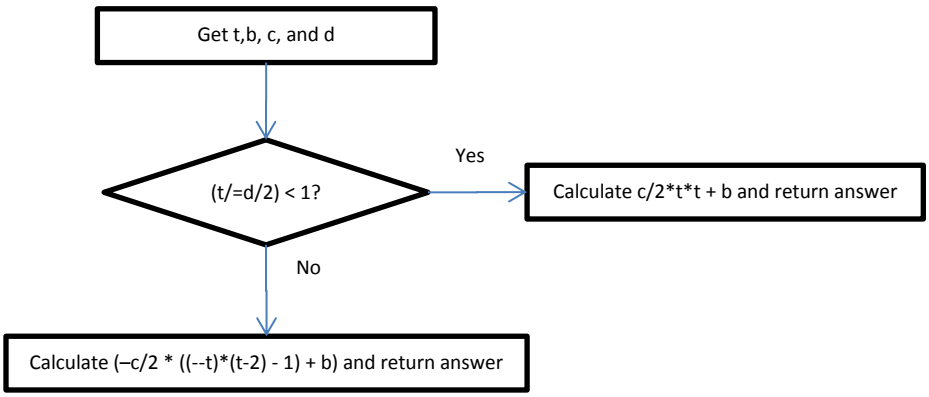


Figure 4-11: Flowchart for quadIO()

4.2.2.2.3 quadO()

This function calculates the next value the “out” easing algorithm and returns it.

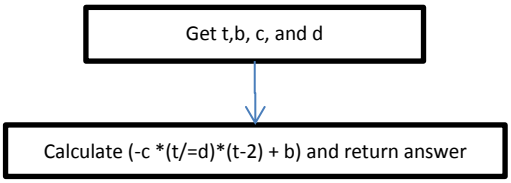


Figure 4-12: Flowchart for quadO

4.2.3 Arduino Uno

4.2.3.1 Main Processing Function

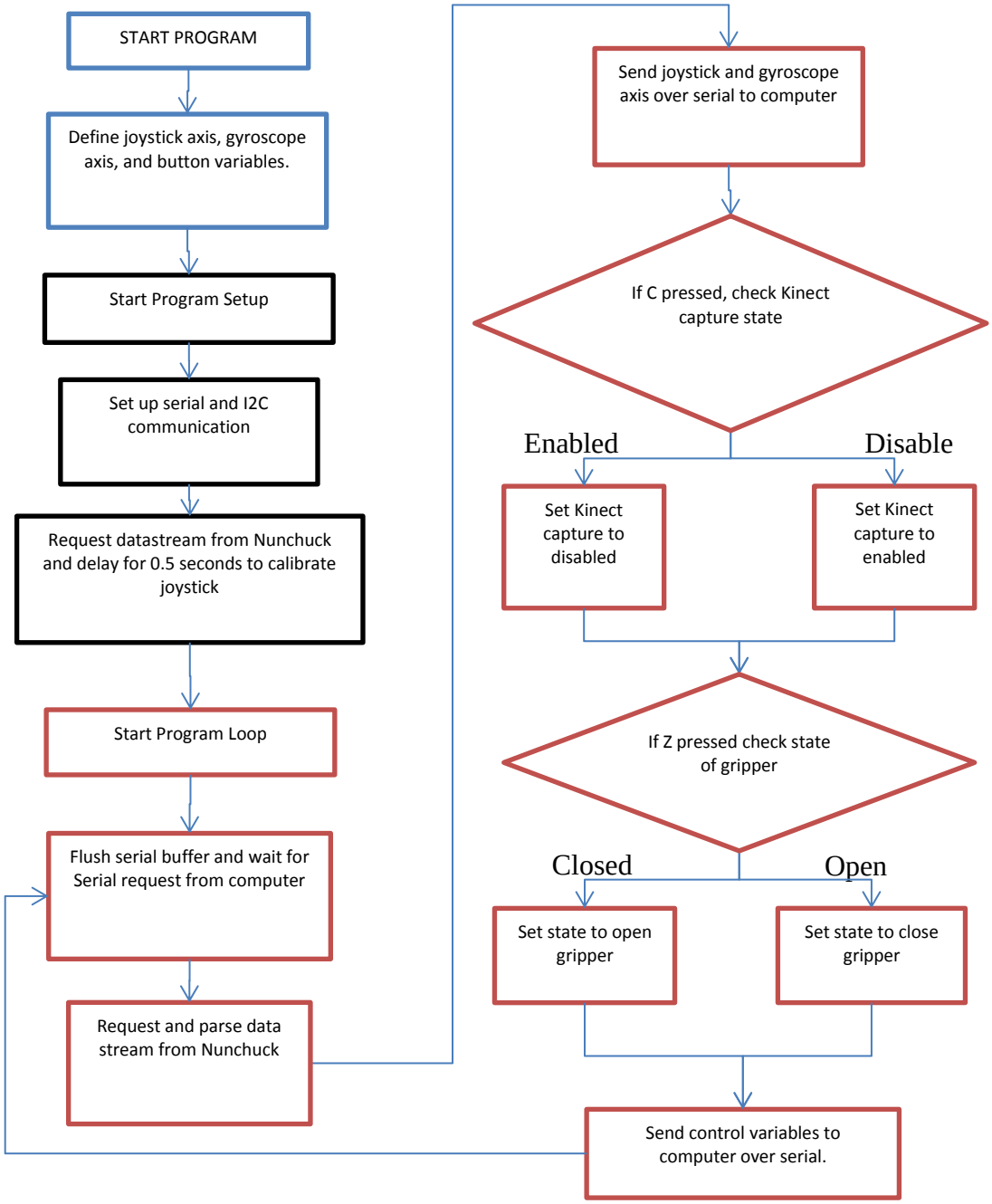
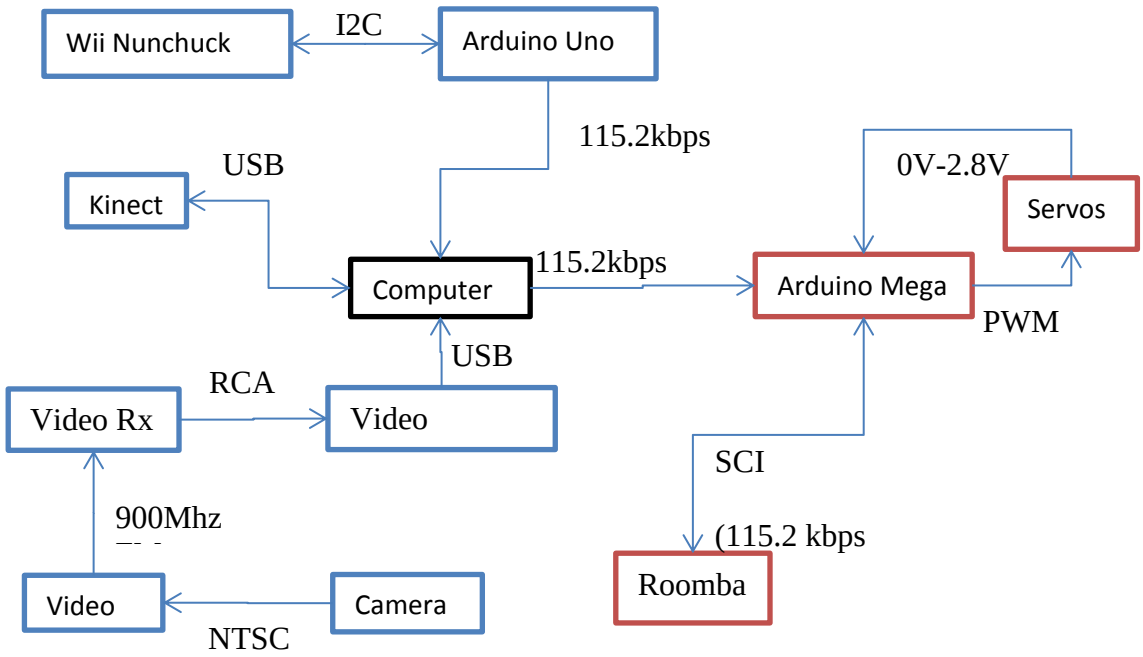


Figure 4-13: Progressed Wii Nunchuck Code

4.3 Schematics

4.3.1 Communication



Chapter 5

Full System Testing

5.1 Overview

With the theory applied and the simulations run, the time has come to put the new program into the control system and test out changes on the test bed robot. Unlike the previous tests, which tested the modular usability of the each enhancement, these tests will involve direct control with all the enhancements in concert and examine if and how they improve the robot's ability to accomplish the required tasks.

5.2 Test 1- Control

5.2.1 Overview

A mechatronics system requires each component to work in synergy with the other components. If there is a failure in one component, there is them a breakdown of the system as a whole. This test checks to ensure that the individually tested systems work together in the system as a whole

5.2.2 Metrics

Each test will be graded on a Pass/Fail metric

5.2.3 Procedure

The start-up and control procedure of the robot system will be walked through.

First, the program will be run. The configuration file information will be entered and the program will enter the while loop. The operation will document if the video GUI shows, verifying the Visual GUI start, and if the process variables change, verifying that the configuration files worked. The operator will then engage active capture. The console output will show the inverse kinematics activity, which verifies the

inverse kinematics engine working, and the robot arm will start moving in relation to the operator's body or by command of the joystick, which verifies the robot arm output. The operator will them record a motion capture.

5.2.4 Initial Results

Test	Pass/Fail
Configuration File Test	Pass
Passes parameters	Pass
Visual GUI start	Pass
Inverse Kinematics	Pass
Computer/Robot Communications	Fail
Robot Output	Pass

Table 5-1: Initial Synergy Testing Results

5.2.5 Discussion and Problem Solving

5.2.5.1 Overview

With the program started up, the operator bravely stood in front of the Kinect and initialed active capture. Nothing responded, yet the servos were jittering (Table 5-1). The code had compiled, the algorithm worked, but the microcontroller on the robot as not responding to input. An investigation to why commenced. A program was written that communicated with the Arduino using text input from the operator.

5.2.5.2 Convert from "if" statements to "switch" cases

A timer was added to the Arduino Mega program which showed the millisecond time the loop began and when the loop finished executing. It showed an over 15ms duration of execution even though the state was 1, which was the shortest conditional. This was longer than the forced 15ms delay for the

servos. The conditionals were then changed from “if/else to “switch/else” statements. The program then ran in 2ms consistently. When testing the input/output, this did not solve either of the problems

5.2.5.3 Faster processor?

The servo twitching was theorized to be due to timing issues. If the time it takes to execute is longer than the PWM timeout, the servos could malfunction. The Arduino Mega is running at 16Mhz [15], and the Arduino Due is running at 84Mhz with a 32bit processor [16]. Most of the programming and hardware is interchangeable between the Mega and the Due. The new prototyping board was purchased, programmed, and swapped in. The servos stopped jittering, yet the microcontroller still was not responding to immediately to commands. However, the Roomba library could not be properly implemented on the Due. As the Due was a Band-Aid to the frustrating issue, and it could not drive the robot mobile base, the design turned back to the Mega. The problem was eventually solved by tweaking some conditionals and changing the Serial parameters.

5.2.5.4 Solution- Serial flushing

The serial buffer and latency have been the most frustrating problems to overcome. There was a similar issue with the Arduino Uno and getting the values form the Wii Nunchuck. A well placed flushing of the serial buffer finally did the trick. It was hoped that the same would hold true here. It did. However, the solution was that the serial flushing had to occur in the transmitting python code and the Arduino receiving code, despite instructions being sent on string at a time, as well as shortening the serial timeout.

5.2.6 Retest

Test	Pass/Fail
Configuration File Test	Pass
Passes parameters	Pass

Visual GUI start	Pass
Inverse Kinematics	Pass
Computer/Robot Communications	Pass
Robot Output	Pass

Table 5-2: Final Synergy Testing Results

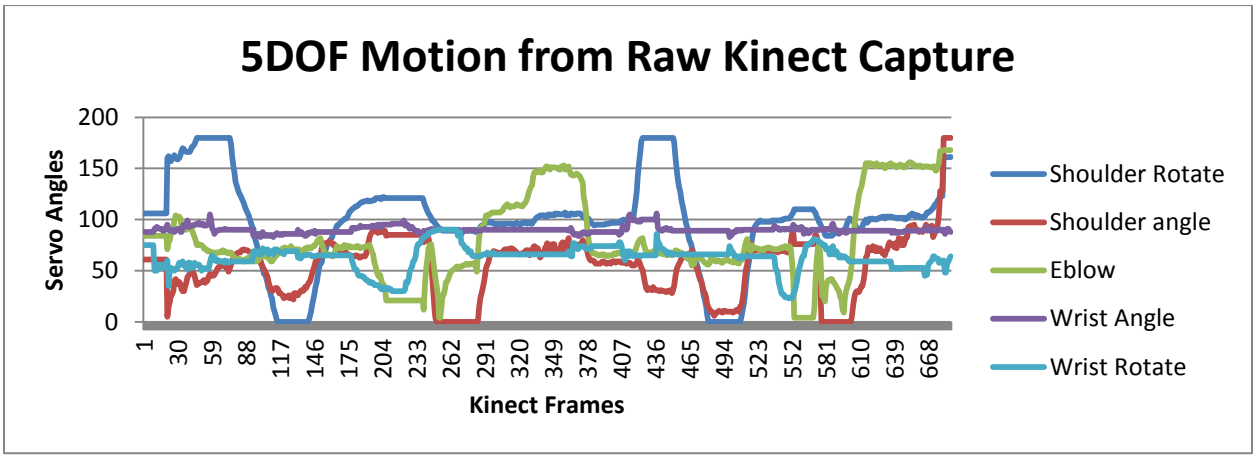


Figure 5-1: Recorded Kinect Output from Test Capture

5.2.7 Discussion

With the communication restored, the control system performed as initially expected and passed the basic start up tests, as shown in Table 5-2. The initial results of this tests show how sensitive this system is concerning the failure of certain key control. For safety, a time out sequence can be found in the robot side firmware that freezes the arm in the last known orientation, as well stops the Roomba motors. If there were issues with some skeletal tracking, as detailed in Section 3.3, the system was able to seamlessly handle it, and even was able to bring the arm back right to the base and then reach out (Figure 5-1). The motion capture recording and playback were successes.

5.3 Test 2- Stability and Dexterity

5.3.1 Overview

This test is the primary test: is the output of the robot arm more controllable and predictable with the new system? If the arm is not smoother or the operator is unable to perform even simple gestures, then the operator will not be able to manipulate objects and cannot be self-reliant. This test is an important test to the combined working relationship of the solutions produced by the above research and design.

5.3.2 Parameters and Metrics

The testing parameters will be a comparison between these capture smoothing states

- Raw value Kinect servos output + Servo PID (no smoothing)
- Smoothed Kinect servos output + Servo PID (Kinect Side Only Smoothing)
- Raw value Kinect servos output + Servo Easing (Robot Side Only Smoothing)
- Smoothed Kinect servos output + Servo Easing (Dual Smoothing)

The operator motion patterns that will be analyzed will be

- Full arm motion from left to right
- Full arm motion from right to left
- Full arm motion from down to up
- Full arm motion from up to down
- “Reach out and touch target”

The overall stability of the system, as measured from the arm, will be charted. A count and the amplitude of unnecessary movements, bounces or sways, will be recorded and compared. The system with the lowest amplitude and count will be determined to be the most “stable”.

5.3.3 Procedure

The operator will move in the following patterns:

- Full arm motion from left to right
- Full arm motion from right to left
- Full arm motion from down to up
- Full arm motion from up to down
- “Reach out and touch target”

These motions will be recorded. The appropriate smoothing parameters will be adjusted on both the computer software and the robot firmware. A triple axis accelerometer and placed horizontally in the gripper. The recorded motions will then be played back for each robot and Kinect output smoothing state and the accelerometer data will be analyzed and compared.

The frames in this test are based on the 115200 bps, the highest refresh rate, on the accelerometer. The Accelerometer‘s axis position is a relative position that is internal to the accelerometer.

5.3.4 Results

5.3.4.1 Test Data

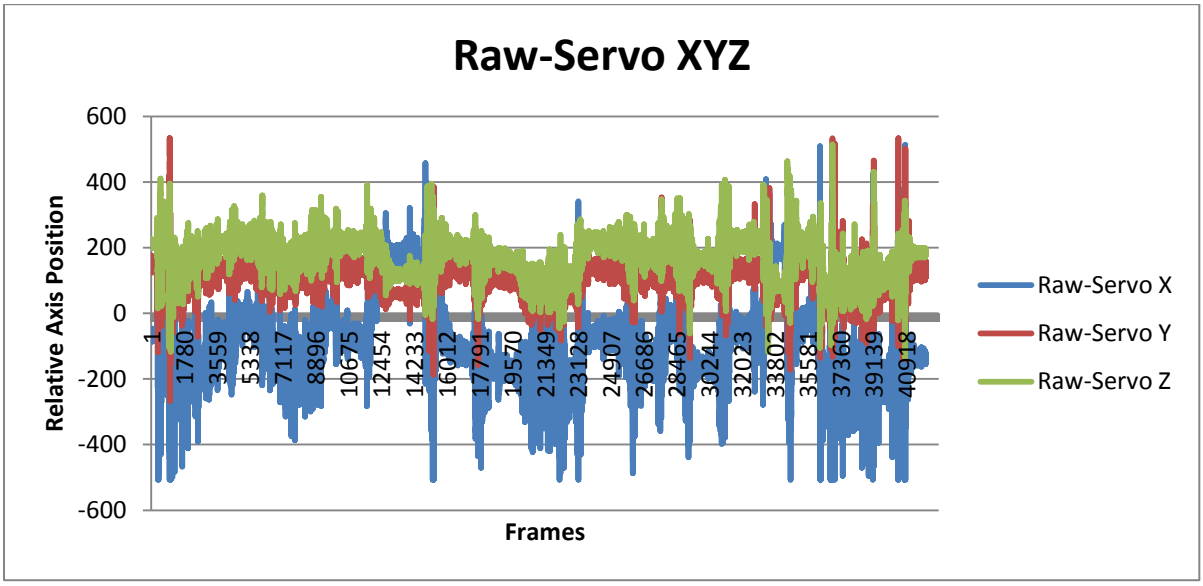


Figure 5-2: Original System Configuration (Raw Kinect and Servo PID)

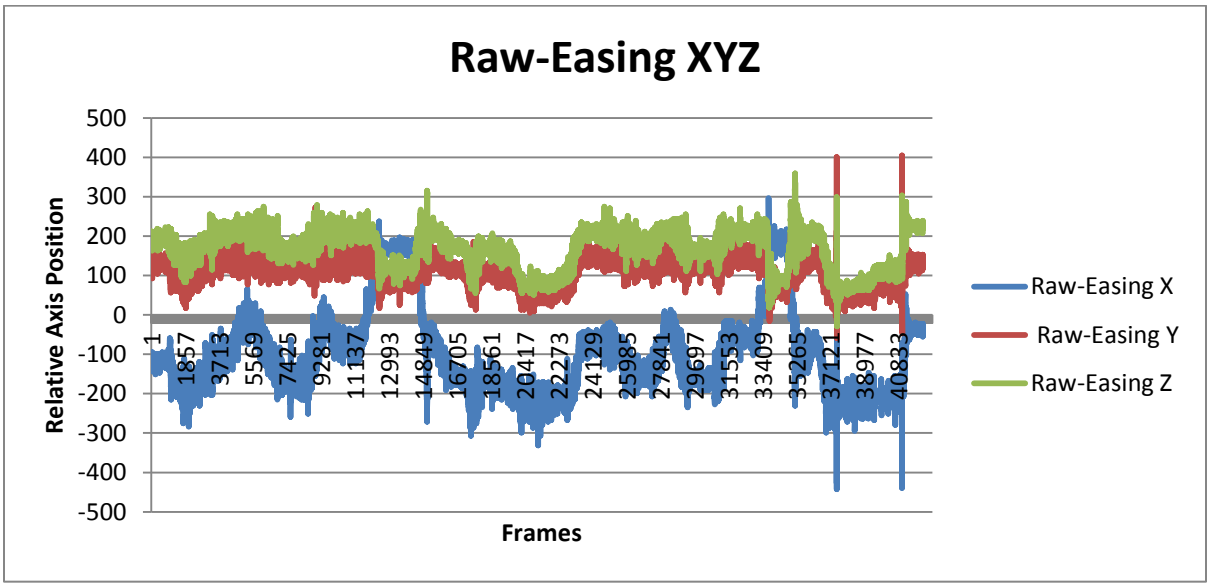


Figure 5-3: Raw Kinect and Servo Easing Algorithm Results

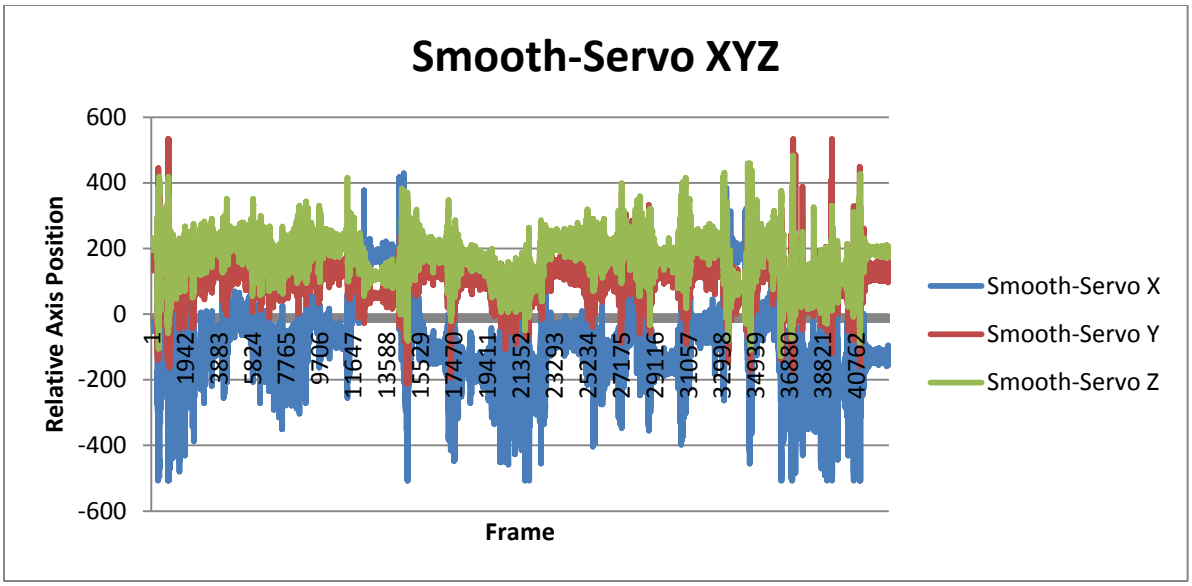


Figure 5-4: Smoothed Kinect Output and Servo PID Results

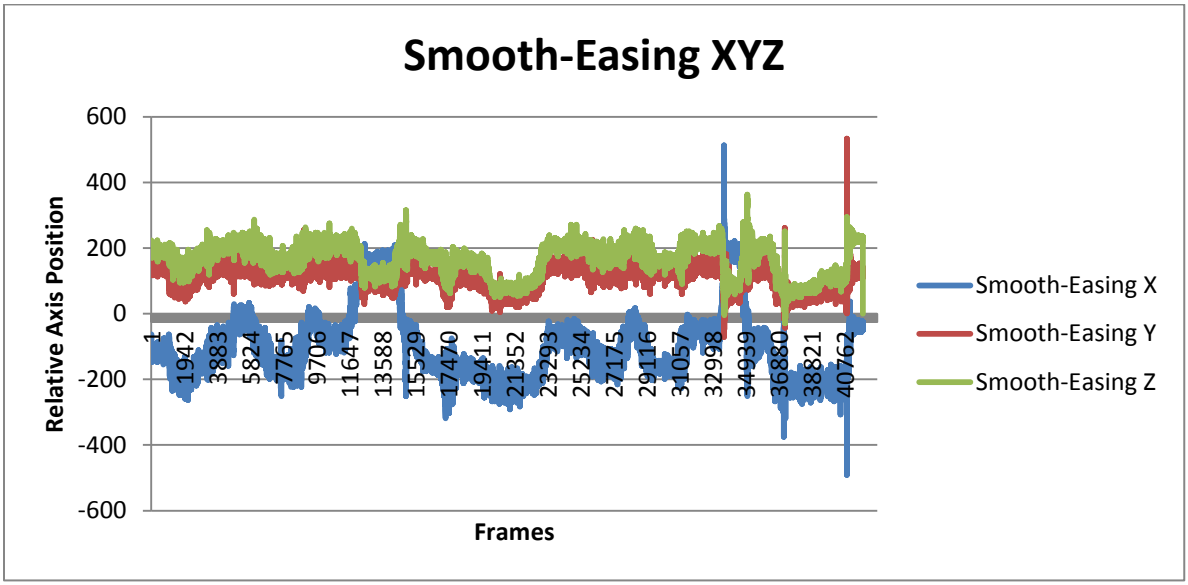


Figure 5-5: Smoothed Kinect Output and Servo Easing

5.3.4.2 X Axis

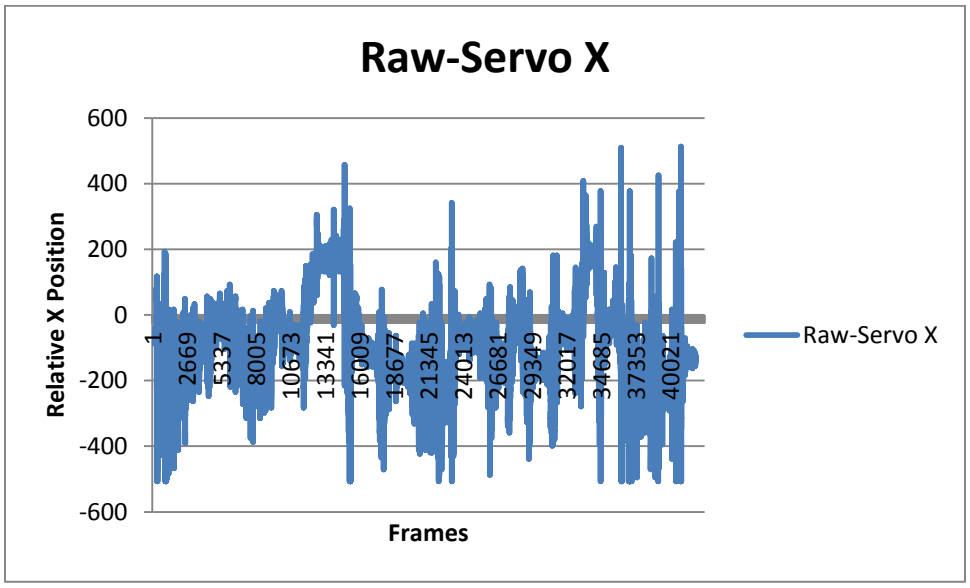


Figure 5-6: Raw Kinect Output and Servo PID X Axis

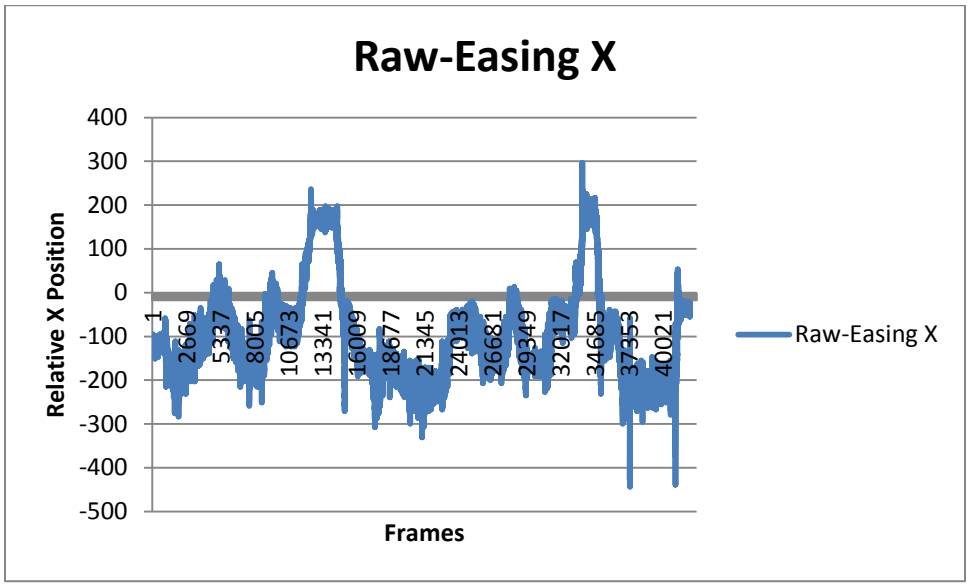


Figure 5-7: Raw Kinect Output and Servo Easing X Axis

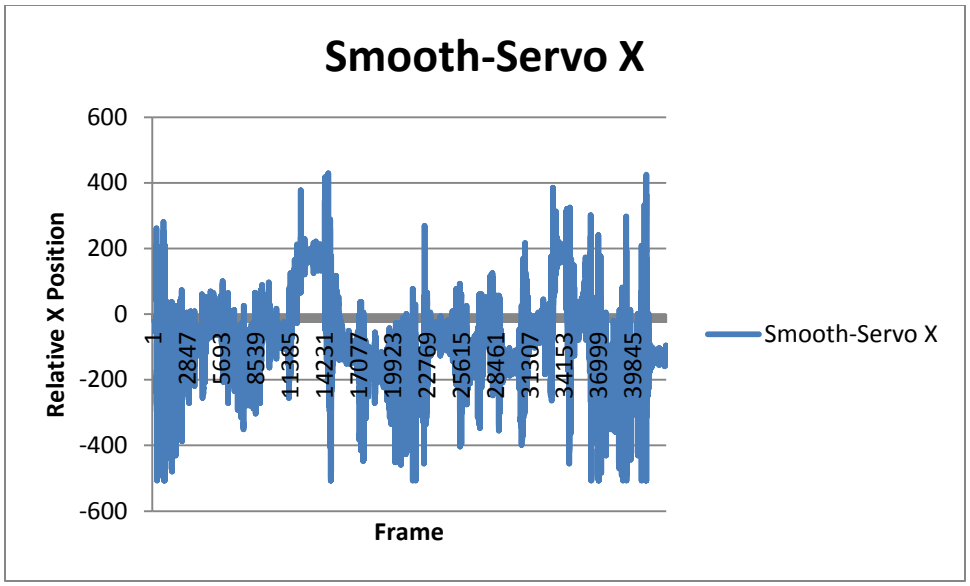


Figure 5-8: Smoothed Kinect Output and Servo PID X Axis

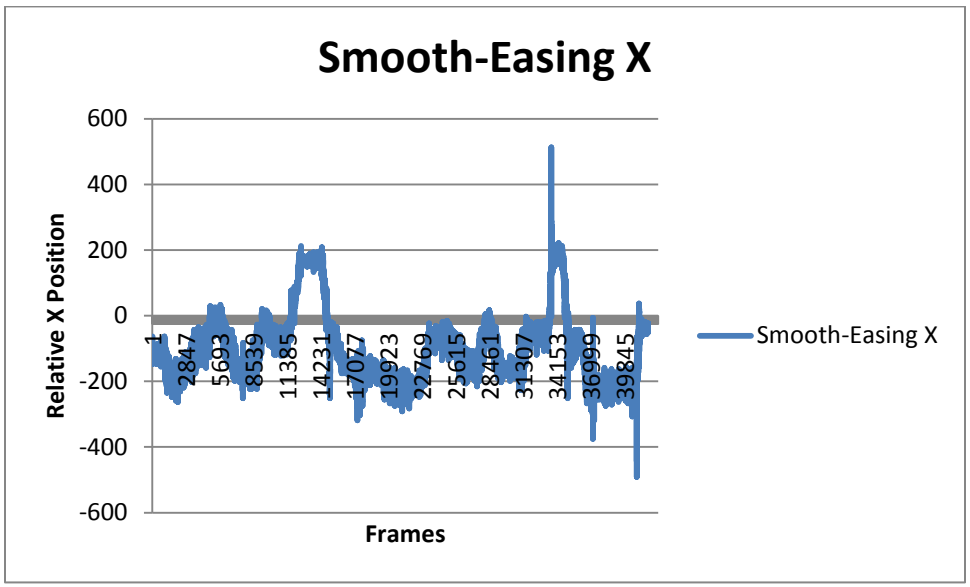


Figure 5-9: Smoothed Kinect Output and Servo Easing X Axis

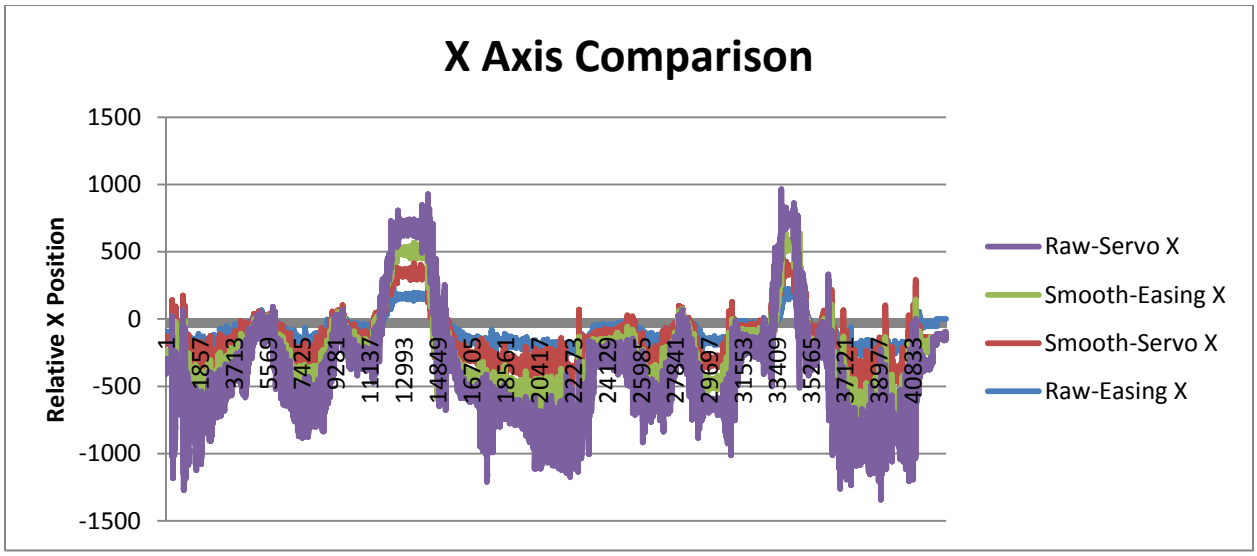


Figure 5-10: X Axis Comparison

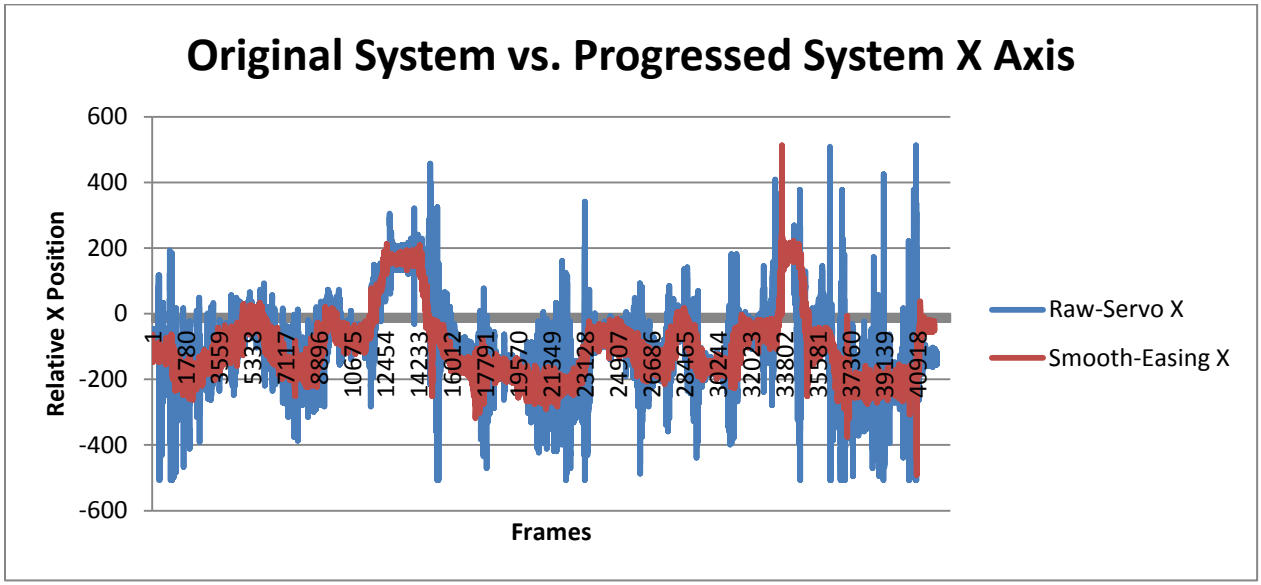


Figure 5-11: Comparison between the Original and the Progressed System Smoothing Algorithm for the X Axis

5.3.4.3 Y Axis

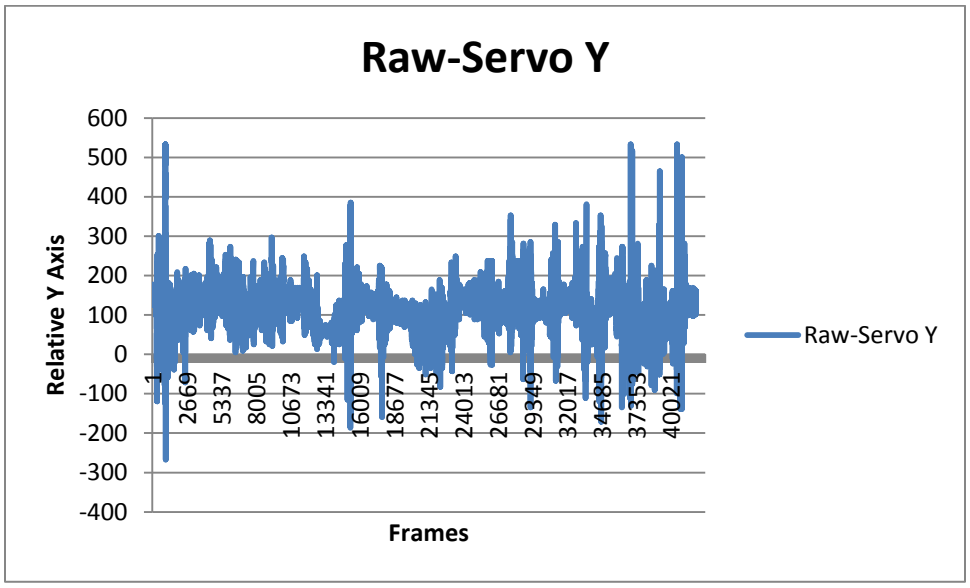


Figure 5-12:Raw Kinect Output and Servo PID Y Axis

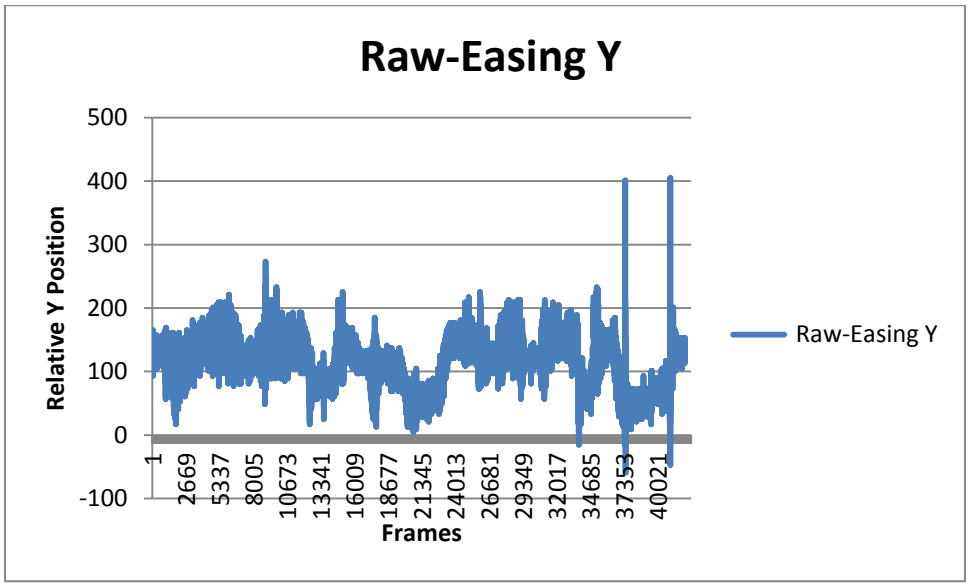


Figure 5-13: Raw Kinect Output and Servo Easing Y Axis

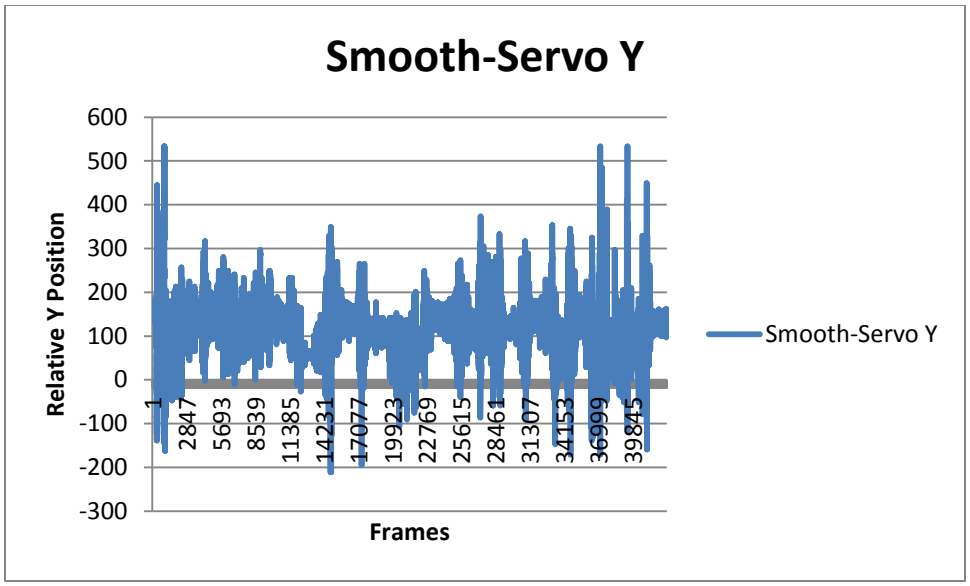


Figure 5-14: Smoothed Kinect Output and Servo PID Y Axis

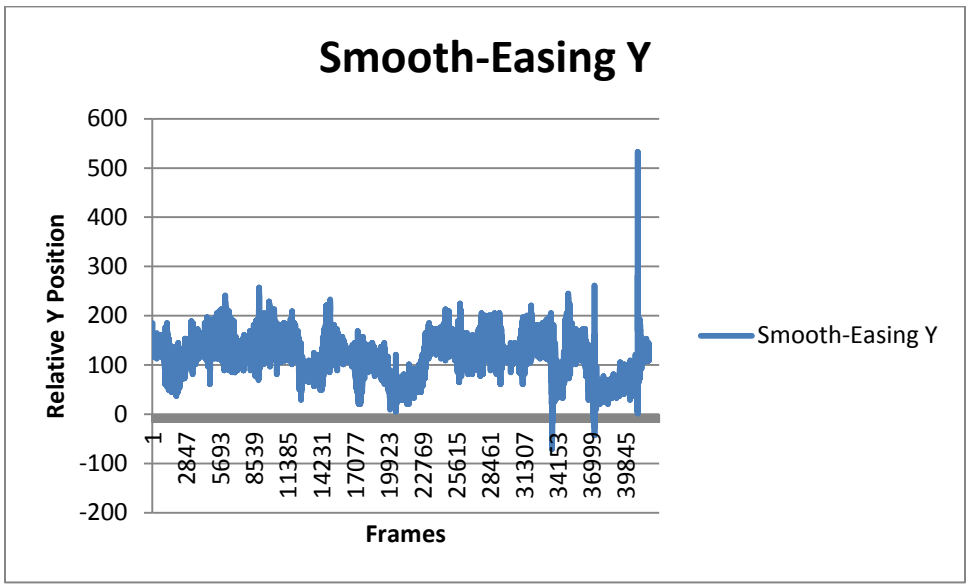


Figure 5-15: Smoothed Kinect Output and Servo Easing Y Axis

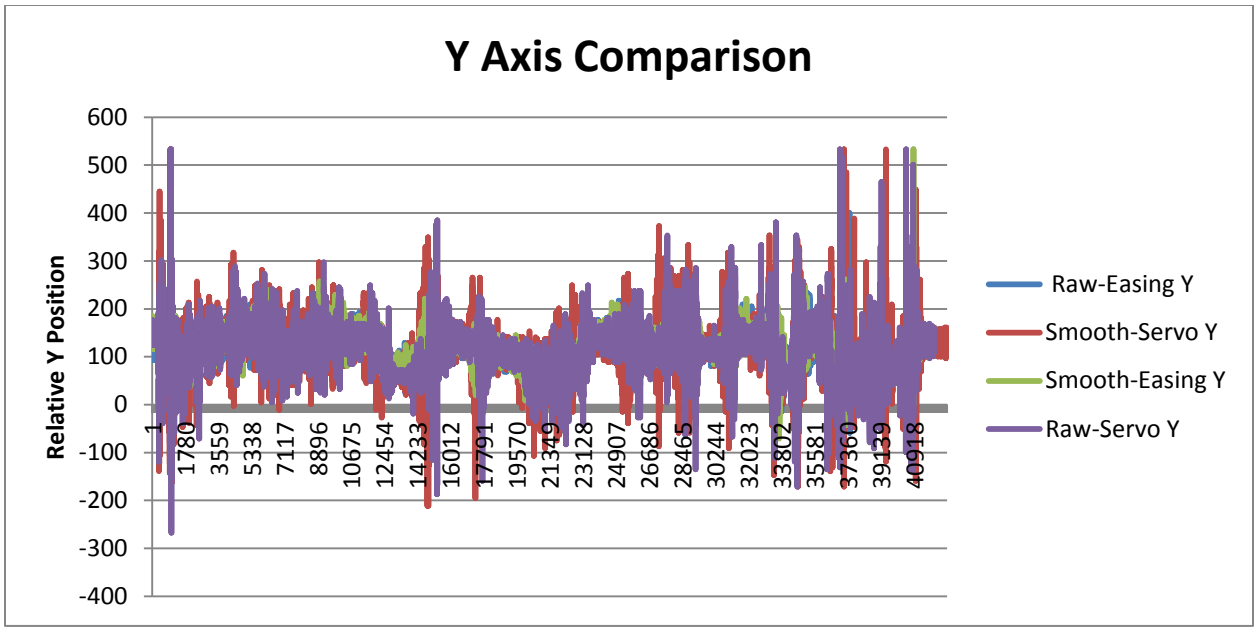


Figure 5-16: Y Axis Comparison

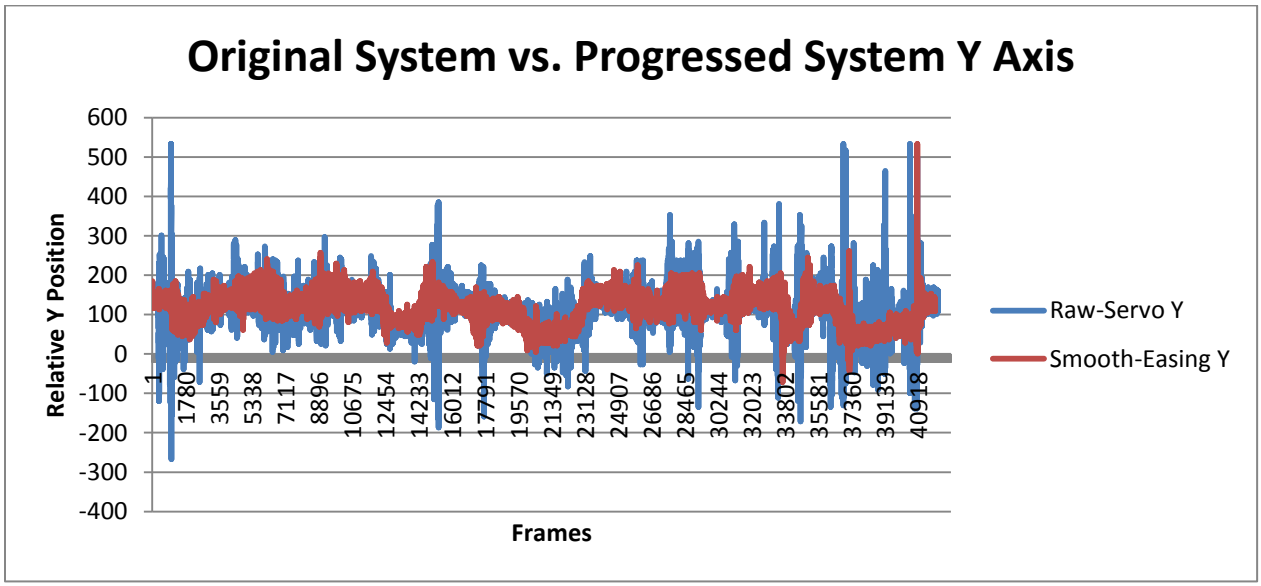


Figure 5-17: Comparison between the Original and the Progressed System Smoothing Algorithm for the Y Axis

5.3.4.4 Z Axis

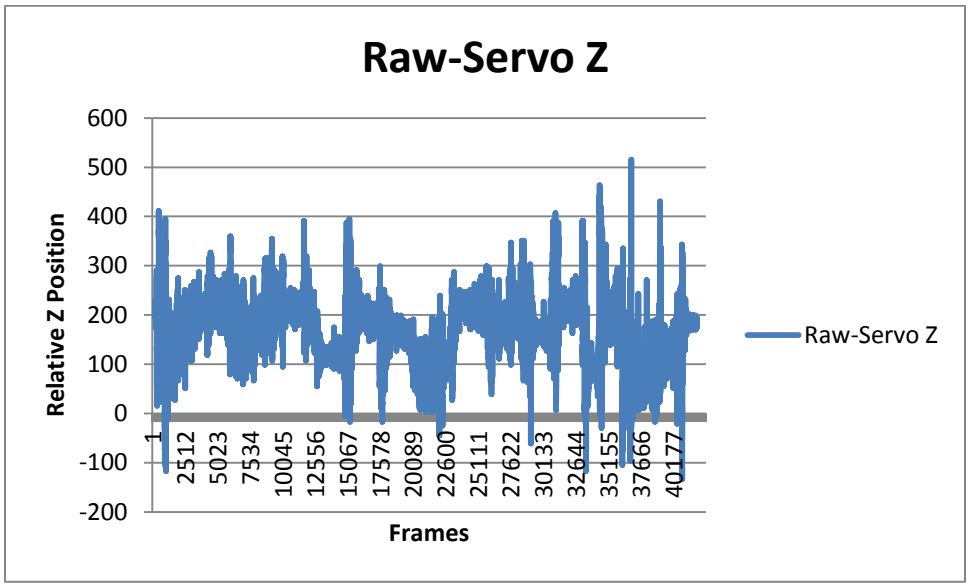


Figure 5-18: Raw Kinect Output and Servo PID Z Axis

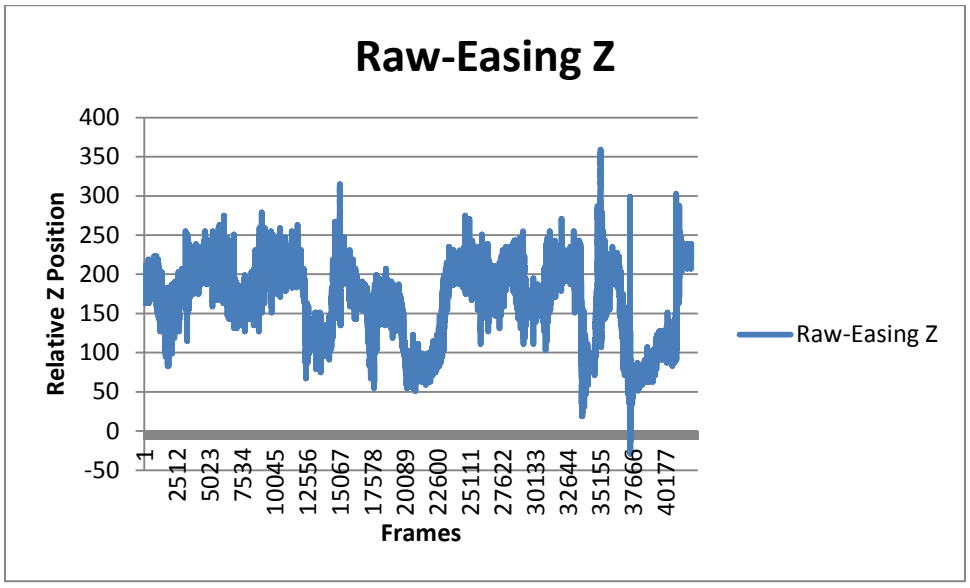


Figure 5-19: Raw Kinect Output and Servo Easing Z Axis

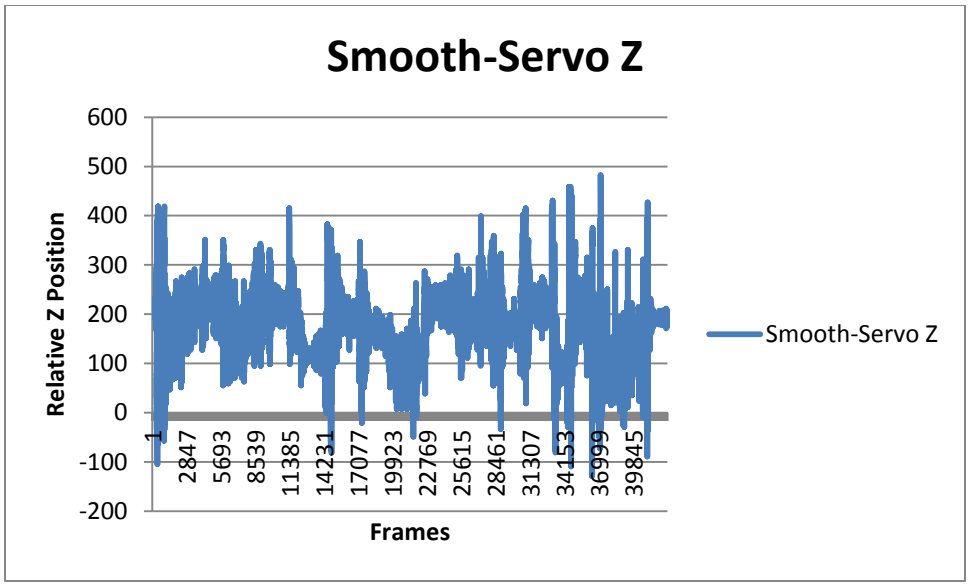


Figure 5-20: Smoothed Kinect Output and Servo PID Z Axis

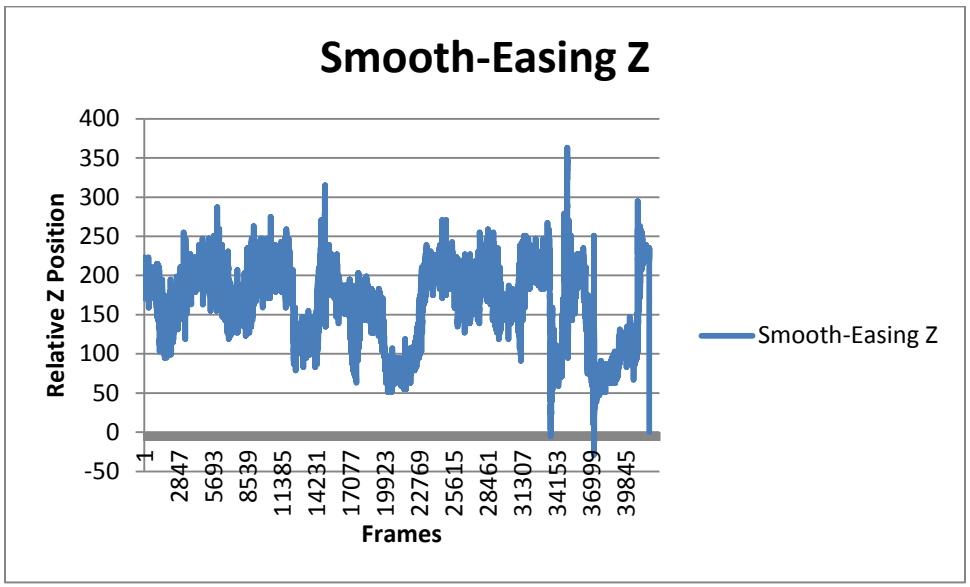


Figure 5-21: Smoothed Kinect Output and Servo Easing Z Axis

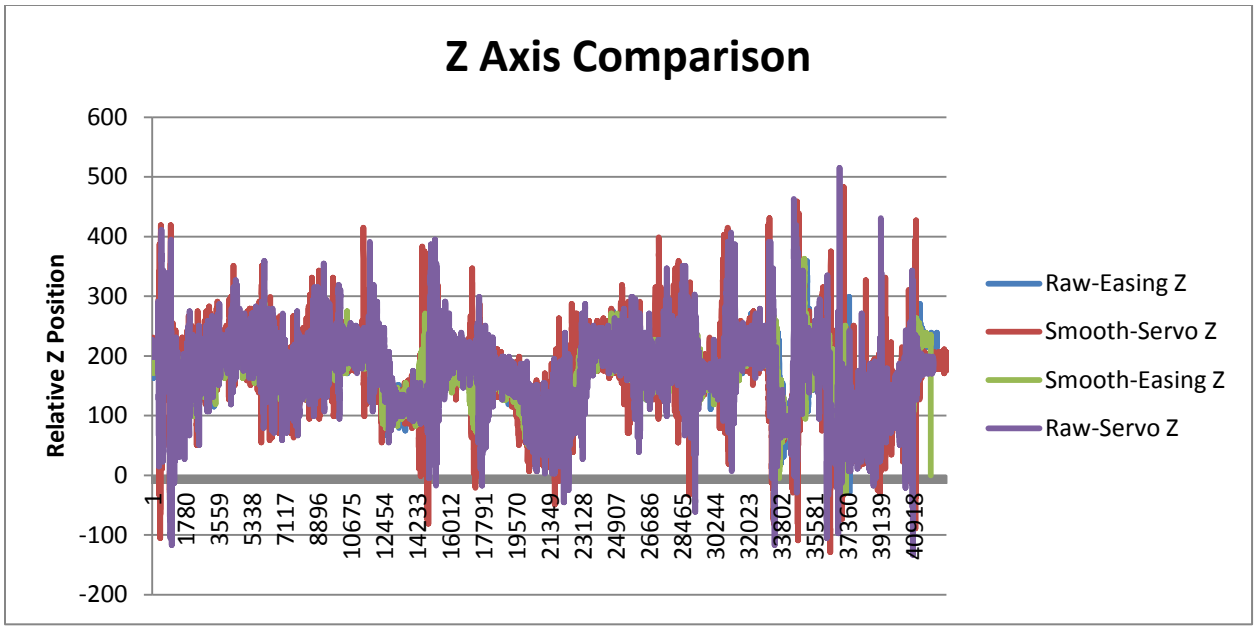


Figure 5-22: Z Axis Comparison

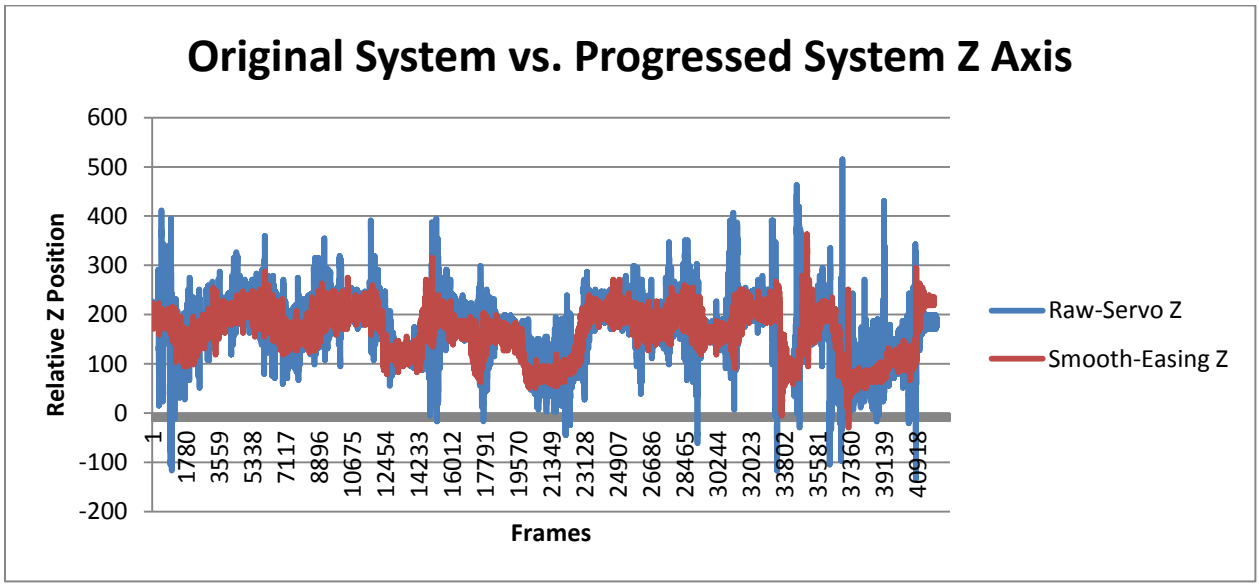


Figure 5-23: Comparison between the Original and the Progressed System Smoothing Algorithm for the Z Axis

5.3.5 Discussion

An overview of the graphical data illustrates that the X position was the most dynamic of the positions (Section 5.3.4.2). It held the most noise in motion, as well as the most movement. The data shows that any system that uses the servo easing algorithm was considerably smoother than a comparable system

that didn't. There were considerably fewer spikes in motion, as well as a tighter raw motion profile for the systems with servo easing than those without. In the Y values, as shown in Figure 5-13 and Figure 5-15, it seemed that the smoothed Kinect output added more noise than the Raw Kinect values did. This is most likely because of the easing parameters slowing down and speeding up instead of the more linear motion profile of the servo motor PID. Thus, it is not really noise, but an effect of the system motion on the accelerometer. The results concluded that the combination that had the best system response was the smoothed Kinect output coupled with the servo easing algorithm. There was a drastic improvement over the original system's performance and stability versus the new system's performance and stability (Figure 5-11, Figure 5-17, and Figure 5-23).

5.4 Test 3- Usability

5.4.1 Overview

An assistive device should be able to retrieve objects easily. In this test, the real output of the robot will be examined. Similar to the Senior Design Final Test, the robot must traverse a room and pick up and object and retrieve it for the operator. This test focuses on the effectiveness of the addition of the Visual GUI on the use of the robot.

5.4.2 Procedure

The robot will move from the starting position and, with the operator being able to directly observing the robot, the operator must pick up an object and return it and the robot back to the starting position. The test will be done with and without the Visual GUI. This is a timed test. The attempts towards grabbing the target object as well as the primary method of viewing the robot's action will be recorded.

5.4.3 Results

Test	Time	Tries	Primary Visual Usage
Original System	82 seconds	4	At Robot
With Visual GUI	57 seconds	2	At GUI

Table 5-3: Visual GUI Effectiveness Test Results

5.4.4 Discussion

With the kinks worked out in Test 1 and the verified stability shown in in Test 2, the robot arm performed as expected. The new found system stability greatly increased the operator's ability to execute the task, as the target was quickly found and retrieved with fewer tries and a faster time than without the Visual GUI, as shown in Table 5-3.

It was found best to turn of the direct control of the system while moving and then use it when you had approached the target. This allowed the camera's built in latency to not affect operation, and was easier on the operator in terms on necessary effort and concentration, as they did not have to hold a position and could move around freely, yet guide the robot with the joystick.

5.5 Overall Discussion

After getting the system to properly function in synergy, data was able to be gleaned concerning the usability of the system.

In terms of comparing the smoothing algorithms and their effectiveness,

- Smoothing of Kinect Values + Servo Easing produced the best results
- Servo Easing produced the most dramatic improvement over the controllability and stability of the system.
- Smoothing of Kinect Values had a marginal, yet noticeable improvement over the Raw Kinect Values

The system, concerning the effectiveness of the Visual GUI,

- Using Smoothing of Kinect Values + Servo Easing, and the Visual GUI was able to retrieve the target object faster than without either system.
- The Visual GUI was the most effective improvement in retrieving items due to the increased effective hand eye coordination

Chapter 6

Further Work

6.1 Conclusion

The goal of taking this proof of concept assistive control system and maturing it into a more capable and effective device was satisfied. The ease of performing complex kinematics on the test bed platform opens the door for a new breed of assistive household robots to be researched, developed, and produced. These robots can potentially bring self-reliance to those that are to dependents, and with that comes pride, less depression, and an overwhelmingly higher quality of life for the demography of operators.

To accomplish this goal, a design methodology was used. Research and development of tools, such as the recording system and the output simulator, were necessary to facilitate the successful completion of the assigned goals. The research included testing and parametric analysis of existing "natural interaction" software, embedded systems design and microcontroller programming methodologies, and smoothing techniques on existing systems. The designing included writing the computer side software, controller firmware, and getting these devices to work well together to create something greater than the sum of its parts.

Testing showed that the design implementations worked well together, but had mixed initial results when working in synergy. The OpenNI SDK+NiTE + Python solution was found to still be the SDK of choice for development and deployment of the control system for 3 reasons:

1. it's skeletal joint tracking system was superior that the Official Kinect SDK for the direct control required by this control system

2. It cross-compiles with more operating systems, as do the supporting libraries
3. It works with the growing range of depth sensor, not limiting the system builder to just the present Hardware.

A 3-period, moving, weighted averaging scheme was found to produce the best results for smoothing the inverse kinematics results of the raw inverse kinematics results, in terms of response time, small motion precision, and overall error. It was successfully incorporated into an algorithm on the computer side program and greatly reduced the noise on the output's motion profile. On the robot side, the easing algorithm was successfully incorporated into the microcontroller's firmware in conjunction with using a decision tree. The visual feedback made it easier to control the arm outside of line of sight and gave the operator necessary information about the system while using it. The configuration file allowed the average operator, and researcher, to take advantage of the universal nature of this control system and set their robot parameters to work with the system.

In full system testing, there were some issues that were eventually overcome. The trouble with the serial latency, the timing of the video display function, and its helper functions, to not slow down the system and keep it as real time as possible, and still be informative, and the incorporation of the easing algorithm were all eventually solved. The results of the test were

- The smoothing of both the Kinect output data and the servo easing algorithms considerably improved system stability and performance in comparison to the original system. This combination was also the best improvement.
- The Servo Easy algorithm was the most effective algorithm to use in making the output more stable.
- The Visual GUI improved the time duration and reduced the operator efforts in the retrieval of the target object

While this is not a fully featured and completed final product, it is a vast improvement over the proof of concept in terms of system usability. Further work into looking at more complex implementations of smoothing or better performing inverse kinematics engines can be done, but the goal of the project scope were reached at this point.

6.2 Known Issues

6.2.1 Auto-Start Camera

In Windows XP, Ubuntu, and OSX, the first camera that is available will start. In windows 7 and Windows 8, there is a drop down box form the Video Source. There is no known way around that in this implementation. Further study should be done to get it working.

6.2.2 Compiling OSCeleton

Visual Studio 2008 or 2010 are required to compile OSCeleton. Visual Studio 2012 gives errors. Building from the source code using CMake in any OS has not been tested. CMake or Cygwin may e necessary. Further investigation is needed.

6.2.3 OpenNI2

This "Open Source and Universal" system will not work with the OpenNI2 SDK. The reason being is that OSCeleton only compiles with version 1. The work around is to use NI Mate, a trial based software that costs £198 after 30 days and has given trouble when renewing the free trial license. It will provide the same OSC interface as the free OSCeleton. Email conversations with the CEO of Delimate, Julius Tuomisto, on August 4th showed little desire to implement a cheaper version of the software due to current overhead. Therefore, either a free or cheap alternative needs to be found or further work is needed to get OSCeleton to compile and run with OpenNI2.

6.3 Two Arms

The Kinect, and this code, is capable of tracking both arms. The code is already there The necessary extra changes to the code will take about 5 minutes of coding, and a bit of testing, or a bit longer if the link lengths are different. The second arm will allow for a greater flexibility of tasks and open up the capabilities of remote activities such as moving complex objects, cooking, and other two handed tasks. While the further work depth in designing the system is pretty shallow, plus a two armed robot is necessary, the research potential into that interaction is still vast.

6.4 Different Drive Systems

Only basic tests were done to see and implement a feasible universal system. Steppers and large scaled DC motors may be the drive of choice in a real world system. Further work needs to be done to ensure that it is 100% compatible for those systems.

Testing on the system with digital servos, instead of analog, for mitigating disturbances, should be done. The digital servos have many benefits, from higher holding torques, to faster and better response times. They also have some consequences, like higher power requirements. The servo motors on the observed mounted mobile robot arms at Stony Brook University and all but one servo on the proof of concept test bed robot were analog. Experiments in seeing if a conversion from analog to digital servos will produce better results should be a next step activity for continued work with the available test bed mobile robots.

There are also drop-in, open servo motor controllers that provide feedback from the OpenServo project. These will be useful for experimentations and better for conducting further research. They should be considered for next-gen test beds, if it is servo based.

6.5 Hand device

While the Wii Nunchuck Controller is a quite capable, readily available, and cheap device, it is not without its shortcomings for this project. As there is an Arduino Uno attached to the controller, a variety of different enhancements can be added only hardware alterations to the basic handheld piece. These additions are

- Feedback gripper trigger based on the servo implementation
- Additional Buttons for greater range of "in capture mode" control

6.6 Other Sensors

6.6.1 Asus Xtion series

The Xtion and Xtion Pro Live are similar depth sensor devices powered by the same company that the Kinect uses: PrimeSense. The OpenNI SDK is PrimeSense's SDK, so it will work as a drop in replacement with the Kinect. This sensor boasts a faster frame rate and two options for depth resolution. Its application as a possible depth sensor replacement should be investigated. [11]

6.6.2 Kinect 2

The Kinect 2 is on the horizon and coming out for Windows developers in November 2013 for $399 (hopefully cheaper academically). Unlike the Xbox 360 Kinect, the Xbox One's Kinect 2 will be unable to be hacked there will be no capability to plug it into your computer It boasts performance enhancements that might solve all of the problems with joint detection and tracking that are currently being experienced. It is 3 times as sensitive, a much better skeleton tracking system that is capable of now truly detecting hands and shoulder shrugs. It now also has orientation tracking built in, lowering processing requirements on that. It also has new monitors and capabilities like facial recognition, muscle stressing, and heart beat sensor which can be used to alter the sensitivity of the Kinect as people

tend to focus more and become more tense [10]. Its application as a possible depth sensor replacement should be investigated after its introduction into the marketplace.

6.7 Embedded system

The Raspberry Pi is becoming an extremely popular development platform. It has a HDMI and RCA Video out, and runs Debian Linux, and is capable of running OpenNI, Python, and can communicate with the Arduino Prototyping boards. It also can use the Asus xTion series depth sensor, which now licenses the same PrimeSense's technology that enables skeleton tracking on the Microsoft Kinect. Due to the success of control system's universal hardware and software versatility of the OpenNI+Python SDK, this system, in its current form, can be ported quite easily onto this platform. This will allow for an all-around smaller, embedded version of the testing platform, with only some frames per second compromises.

6.8 GUI

A good, powerful, clean user interface is necessary for ease and efficiency of use. The text based configuration system can be drop box based. The parameters and variables can be always in the user's field of view. The operator can potentially view the Video streams of both the robot FPV and the Kinect's RGB-D and Skeleton at the same time, allowing for ease of use. Also, using OpenNI, LibFreeNect, and the operating system's speech recognition software, the implementation of verbal commands and full control of the Kinect's tilt motor can be achieved.

6.9 Recording

In ways were made into recording the servo output stream and playing it back to control the robot. These can be made into separate files and saved for different tasks, like "Open refrigerator", or "Pick up cup". If the operator requires a more automated task, where all the variables are pretty constant, they can replay the task over and over. Using a speech recognition program, they can call these recordings

into action without having to stress themselves. This is especially convenient for those who require the arm to do a motion that they themselves cannot. Another, more able-bodied, person can be recorded and then the operator can simply call the action when necessary.

6.10 Computer Vision

As this implementation uses OpenCV, which can tie directly into most video streams, including the Kinect's through OpenNI, a host of computer vision based assistive algorithms can be introduced. For example, object detection, where you can select an object on the screen and the system automatically retrieves it, obstacle detection, or even filters to allow for easy remote home surveillance- an important thing to have if you are unable to get out of bed. Other power tools can be created when coupled with the recording capability, such as adaptive playback of picking up an object, opening the fridge door, or even self-navigation back to the point of origin for easy retrieval purposes can be achieved. This is a rich area for further study.

6.11 Better Predictive Algorithms

Some of the potential operators are those with a spastic muscle disorder. This does not allow them to walk very well, but they still have some use of their arms. This special group of operators needs to be addressed, as there may be a way to give them an effective control system without the need of direct control. This may require an adaptive algorithm and enhanced computer vision routines in order to "guess" what that operator is trying to retrieve.

Works Cited

[1] WormFood, "AVRBaudCalc," [Online]. Available: http://www.wormfood.net/avrbaudcalc.php.

[2] K. Khoshelham, "ACCURACY ANALYSIS OF KINECT DEPTH DATA," in *ISPRS Calgary 2011 Workshop*, Calgary, 2011.

[3] Microsoft.com, "Kinect Sensor Setup, Requirements, Support | Kinect for Windows," [Online]. Available: http://www.microsoft.com/en-us/kinectforwindows/purchase/sensor_setup.aspx. [Accessed 16 November 2012].

[4] Kinect for Windows Team, "Near Mode: What it is (and isn't)," 20 January 2012. [Online]. Available: http://blogs.msdn.com/b/kinectforwindows/archive/2012/01/20/near-mode-what-it-is-and-isn-t.aspx?Redirected=true. [Accessed 30 November 2012].

[5] Msdn.microsoft.com., "Tracking Skeletons in Near Depth Range," 2013. [Online]. Available: http://msdn.microsoft.com/en-us/library/hh855353.aspx. [Accessed July].

[6] Microsoft Research Cambrigde & Xbox Incubation, "Real-Time Human Pose Recognition in Parts from Single Depth Images," 2011. [Online].

[7] Microsoft, "http://www.microsoft.com/en-us/kinectforwindows/news/faq.aspx," 2012. [Online]. [Accessed 30 November 2012].

[8] Openni.org, "OpenNI SDK Release Notes | OpenNI," 2012. [Online]. Available: http://www.openni.org/openni-sdk/openni-sdk-release-notes/.

[9] Openni.org, "NiTE 2.2 | OpenNI," 2012. [Online]. Available: http://www.openni.org/files/nite/.

[10] K. Wagner, "Kinect 2 Full Video Walkthrough: The Xbox Sees You Like Never Before," Gizmodo, 21 May 2013. [Online]. Available: http://gizmodo.com/kinect-2-full-video-walkthrough-the-xbox-sees-you-like-509155673 . [Accessed 27 July 2013].

[11] Autoponics.org, "Switching from Kinect to Asus Xtion Pro Live | Autoponics," 13 March 2012. [Online]. Available: http://autoponics.org/?p=160. [Accessed 18 July 2013].

[12] W. Thompson, T. Dyer and Y. Zhou, "Improved Control System using Natural Human Pose Imaging for Robotics and Telepresence Assistive Devices," Simon's Fellowship Presentation, Stony Brook, 2012.

[13] Docs.opencv.org., "HighGUI — OpenCV 2.4.6.0 documentation," 14 July 2011. [Online]. [Accessed 17 May 2013].

[15] Arduino.cc, "Arduino - ArduinoBoardMega2560," Arduino.cc, 2010. [Online]. Available: http://arduino.cc/en/Main/ArduinoBoardMega2560 . [Accessed 3 August 2013].

[16] Arduino.cc, "Arduino - ArduinoBoardDue," Arduino.cc, 2012. [Online]. Available: http://arduino.cc/en/Main/ArduinoBoardDue. [Accessed 2013 August 3].

[17] E. M. Berti, A. J. S. Salmerón and F. Benimeli, "Kalman Filter for Tracking Robotic Arms Using low cost 3D Vision System," in *ACHI 2012 : The Fifth International Conference on Advances in Computer-Human Interactions*, Valencia, 2012.

[18] A. P. L. Bo, M. Hayashibe and P. Poignet, "Joint Angle Estimation in Rehabilitation with Inertial

Sensors," in *Engineering in Medicine and Biology Society*, Boston, 2011.

Appendix A- Copyright Permission

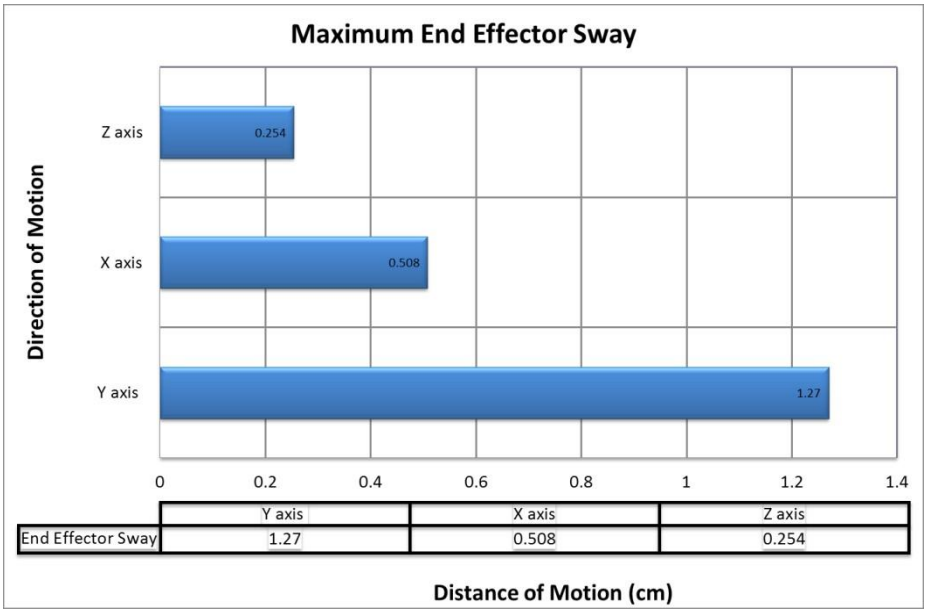
34 Morris Ave,
Huntington, NY, 11743

8//15/13

616 54th Street
Brooklyn NY, 11220

Dear William Thompson,

Hey buddy, I hope you are enjoying packing for college. I am praying only for the best for you! As per our conversation online that is taking place right now, and only mentioned because they are telling me to mention it, I am completing a doctoral dissertation at Stony Brook University entitled "IMPROVED USABILITY FOR A CONTROL SYSTEM USING HUMAN BODY IMAGING AS THE CONTROL METHOD FOR ARM BASED ROBOTICS IN ASSISTIVE TECHNOLOGIES". I would like your permission to reprint in my dissertation excerpts from the following: The citation is that of these two images of the disturbances from the control system that we worked, as follows:



The requested permission extends to any future revisions and editions of my dissertation, including non-exclusive world rights in all languages, and to the prospective publication of my dissertation by UMI. These rights will in no way restrict republication of the material in any other form by you or by others authorized by you. Your signing of this letter will also confirm that you own [or your company owns] the copyright to the above-described material. If these arrangements meet with your approval, please sign this letter where indicated below and return it to me in the enclosed return envelope. Thank you very much. Stay in touch!

Sincerely,

Taurean Dyer

PERMISSION GRANTED FOR THE
USE REQUESTED ABOVE:

William Thompson Date:
____8/15/2013___