

7-2012

Design and Implementation of the Kinect Controlled Electro-Mechanical Skeleton (K.C.E.M.S)

Jason Richard Ekermann

Embry-Riddle Aeronautical University - Daytona Beach

Follow this and additional works at: <https://commons.erau.edu/edt>



Part of the [Mechanical Engineering Commons](#)

Scholarly Commons Citation

Ekermann, Jason Richard, "Design and Implementation of the Kinect Controlled Electro-Mechanical Skeleton (K.C.E.M.S)" (2012). *Dissertations and Theses*. 63.

<https://commons.erau.edu/edt/63>

This Thesis - Open Access is brought to you for free and open access by Scholarly Commons. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of Scholarly Commons. For more information, please contact commons@erau.edu.

Design and Implementation of the Kinect Controlled Electro-Mechanical Skeleton (K.C.E.M.S)

by

Jason Richard Ekermann

A Thesis Submitted to the College of Engineering Department of Mechanical Engineering in
Partial Fulfillment of the Requirements for the Degree of
Master of Science in Mechanical Engineering

Embry-Riddle Aeronautical University
Daytona Beach, Florida
July 2012

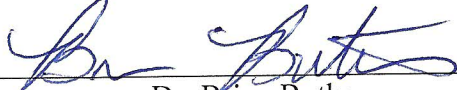
Design and Implementation of the Kinect Controlled Electro-Mechanical Skeleton (K.M.E.S)

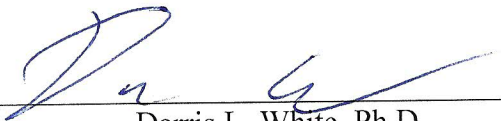
by

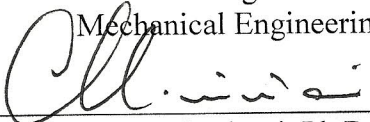
Jason Richard Ekkelmann

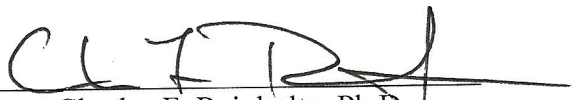
This thesis was prepared under the direction of the candidate's Thesis Committee Chair, Dr. Brian Butka, Professor Electrical Engineering Department, and Thesis Committee Members Dr. Darris White, Professor Mechanical Engineering Department and Dr. Charles Reinholtz, Department Chair, Mechanical Engineering, and has been approved by the Thesis Committee. It was submitted to the Department of Mechanical Engineering in partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering

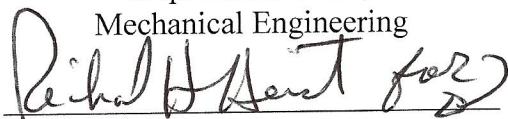
Thesis Review Committee:


Dr. Brian Butka
Committee Chair


Darris L. White, Ph.D.
Committee Member
Graduate Program Chair,
Mechanical Engineering


Maj Mirmirani, Ph.D.
Dean, College of Engineering


Charles F. Reinholtz, Ph.D.
Committee Member
Department Chair,
Mechanical Engineering


Robert Oxley, Ph.D.
Associate Vice President of Academics

8/14/12
Date

Acknowledgements

I would like to thank the following people: Dr. Brian Butka, Dr. Darris White, Dr. Charles Reinholtz, Dylan Rudolph, Siri Karlsen, Bill Russo, and my family.

Dr. Butka has been motivating me to improve myself as an engineer since the day he walked into the robotics lab my sophomore year. Through his motivation and support, I have been able to succeed on many projects including this thesis and the 3 papers we co-authored. Dr. White was instrumental in my time as a graduate student at Embry-Riddle by not only allowing me into the program, but also giving me a chance to work on one of his projects which has now given me my second job. Dr. Reinholtz convinced me to work in the robotics lab the spring of my sophomore year and I have been working with robots ever since. These three professors paved the road for me becoming the engineer I am today.

Without the help of Dylan Rudolph the project would not be where it is today. His work on the software side of this project has been invaluable to its current progress. Bill Russo allowed me to work many hours in the machine shop running the hand mills so that I could fabricate the necessary pieces for this project.

Sheri and Michael Ekelmann have been a huge support throughout my entire life and have always told me to do better than they did; without their support I would not be where I am today. Cassandra Ekelmann has been a source of knowledge when it comes to writing and had a large role in proofing this thesis. Siri Karlsen has been my source of support every day and motivated me when I was unmotivated. Without her I would not have accomplished as much as I did in graduate school.

Abstract

Researcher: Jason Richard Ekelmann

Title: Design and Implementation of the Kinect Controlled Electro-Mechanical Skeleton (K.M.E.S)

Institution: Embry-Riddle Aeronautical University

Degree: Master of Science in Mechanical Engineering

Year: 2012

Mimicking real-time human motion with a low cost solution has been an extremely difficult task in the past but with the release of the Microsoft Kinect motion capture system, this problem has been simplified. This thesis discusses the feasibility and design behind a simple robotic skeleton that utilizes the Kinect to mimic human movements in near real-time. The goal of this project is to construct a 1/3-scale model of a robotically enhanced skeleton and demonstrate the abilities of the Kinect as a tool for human movement mimicry. The resulting robot was able to mimic many human movements but was mechanically limited in the shoulders. Its movements were slower than real-time due to the inability for the controller to handle real-time motions. This research was presented and published at the 2012 SouthEastCon. Along with this, research papers about the formula hybrid accumulator design and the 2010 autonomous surface vehicle were presented and published.

Table of Contents

	Page
Thesis Review Committee	ii
Acknowledgements.....	iii
Abstract	iv
List of Tables	viii
List of Figures	ix
Overview of Thesis	xi
I Introduction.....	1
Significance of the Study	4
Statement of the Problem.....	5
Purpose Statement.....	5
Delimitations.....	5
Limitations and Assumptions	6
Definition of Terms.....	6
List of Acronyms	6
II Review of the Relevant Literature	8
Markerless Motion Capture	8
Motion Capture Using Joint Skeleton Tracking and Surface Estimation	9
MOCA: A Low-Power, Low-Cost Motion Capture System Based on Integrated Accelerometers	11
Altitude Control of a Quadrotor Helicopter Using Depth Map from Microsoft Kinect Sensor	12
Study on the Use of Microsoft Kinect for Robotics Applications	13

	3D Image Reconstruction and Human Body Tracking Using Stereo Vision and Kinect Technology	14
	Accurate Simulation of Hip Joint Range of Motion	15
	Accuracy Analysis of A Novel Humanoid Robot Shoulder Joint ..	16
	Summary	17
	Hypothesis [Research Question or equivalent]	18
III	Methodology	19
	Mechatronics	19
	Construction	21
	Actuator Selection	22
	Bracket Design and Selection	24
	Joint Design	25
	Elbow Design	28
	Knee Design	29
	Hip Design	31
	Shoulder Design	35
	Final Construction	42
	Sensor Selection	43
	Microcontroller Selection	45
	Software	48
	Elbows and Knees	49
	Hips and Shoulders	50
IV	Results	53
	Functional Demo	53

	Reliability Testing.....	60
	Cost Analysis	63
V	Discussion, Conclusions, and Recommendations.....	65
	Conclusions.....	65
	Recommendations.....	65
	References.....	67
Appendices		
A	Source Code.....	70
B	Publications.....	#

List of Tables

	Page
Table	
1 Total system cost.....	64
2 System Ranges	65

List of Figures

		Page
Figure		
1	A body suit used for professional grade motion captures	1
2	Approach to capturing the motion of interactive characters even in the case of close physical contact	2
3	Overview of processing pipeline	9
4	Input image, adapted mesh overlay, and 3D model.....	10
5	Combined method for 3D image reconstruction.....	15
6	Simulation of hip joint range of motion of 3D surface models of bones.....	16
7	The prototype of the shoulder joint on humanoid robot	17
8	A physical skeleton showing the joints targeted in this research.....	19
9	Physical skeleton plastic model on its stand.....	22
10	The Dynamixel AX-12A.....	23
11	Robotic arm built using the AX-12A's and brackets.....	24
12	Robotis bracket design for the AX-12A	25
13	Simple shoulder movements	26
14	Bioid robot used for the small-scale prototype.....	27
15	Movements possible from the elbow joint.....	28
16	CAD model of the elbow joint and upper arm structure.....	28
17	Rotational limits of the knee	29
18	CAD model of the lower leg.....	30
19	Different motions made by the hip joint	31
20	CAD model of the hip joints.....	32

21	Different motions made by the shoulder joint	36
22	CAD model of shoulder joint design 1	37
23	CAD model of shoulder joint design 2	39
24	CAD model of shoulder joint design 3	40
25	CAD model of the final build minus the base and the skeleton body.....	43
26	A 15-point skeletal model and Microsoft Kinect sensor	44
27	The Arbotix microcontroller selected as the controller for this research.....	46
28	Skeletal points imposed on Vitruvian Man.....	48
29	Completed super structure minus the skeleton body	55
30	Final elbow joint construction	56
31	Final knee joint construction.....	56
32	Final hip joint construction	57
33	Final shoulder joint construction	59

Overview of Thesis

The following document will present the research performed by Jason Ekelmann during the time of enrollment for a Masters Degree at Embry-Riddle Aeronautical University.

This document covers the research that went into the K.C.E.M.S. research that was presented at the 2012 SouthEastCon and the research that occurred after the conference.

Chapter 1 covers the introduction, statement of the problem, and the significance of the study. Chapter 2 covers the literature review, which contains brief summaries of relevant research that was compiled over the duration of the project. Chapter 3 covers the methodology of the research, which is broken down into the design of each joint, component selection, and software. Chapter 4 covers the results of the project, which breaks down the final design, testing, and cost analysis. Chapter 5 covers the conclusions, recommendations, and future work for the project. The appendix contains the full source code and the three papers written and published in the proceedings of the 2012 SouthEastCon by Jason Ekelmann and Brian Butka.

The compilation of papers presented at the end of this document is a series of research projects performed by teams of students at Embry-Riddle that Jason Ekelmann was a part of. These research projects relate to one another in the fact that they are all mechatronics project at their core. These papers cover a wide range of subjects such as the Kinect controlled skeleton, an autonomous surface vehicle, and the energy accumulator design for the formula hybrid car. The Kinect controlled skeleton is a mechatronics at its core since it is the combination of mechanical, electrical, and software engineering. The design of the joints and structure being mechanical; the component selection and integration being electrical; and the programming being software.

The surface vehicle project was started to compete in the AUVSI (Association for Unmanned Vehicle Systems International) RoboBoat competitions. Embry-Riddle has been competing in these competitions since the first competition in 2008. The object of this project was to design a surface vehicle or a vehicle that moves on top of water autonomously. The vehicle must then complete a series of tasks in which points are awarded based upon completion. These tasks can range from GPS waypoint navigation to following a series of colored buoys.

The paper written for the conference discussed the design and implementation of the systems seen on the 2010 surface vehicle. This document covers the hull design which involved hydrodynamics and mechanical engineering; the sensor selection and system integration which required electrical engineering; and the programming of the system which was software engineering. Once again the document relates to the rest of the projects through mechatronics.

The final paper attached to this document is a project which is comprised mostly of mechanical and electrical engineering. This paper is comprised of the research and work that went into the energy storage system for the 2012 Embry-Riddle Formula Hybrid car. The basis of this research was to design and build a system which will house the high voltage system for the car.

This system contains batteries, controllers, and safety monitoring systems. The paper covers the components that went into the design, along with the CAD models of the overall accumulator design. These designs are then

supported by a series of calculations and safety considerations. Once again the paper is related to the others through the principles of mechatronics.

Chapter I

Introduction

Robots that mimic human movement have been depicted as the robots of the future in literature and film for a long time. The recent Hollywood movie “Real Steel” features a robot that mimics human movements through watching a person move and then performing the same movements simultaneously. Although the movie is currently science fiction, current research shows the potential for this to become a reality.

The ability to capture and accurately record human motions has been the backbone for many industries such as video game development and animated movie development for a long time. A professional motion capture system was used to digitally capture human movements for the 1995 Atari game “Highlander: The Last of the MacLeods”. These professional level systems require a person to wear a body suit with reflective markers all over it, as seen in Figure 1 [1]. In addition to the custom body suits there is a vast array of sensors and software programs used to capture and compute these movements. Though the accuracies of systems such as Gypsy 7 are excellent, the hardware is expensive and the system is not designed to be used in real time applications.



Figure 1 A body suit used for professional grade motion capture systems. Note the reflective markers used to track body motions [1].

These professional motion capture systems utilize a variety of different markers for capturing the movements of objects passive and active markers. Research being performed at MIT, Stanford, and the University of Maryland has led to the development of marker-less systems, which allow users to capture motions without any sort of marker or suit. The thought behind marker-less systems is that marker based systems can in principle capture such motions of interacting subjects, but they suffer from widely known shortcomings, such as errors due to broken marker trajectories, long setup times, and the inability to simultaneously capture dynamic shape and motion of actors in normal clothing [2]. This of course increases with the addition of multiple bodies and objects that require detecting. These marker-less systems often use a series of RGB cameras for the purpose of capturing video and then complex software, which reconstructs the images taken in a 3-D realm. Figure 2 [2] shows captured images from a system developed by Tsinghua University. Even though these systems reduce the cost of having markers, they are still extremely expensive and still aren't being used for real-time applications.

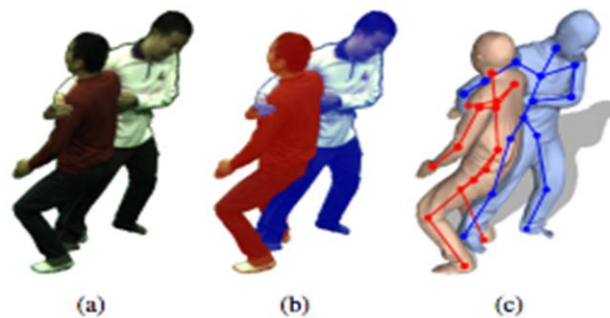


Figure 2 Approach to capturing the motion of interactive characters even in the case of close physical contact: (a) one of the 12 input images, (b) segmentation, (c) estimated skeleton and surface [2].

Since all the professional motion capture systems rely on high-speed cameras that are very expensive this has led to a new field of research for motion capture (MoCap). This research is in the design and development of a low cost MoCap which would bring motion capture systems to the masses. Systems such as these rely heavily on software development and the software doing most of the work, while the professional grade systems have a considerable amount of human intervention. These low cost systems require the use of sensors that are readily available and can be purchased cheaply. A system such as the one developed at the University of Bologna utilizes a series of integrated accelerometers. Accelerometer-based sensing methods are promising technologies for low-cost MoCap systems since they can be implemented with low-power integrated components [3]. These systems are not as accurate as their counterparts but they do not require the complex optical or movement sensor arrays that the larger systems use. This does allow these systems to become more suitable for mobile interaction based motion capture. Since the industry using these systems does not require real time capabilities, the systems themselves are designed for only non-real-time use.

In 2010, Microsoft released a device named the Kinect. This device was to be used in conjunction with the Xbox 360 to provide a new and innovative gaming experience. The Kinect has the ability to capture human movements and relay them to the system in real-time. Costing \$200 dollars, the Kinect made a huge change in the motion capture market, since the Kinect did all of its processing onboard and required only human input for the motions. Unlike many professional systems the Kinect setup tracked movements without any markers and could track 2 people simultaneously. Shortly after

the Kinect's release, many people began writing new code for the sensor so that it could be used in various robotic applications. The University of Canterbury in New Zealand has used the Kinect as a depth sensor for autonomous navigation in a quadrotor system. Systems such as this rely heavily upon computations of depth maps and are common in visual robotic control systems. These computations are used in autonomous navigation, map building and obstacle avoidance [4]. Since the original release of the Kinect, Microsoft has realized that the potential market for this sensor does not revolve around gaming and in 2012 released a windows version of the Kinect. This version has an open source development platform which allows people around the world to develop and share code. Since this release Microsoft has sold 18 million Kinects, which shows the cost effectiveness of this sensor.

This research focuses on developing a system that captures the motions of a human, uses this information to estimate the locations of key bones of the skeleton, and then uses this information to mechanically mimic the skeletal motions on a physical skeleton. Until recently, the technology required to perform this task were well outside of the budget of most museums, but the introduction of the Microsoft Kinect and open source software support allows this project to be performed on a reasonable budget.

Significance of the Study

Children's museums and museums in general have been a great source of knowledge and learning for both adults and children alike. Educating future generations has been and always will be an extremely important undertaking. When dealing with children it is important to make learning fun in order to keep the attention of the

audience. In many children's museums around the world, interactive demonstrations are used to enhance the learning that takes place on their premises. The Daytona Beach Children's museum has many rotating interactive exhibits that are used to teach classes and broaden the horizons of the children that visit the museum. Some of the exhibits include a laser harp and bicycle that a person can ride and power light bulbs. Over the past year, Embry-Riddle Aeronautical University has been working with the museum to create several exhibits that will help children learn and understand the basics of flight. The K.C.E.M.S. project has the ultimate goal of becoming an interactive demonstration, which can be used to teach not only robotics, but also the human skeleton.

Statement of the Problem

Design and build a system, which can capture and mimic human movements in a real time environment. This system must be able to be cyclically loaded in random intervals such as an interactive exhibit in museum would see use. The project must also be able to withstand the abuse that can be seen in environments such as the Daytona Beach Children's Museum.

Purpose Statement

The purpose of this study is to build an interactive demonstration for the Daytona Beach Museum of Arts and Sciences.

Delimitations

During the design and construction of this project the use of commercial off the shelf (COTS) parts will be used whenever possible. This is to increase development and construction speed and decrease cost. Limiting the number of different hardware

components should be strongly accounted for in the design phase of the project. Only readily available software development kits are to be used so that future work can be easily started and prior work can be easily modified.

Limitations and Assumptions

Since the goal of this project is to create a low cost solution, funding will continue to be a limitation. Finding a simple readily available solution for the complexity found in the human shoulder joint turned out to be a severe limitation given the available controllers. Other limitations include the author's limited knowledge of programming languages such as C Sharp and the Microsoft SDK development package.

Definitions of Terms

Femur	A bone of the leg situated between the pelvis and knee
Stereovision	Visual perception of or exhibition in three dimensions
Humerus	The long bone of the arm or forelimb
Ulna	The bone extending from the elbow to the wrist on the side opposite to the thumb
Radius	The bone located on the lateral side of the ulna
Circumduction	The circular movement of a limb

List of Acronyms

MoCap	Motion Capture
COTS	Commercial Off the Shelf
MAP-MRF	Maximum A-Posteriori Markov Random Field
FPS	Frames Per Second
ROI	Region of Interest

RGB	Red Green Blue
LIDAR	Light Detection and Ranging
SDK	Software Development Kit
CNC	Computer Numeric Control
CAD	Computer Assisted Design
CMOS	Complementary Metal Oxide Semiconductor
PWM	Pulse Width Modulation
DFM	Design For Manufacture

Chapter II

Review of the Relevant Literature

Markerless Motion Capture

The ability to perform motion capture without the use of complex systems, which rely on motion markers significantly, reduces the costs of the system that it is being utilized in.

The Automation Department at Tsinghua University has been developing a motion capture system which relies only on the use of cameras. This technology is being used where the current industry standard system aren't being used. These instances are applications where there are multiple objects being looked at and these object contain bodies, which can become twisted together or free flowing alone. A situation such as two people dancing, with one person wearing a dress or skirt, is an ideal application for this system. The people dancing can confuse a marker system and the ability to capture the dress is negated because the objects of interest will be in marker suits.

In this process, the image is segmented using a maximum a-posteriori Markov random field (MAP-MRF) optimization framework. This current system can only handle two person situations. The single person motion capture method was adapted from Motion Capture Using Joint Skeleton Tracking and Surface Estimation [20] to generate the initial skeletons for each body. Once a skeleton is applied to a single body, it is easy to apply and track the second body because the image has already been segmented and categorized. So for each frame the image is segmented into two parts and a skeleton is applied to each body and the image is then combined. This can be seen in Figure 3 [2], which walks through the process of the segmentation and skeletal application.

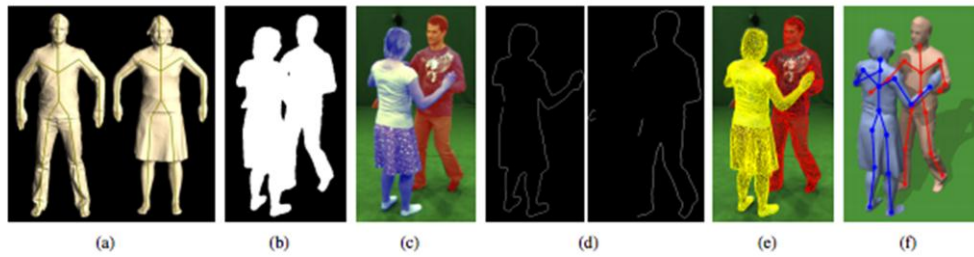


Figure 3 Overview of processing pipeline: (a) articulated template models, (b) input silhouettes, (c) segmentation, (d) contour labels assigned to each person (e) estimated surface, (f) estimated 3D models with embedded skeletons [2].

As can be seen from the above figure, after the segmentation step, the bodies are labeled and then soon after the skeletons are applied. For this application, a series of 12 cameras with a resolution 1296 X 972 pixels at 44 frames per second (fps) was used. This application was also tested with people acting out a combat situation, which hid some of the body; this was shown in Figure 2 above. The robustness of this system is clear but is severely limited by the number of cameras and the number of objects in the field of capture.

Motion Capture Using Joint Skeleton Tracking and Surface Estimation

Research performed at Stanford University consisted of a system which could extract an image and apply a skeleton to a series of images taken from a video. All videos are a series of pictures that are then combined to make continuous motion. For a video that is taken at 30fps, one second of video will yield 30 individual still images. Note that these images can all be the same if there is no change in the object or its environment in the one second the video was taken. By looking at individual images, the process can be simplified because like images can be cut out through the same properties seen in movie compression.

In this system a process, which searches for a particular pose in a frame, is used. To find the body poses in the current frame, the skeletal pose is optimized and a simple

approximate skinning is used to deform the detailed surface of the previous time step into the current time step [20]. From here an adaptive mesh is created and overlaid on the object of interest. This allows for erroneous data and objects to no longer be a part of the processed image. This in turn creates a region of interest (ROI) around the object of interest. This in turn creates a region of interest (ROI) around the object of interest. Finally a 3-D model of the object of interest is created and a skeleton is overlaid. This 3-D model is a model based upon the model used to search for a pose in each frame, which means that the created model may not be identical to the object of interest. A series of images with the analysis performed on them can be seen in Figure 4 [20].

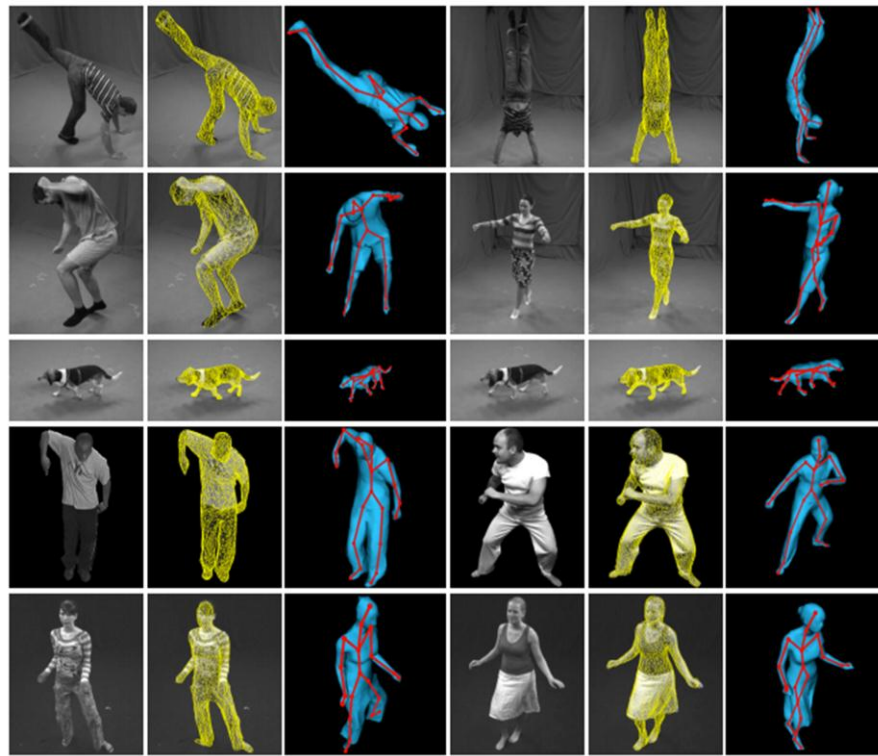


Figure 4 Input image, adapted mesh overlay, and 3D model with estimated skeleton from a different viewpoint respectively [20].

This approach to locating and adapting a skeleton to an image utilizes a series of complex software strategies, which in turn can locate object in images as long as these object have

a rough model to be associated with. This technique does allow for the implementation of a skeleton on objects, such as cats and dogs, without the use of global optimization.

MOCA: A Low-Power, Low-Cost Motion Capture System Based on Integrated Accelerometers

Students at the University of Urbino in Italy performed this research. The premise of this research was to develop a motion capture system based around the use of accelerometers. Accelerometer-based sensing methods are a promising technology for low-cost motion capture systems, since they can be implemented with low-power integrated components [3]. This system utilizes a series of accelerometers placed on the external appendages, such as the arms and the legs. Using accelerometers in this type of system provides a low cost solution and does not have the issues that systems using multiple cameras can experience. Issues with this system include inaccuracies due to the integration and processing, and the bands the sensors are mounted to can move. The resulting experiment was able to categorize motion made with an arm by mounting the sensor on the wrist. This system did not capture the motion of an entire arm, but just the motion of the lower arm in a 3-dimensional space.

This system in conception was going to be used for full motion capture, but in practice did not work, and was reduced to a motion capture system for gesture recognition. The issues that limited the system were the inability to track parts of the body other than arms and legs, and the lack of math and software to support the legs. The advantage of not using cameras does not outweigh the disadvantages of the overall system performance and results.

Altitude Control of a Quadrotor Helicopter Using Depth Map from Microsoft Kinect Sensor

The Microsoft Kinect is an extremely versatile sensor that has many different applications aside from its use in the gaming industry. Students at the University of Canterbury in New Zealand created a quadrotor system, which utilizes the Kinect to perform altitude control based upon the cameras built into the sensor. This method uses passive methods of inferring depth because the Kinect can only capture images and does not utilize active depth systems, such as lasers or ultra sonic sensors. From these methods, the system builds a computational depth map, which is used to create altitude parameters for flight control system of the quadrotor.

For this system, the Kinect needs to be calibrated; taking depth-based images from known distances did this. This process was repeated multiple times over varied ambient light conditions in order to check the sensitivity of the depth measurement to environmental conditions [4]. This application utilizes only the depth camera so the RGB camera is not used. The Kinect is mounted on the bottom of the quadrotor facing the ground. From this position the system can create a depth map, which will be used as the altitude field for controlling the vehicle. An onboard altitude controller is used to stabilize the quadrotor and integrate the depth controls to control the motors. This system shows a unique and simple way to use the Kinect sensor in a dynamic environment. When the Kinect is used for gaming purposes, it remains stationary while objects move in front of it. In this application, the Kinect is mounted to the moving object and can be in an environment filled with moving objects. This of course does not matter because the main focus is the ground or any object that the system may collide with.

Study on the Use of Microsoft Kinect for Robotics Applications

The researched performed by students at the University of California revolves around using the Kinect sensor in a ground robot application. This system has the ability to navigate indoor obstacles. The purpose of the Kinect in this application is to serve as a sensor that can replace the use of a light detection and ranging (LIDAR) sensor. To show the viability of the Kinect as a valid replacement, several experiments were setup to compare the Kinect with a LIDAR system.

The first experiment involved an indoor experiment in which a glass object was placed different distances away from the sensors and data was taken. The data shows the range of the object compared to the sensed range of each sensor. With each different distance, the Kinect was able to match the accuracy of the laser sensor within 5cm except at 40cm because this is outside the operating distance for the normal mode setting of the Kinect. The Kinect has two different operational settings; one for close distance operation and another for normal operations. The default setting has a blind spot from in front of the sensor to 80cm out. This experiment validates the accuracy of the Kinect for sensing objects indoors.

The second experiment was a repeat of the first experiment but it was performed outdoors. The results of this experiment were similar to the first experiment except at 80cm the Kinect was unable to sense the object, while in the first experiment it produced a reading of an object at 81cm. Once again this experiment validates the Kinect as a sensor option for obstacle avoidance. The third set of experiments consisted of several indoor and outdoor runs of a ground vehicle instrumented with a Kinect as the sensor being used for obstacle avoidance. In these tests the robot was let loose in a room with a

chair and some other pieces of furniture in the room. As the robot approaches the chair it stops, performs a 90-degree turn, and continues its navigation of the room. In the outdoor experiment the robot performed as it did indoors. Once again these experiments and this research validate the use of the Kinect as a sensor for robotics.

3D Image Reconstruction and Human Body Tracking Using Stereo Vision and Kinect Technology

The research on the Kinect, in combination with stereovision cameras, was performed at the Illinois Institute of Technology and encompasses research on a system which uses a stereovision camera for imaging and the Kinect as a depth sensor. In this research the Kinect is used in parallel with a stereovision system to identify and track a person. The goal of this research was to create a reconstructed image from the data taken by both sensors. This would allow for the construction of an image that contained depth data, with this depth data a model of what was in the image could be constructed; Figure 5 [27] shows the results of the combined imaging. The images in the figure can then be used for gesture recognition and model construction. This research shows the success of combining the capabilities of the Kinect with the imaging power of an HD camera, resulting in high quality 3-D image reconstruction for real-time streaming videos.

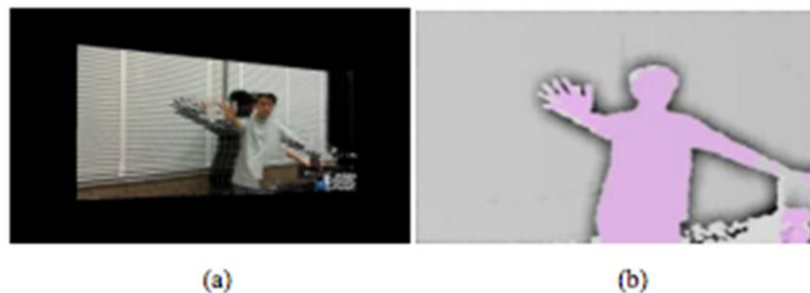


Figure 5 Combined method for 3D image Reconstruction (a) 3D image, (b) Depth map [27].

Accurate Simulation of Hip Joint Range of Motion

This research presents a hip joint motion simulation method using accurate hip joint features and hip range of motion. The purpose of this research is to develop models of the hip in which the maximum range of motion in all directions of the hip can be categorized. By creating models such as these, a better understanding of the limitations seen in the hip can be researched for medical purposes. The medical objective is to quantify hip kinematics in function of hip morphology. Doing so will allow for estimations of the motions that are limited by bone impingements [9]. By quantifying this data, medical professionals can diagnose the reasons for reduced hip movements and joint pains. Thus providing a deep insight to not only the hip joint itself, but the variations in the joint between individuals.

For this modeling, a three-step process was taken to create the model. The first step involves creating a reconstruction of the 3-D bone surfaces of the hip joint. Then estimations of the center of the hip joint are taken. The third and final step involves calculations, which determine the maximum range of motions. These calculations assume the hip joint center as the center for all motions in the ball socket joint. Several different simulations were then performed to calculate the hip joint center. After several simulations, it was determined that the femur head was the joint center of rotation. This would leave the same distance between the femoral head and acetabular rim. The results of this research produced several models, which were comparable to the real values taken from hip joints found in cadavers. Even though the models were not 100 percent accurate, the research led to the development of many models, which can be used in future research; these models can be seen in Figure 6 [9].

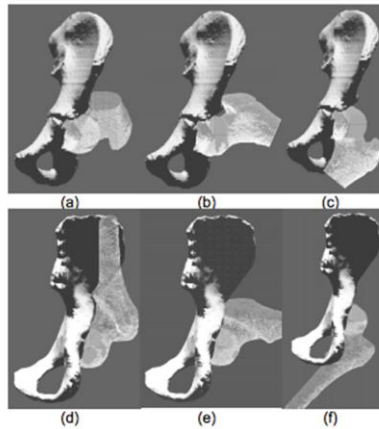


Figure 6 Simulation of hip joint range of motion of 3D surface models of bones: Model from child dataset: (a) fully flexed hip, (b) fully abducted hip (c) fully adducted hip model from young woman dataset: (d) fully flexed hip, (e) fully abducted hip, (f) fully adducted hip [9].

Accuracy Analysis of A Novel Humanoid Robot Shoulder Joint

Students at Yanshan University in China have designed and built a robotic shoulder joint which can replicate the motions found in a human shoulder joint. This research begins the foundation of designing an accurate robotic representation of the human body. Many humanoid robots have some of the capabilities of a human but do not have joints built like a human. This can be seen in solutions such as the robot Asimo that can perform many shoulder movements, but Asimo cannot perform movements such as shoulder shrugs.

The resulting shoulder that was designed and built was a spherical three-degree of freedom joint. This creates a moving platform in which three kinematic chains are connected to a fixed base, which acts as the socket the shoulder joint sits in. This platform is created using three shaft driven servos for each degree of freedom. At the conclusion of this research a joint, which can perform many shoulder movements, was created but had the same lack of movements as the Asimo joint design; Figure 7 shows the fine joint construction.

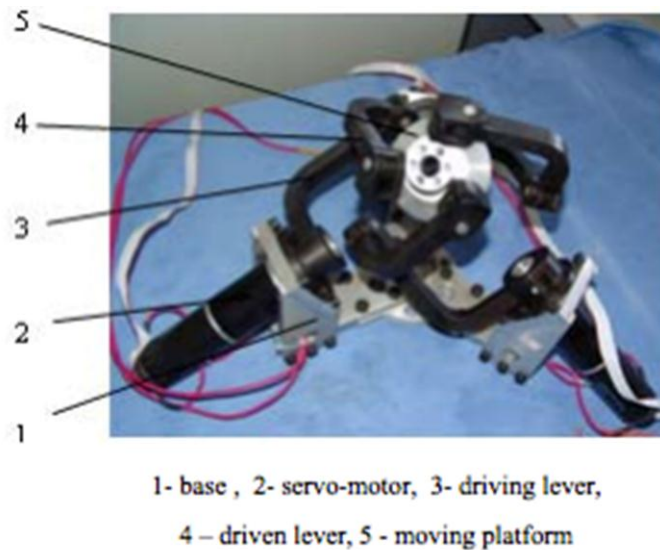


Figure 7 The prototype of the shoulder joint on humanoid robot [10].

Summary

It is clear that there are many different ways to design and build a motion capture system. These systems can range from expensive to inexpensive, but for this research an inexpensive solution is ideal. The Microsoft Kinect provides an ideal low cost sensor package. The use of the Kinect is widespread in robotic-based research. It provides an ideal prototype sensor because of its open source programming and how readily available it is. The Kinect has been seen in flying, ground, and water based applications. When using the Kinect for skeletal tracking, it has built in software and programming to accomplish these tasks.

It is apparent from the research performed that creating a realist robotic universal joint is possible. This can be done in many different ways, with varying amounts of capabilities in the joint. The key in this research is to balance to cost of the joint with the overall performance. It is possible to build an expensive system of robotic joints, but it is

also possible to build a system of inexpensive joint with similar capabilities. By combining the Kinects motion capture abilities with a series of robotic joints, a system, which can replicate human movements can be built.

Hypothesis

It was proposed to build a low cost system which can capture real-time human movements, and replicate them on a robotic platform. This proposed system will be designed for use in a museum thus requiring it to endure a high duty cycle. With the given research that has been completed, it is possible to build the purposed system.

Chapter III

Methodology

Mechatronics

This project will focus on utilizing the captured skeletal maps and mimic the motions on a physical skeleton in real time. Software will analyze the motions of the skeleton 10 times per second. This data will be analyzed to assign specific movements to servo sets for the skeletal points located in the arms and legs. The goal of real-time movements on the physical skeleton requires the use of actuators that are powerful, fast, and accurate. For places on the body where there can be rotation, such as in the shoulder, a pan-tilt motion set up will be used to make the necessary multi-axis movements. Figure 8 [21] shows the actuator locations; the red markings show places where multi-axis actuators are required. It should be noted that only motions of major bones of the skeleton are of interest for this effort. Motions, such as rotations of the wrist and forearm, are not incorporated in this work.



Figure 8 A physical skeleton showing the joints targeted in this research. Black indicates a single axis of motion. Red indicates multi-axis motions [21].

The final design requires the use of 16 actuators. To reduce the number of different parts used in the assembly, the same actuators will be used throughout the design. The actuator selection was based upon 4 different factors; servo speed and accuracy, holding torque, operating angle range, and cost. The holding torque of the actuator was the most crucial factor because in some movements, the actuator is required to hold the weight of entire appendage. The worst-case scenario for holding torque occurs in the leg, since it is the longest and heaviest part of the skeleton. For this requirement, a simple moment calculation was used to determine the holding torque of the actuator needed. The holding torque is given by:

$$\boldsymbol{\tau} = \boldsymbol{r} \times \boldsymbol{F} \quad (1)$$

where τ is the torque, r is the length of the lever arm, and F is the applied force. The worst case occurs when the leg is held straight in front of the body in a kicking motion. For the leg assembly, a mass of 0.3Kg is supported against the pull of gravity yielding a force of 2.94N. For a worst case estimate, the entire mass is assumed to exist at the end of the leg yielding a lever of 0.5m. The worst case holding torque is calculated to be roughly 1.5Nm.

Since the actuators require a controller to interface with the software being written for the other function of this project, an onboard controller must be selected to integrate into the overall design of the system. This component will serve as an interface between the actuators and computer handling the Kinect inputs and algorithms, calculating the rotations needed to position the joints. The selected microcontroller will need to be able to send serial signals to at least 16 actuators and talk to a computer simultaneously.

Prior to the system becoming museum ready, a laptop or another type of computer with a screen will be required to operate the system. The computer will be required to have Microsoft SDK installed for interfacing with the Kinect. The computer will also have some minor hardware specifications, such as requiring 4 gigabytes of RAM and a minimum of a dual core processor. Since the Kinect is interfaced through the laptop, a USB port will be required for the Kinect and another USB port will be needed for the interface to the controller.

Construction

To reduce development time, many COTS (commercial off the shelf) products were used in the construction. The skeletal structure, referred to as the chassis, is a 1m tall plastic model with 25cm and 46cm appendages and was purchased from an anatomical model website. Several different factors had to be taken into account before deciding on the skeleton to be used. Sizing the chassis needed a great deal of consideration due to the size of each appendage; as the chassis becomes larger, the leg and arm appendages grow proportionally. Another deciding factor was that the arms and legs needed to be structural so actuators could be directly mounted to them. This in turn will increase the holding weight required by the servo exponentially since the servos will also become larger and heavier, as will the moments acting on them. Given all these factors, a roughly t scale skeleton was selected for the chassis. The skeleton is constructed from a hard molded resin and has moveable joints in all the areas that will be modified. This chassis is a cheap economical solution that will allow for rapid construction and easy modifications; Figure 9 [21] shows the skeleton that was chosen for the chassis.



Figure 9 Plastic model skeleton on its stand [21].

Actuator Selection

The Dynamixel AX-12A robot actuator was selected for use in this project. The AX-12A has several major advantages over standard hobbyist servos that will be taken advantage of in the construction of the skeleton. These actuators offer a maximum holding torque of 1.6Nm at 12 Volts [5]. When supplying this holding torque, the actuators draw only 900 mA, which allows the use of low cost off the shelf power supplies. Given the overestimates of the required holding torque, it is believed that these actuators are able to hold the entire leg without worry of failure. The AX-12A also offers 360°/continuous operating angles and non-loaded speeds of 0.196sec/60°. These features will allow for near-real-time movements of all the appendages. Along with all the performance features of the AX-12A, there are several built-in features, such as the internal micro-controller, that will be used in this project. The built-in microcontroller

provides feedback of the current angular position and angular velocity, as well as the torque being applied to the load; the availability of these feedback signals and the compact form factor led to the selection of these actuators. A bearing is used at the final axis to ensure no efficiency degradation with high external loads. The actuator also has a built in alarm system that can provide feedback to the higher-level controller when there are issues in current draw, voltage, internal temperature, and torque output. If any anomaly occurs during a high torque hold, the actuator will shut itself off and flash red showing that an error has occurred. The repowering of the system remedies this, and allows the full system to return to normal operation with no damage to the actuator that failed. The case that encloses the mechanics of the actuator has integrated mounting points, which will also be utilized in the assembly of the final design; Figure 10 [5]



Figure 10 The Dynamixel AX-12A is the selected actuator for all joints [5].
shows the AX-12A and a mounting bracket.

The final driving factor for the selection of the actuator is the high number of additional parts and brackets that have been designed around them. The brackets designed for the Dynamixel actuators allow for the construction of multi-axis joint

systems, with relative ease along with providing the proper amount of mounting holes to not cause a fault in the system; Figure 11 [22] shows a robotic arm built utilizing the AX-



Figure 11 Robotic arm built using the AX-12A's and brackets [22].
12A's and their associated brackets.

Bracket Design and Selection

In order to design and build an economical solution, much care needed to be taken in designing the brackets to be used in final construction. By selecting the Dynamixel actuators, a simple and unique solution presented itself. Since these actuators are used to build many robotic projects, a line of plastic brackets have been produced by Robotis. Brackets for the AX-12A's are designed to mount sturdily to the actuators and have various shapes and size. These brackets range in cost from 1 dollar a bracket to 2 dollars a bracket, making these a very quick and economical solution. Since the rough math files for these brackets were readily available, it was simple to reproduce brackets of similar shape and size but with less detail. These reproduced brackets would need to be

machined using either a computer numeric controlled (CNC) milling machine or a hand-milling machine. If custom brackets were to be used, nearly 25 brackets would need to be machined and have CNC code written for them. In order for the machining costs for these pieces to be limited, the parts would need to be manufactured in the school's manufacturing labs. The overall time and effort to make these parts was not an option during the time of the semester the project was being built. Now looking at the construction of the bracket from a purely Design For Manufacturing (DFM) standpoint, making plastic brackets is simpler and less expensive than machine brackets. Plastic brackets such as the bracket shown in Figure 12 [23] are produced quickly using a process called injection molding. Parts like these are cheap to produce in large numbers, but the cost of making 25 different brackets using this method would be extremely high due to tooling costs however this is not of concern on this project since they are COTS. The economical advantage of using COTS brackets and designing around them far



Figure 12 Robotis bracket design for the AX-12A; cost \$1.49 [23].

outweighs the time and cost disadvantage of designing and machining custom brackets.

Joint Design

Since the system needs to be able to replicate human joint movements, a series of actuators needed to be combined to produce the desired range of movements. The main issue with this is the complexity seen in the human shoulder and hip joints. These joints contain a ball joint that is an extremely complex mechanical movement to replicate with a series of single axis actuators. The shoulder contains many more possible movements than the hip joint due to the limited flexibility of the hip and the mechanical limitations of

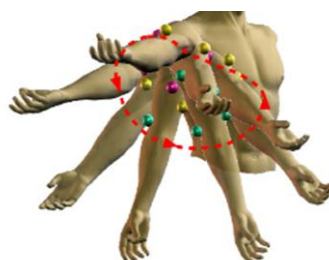


Figure 13 Simple shoulder movements [24].

the hip socket. While the shoulder joint can rotate 360 degrees, the hip joint cannot; Figure 13 [24] shows some simple movements possible with the shoulder joint.

During the initial design phase of this project, a small-scale system was built to prove the concept of capturing human movements using the Kinect and relaying them to actuators. Since the type of actuator had been selected early on, an economical prototype needed to be rapidly designed and built so that the software integration could happen while the final structure was designed and built. The Bioloid, built and designed by Robotis, was the ideal solution for the prototype. This Humanoid robot contained everything that was going to be needed in the final construction of the system. The package included 18 AX-12A's actuators, over one hundred brackets that are designed for the AX-12A's, and the CM-530 robotic controller. The overall capabilities of the

Bioid can perform many of the movements a human can and some movements that they can't; Figure 14 [25] shows the Bioid used for the small-scale prototype.



Figure 14 Bioid robot used for the small-scale prototype [25].

Using the Bioid provided a unique advantage, in that it was a complete system in which the Kinect code could be meshed with its stock controller to produce a functional display. This prototype was shown at the IEEE Southeastcon 2012 to display the concepts of the final system. The other unique advantage the Bioid had was having all the necessary components to build the final design in a quick and easy manner which limited down time between the software development for the prototype and the software implementation on the final design. The final system used many brackets from the Bioid and similar joint designs for the elbows, knees, and hips.

For the final design and software implementation, a strategy of building and testing the simplest joints was taken to allow for a smooth integration of subsystems and components. The elbow and knee joints were developed first because they utilized a single actuator, which allowed for a simple design and minimal fabrication and modification of stock brackets. Since the human elbow can only move roughly 150 degrees given hyper extension [6] as seen in Figure 15 [7], the AX-12A actuator only

needed to be mounted in a way in which it could be connected to the Humerus. The Ulna and Radius (bones that compile the lower arm) bones would then be mounted directly to

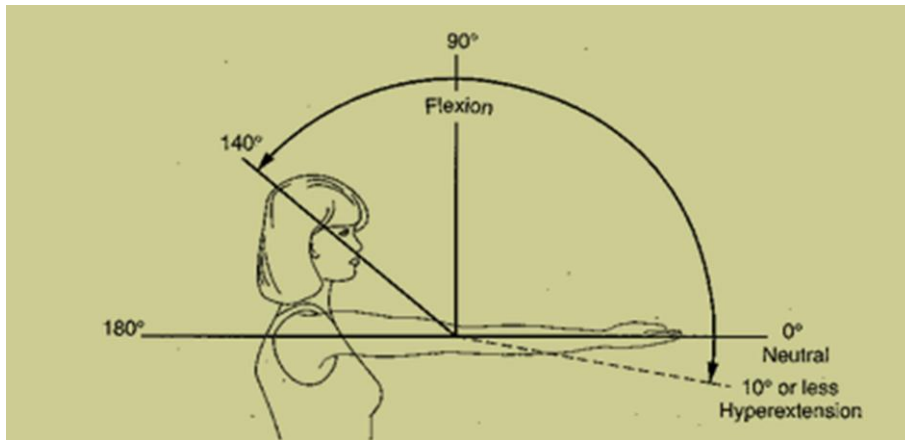


Figure 15 Movements possible from the elbow joint [7].

the actuator.

Since the elbow joint design is simple compared to the shoulder joint, the structure that is used to mount the arm to the shoulder will be included in the elbow joint. In designing the arm structure, a reinforcing bracket was needed in order to mount the Humerus of the skeleton to the actuators. This structure is needed because the model of the Humerus being used is not large enough to support a load; in this case the bones in the arm are purely cosmetic; Figure 16 shows a computer assisted design (CAD) model

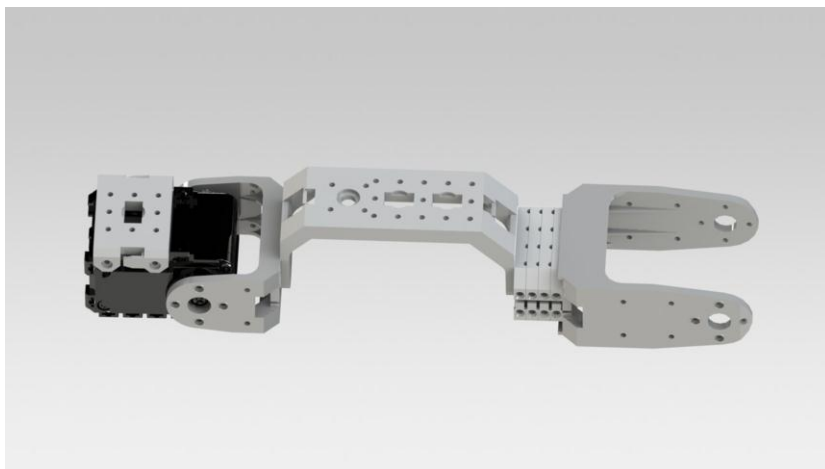


Figure 16 CAD model of the elbow joint and upper arm structure

of the elbow and upper arm design. This design allows for the full range of movements that a human elbow can perform.

Similar to the elbow joint, the knee joint is a simple joint to mechanically replicate using single axis actuators. This is the case because like the elbow, the knee also only moves on a single axis of rotation. The knee has a maximum range of motion of 150 degree in flexion and cannot hyper extend without minor damage [7]. As with the elbow design, the actuator needed to be mounted in a fashion in which the Tibia can be mounted to the front face of the actuator. This will allow for the actuator to properly simulate the knee joint movements; Figure 17 [7] shows the range of motions possible from the knee joint.

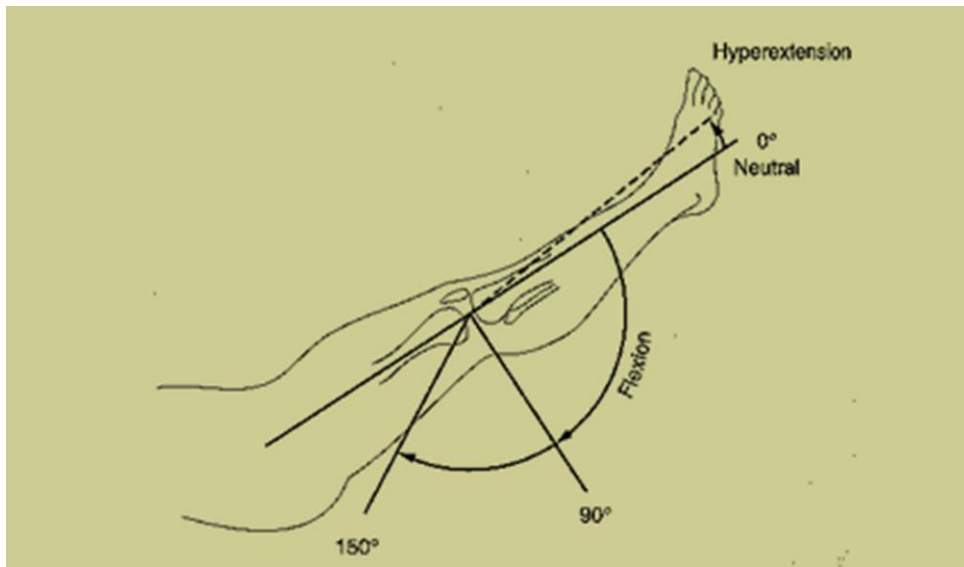


Figure 17 Rotational limits of the knee [7].

For the lower leg structure, the actuator being used for the knee joint will be located at the end of the Femur. For the skeleton being used, the Femur bone was large and strong enough to handle the dynamic forces being imparted from the hip movements. The actuator is mounted to the Femur utilizing two brackets and a $\frac{1}{4}$ inch through bolt to

fasten the brackets to the bone. The upper portion of the Femur is mounted to another series of brackets using another through bolt so that the leg can be attached to the hip

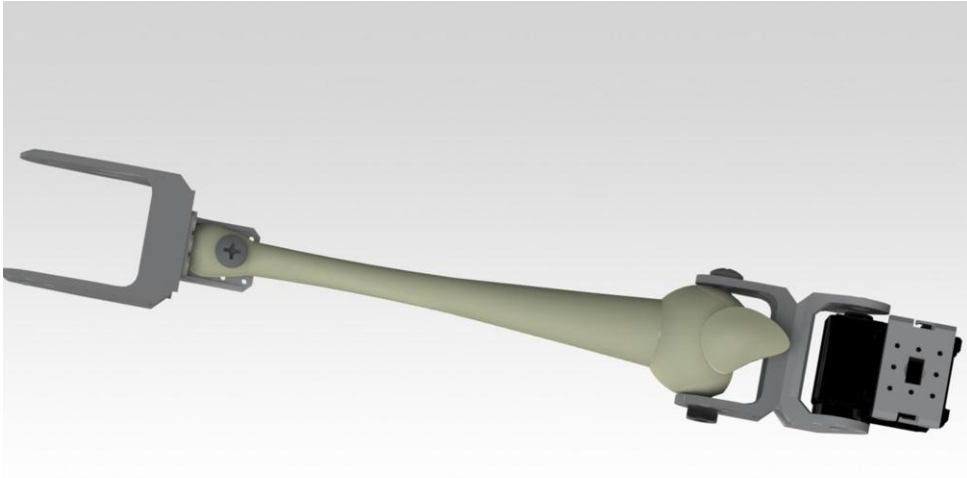


Figure 18 CAD model of the lower leg

joint; Figure 18 shows a CAD model of the lower leg assembly.

The hip joint is the next most complex joint in the human body because it has many of the movements a ball and socket joint can perform but is mechanically limited by the socket the joint sits in. The hipbone and its connecting ligaments limit the range of motion for the entire joint. As with the knee and elbow joint, the hip joint has a range of about 140 degrees between flexion and hyperextension. Since the joint is a ball in socket joint, it also has 80 degrees of motion between abduction and adduction. The hip joint also has the ability to rotate away from the body with about 70 degrees of motion between lateral and medial rotations [8]. As with many other joints in the human body, the overall range of motion is highly dependent on the individual performing the motion. The major factor that affects the range of motion in the hip is flexibility in any direction of movement [9]. This limiting factor can make replicating the hip motion extremely

difficult because in some instances a person can be considerably more flexible than the average person. In these cases, the movement being made can exceed the maximum range of the constructed joint; Figure 19 [8] shows the various forms of motion the hip joint can create.

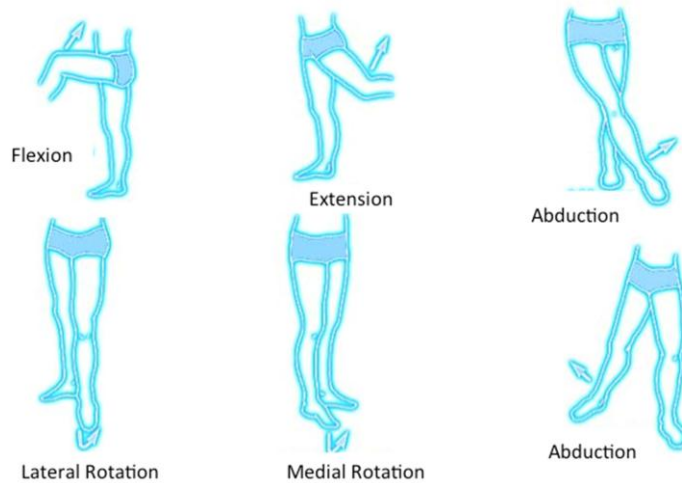


Figure 19 Different motions made by the hip joint [8].

In designing the hip joint for the final build, it was required to construct a joint that was able to follow any movements provided by the Kinect. Since the Kinect can sense Flexion, abduction, and rotation, the hip joint needs to be able to perform all these movements. The difficulty in this is packaging three single axis actuators in a way that does not consume an excessive amount of space and can perform all the necessary functions. The issue with this form of joint is attempting to limit the number of mechanical interferences between mounting brackets in the joint design. Since the actual hip joint is only a single ball joint, there is no worry of mechanical interferences from things such as mounting brackets, unlike the constructed joint. One factor that makes replicating the hip joint easier is its limited range of motion inside its socket. The hip is

comprised of several bones, which in turn form the socket in which the joint sits in. For this project's case, the series of actuators that will comprise the hip joints will be built as one continuous structure. Since the bones from the skeleton will not be used in the construction of the hip joint, they will be mounted in front of the constructed joint to keep the model anatomically correct for educational purposes; Figure 21 shows a labeled CAD model associated with the hip joints.

As seen in Figure 20, the hip is composed of 6 different single axis actuators in order to construct a joint that can perform the movements of both hip joints. The brackets on top of the two actuators are used to mount the hips to the super structure of the skeleton. The hips will be mounted utilizing a total of 8 size M2 bolts with their associating nuts. This many bolts was chosen for the amount of redundancy the extra 4 bolts will provide. In early testing with the Bioloid it was noted that the nuts and bolts had a tendency to loosen and shake themselves out given a lengthy amount of use.

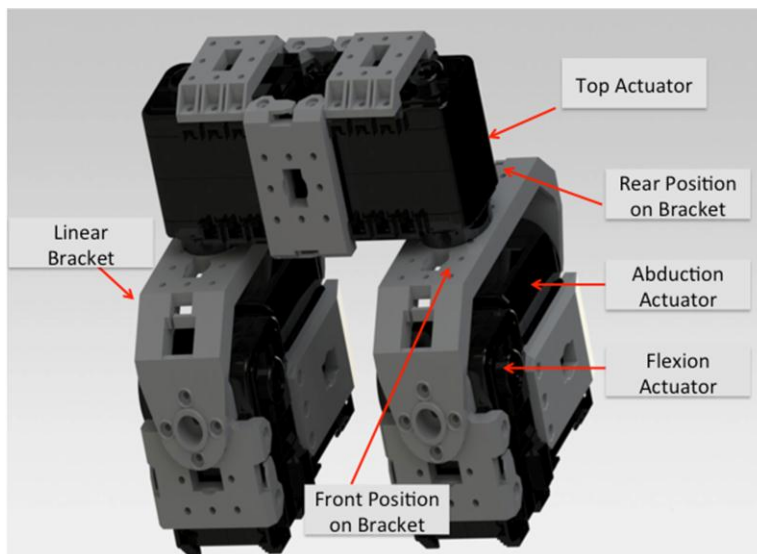


Figure 20 CAD model of the hip joints

The top actuators will be used to impart the necessary movements needed to fulfill the medial and lateral rotations. The placement of the top actuator relative to the connecting bracket needs a large amount of consideration with respect to collisions with the adjacent joint. By placing the actuator in the rear position of the bracket, there is an increased lateral range and a decreased range in medial rotation due to collisions with the adjacent hip performing a medial rotation. When placing the actuator in the forward position on the bracket, an increased range in the medial rotation was created while decreasing the range in lateral hip rotation due to collisions much like placing the actuator in the rear section. By placing the actuator in the center of the bracket the lateral and medial hip rotations are equal. Since the combined rotation needed is 70 degrees based upon the research that was performed, the medial and lateral rotations from off center only need to be 35 degrees. With the given design and the accuracy of the Kinect, this configuration can provide ample amounts of rotation in the hips without any collisions.

When selecting the placements of the last two actuators, several different configurations could have been selected. The final configuration selected utilized the larger linear bracket as seen in the above figure. This allows for two actuators to be mounted inline with one another. From this linear combination the order of the front and rear actuator needed to be decided. By placing the actuator performing the flexion and extension motions in the rear of the bracket, a mechanical interference is created during the flexion motion. Even though there is a collision in the flexion motion, there is an exceedingly large amount of room for motion in the extension range of motion. This does not necessarily discount this design because the leg only needs to be able to perform in

the average range described in the research that was performed. Unfortunately the necessary range of motion in the series of motions is 110 degrees in flexion and 30 degrees in extension. The design, which incorporates the flexion/extension actuator in the rear, is not a valid design in this instance.

Since there is only one other place to locate the flexion/extension actuator, the actuator will be located in the front position in the linear bracket. This provides a mechanical inference with the leg bracket and the rear actuator during extension motions. This is an acceptable issue because the needed motion in extension is only 30 degrees, which this configuration meets. The motion in the flexion motion can also encounter a mechanical interference if the wires are not run properly. If the wires are run in front of the flexion/extension actuator, a pinch point will occur between the leg bracket and the linear bracket with the wire in the middle. This of course is a major issue for two reasons; one being it restricts the motion in the flexion direction to about 45 degrees, and two it can cause major damage to the wires powering the entire leg. With the wires being run behind the actuator a full motion of about 125 degrees in the flexion direction is obtainable and there is no potential for damaged wires.

Since the third and final actuator in the three-actuator hip design is in the abduction/adduction motion, or side-to-side motion, the position of the actuator is relative. Whether it's in the front or rear of the linear bracket makes no difference on performance. Since the flexion/extension actuator is in the front position, the abduction/adduction actuator must be placed in the rear position on the linear bracket. Regardless of position, the limiting factor for this actuator will be the mechanical interference cause by the collision between the actuators and the linear bracket. This

provides a small issue since the needed range of motion is 40 degrees in each direction. As seen in Figure 19 of hip motion, humans have the ability to cross their legs in the adduction motion; this motion will be limited due to possible collisions and the potential for the legs becoming entangled. To avoid such collisions the movements will be limited on the software side.

The final joint is the shoulder joint, which is considered one of the most complex joints in the human body. Although the shoulder is a simple ball joint, on paper it is an extremely difficult joint to animate using single axis mechanical systems [10]. Unlike the hip joint, the shoulder has very few limiting factors and has a considerably larger range of motion than the hip. The shoulder joint can move in all the same motions as the hip joint plus one extra motion; these motions being abduction/adduction, flexion/extension, outward/inward medial rotation, and circumduction. In the hips there exists a lateral rotation while in the shoulders there are only inward and outward medial rotations [11]. Circumduction in anatomy is the ability to move a limb or appendage in a circular motion. This particular motion defines the main motion of the shoulder and many simple motions are built from this ability. The shoulder has the ability to perform a medial rotation both inward and outward; this is also known as a shrug. Since the Kinect cannot detect this motion, its movements will not be incorporated in the final joint design. Since the Kinect can sense the other motions, these will have to be incorporated. When defining the range of motion for these different movements, the average human will have to be examined again. In many cases flexibility and being double jointed can severely affect the maximum range of motion in this joint.

As with the hip, the range of motion in every type of movement the shoulder can make is highly dependent on the flexibility of the person making the movements. For the average person the range of motion has been defined by an approximation of ranges for each motion and these ranges are used throughout the medical industry as a standard [12]. Figure 21 [8] shows the movements possible by the shoulder joint.

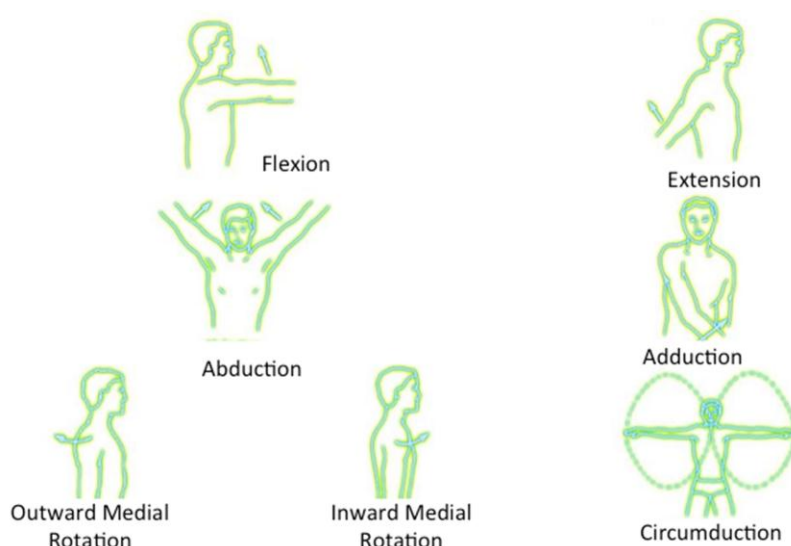


Figure 21 Different motions made by the shoulder joint [8].

Since the shoulder has the ability of circumduction, the average motion for this is 360 degrees. The average range of motion for a shoulder making an extension motion is a maximum of 50 degrees, while in flexion the range is 90 degrees. For the medial rotations, both inward and outward motions are 90 degrees in each direction. Much like the medial rotations, the abduction and adduction movements have a maximum range of 90 degrees. It should be noted that in this single joint there are movements in four unique directions.

The shoulder joint went through a total of three different designs before a final design was chosen. The reason for having built three different shoulder designs was the development of limitations each different design had during testing. The issues seen in each design varied in the joints' ability to perform motions that were going to be anticipated during upcoming demonstrations. The series of motion that are simple to perform with a joint such as the shoulder became difficult to replicate utilizing single axis actuators. Motions that were possible to make using a given design then became difficult or near impossible to perform using the software methods that had been developed. Common motions that were assumed to be made were motions such as waving hands, pointing, clapping, patting the head, and putting the hands on the hips. These motions utilize a combination of individual shoulder motions that provide an impressive and reactive display for children.

The first shoulder joint that was designed, built, and tested was a joint that was comprised of two actuators. This design was based off the shoulder that was found in the Bioliod. This particular design was used originally because the beta version of the software was ported to the Bioliod. So the software was the main driving factor of the

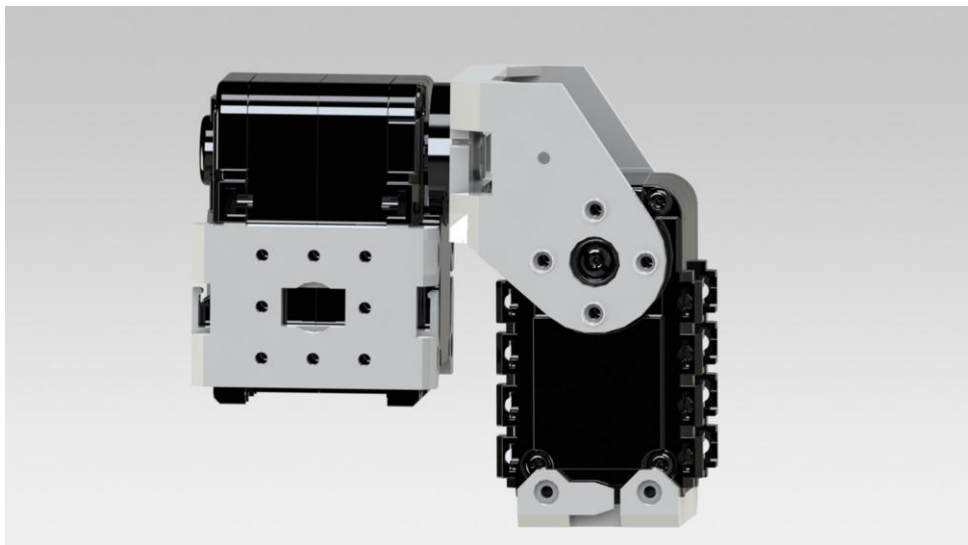


Figure 22 CAD model of shoulder joint design 1

first joint design. By using only two actuators you gain simplicity on the software side but you lose mobility on the mechanical side, which in the long run was deemed unacceptable; Figure 22 shows shoulder joint design 1.

The two-actuator joint allows for the critical circumduction movement to be performed with ease using both actuators. Rotating the bottom actuator to be in line with the axis, the top actuator rotates about, and then rotating the top actuator performs this motion. By only rotating the bottom actuator, the robot is able to mimic abduction and adduction movements. If only the top actuator is rotated the robot can mimic the flexion and extension motions. Though it seems this joint design can perform all the proper movements, not only the motions but the directions the bones are facing are incorrect. These issues create movements that don't seem natural because they cannot follow the path the actual arms are making. Using this joint arrangement would require software that would perform path planning to move the arm to the final position instead of following the input motions. These issues are apparent when the first motion made is a 90-degree abduction and then performing a forward pointing motion or head patting motion. In an instance where the first motion is a 90-degree abduction, the system has a tendency to get stuck on the software side because there is no way to directly translate from that position to any other common position without returning to the arms down position. Since this needs to be a fluid demonstration, where the robot follows the movements the person is making as closely as possible, this design has a fatal flaw and needed to be redesigned.

The second design incorporates a third actuator to gain the ability to make movements that follow actual movements more realistically. The second shoulder design is identical to the hip joint design. This was chosen as the second design for its ability to

simulate the ball joint in the hip. The issue with this design is that the known limitations of the actual hip joint would now be applied to the shoulder joint. Another main disadvantage of this joint setup is the complexity of the software needed to operate it. Since the orientation and overall design of the joint was different from the first design, it required the integration of a third actuator into the software. Along with this integration issue, the overall software strategy needed to be rewritten since the joint movement is

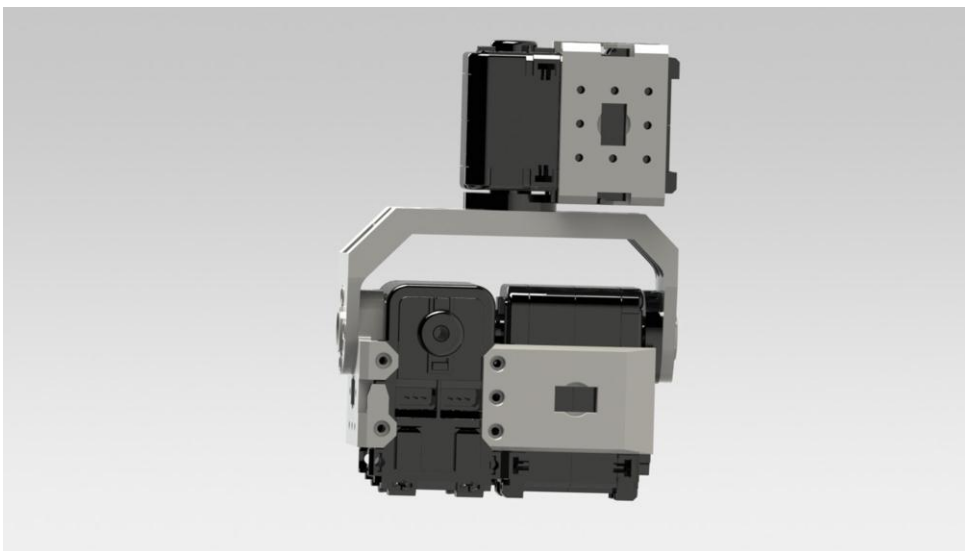


Figure 23 CAD model of shoulder joint design 2

completely different from the first design; Figure 23 shows the CAD model of the second joint design.

Since the above design is identical to the hip joint, the software used was also very similar. The key issue to this was that the hip joint is incapable of performing circumduction. It was decided by the team to continue on with the design because circumduction was not necessarily needed to perform many of the required maneuvers. The first bottom actuator was used to perform maneuvers requiring abduction or adduction. The rear bottom actuator was used to perform the extension and flexion

maneuvers. While the top actuator was used for medial rotation in the hips, it cannot perform the same medial rotation that is made by the shoulder joint. It was clear that this design was incapable of performing the necessary movements because of the limitation that allowed this design to excel as the hip joint. When this joint performs an extension or flexion motion, it has a mechanical interference with the linear bracket that does not allow for the full range of motion necessary for proper shoulder movement. Since these flaws were realized early on, there was limited amount of testing performed on this joint to verify its ability to perform movements such as hand waving and points, neither of which this joint can perform.

The third and final joint design also uses a three-actuator design in a similar arrangement as the second joint design. This design changes the overall orientation and positioning with the top actuator. It still utilizes the linear bracket, which contains two actuators. This design was chosen because the motions it could handle were the motions the team decided the system was most likely to encounter during a demonstration. This was true because many of the limitations given by the orientation of the hip joint were not applicable in the new configuration. Once again, by changing the overall orientation, the

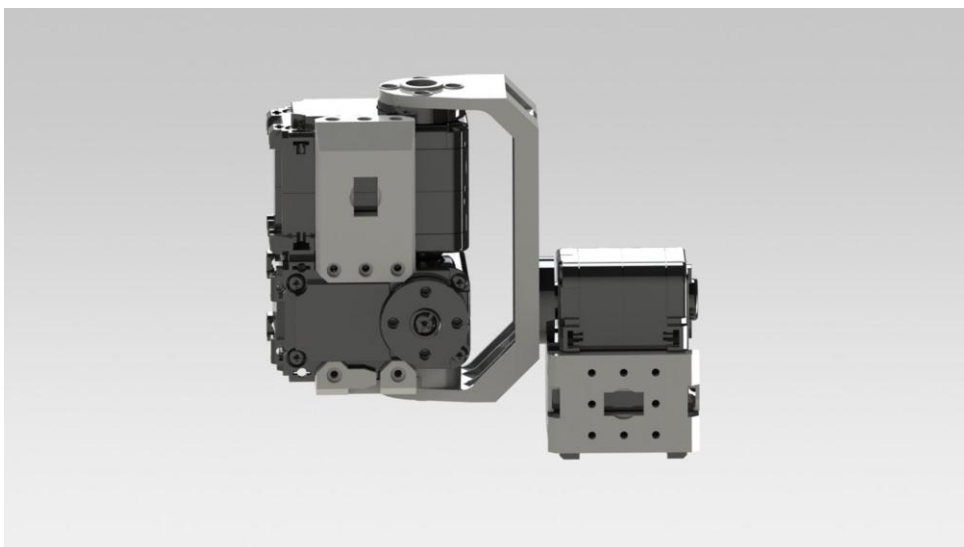


Figure 24 CAD model of shoulder joint design 3

software needed to be rewritten which requires many new position values to be set; Figure 24 shows a CAD model of the third shoulder joint design.

The third design orients the linear bracket vertically, which incorporates the advantages of the first and second designs. The actuator that is not in the linear bracket will be used in combination with the bottom actuator in the linear bracket to perform a circumduction like motion. It should be noted that the size and position of the linear bracket with respect to the first actuator causes the circumduction motion to produce a larger circle. This position of the first actuator on the linear bracket also causes the circumduction motion to be off center, unlike the motion performed by a real shoulder joint. This is acceptable because the accuracies of the sensors selected may not actually detect the difference between a small circumduction movement and a larger circumduction movement. The bottom actuator in the linear bracket is used to perform the abduction and adduction motions while the actuator that is not in the bracket can be used to perform the flexion and extension motions. The second actuator in the linear bracket can perform a medial rotation by definition, but this type of movement in the actual shoulder also has a linear component of motion. This actuator will instead be used to add some amounts of realism to the other motions. In the given configuration, that actuator is highly limited by a mechanical collision with the linear bracket.

When this design performs the abduction and adduction movements, it can perform the entire range required from the studies used for this research. Once again, the adduction movements will be limited on the software side to prevent hang-ups and collisions. The flexion and extension motions, which are performed by the actuator that is not in the linear bracket, are exceeded on the performance side. Since this design can

rotate 360 degrees, it can perform circumduction by this motion. Motions, such as performing a clap, can be accomplished by rotating the first actuator and rotating the bottom linear bracket actuator. The issues seen with this motion is that the sensor does not have the ability to sense the rotation of the bone, so this causes the clapping motion to often have the back of the hands coming together, which is an unnatural motion. Once again, a motion such as waving can be looked at differently because the forearm does not have the ability to rotate. The motion of pointing is easily accomplished, along with motions such as the jumping jack arm motion, and putting the hands on the hips. The issues the first design had are not readily apparent due to the orientation and position of the linear bracket. The possibilities for the software getting stuck in a position which it cannot recover from with this design is still possible but limited and not readily apparent.

Final Construction

In order to build the final system, several different mounting brackets needed to be machined so that the COTS brackets could mount to them. These parts were built using a milling machine in the schools' machine shop. In order for the skeleton to be held upright, a long bar was built to mount all the different components to. This bar was constructed from a 3 foot long piece of 1/2-inch steel square stock. Four holes were machined into the square stock so that the leg bracket and shoulder bracket could be mounted. The first bracket is the bracket that will be used to mount the hips and legs to the upright bar. This bracket was constructed of a piece of 1/2-inch thick steel L-channel with 2 inch flanges. This piece of stock had two holes machined into it so it could be mounted to the upright bar, and two sets of four holes that matched the mounting hole found on the COTS brackets. These holes allowed for the mounting of the whole leg

assembly. The second bracket that was machined was designed to mount the two arm assemblies. This bracket was machined using a 1/4-inch thick steel plate that was machined into a cross like shape. The reasoning for this was to eliminate a mechanical collision with the linear brackets in the shoulder joints. The piece of steel plate then had ten holes machined into it. Eight of these holes matched the holes found on the COTS bracket; two holes for mounting to the upright bar and two holes for mounting the upper body of the skeleton to the steel plate. The robot also needed a base so it could be free standing; this base was built using a large piece of rectangle stock, which provided enough size and weight to support the robot in motion. The bones from the skeleton that were not used as structural components of the design were machined and mounted to brackets, which were affixed to the appropriate actuators. The main body of the skeleton was not strong enough to bear any sort of major loading so this piece was also affixed to the main super structure through bolts, which were mounted to the steel plate; Figure 25 shows a CAD model of the final design without the base or ornamental bones.



Figure 25 CAD model of the final build minus the base and the skeleton body.

Sensor Selection

Since this project is based around the Kinect, that was the sensor that was selected. At a cost of \$200, the Microsoft Kinect has the ability to track the movements of 24 distinct skeletal points on the human body. These points include the head, hands, arms, and legs. Along with these 24 skeletal points, the Kinect can track two people at the same time and has voice recognition capabilities [13]. This project only requires the tracking of less than 15 skeletal points for a single user. Figure 26 [14] shows the Kinect and a skeletal map.



Figure 26 A 15-point skeletal model (left) produced by a Microsoft Kinect sensor (right) [14].

The Kinect sensor generates the skeletal map by reading data from an array of sensors including: a depth sensor, an accelerometer, a multi-array microphone, and a RGB camera [14]. The microphone was originally not going to be used for this application but voice commands were added to assist with testing and user interface issue. Commands such as stop and pause are used to stop the demo; resume game is used to resume the demo. Commands such as faster and slower may be implemented to adjust the speed of the actuators during testing. The main driving sensors on the Kinect are the depth sensor and the cameras. The depth sensor is a Micron 1/2-Inch Megapixel CMOS Digital Image Sensor that consists of an infrared laser projector and a CMOS

(Complementary metal–oxide–semiconductor) sensor. This CMOS is considered an active pixel sensor and is capable of capturing 3D video data in ambient light [13,15].

The main bulk of the data used to construct the skeletal points is taken from the two RGB cameras in the Kinect [27]. These cameras are Aptina 1/4-Inch 1.3- Megapixel SOC CMOS Digital Image Sensors, which give the Kinect a viewing range of roughly 11ft [26]. Along with the mentioned sensor, the Kinect is equipped with a motorized pivot that allows the Kinect to physically move as it tracks targets. This pivot is not used in the demo because the Kinect will ideally be mounted so that pivoting the sensor will not be necessary.

Microcontroller Selection

Since the skeleton requires a total of 16 actuators in order to perform all necessary movements, a microcontroller that can handle a minimum of 16 actuators will be required. The AX-12A requires TTL level serial communications to send and receive signals. This project utilizes a total of 1 AX-12A's to be controlled in realtime. Although a controller is available from Robotis (the manufacturer of the AX-12A actuators), it is unclear if the controller will be able to perform all of the necessary analysis of skeletal motion in realtime. It was anticipated that as the project neared completion, a more powerful controller such as Vanadium Labs ArbotiX Robocontroller would be required. However, in the interests of speeding development, the Robotis controller and software was used as the development platform.

The Robotis CM-5 controller is the control that was used with the Bioloid robots. This controller was able to handle moving the 18 actuators with ease. The CM-5 utilizes the ATMEGA128 for its microprocessor. The high-performance, low-power Atmel 8-bit

AVR RISC-based microcontroller combines 128KB of programmable flash memory, 4KB SRAM, a 4KB EEPROM, an 8-channel 10-bit A/D converter, and a JTAG interface for on-chip debugging. The device supports throughput of 16 MIPS at 16 MHz and operates between 4.5-5.5 volts [16]. The CM-5 also has the advantage of having the Bioloid software preloaded onto it, which allows for an easy interface between the kinect software and output commands to the actuators. The main flaw with using this controller is the inability to change the speed of the actuator, since commands are being sent to the controller and the controller is moving them as if it were connected to the Bioloid.

The advantage of the ArbotiX controller over many other popular microcontrollers such as the Arduino family is that this ArbotiX controller is designed with the Dynamixel AX-12 servos in mind. This microcontroller boasts the ability to control more than 24 AX-12 servos simultaneously using its integrated Atmega644p processor [17]. The ArbotiX also has the ability to incorporate an XBee system for wireless communications. If needed there are motor drivers, encoder headers, and 32 analog headers equipped to this board allowing the use of PWM (pulse width modulation) servos if needed; Figure 27 [17] shows the ArbotiX microcontroller



Figure 27 The Arbotix microcontroller selected as the controller for this research [17].

Software

Before writing software that interfaced with the Kinect, the team needed to understand exactly what the native software on the Kinect was doing. It is given that the Kinect senses the body and then output a series of points that represent joints and positions on the body. That being said, the software development kit (SDK) provided by Microsoft allows for interfacing with the Kinect without needing to know exactly what is going on in the background. The Kinect uses two cameras to capture real-time images of the environment in the field of view. First, the Kinect creates a depth map using structured light and then infers body position from a technique called machine learning [18]. From this inferred body position, a skeletal map is built by estimating the positions of 20 different skeletal points. The points represent the major joints such as the knees and elbows, and minor joints such as the wrists and ankles. The Kinect then tracks these skeletal points and the skeletal map is continuously updated based upon changes in the position of these points. By utilizing these points, the software for controlling the actuators in the robot can relate the skeletal movements recorded by the Kinect to rotations needed to be performed by the actuators.

In order to write the software interface between the Kinect and the actuator, the proper language needs to be selected. The language chosen was C# because of its ability to easily call the Microsoft SDK library. Additionally, C# allows for a compiled .exe with sophisticated graphical user interfaces (GUI) along with the sample source code for the Kinect being provided in C#. C# also allows for a quick run-time, which is necessary for

a computationally complex task such as this. Since there is an open source SDK for the Kinect, all interfaces on that side will be done through that. This SDK has some built in higher level functions that can be utilized, such as obtaining the relative positions of all of the joints of a human in the field of view of the Kinect. The SDK also allows for various graphical outputs for debugging. Among these is the display of all of the skeletons in 3D space. The .NET framework is used for hardware interfaces because it is well integrated into the C# language, and is easy to access using Visual Studio. Any function that is not possible natively using C#'s built-ins or the Kinect SDK can be done with .NET. The .NET frameworks will be utilized for all math based operations and the serial communications with the CM-5 controller.

In order to relate the actuators to the corresponding positions on the skeletal map, an understanding of all the positions in relation to the body should be made; Figure 28 [19] shows these relations.

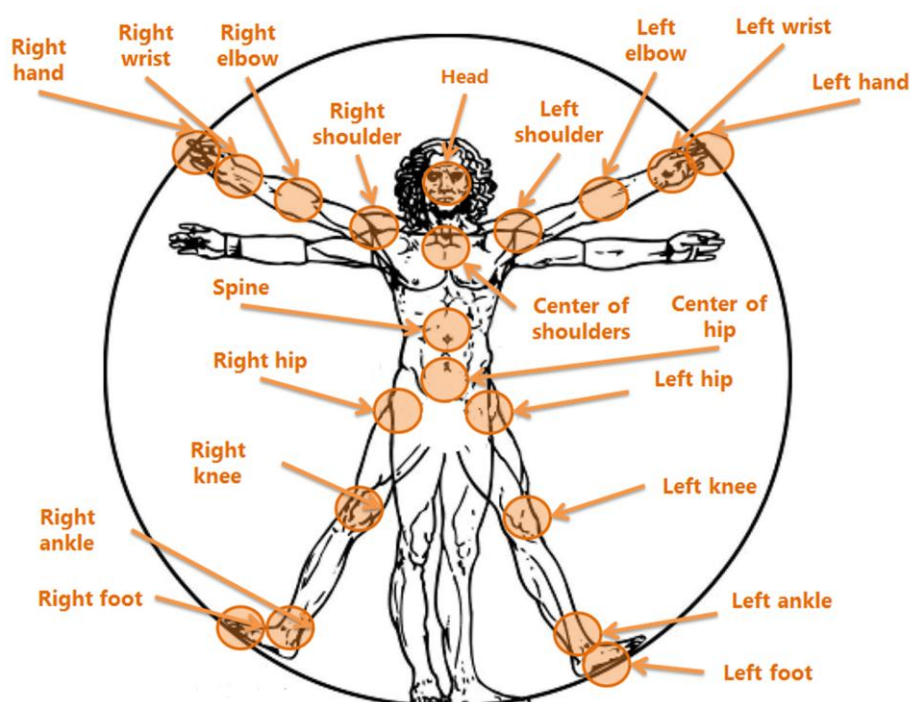


Figure 28 Skeletal points imposed on Vitruvian Man [19].

With the skeletal map given, each of the joints are indexed and their X, Y, and Z positions are known relative to the dimensional space the Kinect has created. For this project, the angles of the knees, elbows, shoulders, and hips need to be calculated from the skeletal points so that the angles can be used to operate the actuators. Since the given coordinates are relative Cartesian coordinates of each of the joints in the above picture, angles that approximate the proper servo settings in order to best match the position of the robot to that of the operator must be produced.

When designing the software for the system, the software components for the knees and elbows were designed first because of their similarity and overall straight forwardness in relating the Kinect outputs to the actuators. For the purposes of this system, the knee and elbow methodology is identical to one another with some differences in limits and starting positions. The goal of the software is to calculate the angle for the elbow joints; this can be done because the positions of the shoulder, elbow, and wrist joints are known. In this situation, the vector pointing from the wrist to the elbow will be defined as vector “a”. The vector pointing from the elbow to the shoulder will be defined as vector “b”. By solving for theta in the following equation, the angle for the elbow can be determined.

$$A = |\mathbf{a} \times \mathbf{b}| = |\mathbf{a}||\mathbf{b}| \sin \theta_1 \quad (2)$$

By taking this equation and solving for theta, you obtain the following equation.

$$\theta_1 = \text{Arcsin}\left(\frac{|\mathbf{a} \times \mathbf{b}|}{|\mathbf{a}||\mathbf{b}|}\right) \quad (3)$$

By adding a scaling factor, theta can be directly applied to the actuators, which will allow for movements that coincide with elbow motions captured by the Kinect. The difference between the elbow and the knee is that vector “a” is the vector pointing from the knee to

the ankle, while vector “b” is the vector pointing from the knee to the hip. As with the elbow, theta requires a scaling factor before it can be used, though this factor can be different than the elbow. Since software limits also need to be put in place, these limits contain the maximum range in which these joints can move. Along with limits, the start positions are also defined. These positions are the at-rest positions or the positions the actuators return to when they become unpowered. In the case of the knee, the angle between the upper leg and the lower leg will be 180 degrees. This position also is identical for the elbow joint with relation to the upper and lower arm.

The software design for the hips and shoulders is considerably different than the design of the elbows and knees. These joints require a different methodology because both joints are universal ball joints, while the robots joints are not. Before the math can be discussed, the axis of motion must be defined. The first axis will be called the “Lifter axis”, which produces the motion to raise the arms in front of the body. The second axis will be called the “Flexor axis”, which produces the motion to raise the arms up the side of the body creating a “T”. Next, the vectors that will be used in the calculation must be defined. The vector, which points from the shoulder to the elbow, will be defined as vector “a”. Vector “b” is the vector that points from the shoulder to the hip. Vector “c” is the vector that points from the center of the shoulders to the shoulders. The first angle of interest is the angle between vectors “a” and “b”. This can be calculated using the same equation used in the elbow calculations and solving for θ_1 . The second angle is difficult to visualize and is the angle that, when looking down upon a person, their arm is pointing. Zero degrees would be an arm pointing away from the side of the body; ninety degrees would be an arm pointing away from the front of the body. This angle is

calculated by ignoring the z components of the vectors “c” and “a” and finding the angle between them. This angle can be found using the same relation from before but it can be seen below in the following equations.

$$A = |\mathbf{a} \times \mathbf{c}| = |\mathbf{a}||\mathbf{c}| \sin \theta_2 \quad (4)$$

$$\theta_2 = \text{Arcsin}\left(\frac{|\mathbf{a} \times \mathbf{c}|}{|\mathbf{a}||\mathbf{c}|}\right) \quad (5)$$

Once this second angle is determined, proper values for the “Lifter” and “Flexor” servos can be found using the following processes. Taking θ_1 and multiplying it by the cosine of θ_2 will solve for the lifter actuator; this can be seen in the equation below.

$$\theta_L = \theta_1 \cos \theta_2 \quad (6)$$

By taking θ_1 and multiplying it by the sine of θ_2 will solve for the flexor actuator; which can be seen in the equation below.

$$\theta_F = \theta_1 \sin \theta_2 \quad (7)$$

By utilizing these equations, the motions of the skeletal points created by the Kinect software can be replicated by the actuators on the robot.

The final component of the software system is the interface with the Bioloid controller. This is handled by .NET’s serial communication libraries, which make interfacing simple. First an initialization is done upon the program starting, which sets up a COM port for use with the controller. While the program is running, only writing values to the port are necessary. In order to communicate with the CM-5 controller, a few commands are used. These commands replicate the controller being connected to the “roboplus” program. These commands consist of some initializers and terminators, and a command that writes values to the actuators. This command requires that you change either 1 actuator value or all of them. It was decided to update all of the servos after one

cycle. By combining the .NET framework and C#, a software system was developed that has the ability to interface with the Kinect SDK and the CM-5 controller simultaneously, while handling real-time mathematical calculations which drives the angular positions of each of the actuators built into the system. This leads to a robust system that drives a total of sixteen actuators in a real time environment.

Chapter IV

Results

Functional Demo

The objective of this project was to produce a fully functional demonstration that could be displayed in the Daytona Beach Museum of Arts and Sciences. This demonstration was required to not only look like a skeleton, but to function like one. It needed to be able to have a person stand in front of it and perform movement, and the systems would then replicate the motions to some degree of accuracy. This varying accuracy represents the accuracies given from the mechanical and the sensor systems. The overall system is deemed accurate since the motions it makes are representative of the motions that were taken as inputs. This can be confusing because one of the current limitations of the system is its speed. The system cannot replicate someone waving their arms rapidly because the number of total inputs and rate the input are coming is too fast for the actuators and the sensor to be able to replicate and record.

The current form for the system is a functionally complete demo, which needs polishing touches to become museum quality. The overall system is complete with all the actuators and sensors being integrated into the final design and super structure. The quality issue with this system is that the Kinect sensor is not mounted and the robot itself needs to be placed in a case; these issues can be addressed at a later date. The other major issue is that the system currently runs off a personal computer so a demonstration computer needs to be purchased. The system is currently not operable without a user to turn the system on. The issues that separate the system from being museum ready do not separate the system from being complete as far as the research aspect is concerned.

Conceptually and physically the system is ready to perform demonstrations and the overall system and software is being tested for robustness. The system needs to be mechanically robust since it will be expected to operate 8 hours a day at the museum. Currently there is no way to test the system for full days because of the time needed to test this. While testing and integrating the software, the overall system performance is also being tested. As with many demonstrations that Embry-Riddle Aeronautical University makes for the museum, this demonstration will be fully stress tested in its actual environment with sample group sizes from the classes that are being taught at the museum, instead of being setup as a display immediately. This sample testing will allow for the team to see flaws and potential software and hardware malfunctions. These software and hardware failures will then be able to be fixed or redesigned to be more robust.

The final design came together as expected and matches the overall design shown in the CAD models made by the team. The final system features a total of 16 fully functional mechanical single axis actuators to create a combination of eight functional joint and four individual joint types. This completes a third scale human skeleton, which can replicate the motions of a human to some extent. The final super structure is built out of three sub-structures; those being the arm structure, the leg structure, and the base structure. The leg and arm structures are hard mounted to the base structure to complete the final super structure; the final system can be seen in Figure 29. Within the structure, the controller is mounted and the wiring is run throughout the different sub-structures; this wiring style allows for a reduced risk of kinks and snags during operation.

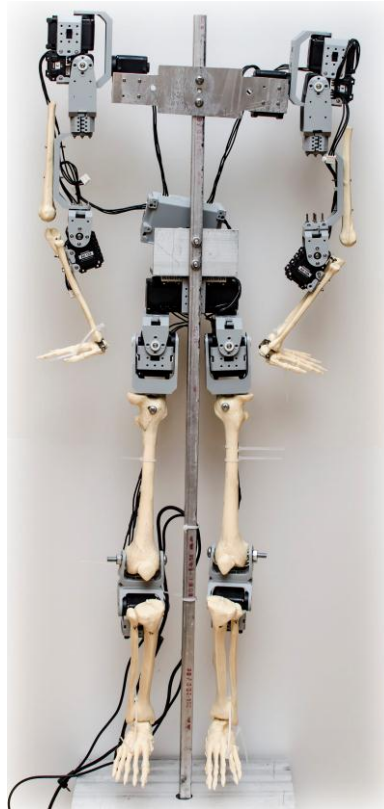


Figure 29 Completed super structure minus the skeleton body

In order for the demonstration to be considered successful, each of the joints needed to meet a certain specification based upon the average range of motion the corresponding joint can move on the human body. The first joint that was discussed was the elbow joint, with a range of motion of 140 degrees plus 10 degrees in hyperextension. The 10 degrees is noted separately because the joint designed for the robot was designed to operate in the normal range of motion for the elbow joint. By definition, hyperextension is a motion that is greater than normal extension. The designed joint has a range of motion of roughly 150 degrees, with the extra 10 degrees in the flexion range of motion. The reason the joint can't go into the hyperextension range is because of a mechanical collision between the two bones that are mounted on the fore arm and upper arm; Figure 30 shows the final elbow joint construction.



Figure 30 Final elbow joint construction

The knee joint was nearly identical in design as the elbow joint and displays the same range of motion, minus the hyperextension as well. The maximum range of motion for the knee is 150 degrees in the flexion motion. The designed knee joint was able to exceed the range of motion needed by having a total range of motion of about 155 degrees in flexion and another 5 degrees in hyperextension. This is due to the mechanical features of the actuator in comparison to the ligaments that restrict motion in the knee joint; Figure 31 shows the final knee joint construction.



Figure 31 Final knee joint construction

The hip joint had several different motions that needed to be categorized since it was a different type of joint than the both the elbow and knee joints. The hip joint is a ball joint that had three different types of motions; these motions consisted of medial/lateral rotations, flexion/extension movements, and abduction/adduction motions. The designed joint needed to be able to reproduce a range of motion of 140 degrees in flexion/extension, 80 degrees of rotation in the abduction/adduction, and 70 degrees in the medial/lateral rotations. The hip joint that was built and designed was able to meet and exceed the range of motion needed in the flexion and extension range by producing a maximum range of about 160 degrees. The medial and lateral rotations could also be reproduced exactly. The abduction and adduction motions need to be specially noted because the range of motion reproduced meets the 80 degrees necessary, but the adduction motion has the ability to cross the adjacent leg and this motion has been disabled to prevent the limbs from tangling with one another. In practice and testing, the hip joint can closely follow the motion made by an actual human hip; Figure 32 shows the final hip joint construction.



Figure 32 Final hip joint construction

As with the hip joint, the shoulder joint is an extremely versatile joint with respect to its ability to produce four different and unique types of movements. This ball joint, unlike the hip, is unrestricted as far as movements are concerned and has motions that consist of circumduction, abduction/adduction, medial rotations, and extension/flexion motions. Since the shoulder can perform the circumduction motion, the designed joint needed to be able to make a circular motion also; this could be completed to some degree. The limiting factor on the designed joint ability to perform circumduction is the radius of the circumduction being performed. For instance, a minor circumduction or a very small circle could not even be sensed, let alone it could not be performed accurately; while a full circumduction or the largest circle the arms can make can be performed up to 359 degrees of rotation. This full circle limitation is a software check to prevent wires from binding. The act of performing any sort of medial rotation was not factored into the design because this motion is a shrug and the Kinect cannot sense this type of movement. If this motion were necessary, the current design would need to be rethought because there is only rotation movements in the joint and a system with a cam might need to be integrated to perform the shrugging motion. The abduction/adduction motion in a human shoulder is 90 degrees in each direction. The designed joint can perform a combined range of about 150 degrees because it is limited by software. The extra 30 degrees crosses the body and this could be a potential area for entanglement. The flexion motion is 90 degrees, while the extension motion is only 50 degrees. The designed joint can perform 359 degrees in this type of motion because of the abilities shown in a purely rotational joint. Given all the movements that can be made by a shoulder versus the movements

deemed necessary, the designed joint meets the requirements of the projects; Figure 33 shows the final shoulder joint construction.

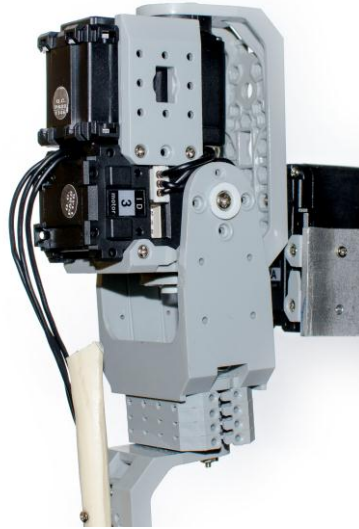


Figure 33 Final shoulder joint construction

Once the system was fully constructed, full scale testing began. The original specifications of the project were to produce near real time movements that mimicked human inputs taken from the Kinect. Unfortunately, the current controller the system is using does not allow for the movement speed necessary to create near real time mimicry. Another limiting factor for this specification is the potential for the system to damage itself. This slowness is also caused by the use of the CM-5 controller, which still has the Bioloid software loaded onto it and uses the same movement speeds and algorithms from the Bioloid system. Since the system and the software are both in their prototype version, the overall robustness of the system has yet to be tested. Until the system is completely tested, utilizing slower speeds will lead to less mechanical issues and less down time. The plan is to incorporate a new controller, which will allow for increased operational speeds later in the project's life.

Reliability Testing

Since this system is being designed for use as a museum demonstration, it is imperative that the overall robustness of the system is known. The idea for this project is to have it operating during all hours the museum is open. This of course can lead to many hours of use being put on every mechanical component of the system. The system itself can be broken down in two possible subsystems that need to be tested. The mechanical system, which encompasses everything that moves and if broken would impair the systems overall functionality. The other subsystem would be the hardware and software systems. This system encompasses the sensors, the controller, and the software itself. Unfortunately, with this systems design, if one component of these subsystems fails the whole system will fail.

The mechanical subsystem has many opportunities to fail, mainly in each actuator being used in the system. Fortunately enough, these particular actuators have safety systems built into them that attempt to mitigate any damage that the actuator may incur by shutting them down before the actuator is damaged. An example of this safety system in action comes when a leg raise is being performed. This is inherently the worst-case scenario of the actuators. This being that the leg is the longest and heaviest limb on the skeleton thus generating the highest required holding torque. In a electric actuator such as the ones being used, high torque holding loads require more power and more power generates more heat. After about 15-20 seconds the actuator will power itself down and flash red showing it has disabled itself. The actuator at this point is not damaged but requires the system to be reset. The leg is the only instance in which holding a position will cause the actuator to eventually shut itself down. Other cases of the actuator

disabling itself are if it isn't properly fastened and if the actuator becomes stuck. An example of the actuator becoming stuck is if the arm becomes tangled with the rib cage. This in turn provides an infinite amount of resistance that the actuator cannot overcome; in this instance the actuator will also shut itself down. Unfortunately, resetting the system will not overcome this issue because the arm will still be entangled with the rib cage so the arm will have to be physically removed. To prevent issues such as these, software stops have been put in place at the expense of movements such as crossing legs and arms. To prevent the actuators from becoming loose, Loctite should be used on all fasteners.

Another type of failure in the mechanical system would be the permanent failure of any actuator. During testing and development this was seen on one actuator. After performing an analysis on the actuator, it was deemed that a particle made its way into the motor and seized it. Given these actuators were not designed to handle such work cycles, it is a matter of time in which they will fail. The only way to solve this issue is to replace the broken actuators or replace them when the whole system receives service. Replacing actuators before they fail will allow the system to operate continuously but it comes with an added cost. While only replacing broken actuators reduces costs but increases the potential for down time. Unfortunately there is no way to avoid the eventual failure of each actuator. This being known, the system has been designed to allow for the replacement of the actuators with minimal hassle.

The final mechanical failure would be any of the brackets failing. This would result in actuators becoming loose or falling off. This is an unlikely situation but if it were to occur, the only solution would be to replace the bracket. This was one of the driving factors behind selecting COTS brackets because replacing them is cheap and

simple. Replacing some brackets is easier than replacing others but the risk of failure is limited. In order to test the system on the mechanical side, the whole system must be tested. The test plan for the mechanical side is to test during development and then to run small group tests until the team feels the system is ready for full scale testing. Issues and failures will be documented and if needed, a redesign can be made.

Failures to the hardware and software subsystem are just as debilitating as having a mechanical failure. The only sensor the system uses is the Kinect and if that were to fail, the whole system would be incapable of performing what it was meant to do. Fortunately Microsoft did a lot of the reliability testing on the Kinect already. If the Kinect fails, it will need to be replaced, otherwise there will be no sensor for motion capture. The system will still be able to function as far as being manually operated but the demo will be useless. The Kinect is designed to last many years and should not require the maintenance or have the risk of failing as some of the mechanical components do.

The CM-5 controller is another piece of hardware that if it fails, the system will not operate. This piece can fail in many different ways such as having a short or overheating. In either of these instances the controller will not function or function properly, which means there is no way to control the actuators. If the controller fails, it will need to be replaced. Once again, this is an easily obtainable part and is easy to install. The reliability of this component once again has been tested by Robotis and has less of a chance of failure than some components on the mechanical side. Nonetheless, if this system is to be used as a museum demonstration, a spare controller or two should be kept on hand just in case a failure in the current controller should happen.

The final component in the hardware software subsystem that could fail is the software itself. Software in terms can fail, however the software never worked to begin with if it fails. These are called glitches as apposed to failures and if one of these were to occur, tests to reproduce the issue would need to be performed. This needs to be done to find the failure mode of the glitch. When this is found, a software patch will need to be written and the software will then need to be updated. Unfortunately, if this occurs while the system is at the museum, it will not be a quick fix because someone that is familiar with the system will need to fix it or someone who is unfamiliar will need to come up to speed with the system before they can fix it. This is why a long-term small group test will need to be performed in order to discover these glitches and solve them before the system is released as a final product.

Cost Analysis

Along with designing and building a system that can perform all the necessary requirements, it was imperative to keep costs down. This was necessary because the project was internally funded and was going to be for a museum. The overall cost of the project does not factor in engineering time, which can be extremely expensive compared to the rest of the costs on the project. Fortunately, using COTS brackets, which came in the Bioloid kit, reduced a lot of the costs of the project. The kit consisted of 18 actuators, the controller, and all the brackets that were needed to build the system. The skeleton was purchased from a medical display website and the aluminum that was used was given to the project; Table 1 shows the cost of the project. As can be seen from the table below, the overall system cost is extremely low.

Table 1 Total system cost

Bioid	\$1,200
Skeleton	\$45
Misc items /Kinect	\$300
Total	\$1,545

It should be noted that this was the cost to build the system and does not factor in the necessary things to turn this into a museum quality demonstration. In order for this to be put into a museum, a computer will need to be purchased for the purpose of running the software. This computer can be purchased for around \$1000. The other necessary item is a case to enclose the system. If the system were not put in a protective case, people trying to interact with it would surely damage it. A custom case can vary in cost because of material selection and overall appearance. The custom case should cost no more than \$2000. That being said, a complete museum solution would cost in the range of \$5000 to produce. Given the average museum demo costs tens of thousands of dollars, this can be considered a low cost piece for a museum to fund or purchase.

Chapter V

Conclusions and Recommendations

Conclusions

The goal of this project was to design a low cost system that can mimic human movements. Through this process, the design team used readily available cutting edge technology to accomplish the overall goals set for the project. Creating a system which can mimic every type of motion the major joint in the body can do is an extremely difficult task. The original specifications of the project ended up being over ambitious and the final product did not meet 100 percent of the original specifications set forth in the beginning of the project Table 2 shows the movement capabilities of the system.

Table 2 System Ranges

Body Part	Movement	Range Required	Range Performed	Percent Error
Elbow	Flexion/Extension	140°	140°	0.00%
Elbow	Hyper Extension	10°	10°	0.00%
Knee	Flexion/Extension	150°	150°	0.00%
Hip	Flexion/Extension	140°	135°	3.57%
Hip	Abduction/Adduction	160°	150°	6.25%
Hip	Medial Rotation	70°	35°	50.00%
Shoulder	Circumduction	360°	360°	0.00%
Shoulder	Flexion/Extension	140°	140°	0.00%
Shoulder	Abduction/Adduction	160°	150°	6.25%
Shoulder	Medial Rotation	180°	0°	100.00%

The final product required making balanced decisions for performance, speed, and cost to build the overall system. Designing a system that could incorporate all the motions of the shoulder would have been costly and the end product would have been more complex than the current system. The speed issues were an oversight in design

when selecting the controller, which became the limiting factor in movement speed. The overall cost of the project did fall below original estimates by \$2000 dollars. The important issue to realize for this system is that the final system is meant to be a public display. In the showings and small-scale group testing that has been performed, the audience and users have been impressed with the demonstration. This is a valid representation of what could be expected from future museum patrons.

Recommendations

Before the system is ready for full time use, a long duration stress test should be performed. The robustness of the system should be demonstrated to museum representatives before placing the system in the museum. This should be done by leaving the system on for twelve hours at a time and allowing people to use it. Before the system can be placed in a public scenario, a computer must be purchased to run the system and a user interface must be designed. This will allow the system to be a near turnkey system for whoever uses it. The robot must also be enclosed so no tampering can occur. A service plan should also be created so that people who were not related in the design of the system can service and replace broken parts if needed. Since the long-term goal is for this system to become a product, these key issues listed above are necessary in the product testing and evaluation stages.

References

- [1]Jeong, Woong, Yong-Ho Seo, and Hyun Yang. "Effective Humanoid Motion Generation based on Programming-by-Demonstration Method for Entertainment Robotics." Virtual Systems and Multimedia (VSMM), 2010 16th International Conference on. (2010): 286-292. Print.
- [2]Yebin Liu; Stoll, C.; Gall, J.; Seidel, H.-P.; Theobalt, C.; , "Markerless motion capture of interacting characters using multi-view image segmentation," Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on , vol., no., pp.1249-1256, 20-25 June 2011
- [3]Elisabetta Farella, Luca Benini, Bruno Riccò, and Andrea Acquaviva, "MOCA: A Low-Power, Low-Cost Motion Capture System Based on Integrated Accelerometers," Advances in Multimedia, vol. 2007, Article ID 82638, 11 pages, 2007.
- [4]Stowers, J.; Hayes, M.; Bainbridge-Smith, A.; , "Altitude control of a quadrotor helicopter using depth map from Microsoft Kinect sensor," Mechatronics (ICM), 2011 IEEE International Conference on , vol., no., pp.358-362, 13-15 April 2011
- [5]AX-12a Datasheet. Nov 23, 2011. <http://www.trossenrobotics.com/>
- [6] Yangming Xu; Hollerbach, J.M.; , "A robust ensemble data method for identification of human joint mechanical properties during movement," *Biomedical Engineering, IEEE Transactions on* , vol.46, no.4, pp.409-419, April 1999
- [7] Hamilton, H. (1997) "Range of Motion." *web based labs*. Roawn, n.d.
- [8] MACKENZIE, B. (2004) *Range of Movement (ROM)* [WWW] Available from: <http://www.brianmac.co.uk/musrom.htm>
- [9] MyungJin Kang; Sadri, H.; Mocozet, L.; Magnenat-Thalmann, N.; Hoffmeyer, P.; , "Accurate simulation of hip joint range of motion," Computer Animation, 2002. Proceedings of. Vol., no., pp.215-219, 2002
- [10] BingYan Cui; Zhenlin Jin; , "Accuracy analysis of a novel humanoid robot shoulder joint," Reconfigurable Mechanisms and Robots, 2009. ReMAR 2009. ASME/IFTOMM International Conference on , vol., no., pp.423-427
- [11] Calais-Germain, Blandine. "Anatomy of Movement", Eastland Press, 1993. ISBN 0-939616-17-3

- [12] Mustafa, S.K.; Song Huat Yeo; Cong Bang Pham; Guilin Yang; Wei Lin; , "A biologically-inspired anthropocentric shoulder joint rehabilitator: workspace analysis & optimization," *Mechatronics and Automation*, 2005 IEEE International Conference , vol.2, no., pp. 1045- 1050 Vol. 2, 29 July-1 Aug. 2005
- [13] Lu Xia; Chia-Chih Chen; Aggarwal, J.K.; , "Human detection using depth information by Kinect," *Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2011 IEEE Computer Society Conference on , pp.15-22, 20-25 June 2011
- [14] "Kinect Hardware."Open Kinect. N.p., n.d. Web. Oct 31st 2011. <http://openkinect.org/wiki/Hardware_info>.
- [15] Rakprayoon, Panjawee; Ruchanurucks, Miti; Coundoul, Ada; , "Kinect- based obstacle detection for manipulator," *System Integration (SII)*, 2011 IEEE/SICE International Symposium on , pp.68-73, 20-22 Dec. 2011
- [16] Atmega128 Datasheet. Nov 2011 <http://www.atmel.com/devices/atmega128.aspx>
- [17] Arbotix Datasheet Dec 29, 2011 <http://www.vanadiumlabs.com/arbotix>
- [18] MacCormick, John. "How Does the Kinect Work." 27 April 2012. <<http://users.dickinson.edu/~jmac/selected-talks/kinect.pdf>>.
- [19] Vitruvian Man. Graphic. Web. 20 June 2012. <<http://www.edalabs.com/products-mainmenu-38/vitruvian-dentures>>.
- [20] J. Gall, C. Stoll, E. Aguiar, C. Theobalt, B. Rosenhahn, and H.-P. Seidel. Motion capture using joint skeleton tracking and surface estimation. In *CVPR*, pages 1746–1753, 2009.
- [21] Mr Thrifty Skeleton. N.d. Photograph. Skeletons and More, Web. Jul2011. <<http://www.skeletonsandmore.com/securecart/index.php>>
- [22] AX-12 Dual Gripper Hardware Kit. N.d. Photograph. Crust Crawler, Web. May 2012. <<http://crustcrawler.com/products/dualgrripper/index.php>>.
- [23] AX Series Dynamixel Brackets . N.d. Photograph. Trossen Robotics, Web. June 2011. <<http://www.trossenrobotics.com/store/p/6179-Bioloid-Frame-F1>>
- [24] Joints and Movements Fact Sheet. N.d. Photograph. Hoyle Science, Web. June 2012. <<http://hoylescience.pbworks.com/w/page/39944954/Joints>>.
- [25] Bioloid Premium. N.d. Photograph. Robot Shop, Web. April 2012. <<http://www.robotis-shop-en.com/shop/step1.php?number=733>>.

[26] El-laithy, Riyad A.; Huang, Jidong; Yeh, Michael; , "Study on the use of Microsoft Kinect for robotics applications," Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION , vol., no., pp.1280-1288, 23-26 April 2012

[27] Jia, Weidi; Yi, Won-Jae; Saniie, Jafar; Oruklu, Erdal; , "3D image reconstruction and human body tracking using stereo vision and Kinect technology," Electro/Information Technology (EIT), 2012 IEEE International Conference on , vol., no., pp.1-4, 6-8 May 2012

Appendix A

Source Code

```
//-----  
// <copyright file="MainWindow.xaml.cs" company="Microsoft">  
//   Copyright (c) Microsoft Corporation. All rights reserved.  
// </copyright>  
//-----  
// This module contains code to do Kinect NUI initialization,  
// processing, displaying players on screen, and sending updated player  
// positions to the game portion for hit testing.  
namespace ShapeGame  
{  
    using System;  
    using System.Text;  
    using System.IO.Ports;  
    using System.Collections.Generic;  
    using System.ComponentModel;  
    using System.IO;  
    using System.Linq;  
    using System.Media;  
    using System.Runtime.InteropServices;  
    using System.Threading;  
    using System.Windows;  
    using System.Windows.Controls;  
    using System.Windows.Threading;  
    using Microsoft.Kinect;  
    using ShapeGame.Speech;  
    using ShapeGame.Utils;  
    /// <summary>  
    /// Interaction logic for MainWindow.xaml  
    /// </summary>  
    ///  
    public partial class MainWindow : Window  
    {
```

```

#region Private State

private const int TimerResolution = 2; // ms

private const int NumIntraFrames = 3;

private const int MaxShapes = 80;

private const double MaxFramerate = 70;

private const double MinFramerate = 15;

private const double MinShapeSize = 12;

private const double MaxShapeSize = 90;

private const double DefaultDropRate = 2.5;

private const double DefaultDropSize = 32.0;

private const double DefaultDropGravity = 1.0;

private readonly Dictionary<int, Player> players = new Dictionary<int, Player>();

private readonly SoundPlayer popSound = new SoundPlayer();

private readonly SoundPlayer hitSound = new SoundPlayer();

private readonly SoundPlayer squeezeSound = new SoundPlayer();

private double dropRate = DefaultDropRate;

private double dropSize = DefaultDropSize;

private double dropGravity = DefaultDropGravity;

private DateTime lastFrameDrawn = DateTime.MinValue;

private DateTime predNextFrame = DateTime.MinValue;

private double actualFrameTime;

private Skeleton[] skeletonData;

// Player(s) placement in scene (z collapsed):

private Rect playerBounds;

private Rect screenRect;

private double targetFramerate = MaxFramerate;

private int frameCount;

private bool runningGameThread;

private FallingThings myFallingThings;

private int playersAlive;

public int count = 0;

public int RightElbowIndex = 5; public int LeftElbowIndex = 6; public int RightKneeIndex = 11; public int LeftKneeIndex = 13;

public int RightShoulderFlexorIndex = 3; public int LeftShoulderFlexorIndex = 4; public int RightShoulderSpinnerIndex = 10;
public int LeftShoulderSpinnerIndex = 1;

public int RightShoulderLifterIndex = 14; public int LeftShoulderLifterIndex = 11; public int RightHipLifterIndex = 16; public
int LeftHipLifterIndex = 15;

```

```

    public int RightHipFlexorIndex = 18; public int LeftHipFlexorIndex = 17; public int RightHipSpinnerIndex = 7; public int
    LeftHipSpinnerIndex = 8;
    public int RightElbowStart = 517; public int LeftElbowStart = 550; public int RightKneeStart = 158; public int LeftKneeStart =
    527;

    public int RightShoulderFlexorStart = 504; public int LeftShoulderFlexorStart = 810; public int RightShoulderSpinnerStart =
    833; public int LeftShoulderSpinnerStart = 852;

    public int RightShoulderLifterStart = 194; public int LeftShoulderLifterStart = 670; public int RightHipLifterStart = 519; public
    int LeftHipLifterStart = 193;

    public int RightHipFlexorStart = 510; public int LeftHipFlexorStart = 528; public int RightHipSpinnerStart = 237; public int
    LeftHipSpinnerStart = 522;

    public int RightElbowMin = 187; public int LeftElbowMin = 550; public int RightKneeMin = 500; public int LeftKneeMin =
    500;

    public int RightShoulderFlexorMin = 219; public int LeftShoulderFlexorMin = 550; public int RightShoulderSpinnerMin = 500;
    public int LeftShoulderSpinnerMin = 500;

    public int RightShoulderLifterMin = 500; public int LeftShoulderLifterMin = 300; public int RightHipLifterMin = 500; public int
    LeftHipLifterMin = 500;

    public int RightHipFlexorMin = 500; public int LeftHipFlexorMin = 500; public int RightHipSpinnerMin = 500; public int
    LeftHipSpinnerMin = 500;

    public int RightElbowMax = 489; public int LeftElbowMax = 847; public int RightKneeMax = 500; public int LeftKneeMax =
    500;

    public int RightShoulderFlexorMax = 505; public int LeftShoulderFlexorMax = 820; public int RightShoulderSpinnerMax = 500;
    public int LeftShoulderSpinnerMax = 500;

    public int RightShoulderLifterMax = 500; public int LeftShoulderLifterMax = 700; public int RightHipLifterMax = 500; public
    int LeftHipLifterMax = 500;

    public int RightHipFlexorMax = 500; public int LeftHipFlexorMax = 500; public int RightHipSpinnerMax = 500; public int
    LeftHipSpinnerMax = 500;

    public int[] StartingValues = new int[19];

    public int[] CurrentValues = new int[19];

    public int[] MinValues = new int[19];

    public int[] MaxValues = new int[19];

    private SpeechRecognizer mySpeechRecognizer;

    SerialPort serialPort1;

    #endregion Private State

    #region ctor + Window Events

    public MainWindow()

    {

        InitializeComponent();

        this.RestoreWindowState();

    }

    // Since the timer resolution defaults to about 10ms precisely, we need to

    // increase the resolution to get framerate above between 50fps with any

    // consistency.

```

```

[DllImport("Winmm.dll", EntryPoint = "timeBeginPeriod")]

private static extern int TimeBeginPeriod(uint period);

private void RestoreWindowState()
{
    // Restore window state to that last used

    Rect bounds = Properties.Settings.Default.PrevWinPosition;

    if (bounds.Right != bounds.Left)
    {
        this.Top = bounds.Top;

        this.Left = bounds.Left;

        this.Height = bounds.Height;

        this.Width = bounds.Width;
    }

    this.WindowState = (WindowState)Properties.Settings.Default.WindowState;
}

private void WindowLoaded(object sender, EventArgs e)
{
    serialPort1 = new SerialPort("COM3", 57600);

    serialPort1.DataBits = 8;

    serialPort1.Parity = Parity.None;

    serialPort1.StopBits = StopBits.One;

    serialPort1.Open();

    StartingValues[3] = 0; StartingValues[9] = 0;

    StartingValues[RightElbowIndex] = RightElbowStart; StartingValues[LeftElbowIndex] = LeftElbowStart;
    StartingValues[RightKneeIndex] = RightKneeStart; StartingValues[LeftKneeIndex] = LeftKneeStart;

    StartingValues[RightShoulderFlexorIndex] = RightShoulderFlexorStart; StartingValues[LeftShoulderFlexorIndex] =
    LeftShoulderFlexorStart;

    StartingValues[RightShoulderSpinnerIndex] = RightShoulderSpinnerStart; StartingValues[LeftShoulderSpinnerIndex] =
    LeftShoulderSpinnerStart;

    StartingValues[RightShoulderLifterIndex] = RightShoulderLifterStart; StartingValues[LeftShoulderLifterIndex] =
    LeftShoulderLifterStart;

    StartingValues[RightHipFlexorIndex] = RightHipFlexorStart; StartingValues[LeftHipFlexorIndex] = LeftHipFlexorStart;

    StartingValues[RightHipSpinnerIndex] = RightHipSpinnerStart; StartingValues[LeftHipSpinnerIndex] = LeftHipSpinnerStart;

    StartingValues[RightHipLifterIndex] = RightHipLifterStart; StartingValues[LeftHipLifterIndex] = LeftHipLifterStart;

    CurrentValues[RightElbowIndex] = RightElbowStart; CurrentValues[LeftElbowIndex] = LeftElbowStart;
    CurrentValues[RightKneeIndex] = RightKneeStart; CurrentValues[LeftKneeIndex] = LeftKneeStart;

    CurrentValues[RightShoulderFlexorIndex] = RightShoulderFlexorStart; CurrentValues[LeftShoulderFlexorIndex] =
    LeftShoulderFlexorStart;

```

CurrentValues[RightShoulderSpinnerIndex] = RightShoulderSpinnerStart; CurrentValues[LeftShoulderSpinnerIndex] = LeftShoulderSpinnerStart;

CurrentValues[RightShoulderLifterIndex] = RightShoulderLifterStart; CurrentValues[LeftShoulderLifterIndex] = LeftShoulderLifterStart;

CurrentValues[RightHipFlexorIndex] = RightHipFlexorStart; CurrentValues[LeftHipFlexorIndex] = LeftHipFlexorStart;

CurrentValues[RightHipSpinnerIndex] = RightHipSpinnerStart; CurrentValues[LeftHipSpinnerIndex] = LeftHipSpinnerStart;

CurrentValues[RightHipLifterIndex] = RightHipLifterStart; CurrentValues[LeftHipLifterIndex] = LeftHipLifterStart;

MinValues[RightElbowIndex] = RightElbowMin; MinValues[LeftElbowIndex] = LeftElbowMin;
MinValues[RightKneeIndex] = RightKneeMin; MinValues[LeftKneeIndex] = LeftKneeMin;

MinValues[RightShoulderFlexorIndex] = RightShoulderFlexorMin; MinValues[LeftShoulderFlexorIndex] = LeftShoulderFlexorMin;

MinValues[RightShoulderSpinnerIndex] = RightShoulderSpinnerMin; MinValues[LeftShoulderSpinnerIndex] = LeftShoulderSpinnerMin;

MinValues[RightShoulderLifterIndex] = RightShoulderLifterMin; MinValues[LeftShoulderLifterIndex] = LeftShoulderLifterMin;

MinValues[RightHipFlexorIndex] = RightHipFlexorMin; MinValues[LeftHipFlexorIndex] = LeftHipFlexorMin;

MinValues[RightHipSpinnerIndex] = RightHipSpinnerMin; MinValues[LeftHipSpinnerIndex] = LeftHipSpinnerMin;

MinValues[RightHipLifterIndex] = RightHipLifterMin; MinValues[LeftHipLifterIndex] = LeftHipLifterMin;

MaxValues[RightElbowIndex] = RightElbowMax; MaxValues[LeftElbowIndex] = LeftElbowMax;
MaxValues[RightKneeIndex] = RightKneeMax; MaxValues[LeftKneeIndex] = LeftKneeMax;

MaxValues[RightShoulderFlexorIndex] = RightShoulderFlexorMax; MaxValues[LeftShoulderFlexorIndex] = LeftShoulderFlexorMax;

MaxValues[RightShoulderSpinnerIndex] = RightShoulderSpinnerMax; MaxValues[LeftShoulderSpinnerIndex] = LeftShoulderSpinnerMax;

MaxValues[RightShoulderLifterIndex] = RightShoulderLifterMax; MaxValues[LeftShoulderLifterIndex] = LeftShoulderLifterMax;

MaxValues[RightHipFlexorIndex] = RightHipFlexorMax; MaxValues[LeftHipFlexorIndex] = LeftHipFlexorMax;

MaxValues[RightHipSpinnerIndex] = RightHipSpinnerMax; MaxValues[LeftHipSpinnerIndex] = LeftHipSpinnerMax;

MaxValues[RightHipLifterIndex] = RightHipLifterMax; MaxValues[LeftHipLifterIndex] = LeftHipLifterMax;

Thread.Sleep(4000);

serialPort1.Write("v E List\r\n");

Thread.Sleep(5000);

serialPort1.Write("on\r\n");

Thread.Sleep(100);

RobotWrite(CurrentValues);

Thread.Sleep(1000);

playfield.ClipToBounds = true;

this.myFallingThings = new FallingThings(MaxShapes, this.targetFramerate, NumIntraFrames);

this.UpdatePlayfieldSize()
this.myFallingThings.SetGravity(this.dropGravity);

this.myFallingThings.SetDropRate(this.dropRate);


```

        this.myFallingThings.SetSize(this.dropSize);

        this.myFallingThings.SetPolies(PolyType.All);

        this.myFallingThings.SetGameMode(GameMode.Off);

        SensorChooser.KinectSensorChanged += this.SensorChooserKinectSensorChanged;

        this.popSound.Stream = Properties.Resources.Pop_5;

        this.hitSound.Stream = Properties.Resources.Hit_2;

        this.squeezeSound.Stream = Properties.Resources.Squeeze;

        this.popSound.Play();

        TimeBeginPeriod(TimerResolution);

        var myGameThread = new Thread(this.GameThread);

        myGameThread.SetApartmentState(ApartmentState.STA);

        myGameThread.Start();

        FlyingText.NewFlyingText(this.screenRect.Width / 30, new Point(this.screenRect.Width / 2, this.screenRect.Height / 2),
"Shapes!");
    }

    private void WindowClosing(object sender, CancelEventArgs e)
    {
        this.runningGameThread = false;

        Properties.Settings.Default.PrevWinPosition = this.RestoreBounds;

        Properties.Settings.Default.WindowState = (int)this.WindowState;

        Properties.Settings.Default.Save();
    }

    private void WindowClosed(object sender, EventArgs e)
    {
        SensorChooser.Kinect = null;

        serialPort1.Close();
    }

    #endregion ctor + Window Events

    #region Kinect discovery + setup

    private void SensorChooserKinectSensorChanged(object sender, DependencyPropertyChangedEventArgs e)
    {
        if (e.OldValue != null)
        {
            this.UninitializeKinectServices((KinectSensor)e.OldValue);
        }
    }

```

```

// Only enable this checkbox if we have a sensor
enableAec.IsEnabled = e.NewValue != null;
if (e.NewValue != null)
{
    this.InitializeKinectServices((KinectSensor)e.NewValue);
}
}

// Kinect enabled apps should customize which Kinect services it initializes here.
private KinectSensor InitializeKinectServices(KinectSensor sensor)
{
    // Application should enable all streams first.
    sensor.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);
    sensor.SkeletonFrameReady += this.SkeletonsReady;
    sensor.SkeletonStream.Enable(new TransformSmoothParameters()
    {
        Smoothing = 0.5f,
        Correction = 0.5f,
        Prediction = 0.5f,
        JitterRadius = 0.05f,
        MaxDeviationRadius = 0.04f
    });
    try
    {
        sensor.Start();
    }
    catch (IOException)
    {
        SensorChooser.AppConflictOccurred();
        return null;
    }

    // Start speech recognizer after KinectSensor.Start() is called
    // returns null if problem with speech prereqs or instantiation.
    this.mySpeechRecognizer = SpeechRecognizer.Create();
    this.mySpeechRecognizer.SaidSomething += this.RecognizerSaidSomething;
    this.mySpeechRecognizer.Start(sensor.AudioSource);

    enableAec.Visibility = Visibility.Visible;
    this.UpdateEchoCancellation(this.enableAec)

    return sensor;
}

```

// Kinect enabled apps should uninitialize all Kinect services that were initialized in InitializeKinectServices() here.

```
private void UninitializeKinectServices(KinectSensor sensor)
{
    sensor.Stop();

    sensor.SkeletonFrameReady -= this.SkeletonsReady;

    if (this.mySpeechRecognizer != null)
    {
        this.mySpeechRecognizer.Stop();

        this.mySpeechRecognizer.SaidSomething -= this.RecognizerSaidSomething;

        this.mySpeechRecognizer.Dispose();

        this.mySpeechRecognizer = null;
    }

    enableAec.Visibility = Visibility.Collapsed;
}
```

#endregion Kinect discovery + setup

#region Kinect Skeleton processing

```
private void SkeletonsReady(object sender, SkeletonFrameReadyEventArgs e)
{
    using (SkeletonFrame skeletonFrame = e.OpenSkeletonFrame())
    {
        if (skeletonFrame != null)
        {
            int skeletonSlot = 0;

            if ((this.skeletonData == null) || (this.skeletonData.Length != skeletonFrame.SkeletonArrayLength))
            {
                this.skeletonData = new Skeleton[skeletonFrame.SkeletonArrayLength];
            }

            skeletonFrame.CopySkeletonDataTo(this.skeletonData);

            foreach (Skeleton skeleton in this.skeletonData)
            {
                if (SkeletonTrackingState.Tracked == skeleton.TrackingState)
                {
                    Player player;
```

```

if (this.players.ContainsKey(skeletonSlot))
{
    player = this.players[skeletonSlot];
}
else
{
    player = new Player(skeletonSlot);
    player.SetBounds(this.playerBounds);
    this.players.Add(skeletonSlot, player);
}

player.LastUpdated = DateTime.Now;

// Update player's bone and joint positions
if (skeleton.Joints.Count > 0)
{
    if (count == 1)
    {
        //Left Shoulder and Arm

        CurrentValues[LeftElbowIndex] = GetAngle4Points(skeleton.Joints[JointType.WristLeft].Position,
skeleton.Joints[JointType.ElbowLeft].Position, skeleton.Joints[JointType.ShoulderLeft].Position,

        skeleton.Joints[JointType.ElbowLeft].Position, LeftElbowIndex, LeftElbowStart, 1.0);

        double ShoulderLeftAngle2D = GetAngle4Points_2D_XY(skeleton.Joints[JointType.ShoulderRight].Position,
skeleton.Joints[JointType.ShoulderLeft].Position, skeleton.Joints[JointType.ElbowLeft].Position,

        skeleton.Joints[JointType.ShoulderLeft].Position);

        CurrentValues[LeftShoulderFlexorIndex] = GetAngle4Points(skeleton.Joints[JointType.ElbowLeft].Position,
skeleton.Joints[JointType.ShoulderLeft].Position, skeleton.Joints[JointType.HipLeft].Position,

        skeleton.Joints[JointType.ShoulderLeft].Position, LeftShoulderFlexorIndex, LeftShoulderFlexorStart+150, -
Math.Cos(ShoulderLeftAngle2D));

        CurrentValues[LeftShoulderLifterIndex] = GetAngle4Points(skeleton.Joints[JointType.ElbowLeft].Position,
skeleton.Joints[JointType.ShoulderLeft].Position, skeleton.Joints[JointType.HipLeft].Position,

        skeleton.Joints[JointType.ShoulderLeft].Position, LeftShoulderLifterIndex, LeftShoulderLifterStart, -
Math.Sin(ShoulderLeftAngle2D));

        //Right Shoulder and Arm

        CurrentValues[RightElbowIndex] = GetAngle4Points(skeleton.Joints[JointType.WristRight].Position,
skeleton.Joints[JointType.ElbowRight].Position, skeleton.Joints[JointType.ShoulderRight].Position,

        skeleton.Joints[JointType.ElbowRight].Position, RightElbowIndex, RightElbowStart, -1.0);

        double ShoulderRightAngle2D = GetAngle4Points_2D_XY(skeleton.Joints[JointType.ShoulderLeft].Position,
skeleton.Joints[JointType.ShoulderRight].Position, skeleton.Joints[JointType.ElbowRight].Position,

        skeleton.Joints[JointType.ShoulderRight].Position);

```

```

        CurrentValues[RightShoulderFlexorIndex] = GetAngle4Points(skeleton.Joints[JointType.ElbowRight].Position,
skeleton.Joints[JointType.ShoulderRight].Position, skeleton.Joints[JointType.HipRight].Position,

        skeleton.Joints[JointType.ShoulderRight].Position, RightShoulderFlexorIndex, RightShoulderFlexorStart +
150, -Math.Cos(ShoulderRightAngle2D));

        CurrentValues[RightShoulderLifterIndex] = GetAngle4Points(skeleton.Joints[JointType.ElbowRight].Position,
skeleton.Joints[JointType.ShoulderRight].Position, skeleton.Joints[JointType.HipRight].Position,

        skeleton.Joints[JointType.ShoulderRight].Position, RightShoulderLifterIndex, RightShoulderLifterStart, -
Math.Sin(ShoulderRightAngle2D));

//Right Hip and Leg

        CurrentValues[RightKneeIndex] = GetAngle4Points(skeleton.Joints[JointType.AnkleRight].Position,
skeleton.Joints[JointType.KneeRight].Position, skeleton.Joints[JointType.HipRight].Position,

        skeleton.Joints[JointType.KneeRight].Position, RightKneeIndex, RightKneeStart, -1.0);

        double HipRightAngle2D = GetAngle4Points_2D_XY(skeleton.Joints[JointType.HipLeft].Position,
skeleton.Joints[JointType.HipRight].Position, skeleton.Joints[JointType.KneeRight].Position,

        skeleton.Joints[JointType.HipRight].Position);

        CurrentValues[RightHipFlexorIndex] = GetAngle4Points(skeleton.Joints[JointType.KneeRight].Position,
skeleton.Joints[JointType.HipRight].Position, skeleton.Joints[JointType.ShoulderRight].Position,

        skeleton.Joints[JointType.HipRight].Position, RightHipFlexorIndex, RightHipFlexorStart + 150, -
Math.Cos(HipRightAngle2D));

        CurrentValues[RightHipLifterIndex] = GetAngle4Points(skeleton.Joints[JointType.KneeRight].Position,
skeleton.Joints[JointType.HipRight].Position, skeleton.Joints[JointType.ShoulderRight].Position,

        skeleton.Joints[JointType.HipRight].Position, RightHipLifterIndex, RightHipLifterStart, -
Math.Sin(HipRightAngle2D));

//Left Hip and Leg

        CurrentValues[LeftKneeIndex] = GetAngle4Points(skeleton.Joints[JointType.AnkleLeft].Position,
skeleton.Joints[JointType.KneeLeft].Position, skeleton.Joints[JointType.HipLeft].Position,

        skeleton.Joints[JointType.KneeLeft].Position, LeftKneeIndex, LeftKneeStart, -1.0);

        double HipLeftAngle2D = GetAngle4Points_2D_XY(skeleton.Joints[JointType.HipRight].Position,
skeleton.Joints[JointType.HipLeft].Position, skeleton.Joints[JointType.KneeLeft].Position,

        skeleton.Joints[JointType.HipLeft].Position);

        CurrentValues[LeftHipFlexorIndex] = GetAngle4Points(skeleton.Joints[JointType.KneeLeft].Position,
skeleton.Joints[JointType.HipLeft].Position, skeleton.Joints[JointType.ShoulderLeft].Position,

        skeleton.Joints[JointType.HipLeft].Position, LeftHipFlexorIndex, LeftHipFlexorStart + 150, -
Math.Cos(HipLeftAngle2D));

        CurrentValues[LeftHipLifterIndex] = GetAngle4Points(skeleton.Joints[JointType.KneeLeft].Position,
skeleton.Joints[JointType.HipLeft].Position, skeleton.Joints[JointType.ShoulderLeft].Position,

        skeleton.Joints[JointType.HipLeft].Position, LeftHipLifterIndex, LeftHipLifterStart, -
Math.Sin(HipLeftAngle2D));

        RobotWrite(CurrentValues);

        count = 0
    }

```

```

else

{
    count++;
}

player.IsAlive = true;

// Head, hands, feet (hit testing happens in order here)

player.UpdateJointPosition(skeleton.Joints, JointType.Head);

player.UpdateJointPosition(skeleton.Joints, JointType.HandLeft);

player.UpdateJointPosition(skeleton.Joints, JointType.HandRight);

player.UpdateJointPosition(skeleton.Joints, JointType.FootLeft);

player.UpdateJointPosition(skeleton.Joints, JointType.FootRight);

// Hands and arms

player.UpdateBonePosition(skeleton.Joints, JointType.HandRight, JointType.WristRight);

player.UpdateBonePosition(skeleton.Joints, JointType.WristRight, JointType.ElbowRight);

player.UpdateBonePosition(skeleton.Joints, JointType.ElbowRight, JointType.ShoulderRight);

player.UpdateBonePosition(skeleton.Joints, JointType.HandLeft, JointType.WristLeft);

player.UpdateBonePosition(skeleton.Joints, JointType.WristLeft, JointType.ElbowLeft);

player.UpdateBonePosition(skeleton.Joints, JointType.ElbowLeft, JointType.ShoulderLeft);

// Head and Shoulders

player.UpdateBonePosition(skeleton.Joints, JointType.ShoulderCenter, JointType.Head);

player.UpdateBonePosition(skeleton.Joints, JointType.ShoulderLeft, JointType.ShoulderCenter);

player.UpdateBonePosition(skeleton.Joints, JointType.ShoulderCenter, JointType.ShoulderRight);

// Legs

player.UpdateBonePosition(skeleton.Joints, JointType.HipLeft, JointType.KneeLeft);

player.UpdateBonePosition(skeleton.Joints, JointType.KneeLeft, JointType.AnkleLeft);

player.UpdateBonePosition(skeleton.Joints, JointType.AnkleLeft, JointType.FootLeft);

player.UpdateBonePosition(skeleton.Joints, JointType.HipRight, JointType.KneeRight);

player.UpdateBonePosition(skeleton.Joints, JointType.KneeRight, JointType.AnkleRight);

player.UpdateBonePosition(skeleton.Joints, JointType.AnkleRight, JointType.FootRight);

player.UpdateBonePosition(skeleton.Joints, JointType.HipLeft, JointType.HipCenter);

player.UpdateBonePosition(skeleton.Joints, JointType.HipCenter, JointType.HipRight);

// Spine

player.UpdateBonePosition(skeleton.Joints, JointType.HipCenter, JointType.ShoulderCenter);

}

}

```

```

        skeletonSlot++;

    }

}

}

}

public int GetAngle4Points(SkeletonPoint Vector1Point1, SkeletonPoint Vector1Point2, SkeletonPoint Vector2Point1,
SkeletonPoint Vector2Point2, int JointIndex, double offset, double scaling)
{
    float Vec1X = Vector1Point1.X - Vector1Point2.X;
    float Vec1Y = Vector1Point1.Y - Vector1Point2.Y;
    float Vec1Z = Vector1Point1.Z - Vector1Point2.Z;
    float Vec2X = Vector2Point1.X - Vector2Point2.X;
    float Vec2Y = Vector2Point1.Y - Vector2Point2.Y;
    float Vec2Z = Vector2Point1.Z - Vector2Point2.Z;
    float CrossX = Vec1Y * Vec2Z - Vec1Z * Vec2Y;
    float CrossY = Vec1X * Vec2Z - Vec1Z * Vec2X;
    float CrossZ = Vec1X * Vec2Y - Vec1Y * Vec2X;

    double MagCross = Math.Sqrt(Math.Pow(CrossX, 2) + Math.Pow(CrossY, 2) + Math.Pow(CrossZ, 2));
    double Mag2 = Math.Sqrt(Math.Pow(Vec2X, 2) + Math.Pow(Vec2Y, 2) + Math.Pow(Vec2Z, 2));
    double Mag1 = Math.Sqrt(Math.Pow(Vec1X, 2) + Math.Pow(Vec1Y, 2) + Math.Pow(Vec1Z, 2));
    double Angle = (Math.Asin(MagCross / (Mag1 * Mag2)) * 300) * scaling + offset;

    if (Angle > 1023)
    {
        Angle = Angle - 1023;
    }

    if (Angle < 0)
    {
        Angle = Angle + 1023;
    }
    if (Angle > MaxValues[JointIndex] || Angle < MinValues[JointIndex])
    {
        Angle = CurrentValues[JointIndex];
    }

    return ((int)Angle);
}

```

```

public double GetAngle4Points_2D_XY(SkeletonPoint Vector1Point1, SkeletonPoint Vector1Point2, SkeletonPoint
Vector2Point1, SkeletonPoint Vector2Point2)

```

```

{
    float Vec1X = Vector1Point1.X - Vector1Point2.X;
    float Vec1Y = Vector1Point1.Y - Vector1Point2.Y;
    float Vec1Z = 0;
    float Vec2X = Vector2Point1.X - Vector2Point2.X;
    float Vec2Y = Vector2Point1.Y - Vector2Point2.Y;
    float Vec2Z = 0;
    float CrossX = Vec1Y * Vec2Z - Vec1Z * Vec2Y;
    float CrossY = Vec1X * Vec2Z - Vec1Z * Vec2X;
    float CrossZ = Vec1X * Vec2Y - Vec1Y * Vec2X;
    double MagCross = Math.Sqrt(Math.Pow(CrossX, 2) + Math.Pow(CrossY, 2) + Math.Pow(CrossZ, 2));
    double Mag2 = Math.Sqrt(Math.Pow(Vec2X, 2) + Math.Pow(Vec2Y, 2) + Math.Pow(Vec2Z, 2));
    double Mag1 = Math.Sqrt(Math.Pow(Vec1X, 2) + Math.Pow(Vec1Y, 2) + Math.Pow(Vec1Z, 2));
    double Angle = (Math.Asin(MagCross / (Mag1 * Mag2)));
    return (Angle);
}

```

```

public void RobotWrite(int[] CurrentValues)

```

```

{
    serialPort1.Write("go ---- ");
    for (int i = 1; i < 19; i++)
    {
        if (i == 2)
        {
            serialPort1.Write(" ----");
        }
        else if (i == 12)
        {
            serialPort1.Write(" ----")
        }
        else if (i == 9)
        {
            serialPort1.Write(" ----");
        }
        else
        {

```



```

        //serialPort1.Write(i.ToString());

        serialPort1.Write(" ");

        string printedval = (String.Format("{0:000.}", CurrentValues[i]));

        serialPort1.Write(printedval);

    }

}

serialPort1.Write(" ---- -\r\n");

}

private void CheckPlayers()
{
    foreach (var player in this.players)
    {
        if (!player.Value.IsAlive)
        {
            // Player left scene since we aren't tracking it anymore, so remove from dictionary
            this.players.Remove(player.Value.GetId());

            break;
        }
    }

    // Count alive players
    int alive = this.players.Count(player => player.Value.IsAlive);

    if (alive != this.playersAlive)
    {
        if (alive == 2)
        {
            this.myFallingThings.SetGameMode(GameMode.TwoPlayer);
        }

        else if (alive == 1)
        {
            this.myFallingThings.SetGameMode(GameMode.Solo);
        }

        else if (alive == 0)
        {
            this.myFallingThings.SetGameMode(GameMode.Off);
        }
    }
}

```

```

        if ((this.playersAlive == 0) && (this.mySpeechRecognizer != null))
        {
            BannerText.NewBanner(
                Properties.Resources.Vocabulary,
                this.screenRect,
                true,
                System.Windows.Media.Color.FromArgb(200, 255, 255, 255));
        }
        this.playersAlive = alive;
    }
}

private void PlayfieldSizeChanged(object sender, SizeChangedEventArgs e)
{
    this.UpdatePlayfieldSize();
}

private void UpdatePlayfieldSize()
{
    // Size of player wrt size of playfield, putting ourselves low on the screen.
    this.screenRect.X = 0;
    this.screenRect.Y = 0;
    this.screenRect.Width = this.playfield.ActualWidth;
    this.screenRect.Height = this.playfield.ActualHeight;
    BannerText.UpdateBounds(this.screenRect);
    this.playerBounds.X = 0;
    this.playerBounds.Width = this.playfield.ActualWidth;
    this.playerBounds.Y = this.playfield.ActualHeight * 0.2;
    this.playerBounds.Height = this.playfield.ActualHeight * 0.75;
    foreach (var player in this.players)
    {
        player.Value.SetBounds(this.playerBounds);
    }

    Rect fallingBounds = this.playerBounds;
    fallingBounds.Y = 0;
    fallingBounds.Height = playfield.ActualHeight;
    if (this.myFallingThings != null)

```

```

    {
        this.myFallingThings.SetBoundaries(fallingBounds);
    }
}

#endregion Kinect Skeleton processing

#region GameTimer/Thread

private void GameThread()
{
    this.runningGameThread = true;

    this.predNextFrame = DateTime.Now;

    this.actualFrameTime = 1000.0 / this.targetFramerate;

    // Try to dispatch at as constant of a framerate as possible by sleeping just enough since
    // the last time we dispatched.

    while (this.runningGameThread)
    {
        // Calculate average framerate.

        DateTime now = DateTime.Now;

        if (this.lastFrameDrawn == DateTime.MinValue)
        {
            this.lastFrameDrawn = now;
        }

        double ms = now.Subtract(this.lastFrameDrawn).TotalMilliseconds;

        this.actualFrameTime = (this.actualFrameTime * 0.95) + (0.05 * ms);

        this.lastFrameDrawn = now;

        // Adjust target framerate down if we're not achieving that rate

        this.frameCount++;

        if ((this.frameCount % 100 == 0) && (1000.0 / this.actualFrameTime < this.targetFramerate * 0.92))
        {
            this.targetFramerate = Math.Max(MinFramerate, (this.targetFramerate + (1000.0 / this.actualFrameTime)) / 2);
        }

        if (now > this.predNextFrame)
        {
            this.predNextFrame = now;
        }

        else

```

```

    {
        double milliseconds = this.predNextFrame.Subtract(now).TotalMilliseconds;
        if (milliseconds >= TimerResolution)
        {
            Thread.Sleep((int)(milliseconds + 0.5));
        }
    }

    this.predNextFrame += TimeSpan.FromMilliseconds(1000.0 / this.targetFramerate);
    this.Dispatcher.Invoke(DispatcherPriority.Send, new Action<int>(this.HandleGameTimer), 0);
}

private void HandleGameTimer(int param)
{
    // Every so often, notify what our actual framerate is
    if ((this.frameCount % 100) == 0)
    {
        this.myFallingThings.SetFramerate(1000.0 / this.actualFrameTime);
    }

    // Advance animations, and do hit testing.
    for (int i = 0; i < NumIntraFrames; ++i)
    {
        foreach (var pair in this.players)
        {
            HitType hit = this.myFallingThings.LookForHits(pair.Value.Segments, pair.Value.GetId());
            if ((hit & HitType.Squeezed) != 0)
            {
                this.squeezeSound.Play()
            }
            else if ((hit & HitType.Popped) != 0)
            {
                this.popSound.Play();
            }
            else if ((hit & HitType.Hand) != 0)
            {
                this.hitSound.Play();
            }
        }
    }
}

```

```

    }
    this.myFallingThings.AdvanceFrame();
}

// Draw new Wpf scene by adding all objects to canvas
playfield.Children.Clear();

this.myFallingThings.DrawFrame(this.playfield.Children);

foreach (var player in this.players)
{
    player.Value.Draw(playfield.Children);
}

BannerText.Draw(playfield.Children);

FlyingText.Draw(playfield.Children);

this.CheckPlayers();
}

#endregion GameTimer/Thread

#region Kinect Speech processing

private void RecognizerSaidSomething(object sender, SpeechRecognizer.SaidSomethingEventArgs e)
{
    FlyingText.NewFlyingText(this.screenRect.Width / 30, new Point(this.screenRect.Width / 2, this.screenRect.Height / 2),
e.Matched);

    switch (e.Verb)
    {
        case SpeechRecognizer.Verbs.Pause:
            serialPort1.Write("off\r\n");
            Thread.Sleep(5000);
            this.myFallingThings.SetDropRate(0);
            this.myFallingThings.SetGravity(0);
            break;

        case SpeechRecognizer.Verbs.Resume:
            serialPort1.Write("on\r\n");
            Thread.Sleep(100);
            this.myFallingThings.SetDropRate(this.dropRate);
            this.myFallingThings.SetGravity(this.dropGravity);
            break;

        case SpeechRecognizer.Verbs.Reset:
            this.dropRate = DefaultDropRate;
    }
}

```

```

        this.dropSize = DefaultDropSize;

        this.dropGravity = DefaultDropGravity;

        this.myFallingThings.SetPolies(PolyType.All);

        this.myFallingThings.SetDropRate(this.dropRate);

        this.myFallingThings.SetGravity(this.dropGravity);

        this.myFallingThings.SetSize(this.dropSize);

        this.myFallingThings.SetShapesColor(System.Windows.Media.Color.FromRgb(0, 0, 0), true);

        this.myFallingThings.Reset();

        break;

    case SpeechRecognizer.Verbs.DoShapes:

        this.myFallingThings.SetPolies(e.Shape);

        break;

    case SpeechRecognizer.Verbs.RandomColors:

        this.myFallingThings.SetShapesColor(System.Windows.Media.Color.FromRgb(0, 0, 0), true);

        break;

    case SpeechRecognizer.Verbs.Colorize:

        this.myFallingThings.SetShapesColor(e.RgbColor, false);

        break;

    case SpeechRecognizer.Verbs.ShapesAndColors:

        this.myFallingThings.SetPolies(e.Shape);

        this.myFallingThings.SetShapesColor(e.RgbColor, false);

        break;

    case SpeechRecognizer.Verbs.More:

        this.dropRate *= 1.5;

        this.myFallingThings.SetDropRate(this.dropRate);

        break;

    case SpeechRecognizer.Verbs.Fewer:

        this.dropRate /= 1.5;

        this.myFallingThings.SetDropRate(this.dropRate);

        break;

    case SpeechRecognizer.Verbs.Bigger:

        this.dropSize *= 1.5;

        if (this.dropSize > MaxShapeSize)

        {

            this.dropSize = MaxShapeSize;

```

```
    }

    this.myFallingThings.SetSize(this.dropSize);

    break;

case SpeechRecognizer.Verbs.Biggest:

    this.dropSize = MaxShapeSize;

    this.myFallingThings.SetSize(this.dropSize);

    break;

case SpeechRecognizer.Verbs.Smaller:

    this.dropSize /= 1.5;

    if (this.dropSize < MinShapeSize)

    {

        this.dropSize = MinShapeSize;

    }

    this.myFallingThings.SetSize(this.dropSize);

    break;

case SpeechRecognizer.Verbs.Smallest:

    this.dropSize = MinShapeSize;

    this.myFallingThings.SetSize(this.dropSize);

    break;

case SpeechRecognizer.Verbs.Faster:

    this.dropGravity *= 1.25;

    if (this.dropGravity > 4.0)

    {

        this.dropGravity = 4.0;

    }

    this.myFallingThings.SetGravity(this.dropGravity);

    break;

case SpeechRecognizer.Verbs.Slower:

    this.dropGravity /= 1.25;

    if (this.dropGravity < 0.25)

    {

        this.dropGravity = 0.25;

    }

    this.myFallingThings.SetGravity(this.dropGravity);
```

```

        break;
    }
}

private void EnableAecChecked(object sender, RoutedEventArgs e)
{
    CheckBox enableAecCheckBox = (CheckBox)sender;
    this.UpdateEchoCancellation(enableAecCheckBox);
}

private void UpdateEchoCancellation(CheckBox aecCheckBox)
{
    this.mySpeechRecognizer.EchoCancellationMode = aecCheckBox.IsChecked != null && aecCheckBox.IsChecked.Value
        ? EchoCancellationMode.CancellationAndSuppression
        : EchoCancellationMode.None;
}

#endregion Kinect Speech processing
}
}

```


Appendix B

Publications

Kinect Controlled Electro-Mechanical Skeleton

Jason Eckelmann and Brian Butka
Mechanical and Electrical Engineering Departments
Embry-Riddle Aeronautical University
Daytona Beach, FL USA
butkab@erau.edu

Abstract—Mimicking real-time human motion with a low cost solution has been an extremely difficult task in the past but with the release of the Microsoft Kinect motion capture system this problem has been simplified. This paper discusses the feasibility and design behind a simple robotic skeleton which utilizes the Kinect to mimic human movements in real-time. The long-term goal of this project is to construct a $\frac{1}{2}$ scale model of a full robotically enhanced skeleton and demonstrate the abilities of the Kinect as a tool for human movement mimicry.

Keywords— Robotics; Kinect; Mechatronics; Motion Capture;

I. INTRODUCTION

Robots that mimic human movement have been depicted as the robots of the future in literature and film for a long time. The recent Hollywood movie *Real Steel* features a robot that mimics human movements through watching a person move and then performing the same movements simultaneously. Although the movie is currently science fiction, this research investigates the development of a low-cost system motion capture system for use as a display in a children's science museum.

This research focuses on developing a system that captures the motions of a human, uses this information to estimate the locations of key bones of the skeleton and then uses this information to mechanically mimic the skeletal motions on a physical skeleton. Until recently, the technology required to perform this task were well outside of the budget of most museums, but the introduction of the Microsoft Kinect and open source software support allow this project to be performed on a reasonable budget.

II. CURRENT STATE OF THE ART

Professional motion capture systems have been used to digitally capture human movements for use in animation since the 1995 Atari game *Highlander: The Last of the MacLeods*. These professional level systems require a person to wear a body suit with reflective markers all over it as seen in Figure 1 [4]. In addition to the custom body suits there is a vast array of sensors and software programs used to capture and compute these movements. Though the accuracies of systems such as Gypsy 7 are excellent, the hardware is expensive and the system is not designed to be used in real time applications.



Figure 1 A body suit used for professional grade motion capture systems. Note the reflective markers used to track body motions.

III. LOW COST MOTION CAPTURE

At a cost of \$200 the Microsoft Kinect has the ability to track the movements of 24 distinct skeletal points on the human body. These points include the head, hands, arms, and legs. Along with these 24 skeletal points the Kinect can track two people at the same time and has voice recognition capabilities [7]. This project only requires tracking of less than 15 skeletal points for a single user. Figure 2 shows the Kinect and a skeletal map.



Figure 2 A 15 point skeletal model (left) produced by a Microsoft Kinect sensor (right).

The Kinect sensor generates the skeletal map by reading data from an array of sensors including: a depth sensor, an accelerometer, a multi-array microphone, and two RGB cameras [1]. The microphone is currently not used for this application. The main driving sensors on the Kinect are the depth sensor and the cameras. The depth sensor is a Micron

1/2-Inch Megapixel CMOS Digital Image Sensor that consists of an infrared laser projector and a CMOS (Complementary metal-oxide-semiconductor) sensor. This CMOS is considered an active pixel sensor and is capable of capturing 3D video data in ambient light [7,8]. The main bulk of the data used to construct the skeletal points is taken from the two RGB cameras in the Kinect. These cameras are Aptina 1/4-Inch 1.3-Megapixel SOC CMOS Digital Image Sensors, which give the Kinect a viewing range of roughly 11ft. Along with the mentioned sensor the Kinect is equipped with a motorized pivot that allows the Kinect to physically move as it tracks targets.

IV. MECHATRONICS

This project will focus on utilizing the captured skeletal maps and mimic the motions on a physical skeleton in real time. Software will analyze the motions of the skeleton 10 times per second. This data will be analyzed to assign specific movements to servo sets for the skeletal points located in the arms and legs. The goal of real-time movements on the physical skeleton requires the use of actuators that are powerful, fast and accurate. For places on the body where there can be rotation such as in the shoulder a pan-tilt motion set up will be used to make the necessary multi-axis movements. Figure 3 shows the actuator locations, the red markings show places where multi-axis actuators are required. It should be noted that only motions of major bones of the skeleton are of interest for this effort. Motions such as rotations of the wrist and forearm are not incorporated in this work.



Figure 3 A physical skeleton showing the joints targeted in this research. Black indicates a single axis of motion. Red indicates multi-axis motions.

The final design requires the use of 12 actuators. To reduce the number of different parts used in the assembly the same

actuators will be used throughout the design. The actuator selection was based upon 4 different factors; servo speed and accuracy, holding torque, operating angle range and cost. The holding torque of the actuator was the most crucial factor because in some movements the actuator is required to hold the weight of entire appendage. The worst-case scenario for holding torque occurs in the leg since it is the longest and heaviest part of the skeleton. For this requirement a simple moment calculation was used to determine the holding torque of the actuator needed. The holding torque is given by!

$$\tau = r \times F \quad (1)$$

where τ is the torque, r is the length of the lever arm and F is the applied force. The worst case occurs when the leg is hold straight in front of the body in a kicking motion. For the leg assembly a mass of 0.3Kg is supported against the pull of gravity yielding a force of 2.94N. For a worst case estimate, the entire mass is assumed to exist at the end of the leg yielding a lever of 0.5m. The worst case holding torque is calculated to be roughly 1.5Nm.

A. Selecting the Actuator

The Dynamixel AX-12A robot actuator was selected for use in this project. The AX-12A has several major advantages over standard hobbyist servos that will be taken advantage of in the construction of the skeleton. These actuators offer a maximum holding torque of 1.6Nm at 12 Volts [2]. When supplying this holding torque the actuators draw only 900 mA which allows the use of low cost off the shelf power supplies. Given the overestimates of the required holding torque it is believed that these actuators are able to hold the entire leg without worry of failure. The AX-12A also offers 300°/continuous operating angles and non-loaded speeds of 0.196sec/60°. These features will allow for near-real-time movements of all the appendages. Along with all the performance features of the AX-12A there are several feature built-in features such as the internal micro-controller that will be used in this project. The built-in microcontroller provides feedback of the current angular position and angular velocity as well as the torque being applied to the load. These availability of these feedback signals in a compact footprint drive the use of these actuators. A bearing is used at the final axis to ensure no efficiency degradation with high external loads. The actuator also has a built in alarm system that can feedback to the higher-level controller when there are issues in current draw, voltage, internal temperature, and torque output. The case that encloses the mechanics of the actuator has integrated mounting points, which will also be utilized in the assembly of the project; Figure 4 shows the AX-12A and a mounting bracket.



Figure 4 The Dynamixel AX-12+ is the selected actuator for all joints.

Since the skeleton requires a total of 12 servos in order to perform all necessary movements a microcontroller that can handle a minimum of 12 servos will be required. The AX-12A requires TTL level serial communications to send and receive signals. This project utilizes a total of 12 AX-12A's to be controlled in realtime. Although a controller is available from Robotis (the manufacturer of the AX-12A actuators), it is unclear if the controller will be able to perform all of the necessary analysis of skeletal motion in realtime. It is anticipated that as the project nears completion, a more powerful controller such as Vanadium Labs ArbotiX Robocontroller will be required. However, in the interests of speeding development, the Robotis controller and software will be used as the initial development platform. The advantage of the ArbotiX controller over many other popular micro-controllers such as the Arduino family is that this ArbotiX controller is designed with the Dynamixel AX-12 servos in mind. This microcontroller boasts the ability to control more than 24 AX-12 servos simultaneously using its integrated Atmega644p processor [3]. The ArbotiX also has the ability to incorporate an XBee system for wireless communications. If needed there are motor drivers, encoder headers, and 32 analog headers equipped to this board allowing the use of PWM (pulse width modulation) servos if needed; Figure 5 shows the ArbotiX microcontroller.



Figure 5 The Arbotix microcontroller selected as the controller for this research.

V. CONSTRUCTION

To reduce development time many COTS (commercial off the shelf) products were used in the construction. The skeletal structure referred to as the chassis is a 1m tall plastic model that was purchased from a anatomical model website. Several different factors had to be taken into account before deciding on the skeleton to be used. Sizing the chassis needed a great deal of consideration due to the size of each appendage; as the chassis becomes larger the leg and arm appendages grow proportionally. Since another deciding factor was that the arms and legs needed to be structural. This in turn will increase the holding weight required by the servo exponential since the servos will also become larger and heavier as will the moments acting on them. Given all these factors a roughly half scale skeleton was selected for the chassis. A 1m tall skeleton was selected for the chassis; which has 25cm and 46cm appendages. The skeleton is constructed from a hard molded resin and has moveable joints in all the areas that will be modified. This chassis is a cheap economical solution that will allow for rapid construction and easy modifications.



Figure 6 Physical skeleton plastic model on its stand.

The skeleton has wire joints built into several key joints so structural modifications to the joints must be made. Large machined rods will replace all the wire joints in the shoulders, elbows, knees, and hips. This requires some machining of the stock plastic skeletal frame; metal rods are used for actuator mounts. Figure 7 shows a typical joint. The shoulder bracket will have the threaded rod run through the bracket's center holes.



Figure 7 Mechanical design of a typical joint showing mounting points.

For multi-axis actuation, a second actuator will be affixed with a 90 degree offset to the above actuator. From the second servo another bracket similar in make to the shoulder bracket will connect the arm or upper leg to the servo system. Figure 8 shows a pan/tilt servo set utilizing off the shelf brackets.

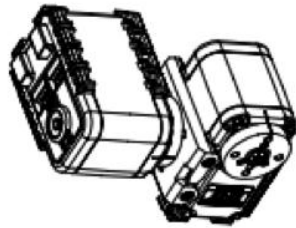


Figure 8 A possible pan-tilt configuration of actuators.

VI. SOFTWARE

The software used in the system consists of three major elements: motion capture and extraction of skeletal joint positions, real-time analysis and path planning for 12 joints, and actuator control software. Much of the motion capture and joint position extraction is performed in the Kinect hardware. Once the hardware is correctly configured, the Kinect hardware will provide a continuous stream of joint positions that is

updated multiple times per second. The path-extraction software then needs to determine the position of joint. Once the joint positions are determined the kinematic model must be solved to determine the desired velocity and final position of each of the 12 actuators and transfer this information to the control software. The main function of the control software is to synchronize the motion of the actuators and assure that the actuators are operated within system limits. Humans can perform several motions that would be undesirable in the physical skeleton. Examples of prohibited motions would be striking bones together such as striking the head with a hand or simply clapping 2 hands together. Other prohibited motions would be rapid oscillations of major bones such as rapidly shaking the forearm.

VII. CONCLUSIONS

This paper has described the design of a system that will allow a human skeleton to mimic the motions of a human operator. The Kinect has the potential to revolutionize tele-operated robots by dropping the price from hundreds of thousands of dollars to hundreds of dollars. There are numerous applications for robots that mimic the motions of human operators such as using robots to lift loads beyond the capabilities of humans to a doctor performing surgery from a remote location. What is currently fodder for science fiction movies such as Real Steel will soon be a reality.

ACKNOWLEDGMENTS

Funding for the robot hardware is being provided by NuovoVision. This work described in this paper is part of Jason Eckelmann's Master's thesis.

REFERENCES

- [1] "Kinect Hardware." *Open Kinect*. N.p., n.d. Web. Oct 31st 2011. <http://openkinect.org/wiki/Hardware_info>.
- [2] AX-12a Datasheet. Nov 23, 2011. <http://www.trossenrobotics.com/>
- [3] Arbotix Datasheet Dec 29, 2011 <http://www.vanadiumlabs.com/arbotix>
- [4] Jeong, Woong, Yong-Ho Seo, and Hyun Yang. "Effective Humanoid Motion Generation based on Programming-by-Demonstration Method for Entertainment Robotics." *Virtual Systems and Multimedia (VSMM), 2010 16th International Conference on*. (2010): 286-292. Print.
- [5] K. Konolige and P. Mihelich. Technical description of Kinect-calibration. http://www.ros.org/wiki/kinect_calibration/technical, 2011.
- [6] Smisek, Jan; Jancosek, Michal; Pajdla, Tomas; , "3D with Kinect," *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pp.1154-1160, 6-13 Nov. 2011
- [7] Lu Xia; Chia-Chih Chen; Aggarwal, J.K.; , "Human detection using depth information by Kinect," *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pp.15-22, 20-25 June 2011
- [8] Rakprayoon, Panjawee; Ruchanurucks, Miti; Coundoul, Ada; , "Kinect-based obstacle detection for manipulator," *System Integration (SII), 2011 IEEE/SICE International Symposium on*, pp.68-73, 20-22 Dec. 2011

ERAU Formula Hybrid Accumulator Design

Jason Ekelmann and Brian Butka
Mechanical and Electrical Engineering Departments
Embry-Riddle Aeronautical University
Daytona Beach, FL USA
butkab@erau.edu

Abstract—The SAE Formula Hybrid competition is event were students from many different schools put their engineering knowledge to use to design complex hybrid racing systems; Embry-Riddle Aeronautical University has been competing in this competition since its inception. This paper discusses the design on the team's energy storage and accumulator design for the 2012 competition. This design is required to follow all 2012 rules set forth by SAE International and this paper discusses the concerns of the design regarding these rules.

Keywords- Formula Hybrid; Battery; Accumulator; Energy Storage Systems; Embry-Riddle

I. INTRODUCTION

The SAE (Society of Automotive Engineers) has been sponsoring a Formula Hybrid vehicle competition since 2007. In this highly competitive event over 30 different teams competing to build the best Hybrid Formula 1 car. The cars in which each team must build, design, and compete in must be open-wheeled single seat racers. The competition is designed in a way that promotes innovation in fuel efficiency and drivetrain design in high-performance applications such as racers. Before teams can compete with their vehicles they must pass a strict safety inspection where judges make sure all areas of the vehicle are safe [3]. In this portion of the competition there is a major emphasis on the safety of the energy storage system; this impart due to the hazards of dealing with high-voltage storage systems. This paper will go into the design of the ERAU (Embry-Riddle Aeronautical University) Formula Hybrid Team's energy storage and monitoring system design which meets the SAE Formula Hybrid 2012 Rulebook.

II. COMPETITION RULES

The Formula Hybrid competition like many other competitions have a strict set of rules that teams must adhere to in order to compete fairly and safely. For the rulebook given in this competition there is a major focus on the HV (High Voltage) system. As per the rules a system is considered high voltage if it contains or produces a voltage greater than 30 volts [1.5]. Such systems are required to be isolated and physically segregated from the other power systems of the vehicle. The HV storage system must be a self-contained separate part of the vehicle structure and architecture. Along with the system being isolated it must contain various safety

features as described in the rules. The first major safety system required is a GFD (Ground Fault Detector), which is utilized to detect any faults below 500 ohms/volt or 40k Ω . If such a fault is detected the immediate shutdown of all electrical systems is required. The Bender 475LY shown in Figure 1 is such an example.

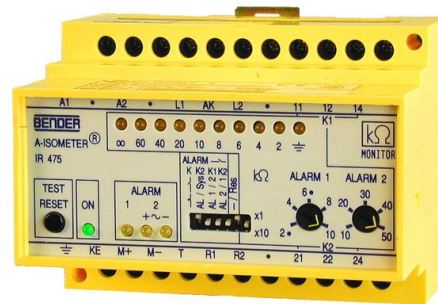


Figure 1 The Bender 475LY ground fault detection system.

The second major set of rules involving safety pertains to the accumulator design. The accumulator must contain a monitoring system that varies depending on the energy accumulator type. The AMS (Accumulator Monitoring System) that will be used on the ERAU vehicle will be utilizing the rules for Lilon (Lithium Ion) batteries. The AMS is required to monitor the accumulator at time that energy is flowing into or out of the storage system. This system is to be used to prevent hazardous thermal conditions such as overheating and overcharging [6]. This is to prevent dangerous situations such as batteries catching fire or melting during charging and high load situations such as the acceleration run. For the Lilon accumulator type teams are required to build an AMS that can monitor the temperature of each battery module and voltage monitoring of each individual battery cell. This safety system must be able to disable the storage system by opening the contactors inside the accumulator. This can be caused by any of the specified hazardous conditions such as over-voltage, under-voltage, overheating, or cell reversal.

Though not required by the rules a balancing system is recommended for the Lilon accumulator setup.

The mechanical design of the accumulator is specified in the electrical rules because it is necessary to have the storage system in a container that is isolated from the rest of the vehicle. The energy storage system must be in a closed container and utilize contactors for any connections leaving the enclosure. The mechanical properties required of the storage enclosure are clearly stated in the rules. The enclosure and mounts must withstand a 20g static load in front/back and sides and an 8g static load in the vertical direction. The enclosure must also be considered mechanically robust, fireproof, and must fully enclose the accumulator. Along with all these internal features the storage container must also have a fireproof barrier equipped between it and the cockpit.

III. ERAU DESIGN OVERVIEW

The ERAU design utilizes the A123 M1A cylindrical cells. These cells will be housed in aluminum battery tubes. Between each cell contact plates will be used to allow connections to the AMS. For this accumulator design a total of nine tubes will be used. The goal of this design was to create a lightweight mounting structure that will meet all the rules and requirements as set by the competition officials; this will be accomplished by using an Aluminum Isogrid to construct the mounting structure. A series of L brackets will be used to construct the external box structure with fiberglass sides. The outward facing side will utilize a Lexan sheet to allow for visibility into the enclosure. Internally the system components will be mounted to the Isogrid using insulated stand offs. This entire enclosure will be mounted to the vehicle chassis via aluminum mounts which will be bonded to the enclosure; Figure 2 shows a basic model of the ERAU design.

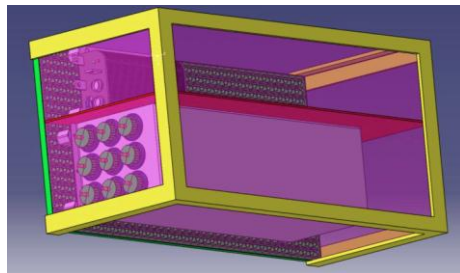


Figure 2 Physical layout of the accumulator design. Visible are the cylindrical battery packs.

IV. BATTERY SELECTION

The A123 M1A cylindrical cells were selected for this design for several reasons, one of which is these cells are commonly used by many teams at ERAU and there is a wealth of

experience using these batteries. Experience aside the A123 cells have several unique features that the team has found to be beneficial. These particular cells have a high power density over a broad SOC (State of Charge) [2]. They are capable of handling a high amount of physical abuse and have an extremely stable chemical composition. A123 technology is widely used in high performance vehicles around the world. The selected cells have a nominal voltage of 3.3 volts and a specific power of 2700 W/kg. The team's design will use cells that have been extracted from DeWalt drill packs; in order to prevent any conductive paths from the cell casing the stock manufacturers paper coving will be retained. Each cell will then be wrapped in 6mm thick PVC (Polyvinyl Chloride) shrink-wrap to reduce the radius of the positive terminal, which will ensure isolation. The system design requires a total of 72 individual cells; Figure 3 shows several unwrapped cells.



Figure 3 The negative and positive terminals of a typical A123 battery cell.

V. BATTERY CONTAINMENT TUBE DESIGN

A key feature of the accumulator pack design is the battery tubes, which house the individual cells. Aluminum was chosen for the battery tubes due to its availability, mechanical properties, and thermal conductivity. The other option was a plastic housing which would have needed to be a custom made tube in order to be used in this application; in turn this would have been a costly alternative. In selecting aluminum as the tube material there raises a concern that this may be in violation of the rules due to the fact a conductive pathway may develop between the cells and the tubing; further correspondence with the competition organizers will clear or verify these concerns. The tubes will have slots machined in them to allow for the contactor plates to be inserted easily between cells and for the AMS wires to leave the tubing. Each tube will consist of eight A123 cells and seven contactor plates. At the end of each tube will be a plastic end cap with terminals that will be used to connect each tube to the overall

system; Figure 4 shows the design of nine tubes in the holding chamber.

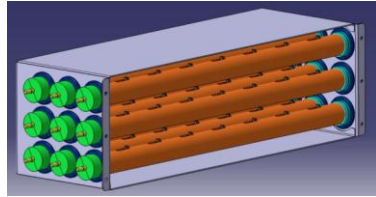


Figure 4 The battery containment system.

VI. CONTACTOR PLATE, END CAP, AND TUBE RETAINMENT DESIGN

In each battery tube contains 8 battery cells between each of these cells is a contactor plate. This plate is used to not only allow the batteries to make contact with each other but to allow the gathering of all the necessary data for the AMS. Each contactor plate is constructed of a conductive material in a plastic housing; a hole is drill in the tab of the contactor so that a wire can be connected to the AMS. During the tube assembly the plates can be easily inserted into the battery tubes then rotated to make contact with the next cell. This design brings yet another concern with a violation in the rules; with the given design the contactor plate if partial exposed which may provide a safety hazard; Figure 5 shows the contactors in the battery tube highlighting the concern.

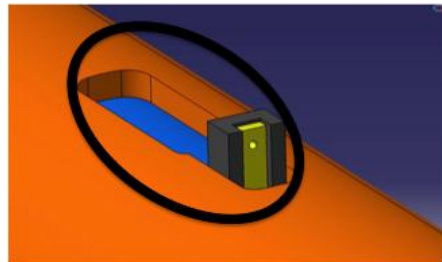


Figure 5 Close up of the battery monitoring contactor.

As with the contactors each tube will have two custom-built end caps at each end. The ends of each tube will be threaded so that the end caps can be affixed to them. The end caps were designed to handle the thermal expansion of the batteries by incorporating springs at negative end of the tube. The springs used are Belleville disc springs and will give the system a preload of roughly 6 lbs.; this will ensure each battery makes contact with its following contactor. Both end caps will also contain the terminals used to connect the tubes together; Figure 6 shows the positive and negative end cap designs.

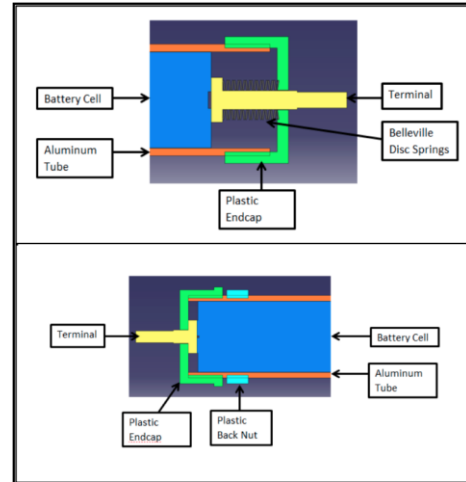


Figure 6 The positive (top) and negative (bottom) endcap designs for the battery containment tubes.

Since there are nine battery tubes a simple custom retainment housing was design to hold each battery tube safely. The retainment housing is simple five-sided sheet metal enclosure with nine circular holes cut in the front and backsides to hold the battery tubes. The prevent chaffing on the tubes rubber grommets will be inserted into each hole prior to the battery tubes being placed. Once the tubes are placed in their individual slots the plastic end caps will be tightened down to the tube securing them to the retainment enclosure; Figure 7 shows the retainment enclosure and Figure 8 the final position of the battery tubes.

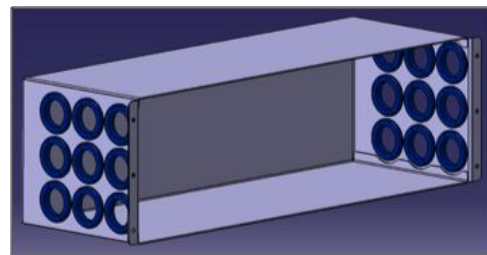


Figure 7 External battery containment system.

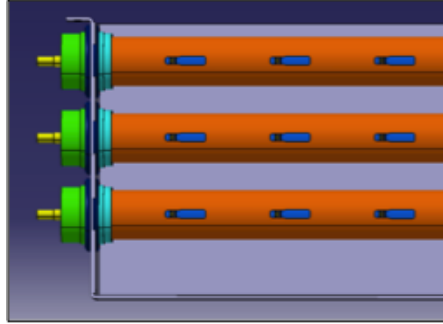


Figure 8 The battery containment system showing the end-cap positioning.

VII. HIGH VOLTAGE ENCLOSURE DESIGN

Special consideration has been taken when designing the external enclosure because a vast amount of weight can be saved with a well thought out design. The main part of the enclosure will be the aluminum isogrid, which will be used to structurally mount all the internal components. The nodes of the enclosure will have helicoiling for any steel bolts or will be tapped for aluminum bolts used in the attachment of components. Since the aluminum isogrid is naturally conductive it will be covered in an insulating fiberglass. From this all the components will be mounted using insulated standoffs. These standoffs will be used to prevent conductive connections from forming between components and the grid via bolt connections. Figure 9 shows the planned layout of components on the isogrid. The components are blocked out and the high voltage routing has been highlighted. All wiring will be routed using standoffs the height of the standoff will be complaint with the requirements stated in the rulebook. This will insure if the grid becomes electrified, all components and wiring will still be compliant with the rules.

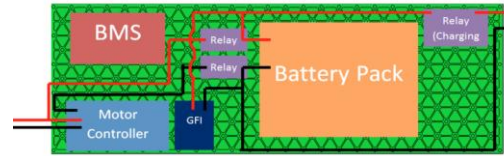


Figure 9 High-voltage enclosure battery pack, battery management system and required isolation relays.

Since the enclosure has two main sections; those being the battery pack and the rest of the electrical components it will be divided by a fiberglass panel. This will provide a level of separation when servicing the system. Mainly the design called for the motor controller and battery pack to be able to be serviced and inspected independently. Next is the defining structure, which will for the actual box enclosure. This is constructed from a series of L-brackets constructed in the shape of a box. These brackets will mount to the isogrid backing to finish the structure. As with the isogrid material the L-brackets will need to be covered in a fiberglass sheet to act as an insulator this will also close of the open sides of the structure. The final outward facing side will be constructed of 2 pieces of Lexan, which will be hinged to allow work on either the motor controller or the battery pack. The entire enclosure will then have four aluminum mounts fixed to the outside of the structure. These mounts retain spherical bearings, which will allow for the mounting to the chassis; Figure 10 shows the enclosure on the chassis.

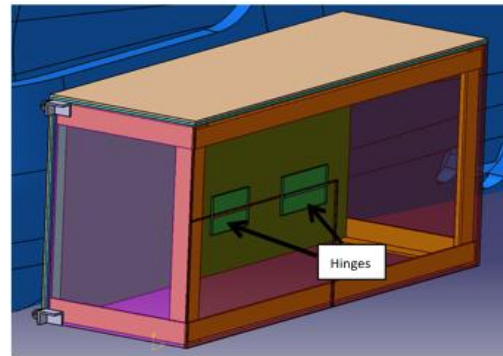


Figure 10 Enclosure chassis showing the hinge positions.

VIII. CONCLUSIONS

This paper describes the design of the energy accumulator system for the ERAU formula hybrid vehicle. With deadlines fast approaching a valid energy storage system design is required and compromises must be made in the interests of time. Although the prototypes of the designed have functioned well, there are areas of ERAU, which will require further analysis to assure compliance with all competition safety rules.

ACKNOWLEDGMENTS

The authors would like to thank the ERAU Formula Hybrid team and Dr. White for all their hard work on this project. Special thanks to Matthew Nowacki for all the documentation and design elements used in this paper.

REFERENCES

- [1] SAE International, *2012 Formula Hybrid Rules*. 1st. SAE International, 2011.
- [2] A123-M1a Datasheet. Feb 1, 2012. <http://www.a123systems.com/>
- [3] Sibley, J.; Wirasingha, S.G.; Emadi, A.; , "Formula Hybrid racing at Illinois Institute of Technology: Academic Year 2007/2008," *Vehicle Power and Propulsion Conference*, 2008. VPPC '08. IEEE , pp.1-6, 3-5 Sept. 2008
- [4] Benson, K.W.; Fraser, D.A.; Hatridge, S.L.; Monaco, C.A.; Ring, R.J.; Sullivan, C.R.; Taber, P.C.; , "The hybridization of a Formula race car," *Vehicle Power and Propulsion, 2005 IEEE Conference* , pp. 295- 299, 7-9 Sept. 2005
- [5] Maggetto, G.; Van Mierlo, J.; , "Electric and electric hybrid vehicle technology: a survey," *Electric, Hybrid and Fuel Cell Vehicles (Ref. No. 2000/050), IEE Seminar* .pp.1/1-111, 2000
- [6] Ikwhang Chang; Namwook Kim; Daeheung Lee; Suk Won Cha; , "Designing and manufacturing of Formula SAE-Hybrid racecar for a new engineering education program," *Vehicle Power and Propulsion Conference (VPPC), 2010 IEEE* , pp.1-6, 1-3 Sept. 2010

Seagle 3.0 An Autonomous Surface Vehicle

Jason Eckelmann and Brian Butka
Mechanical and Electrical Engineering Departments
Embry-Riddle Aeronautical University
Daytona Beach, FL USA
butkab@erau.edu

Abstract— The AUVSI Autonomous Surface vehicle event is a student-based competition where teams design, build and compete with fully autonomous surface vessel. These vehicles are required to perform many different tasks that vary from competition to competition but the ability to navigate channels marked by red and green markers and perform GPS based navigation is always a constant. Embry-Riddle Aeronautical University has been competing in this competition since it starts. This paper discusses the design of the Seagle 3.0 platform.

Keywords— Robotics; Autonomy; Autonomous Surface Vehicle; Computer Vision; Mechatronics;

I. INTRODUCTION

An Autonomous Surface Vehicle is a floating, untethered robot capable of performing complex tasks without human interaction [7]. Seagle 3.0 represents a major advance in technology compared to the original Embry Riddle platforms that competed in past competitions. Seagle 3.0 is new above and below the waterline, including a new central processing computer, upgraded sensors, a water cooling system, and greatly enhanced and refined software. The vessel itself is a planning hull design intended to maximize the speed-to-thrust ratio. It uses a relatively flat large-wetted-surface area foam core covered with S-Glass laminate. All four sensors (DGPS, digital compass, and two cameras) are located on masts above the deck.

Components inside the Electronics Enclosure include the onboard computer, a wireless router for communication during testing and debugging, a Devantech two-axis motor controller, a Parallax servo controller, an RxMux servo multiplexer, an Onboard Health Monitoring System (or OHMS for short), and batteries.

II. DESIGN OVERVIEW

Seagle 3.0 was developed to meet the requirements specified in the 2010 Autonomous Surface Vehicle Competition rules [2] and has since been used as a developmental platform for current teams. During the design stages emphasis was put on safety, performance, simplicity of design, operational effectiveness, and reliability. Figure 1 shows Seagle 3.0 in the pool. Although Seagle 3.0 is intended to perform its mission autonomously, it must also be launched, prepared and recovered by a shore-based team. Seagle 3.0 is a small electrically propelled flat-bottom boat known as a skiff that is differentially driven by two SeaBotix BTD150 thrusters.

Seagle 3.0 is 1.5m long, .5m wide and .75m tall. The entire vessel, including batteries, internal hardware, and competition necessary hardware weighs 19 kg. Where appropriate, Seagle 3.0 incorporates commercial-off-the-shelf (COTS) components to help ensure reliability. A Sea Horse watertight case provides a dry environment for the onboard electronics, including a custom built computer, a Devantech motor controller, a Parallax USB Servo Controller, an RxMux multiplexer (for switching from remote to autonomous operation), an onboard health monitoring system, four sealed custom Nanophosphate lithium ion battery packs, a Linksys 2.4 GHz wireless router with a high gain antenna for faster remote desktop streaming while testing, a computer controlled switch for the water cannon and a water-cooling system. Seagle 3.0 includes two Axis 207MW cameras for buoy and target perception as well as obstacle avoidance, a DGPS, and a digital compass for navigation to specified points on the course; Figure 1 shows Seagle 3.0 during testing.

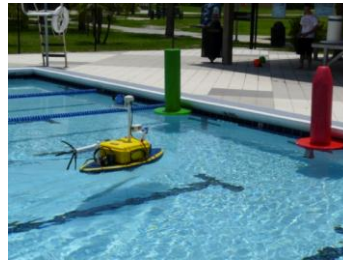


Figure 1 Seagle 3.0 during testing

III. PLATFORM DESIGN

The skiff design used on Seagle 3.0 was fabricated from Extruded Polystyrene. This closed-cell foam material machines well, is resistant to salt water and most common chemicals, has a low coefficient of water absorption and is exceedingly buoyant, having a density about 1/30 that of water. The hull was milled out of the EPS foam. The final shape was obtained through sanding and rail rounding. The hull was then painted using an acrylic-based paint to seal the foam and laminated using 6oz. S-Glass cloth for structure. Motor keel mounts and mast plugs were inserted using an epoxy micro-balloon mix.

After the laminating was completed, the vessel was hot coated using a pure epoxy mix; it was then finish sanded and sealed again with a clear coat. Lexan keels were machined and mounted to provide a secure structure for the thrusters.

A. Electronics Enclosure

The Electronics Enclosure is a modified Sea Horse watertight case mounted directly to the deck. The enclosure is shown in Figure 2. The stock case is watertight, and the use of water resistant connectors along with careful attention to sealing around penetrations provides a reliable water resistant enclosure for the electronics. The case has been modified to have two mounting layers, with the wiring and control boards below and the computer, power supply, and batteries above. The case is also equipped with a water-cooling system.

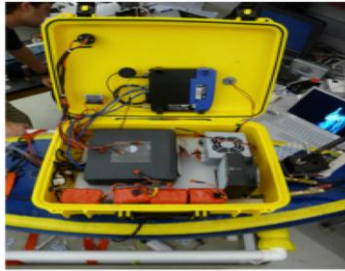


Figure 3 Top-level of the electronics enclosure; second level hidden

IV. ELECTRICAL DESIGN

Seagle 3.0 features various cost effective and high performance electrical components such as custom battery packs, GPS, digital compass, and network cameras that are all controlled by a custom built computer. Many of the components are advanced sensors used in the navigation algorithms.

A. Cameras

The cameras on Seagle 3.0 are critical components for navigation, obstacle avoidance, targeting, and payload retrieval. Seagle 3.0 is equipped with two Axis 207MW network cameras. The Axis 207MW has a horizontal field of view of 74 degrees, a maximum resolution of 1280 x 1024 at a frame rate of 12 frames per second. One camera is mounted facing forward on the deck and the second camera is mounted on a servo on the center mast. The second camera is controlled to face forward or backward depending on challenge requirements; Figure 3 shows an Axis 207MW.



Figure 2 Axis 207MW network camera

B. Propulsion

Propulsion is a key component to navigation so a differential drive system was developed utilizing two SeaBotix thrusters. The SeaBotix thrusters deliver a continuous Bollard thrust of 2.2 kgf at only 4.25 amps. A peak thrust of 2.9 kgf can be attained for short periods by increasing the current. At 4.25 amps and 19 VDC, the BTD150 thrusters use only 81 watts of power.

C. Batteries

Seagle 3.0 is powered by four lithium ion Nanophosphate battery packs, which were custom assembled by using eight A123 M1A cylindrical cells. These cells were selected for their size and energy output of their SOC (State of Charge) [1]. The battery packs consist of six 3.3V cells wired in series to achieve 19.8V. The Packs are then wired in parallel to reach a run time of 1.5 hours.

D. Computer

To increase data processing speeds, the team built a custom small form factor computer using commercially available components. This new computer contains a 2.5 GHz Intel quad core processor; 4 gigabytes of DDR2 Ram, and a 320 GB hard drive. A 250 Watt DC to DC power supply that has a low-voltage cut off feature powers it. The low-voltage cutoff safely shuts down the operating system when the input voltage drops below 13 volts. This onboard computer runs the Windows XP operating system and National Instrument LabVIEW programming environment, which is used for all mission-task programming.

E. Wireless Communications

A Linksys 2.4 GHz wireless router is connected to the computer to provide remote access to the software and vehicle systems. Testing and changes to the code can be accomplished conducted without having a physical connection to the vehicle. This allows for the monitoring Seagle 3.0 from the shore through a ground station. To increase range of the communications to the ground station a high gain antenna was equipped to the electronics enclosure.

F. GPS and Compass

The Novatel Smart Antenna with OmniSTAR corrections is a compact, lightweight and weatherproof package that gives a 0.6-meter Circular Error Probable (CEP) accuracy. The Pacific Navigation Instruments TCM2.5 tilt compensated 3-axis digital compass has an accuracy of 0.8 degrees. The GPS and Compass are used together for waypoint navigation. Since GPS is incapable of generating heading information when the vehicle is stationary [8], the compass is used to determine heading at low speeds. These sensors are used for heading hold navigation and waypoint navigation.

G. Servo Controller

A USB 16-Channel Parallax Servo Controller accepts USB output from the control computer and converts this to the pulse-width-modulated signals needed to command the Devantech motor controller and the Team Delta RC relay switch, which has been used to activate competition critical components.

H. Motor Controller

The MD22 Devantech Motor Controller is a robust two-axis motor driver. The driver is designed to supply power to two independent motors, allowing the vessel to be differentially driven. By allowing the vehicle to be differentially driven zero radius turns can be preformed during the obstacle avoidance.

I. Onboard Health Monitoring System (OHMS)

The onboard health monitoring system includes an Arduino Pro-Mini microprocessor, an AttoPilot current and voltage sensor and an analog thermometer. With this system the team is able to monitor the battery packs and electronics case temperature in real time. The Arduino is used to interpret temperature, voltage and current sensor data. It sends this information to the main vehicle computer as a serial string. LabVIEW is used to display this information on a graphical user interface. Also, a warning message is displayed when the system voltage drops to an unsafe level and LabVIEW initiates a shutdown sequence.

V. SOFTWARE DESIGN

The intelligent navigation software that operates Seagle 3.0 is preloaded on the onboard computer prior to deployment. During development, setup and testing, an operator can interface with the onboard computer using Remote Desktop running over a standard 802.11 network. At the start of a competition run, the software is set up and running before switching into autonomous mode; for real world seniors a permanent link with the system can be established utilizing the testing setup. The software provides feedback (viewed on the remote desktop) to verify that the cameras operational and the software is attempting to correctly control the thrusters. Once all systems, including the onboard health monitoring system, are checked, the autonomous/manual switch on the RC transmitter is set to autonomous mode. The vessel then executes its mission autonomously. If at any time the ground station operators deem the system is in danger of harming itself or the environment around it a switch on the RC controller can be flipped and remote control of the vessel is reestablished.

Should this system not work there is both a local and remote kill switch. The local kill switch is mounted on the electronics enclosure and cut power to the motor. While the remote kill switch is a hand held box that is armed before vehicle deployment and can be activated with the press of emergency stop at anytime. This remotely kills power to the motor controller.

A. Mission Strategy and Software

For each given competition a new set of mission objectives is given usually building on prior competitions. These objectives often require navigation through buoys, avoiding obstacles, find and shooting targets, performing GPS navigations, docking, and returning to the starting locations. A unique and innovative software system was developed to allow new challenges to be integrated with the old system. By utilizing a state driven software system the vehicle can move from software state to software state accomplishing a set of prewritten goals. Several of these states, which have been used through the course of this vehicle life, will be discuss.

It is common in the competition for the vehicle to have to preform an autonomous speed run through a series of large colored gates. These gates tend to be marked by a red and green buoy. For this state the software will use both vision and Gps. Drive points generated by the vision code are used to control heading and speed, and GPS data is used to determine the distance traveled. Immediately after traveling the distance of the speed gate the software will switch into another state; lets say the next state is the buoy channel navigation. In order for the vehicle to navigate the buoy channel it must first find the start of the channel. This is done by performing a series of preset movement that have been developed to allow the vision system to find the buoys. Once a buoy is found the software switches from its buoy hunting state to the buoy navigation state. In this state the software utilizes both cameras to find red and green buoys and plot Gps based drive points for the vehicle to navigate. In this state the vehicle searches for a set number of buoy or times out in which the software will switch to another state. After all the required states are performed the software switches to the final state which required the vehicle to return to its starting position. This is done by navigating to a preset Gps waypoint and using the cameras for obstacle avoidance.

B. Navigation Algorithm

The vision-based navigation code generally uses a simple algorithm to determine motor thrust commands. When the vessel senses a single red buoy and a single green buoy, it will calculate a point equidistant between them and drive towards it. If the vessel only sees a red buoy, a drive point offset a user-specified distance to the left of the buoy will be selected and the vehicle will drive towards that point. If the vessel only sees a green buoy, a drive point offset a user-specified distance to the right of the buoy will be selected and the vehicle will drive towards that drive point. If the vessel does not see any buoys the drive point will be set to (0,0), the current location of the vessel, and the vehicle will turn in place in an attempt to acquire the buoys. Figure 4 shows the buoy navigation algorithm.

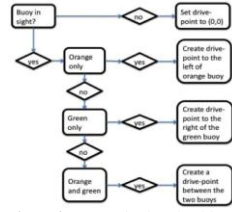


Figure 4 Buoy navigation algorithm

A proportional-derivative control law is used to determine how the vessel drives to a specified drive point. Using a local, vehicle- based coordinate frame, the distance and angle to the point of interest are calculated. The following equations are then used to determine the thrust command values for each thruster.

$$V_{L-M} = V_{L-\theta} + \frac{180 - |\theta|}{180} * V_{L-F} \quad (1)$$

$$V_{R-M} = V_{R-\theta} + \frac{180 - \theta}{180} * V_{R-V} \quad (2)$$

Where V_{L-F} and V_{R-F} are the forward components of thrust with the proportional term that factors in the distance to the drive point. Those values are then applied to the two overall equations that we developed that give us the overall thrust for each thruster.

The total thrust commanded from each thruster includes a forward drive component and a turning drive component. The forward thrust commanded from each thruster is proportional to the distance from the vehicle to the drive point. The farther away the drive point, the faster the vehicle will drive. As the vehicle approaches the drive point, it will begin to slow down. The turning component commanded from each thruster is based on the heading angle to the drive point. The greater the heading angle to the drive point, the greater the difference in thrust. A derivative control term has been added to the thrust command algorithm to reduce overshoot [6]. This variable is a damping system applied to the forward thrust to prevent over-corrections. A user-specified dead band on the turning component of proportional control prevents the vessel from hunting back and forth when the turn angle is near zero. A throttle control function has also been added, which allows the user the set the total percent of throttle that the vehicle applies to the thrusters.

C. Vision Algorithm

Seagle 3.0 is using several vision algorithms for many of the different challenges it has to perform such as obstacle avoidance and vision based navigation or targeting. The basic computer vision algorithm for each of these tasks is similar. A common user- defined ROI (region of interest) will be set on the front panel by the operator [3]. This ROI allows the user to remove superfluous portions of the image such as the sky and visible parts of the boat. This step allows for higher image processing speeds. The speed gate challenge and the buoy channel navigation challenge use nearly identical vision algorithms based on a hue, saturation, and luminance representation of the color image [5]. A band-pass filter is applied separately to hue, saturation, and luminance. By setting a narrow band, only the pixels that contain values in these three bands will remain. This has proven to be an effective means for eliminating everything but the buoys, due to their small standard deviation. After filtering is complete, several standard LabVIEW morphological computer vision functions are applied, including those to remove small particles, erode, and create a convex hull. These are used to remove noise and combine the reflection of the buoy with the actual buoy. Finally, a circularity filter is applied is used to find buoys in the image. This separates any overlapping circles and classifies them based on their radius, area and perimeter. The results of applying this image processing technique to an image containing a red buoy are shown in Figure 5.



Figure 5 Vision algorithm results for a red buoy

The targeting challenge such as finding circular targets and shooting them with water use the same basic image processing techniques as the buoy navigation challenge. The primary differences are the band-pass values and the values used of the morphological filters that are applied. The algorithm processes both the red rings and the gray square. During the post processing of the Find Circle command, a targeting line must be drawn to control the servo-mounted cannon. The servo will take the X and Y positions of the center of each circle found and move from each center in a straight path starting at the bottom circle. This allows for the system to hit all targets found even if a false positive is found. The results of applying this image processing technique to an image containing an gray target with red rings are shown in Figure 6.



Figure 6 Vision algorithm results for a gray target

In the past the vehicle was required to pick objects that were mounted to white buoys up. This requires the use of a rear-facing camera for this vehicle. For this example there is a gold ring mounted to a white buoy. Like the other challenges, a band-pass filter is applied to the HSL representation of the images received. The algorithm will initially search for the large white buoy. Once the white buoy is detected, the algorithm has the vessel move closer to the buoy in the forward direction after the distance between the white buoy has been reduced predetermined number the vision algorithm changes from the buoy algorithm to the ring algorithm. At this point the vehicle performs a 180-degree spin using the compass and orients the top camera towards the rear to locate the ring. The heading of the vessel is adjusted to steer directly toward these targets. The results of applying this image processing technique to an image containing the gold ring are shown in Figure 7.

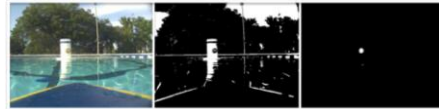


Figure 7 Vision algorithm results for the gold ring mounted to a white buoy

VI. SYSTEM INTEGRATION AND TESTING

Seagle 3.0 is the product of extensive development and design optimization. Such a complex, multidisciplinary project presents a significant systems integration challenge. Components on the vehicle; each must be able to function individually and in conjunction with the other systems on the vehicle. There are four main systems on Seagle 3.0. They are the electrical system, the hardware (sensors), the software system, and the mechanical system. Each system has a central point of integration; for example, the central point of integration for electrical system is the power distribution board. The central point of integration in the mechanical system is the hull and the hard mounting points. For the sensors and flow of information, the central point of integration is the navigation computer. The LabVIEW programming environment is the central point of software integration. LabVIEW is a critical tool used to receive and organize data from the sensors, and then make the necessary decisions. Software, especially the vision algorithm, was extensively tested in the lab using simulation tools. The team took the vessel out during different weather conditions to create videos of buoys and targets in the water. These videos were converted to Audio Video Interleave (AVI)

format. The team was able to test the code with the videos in the lab without having to set up and run the vessel for every code modification.

VII. CONCLUSIONS AND FUTURE WORK

Seagle 3.0 is a fully autonomous surface vehicle designed and manufactured by engineering students at Embry-Riddle Aeronautical University. In developing Seagle 3.0, the team maintained a mission focus, seeking to meet all the base requirements while providing better than expected overall performance. Seagle 3.0 demonstrates exceptional systems integration, combining proven software and hardware solutions with unique ideas and novel solutions to accomplish the mission tasks. The future of Seagle 3.0 is to be used as a developmental platform for integrating new sensors and systems for future vehicle while they are being constructed.

ACKNOWLEDGMENTS

The authors of this paper would like to give a special thanks to all the students in the Robotics lab and the time and effort that they put in on these student projects.

REFERENCES

- [1] A123-M1a Datasheet, Feb 1, 2012. <http://www.a123systems.com/>
- [2] "2010 Official Rules and Mission," Association for Unmanned Vehicle Systems International. San Diego, California
- [3] Martins, A.; Almeida, J.M.; Ferreira, H.; Silva, H.; Dias, N.; Dias, A.; Almeida, C.; Silva, E.P.; , "Autonomous Surface Vehicle Docking Manoeuvre with Visual Information," *Robotics and Automation, 2007 IEEE International Conference on* , pp.4994-4999, 10-14 April 2007
- [4] Bruce, J.; Balch, T.; Veloso, M.; , "Fast and inexpensive color image segmentation for interactive robots," *Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on* , vol.3, pp.2061-2066 vol.3, 2000
- [5] Dunbabin, M.; Lang, B.; Wood, B.; , "Vision-based docking using an autonomous surface vehicle," *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on* .pp.26-32, 19-23 May 2008
- [6] M. Dunbabin, J. Roberts, K. Usher, G. Winstanley, and P. Corke, "A hybrid AUV design for shallow water reef navigation," in Proceedings of the 2005 International Conference on Robotics and Automation, Barcelona, Apr. 2005, pp. 2117-2122
- [7] Xiaojin Gong; Bin Xu; Reed, C.; Wyatt, C.; Stilwell, D.; , "Real-time Robust Mapping for an Autonomous Surface Vehicle using an Omnidirectional Camera," *Applications of Computer Vision, 2008. WACV 2008. IEEE Workshop on* .pp.1-6, 7-9 Jan. 2008
- [8] M. Agrawal, K. Konolige, and R. C. Bolles. Localization and mapping for autonomous navigation in outdoor terrains: A stereo vision approach. WACV, 2007.

