

Final Design Report
Virtual Integrated Environment for HARMONIE

United States Naval Academy

EE411

Section: 6141

29 April 2015



Frederick Tidwell

Philip Song

Computer and Electrical Engineering Department

Executive Summary:

The object of this project was to create a software simulation of a robotic arm to be used within Johns Hopkins Applied Physics Lab's (APL). Hybrid Augmented Reality Multimodal Operation Neural Integration Environment (HARMONIE) a multi-year project that addresses the needs of paralysis and Amyotrophic Lateral Sclerosis (ALS) patients. The goal of HARMONIE is to use low-cost commercial technologies to assist these patients with activities of daily living. The HARMONIE system consists of a robotic arm mounted onto a wheelchair combined with retina-scanning eyewear that contains software to display augmented reality.

In order to assist software engineers in programming pre-planned limb movements for this system, a software simulator, which we call a Virtual Integrated Environment (VIE) was required to simulate the trajectory of a robotic arm in a three-dimensional environment.

The five major customer requirements for the VIE were: precision trajectory mapping, real time, operational reliability, ease of use, and small program size. The design characteristic that had the most significant impact on decision making was the need for precision in the trajectory mapping of the robotic arm. The related engineering characteristics are position error, velocity error relative to the actual iArm, and program size. The design's essential features, or constraints, include commands inputs, accurate positioning of the robotic arm, and the capability to function without additional hardware.

Two design concepts were developed to meet the constraints and engineering requirements. The first design concept minimized complexity, size, and delay by utilizing the PhysX physics engine to calculate the behavioral information inside of Unity, a commercial game engine interface currently being used in HARMONIE. The second concept used open source software to automatically calculate the kinematics of the robotic arm given the arm's characteristics. The first design concept was chosen because it provides optimal size, speed, and timeliness.

Our overall project could be divided into two main groups. The VIE could be interfaced with the physical iArm, or it could run as a standalone program. Both deliverables run in real-time and both achieved the same level of accuracy. These programs are fully capability and ready to operate within the overall HARMONIE project.

To test our VIE, a prototype test plan was developed in order to compare our project against other similar systems that are currently available. The first test we did was the end point acquisition, for which we input 8 points (one for each quadrant in three-dimensional space). Each of these points consisted of a Cartesian position and corresponding orientation, a total of 6 parameters. We compared the values given from the actual iArm with the values achieved from our VIE, which we found to have an average error of 0.64 mm. Our program size, 431 MB is much lower than our targeted value of 1.5 GB. Finally, we are currently still working on obtaining error rates for the velocity and delay.

Table of Contents

Project Definition.....	1
Need Identification.....	4
Design Concepts.....	10
Unified Unity.....	11
I/O Loop.....	13
Concept Evaluation and Selection.....	17
Embodiment Design.....	18
Prototype Test Plan.....	24
Final Design.....	28
Project Deliverables.....	33
Project Management.....	34
Budget.....	36
Appreciation of Context.....	37
Quad Chart.....	38
References.....	39
Appendix A: House of Quality.....	40
Appendix B: Gantt Chart.....	41
Appendix C: Prototype Test Plan.....	44
Appendix D: Team Charter.....	46

Table of Figures

Figure 1: Joystick-Operated iArm.....	1
Figure 2: Old, Labelled VIE for project HARMONIE using Unity3d with out-of-reach ball.....	3
Figure 3: Display of RoKiSim environment simulator for generic robotic arm.....	6
Figure 4: Example of ROS Gazebo simulating a generic robotic arm.....	6
Figure 5: Johns Hopkins Applied Physics Lab's VIE using Unity.....	7
Figure 6: Apple's Siri Speech Recognition Software.....	8
Figure 7: High level overview of VIE.....	10
Figure 8: Weight Factor Hierarchy.....	10
Figure 9: Robotic arm in Unity.....	11
Figure 10: High level flowchart of Unified Unity design concept.....	11
Figure 11: RoKiSim Screenshot.....	13
Figure 12: High level flowchart of I/O Loop design concept.....	13
Figure 13: Visualization of Denavit-Hartenburg Parameters on sample link configuration	14
Figure 14: Prototype model of viArm in Unity.....	18
Figure 15: Stereolithographic Objects of viArm in Unity.....	18
Figure 16: High level overview of VIE.....	19
Figure 17: Project Architecture.....	19
Figure 18: Two Screenshots of the Unity interface in action.....	20
Figure 19: Inverse Kinematic Equations to determine Cartesian Position.....	21
Figure 20: Sample Grammer.txt file (dictionary for commands).....	22
Figure 21: C# Method that calls executable files into the background.....	23
Figure 22: Flowchart to determine EC#1 Precision Error.....	24
Figure 23: Flowchart to determine EC#2 Velocity Error.....	25
Figure 24: Flowchart to determine EC#3 Delay.....	26
Figure 25: Flowchart to determine EC#4 Program Size.....	27
Figure 26: Unity HingeJoint Default GUI.....	28

Figure 27: VIE Hierarchy of GameObjects.....	29
Figure 28: Code to move individual virtual joints.....	30
Figure 29: Current VIE Model	29

List of Tables

Table 1: Customer Requirements.....	4
Table 2: Engineering Characteristics.....	5
Table 3: Engineering Characteristics Target Values.....	8
Table 4: Technical Assessment.....	9
Table 5: Predicted Unified Unity EC's.....	12
Table 6: Denavit-Hartenburg Parameters for Exact Dynamics' iArm.....	15
Table 7: Predicted I/O Loop EC's.....	15
Table 8: Weighted Decision Matrix.....	17
Table 9: End-Effector Position Test Points Joint Position Commands.....	25
Table 10: Data for Precision Error in Joint Position Commands.....	31
Table 11: Contact Information.....	34
Table 12: Milestones.....	35
Table 13: Budget Breakdown	36

Problem Definition

Before explaining the customer requirements and project details, a background of HARMONIE is needed to fully understand the need of a virtual environment.

HARMONIE Background

Project HARMONIE (Hybrid Augmented Reality Multimodal Operation Neural Integration Environment) was started to find a low cost solution to assist motor-deficient, wheelchair-bound patients with activities of daily living. The end goal for HARMONIE will be a robotic arm, Exact Dynamics' iArm, mounted to a wheelchair that will be controlled by the patient with voice and sight controls. The voice controls will use Windows' Speech Recognition software, while the sight controls will use a commercial product, Tobii Glasses, which scans the retinas alongside an infrared camera that can determine depth. This information is then processed to determine where the patient is looking in a Cartesian coordinate system and then the interface will use this data to perform an action. An example of this in action would be 1) data from the Red-Green-Blue (RGB) and infrared cameras that are the inputs into a computer vision algorithm to locate an object in the workspace, 2) the patient fixates on a particular object and selects it by using an elongated stare or blink, and then a command sent to the iArm to execute the desired function. Figure 1 below shows a patient operating the iArm with a joystick controller. In the envisioned application, the joystick is replaced with the Tobii Glasses as mentioned above.



Figure 1: Joystick-Operated iArm
<http://www.assistive-innovations.com/images/iarm.jpg>

Midshipmen Strachan, Tenne, Song, and Tidwell worked on developing the foundation of project HARMONIE during the summer of 2014 with APL. At this internship, Midshipmen Strachan and Tenne constructed the basic behavior and interface of HARMONIE with the game engine Unity and were followed up by Midshipmen Song and Tidwell to complete the foundation and create user-friendly functions. Some of these functions included making the iArm grab an object in the workspace, creating three-dimensional “keep out” regions that the iArm could not enter, creating visual markers to detect the exact location of the wrist and gripper as the iArm moved, and a self-feeding test so the patient could bring an object to their mouth in a safe manner.

Research departments at numerous universities have created hardware that will autonomously locate and interact with objects using robotic arms for motor-deficient patients. Biomedical and software engineers are in need of a three-dimensional virtual environment to test robotics and prosthetics that will provide an accurate simulation of the hardware in a real world environment without requiring the physical equipment. WALL-E World’s VIE will allow engineers to develop advanced limb control operations to provide more capabilities for patients. The virtual environment software needs to be independent of the physical iArm, operate in real-time, and produce simulated movements that closely match those of the physical iArm.

Problem Statement

The objective of this project is to create a Virtual Integrated Environment (VIE) with voice control that will emulate the motions and surroundings of Exact Dynamic’s iArm without the use of any hardware.

Customers

The primary customer for this Virtual Integrated Environment (VIE) is Johns Hopkins Applied Physics Lab’s Research and Development Department (REDD). REDD will be able to incorporate the VIE into project HARMONIE to reduce the development cost and time associated with providing the motor-deficient patients assistance in activities of daily living.

Information Gathering

The Virtual Integrated Environment is a tool used to emulate a robotic arm and its movement in a three-dimensional workspace. The VIE will take in a command, calculate the needed joint motor angles, and then emulate the iArm’s motion on a screen. The program used to display the virtual robotic arm and its environment is the game engine Unity. Unity is capable of creating and simulating movements of virtual objects (Game Objects). It is a popular choice for developing video games and apps such as Wasteland II and the Temple Run series. In addition, Unity incorporates a graphical user interface (GUI) editor and allows the user to control how objects behave. For example, a user could create a model of a robotic arm in Unity and attach a C# script that determines the coordinated behavior of the objects. Certain behaviors would

include linear movements of the end effector in a Cartesian space and individual motorized joint movements. These movements take advantage of the PhysX physics engine inside of Unity. The physics objects that can be added to segments of the model are “Rigid Bodies” and “Hinge Joints.” Rigid Bodies attach to a Game Objects and give it physical characteristics such as mass, inertia, gravity, and collision detection. Hinge Joints allow Unity users to connect Game Objects together – in our case, segments of the model – and apply a rotational force so that an object may swing around a connected object.

Figure 2 on the following page shows the starting foundation of the VIE that was created during the internship at APL. As seen, the VIE can simulate objects in the workspace (red sphere, bottom left), the maximum range of the iArm (white wire sphere, around figure’s edges), and keep out regions (red box, near center). The two cubes in Figure 2 represent the iArm’s wrist grip tip position in the workspace.

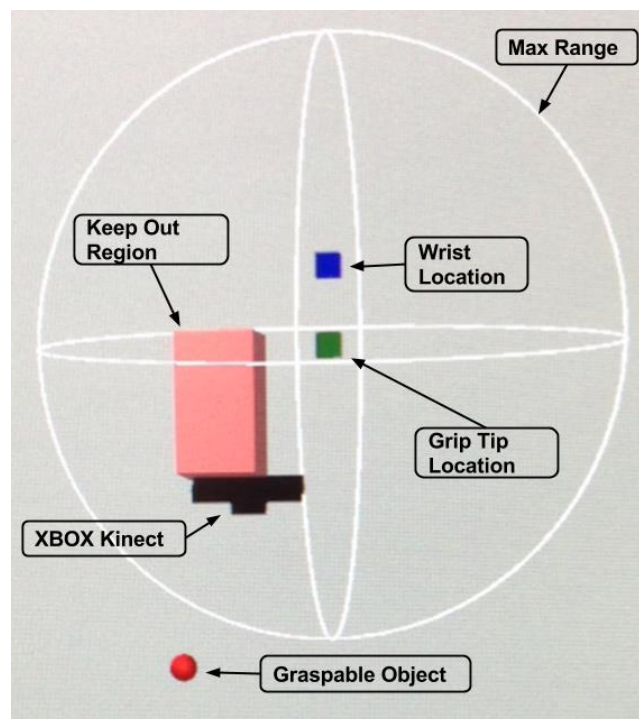


Figure 2: Old, Labelled VIE for project HARMONIE using Unity3d with out-of-reach ball

Resources

Team WALL-E World will be working with the Johns Hopkins Applied Physics Lab’s REDD team, specifically their “Revolutionizing Prosthetics” team. At APL, Robert Armiger and Dr. Brock Wester will act as our project managers and subject matter experts for the VIE. APL will also be supplying their facilities and equipment to make the VIE a reality. In addition, Dr. Currie Wooten and Dr. Justin Blanco at USNA will be assisting as technical experts for the duration of the project. Through the use of portable equipment, WALL-E World will be able to optimize time at both APL and USNA.

Need Identification

Customer Requirements

The customer requirements, observed in Table 1, were determined after meeting with Dr. Brock Wester and Robert Armiger. During the meeting, both mentors came up with their expectations for the VIE and what requirements they believed would contribute the most to project HARMONIE. These customer requirements were ranked and used to determine engineering characteristics in the House of Quality (Appendix A). The five major customer requirements mentioned in the executive summary are bolded in Table 1.

Table 1: Customer Requirements

Customer Requirements
Precise Trajectory Mapping
Real Time
Reliability
Ease of use
Small Program Size
Low Component Cost
Works with Hardware
Integrates Previous Hardware
Speech Recognition

Trajectory mapping precision is, by far, the most crucial customer requirement. APL needs a VIE that can be relied upon to simulate the position and speed of the iArm without the use of hardware. Without it, the VIE is futile. The next highest ranking customer requirement is real time data. This characteristic will determine if the virtual robotic arm will be capable of running alongside the actual iArm simultaneously and let the software engineers realistically see how the system is working without lag. This will allow the engineer to have precise timing information which enables them to make smoother transitions between iArm movements. Small program size will assist engineers share the VIE on multiple systems and, in the future, allow the program to run on embedded devices on a wheelchair with low power consumption. Reliability ties into all the customer requirements in that it provides assurance to the customer that the VIE will do what is expected.

In addition, implementing voice commands will be an add-on. Voice commands had not yet been incorporated into project HARMONIE. After developing the capability to handle voice controls with the iArm within Unity, APL will incorporate the same algorithm into HARMONIE for both the iArm and their Modular Prosthetic Limb (MPL). Voice commands and speech recognition will allow the disabled patient another means of communicating to the iArm which will

supplement the sight controls of the augmented reality video feed for those patients who are able to produce intelligible speech.

To determine how well our design will match up against competitors, the engineering characteristics shown in Table 2 are used to provide numeric comparison between the products. In Table 2, the characteristics are ranked in order of importance, with our top five items in bold. While each engineering characteristic is important to the success of the project, these five will be the determining factors in ranking our design versus others.

Table 2: Engineering Characteristics

Engineering Characteristic	Units	Improvement Direction
Precision Error	cm	↓
Velocity Error	cm/s	↓
Delay	ms	↓
Program Size	GB	↓
Complexity	# of Programs	↓
Training Time	minutes	↓
Total Cost of components	\$	↓
Function Compatibility with Hardware	✓	
Validation Percentage for Speech Recognition	%	↓

Benchmarks

The best approach to understanding the VIE design is to investigate different commercially available robotic simulators. The environment in Figure 3 below is from a robotic kinematic simulator, RoKiSim. RoKiSim takes in different joint angles for a pre-defined robot and computes forward kinematic solutions for the robot to determine end-point position [1]. In addition, the opposite scenario, inverse kinematic computations, can be used to determine joint angles given an end-point position. Only a pre-configured model of the specific robot is needed.

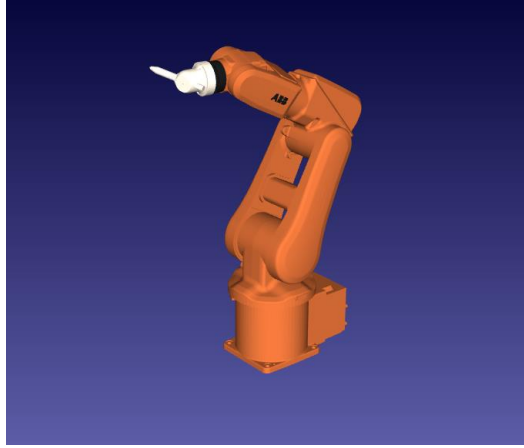


Figure 3: Display of RoKiSim environment simulator for generic robotic arm

This simulated environment can be useful in determining trajectory mappings (tracing the limb movements) between a set of poses. In addition, RoKiSim has extra functionality that allows the user to import object geometries into the workspace.

Other than standalone kinematic simulators, other software is available that can determine the kinematics and provide other features as well. The Robot Operating System (ROS) provides a flexible framework for developing robot software. ROS has a vast set of tools, collections, and conventions that simplify the creation of robot behavior across a variety of robotic platforms [2]. ROS provides a software development interface as well as a visual simulated environment also known as the “Gazebo” to write and test robotic operations as seen in Figure 4.

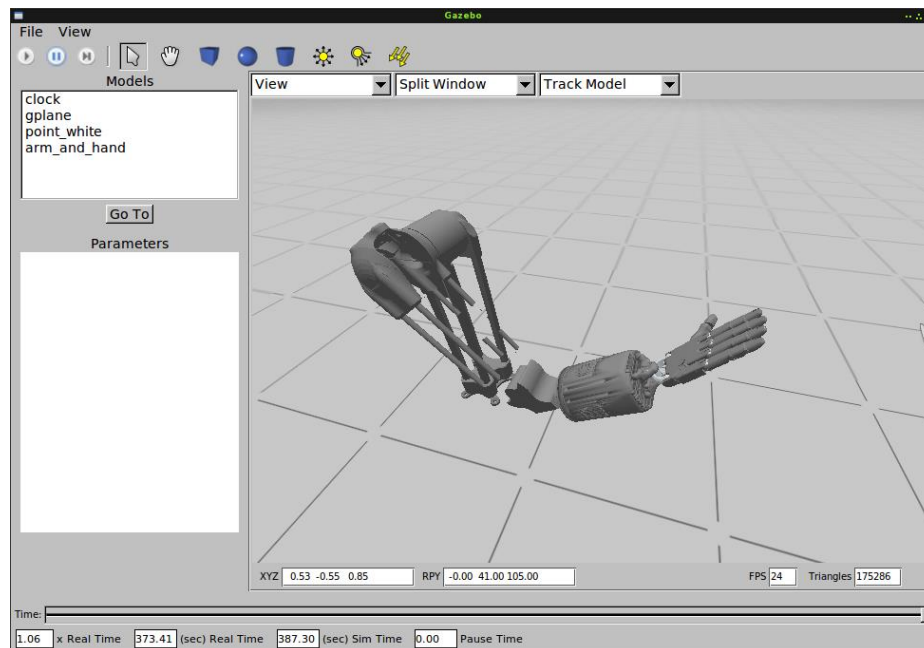


Figure 4: Example of ROS Gazebo simulating a generic robotic arm

While ROS provides a full development system, the current system that APL uses is Unity3d. Within Unity, APL engineers have developed a system for their Modular Prosthetic Limb (MPL) that allows programmers to code functions and methods for the MPL with and without hardware. Unity can be operated using various languages including C#, Javascript, and Boo. In addition, it supports computer-to-computer real-time communication that allows the programmer to incorporate different sensors and hardware in one interface. Figure 5 shows APL's VIE for the MPL [3].



Figure 5: Johns Hopkins Applied Physics Lab's VIE using Unity

Another benchmark system is the University of Massachusetts Lowell's (UML) Manus Arm Emulator [4]. The Manus Arm was developed by Exact Dynamics and was replaced by the iArm. This emulator is very similar to the VIE project, but is unable to function without the physical Manus Arm. The goal of UML's project was to perform testing with the Manus Arm with neural spike inputs or inputs from brain stimulation.

For the benchmarks for speech recognition, we chose to look at Nuance's Dragon NaturallySpeaking 13 and Apple's Siri. Dragon NaturallySpeaking 13 is the top speech recognition software on the market and can be implemented into programming solutions using Nuance's OmniPage SDK and OCR Toolkit. According to third party reviews, the Dragon NaturallySpeaking 13 achieves accuracy around 90 percent while costing \$150. Apple's Siri is a popular speech recognition software that is available on all Apple phones and devices. According to AppleInsider, testing on Siri has achieved accuracy ratings rating from 83 to 94 percent.



Figure 6: Apple's Siri Speech Recognition Software

Target Values and Technical Assessment

In Table 3 on the following page is a list of target values for the VIE. These target values were chosen to compare our VIE with similar projects. These values were discussed with our customers and it was unanimously agreed that meeting these specific engineering characteristics appropriately captures the customer requirements.

Table 3: Engineering Characteristics Target Values

Engineering Characteristics	Target Values
Precision Error (cm)	3
Velocity Error (deg/s)	2
Delay (ms)	200
Program Size (GB)	1.5
Complexity (programs)	1
Training Time (minutes)	20 Minutes
Total Cost of components (\$)	\$16,000
Function Compatibility	Yes
Validation Percentage for Speech Recognition (%)	70

Table below shows a comparison of the prototype target values to the competitor's designs and the most important engineering characteristics.

Table 4: Technical Assessment

EC's Benchmarks	Precision Error (mm)	Velocity Error (mm/s)	Delay (ms)	Program Size (GB)	Training Time (Minutes)
APL's MPL VIE	N/A	N/A	<10	1.6	N/A
ROS	N/A	N/A	N/A	Varies	N/A
UML's Manus Arm Emulator	3.1	N/A	N/A	5.1 MB	N/A
Target Values	3	20	100	<1.5	<20

Table 4 above shows the direct comparison between our VIE's target values compared to other robotic simulators available. As it can be seen, most of the competitors have not accurately assessed their designs according to the characteristics that deemed important for the VIE. For example, no competitor has attempted to determine how long it will take a developer to learn the VIE. In addition, since team WALL-E World's VIE will be connected with the Unity interface to simulate all the hardware, the program size will be extremely high compared to the other competitors. The target values were obtained by deciding how the VIE would effect a patient. For the precision error, the Euclidean distance is checked to see the difference between actual position and expected position. In the event we are doing a self-feeding test, where a patient is bringing the end effector to their mouth, a precision error of greater than 3 mm and velocity error of 20 mm/s could cause bodily harm to the patient if it were to hit the patient in the eye or other sensitive area.

Design Concepts

Two design concepts were considered for mapping the trajectory of the robotic arm. This mapping is crucial in simulating the movement of the arm and emulating the arm's trajectory through the environment.

Each concept assumed that the input to the system is an iArm command and will output to Unity. The four iArm commands that can be given include Cartesian (giving the iArm a wrist position and rotation), Joint (inputting joint angles), Velocity (give a Cartesian direction and speed), and Status (will give feedback to iArm Cartesian position and rotation as well as joint angles). The values in each command must be processed to find the end pose of the iArm, which will then allow virtual limb movement to be executed within Unity. A high level overview of the project is shown in Figure 7 below.

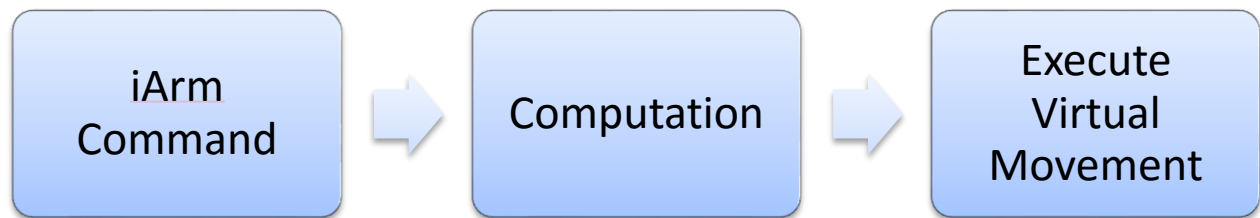


Figure 7: High level overview of VIE

In Figure 7, the iArm Command and Execute Virtual Movement blocks are the input and output of our project, respectively. Both are essential to meet the constraints of the customers and adapt to meet their requirements. The design choice is in determining the transition between the system blocks and designing the behavior of the arm in the center block.

The design concept that met the constraints and optimized the top five engineering requirements was selected using a Weighted Decision Matrix with the Weight Factors illustrated in Figure 8 below.

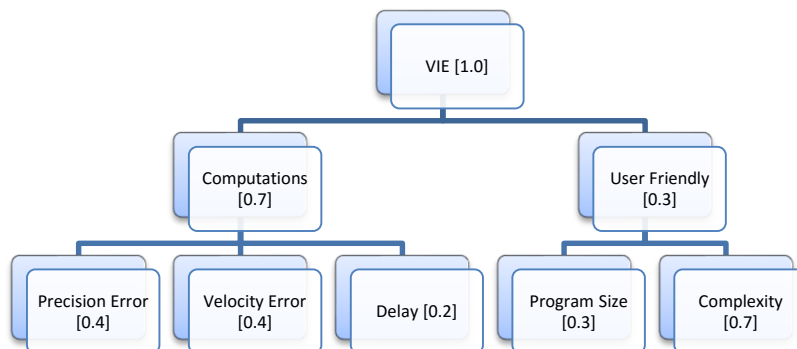


Figure 8: Weight Factor Hierarchy

Design Concept Number 1 “Unified Unity”

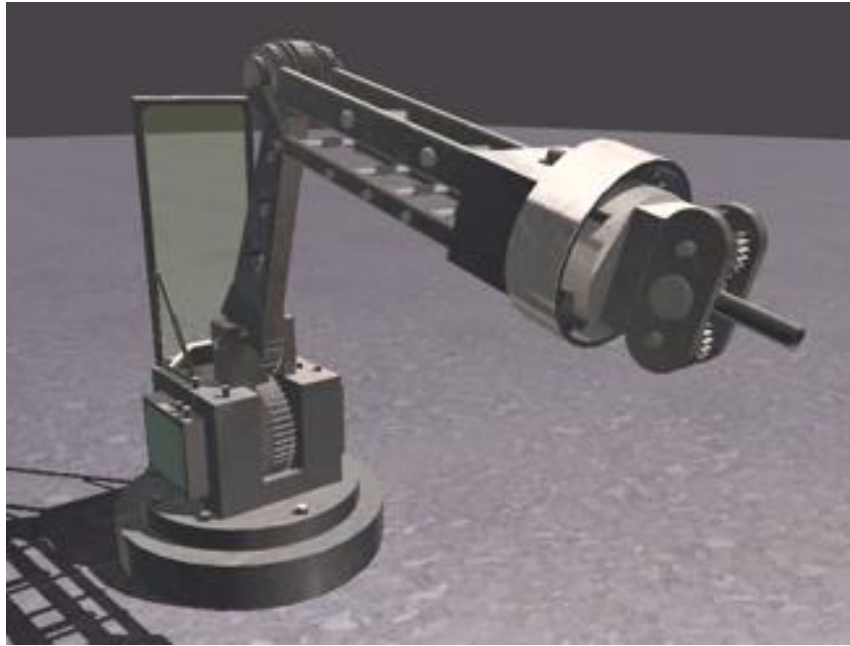


Figure 9: Robotic arm in Unity

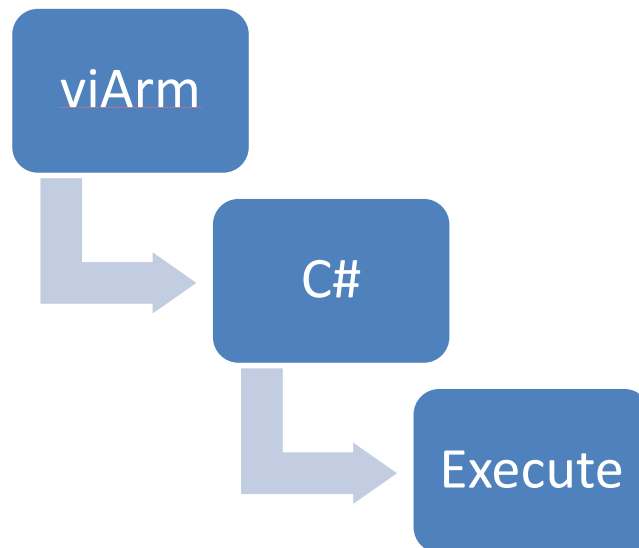


Figure 10: High level flowchart of Unified Unity design concept

“Unified Unity” took advantage of being a modular piece to the Unity interface. In this design concept, the VIE would be listening for a movement command. When one occurred, a C# script would parse the values and use a trajectory algorithm to determine the exact joint angles of all the motors of the virtual iArm (viArm). The script would then use this information to update the viArm until it is in the correct position by incrementing the angles at pre-specified rates in

discrete time intervals. Communication between scripts in the Unity interface would be through public function calls that have setting/getting functionality. This would allow variables to be shared between scripts. Figure 9 shows an example robotic arm inside of Unity that illustrates the high level of description and realistic detail that could be expected out of the WALL-E World's viArm.

Because the VIE would be packaged within Unity, determining a communication method between the Unity interface and the VIE would be unnecessary. In addition, because there is no need for a communication channel, transmission delay would be negligible.

The primary disadvantage to this design concept was the need to use Unity's PhysX physics engine. Due to the rigidity of the objects implemented in the physics engine, PhysX could become unstable and make debugging burdensome. Research and time would be needed to break down the computation process for the physics engine.

Table 5: Predicted Unified Unity EC's

Engineering Characteristic	Units	Value
Precision Error	cm	0
Velocity Error	cm/s	0
Delay	ms	0
Program Size	GB	1.2
Complexity	# of programs	1

Table 5 shows the predicted performance of the top five engineering characteristics of Unified Unity. The precision and velocity error were predicted to be zero assuming that the algorithms were the same that will be computed inside the iArm's microcontroller that instructs the joint motors. The program size stayed relatively small because the only addition to the HARMONIE project is another C# script and a GameObject.

Unified Unity met all constraints. The kinematic equations would allow for a precise trajectory mapping. Because the VIE was packaged within Unity, it would not add any transmission delay, allowing the project to run at real-time. The setup of this project was not complex, requiring only one program, Unity, to run the project.

Design Concept Number 2 “I/O Loop”

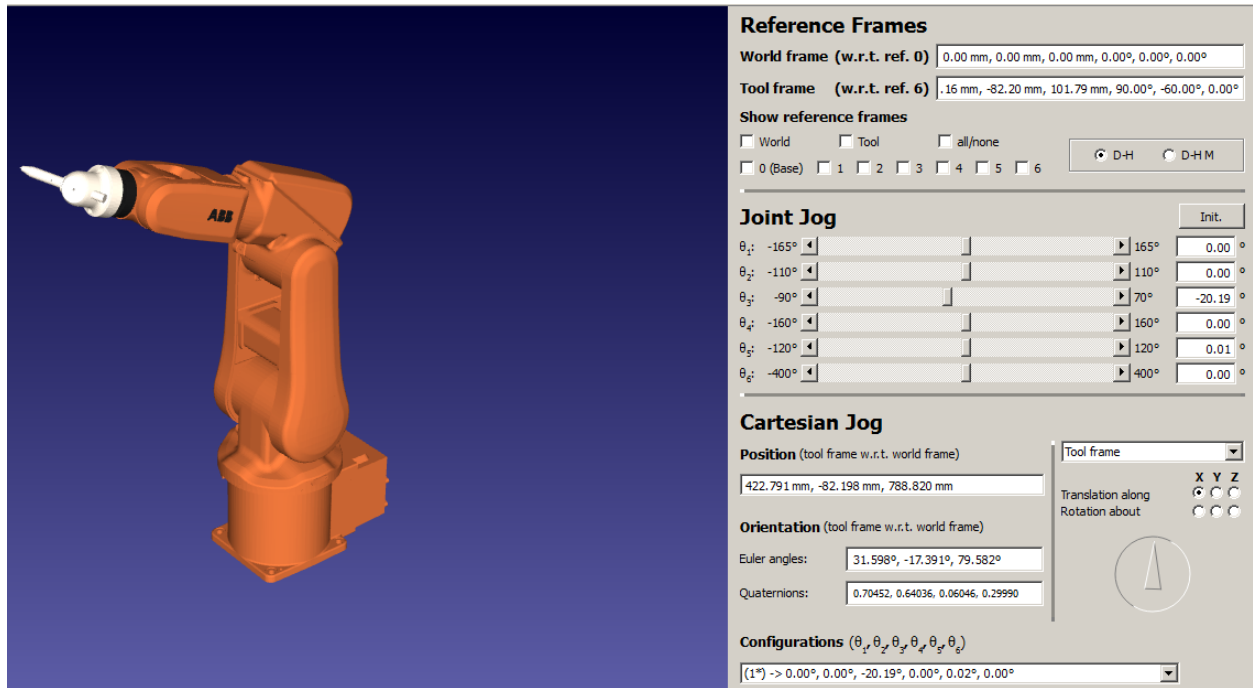


Figure 11: RoKiSim Screenshot

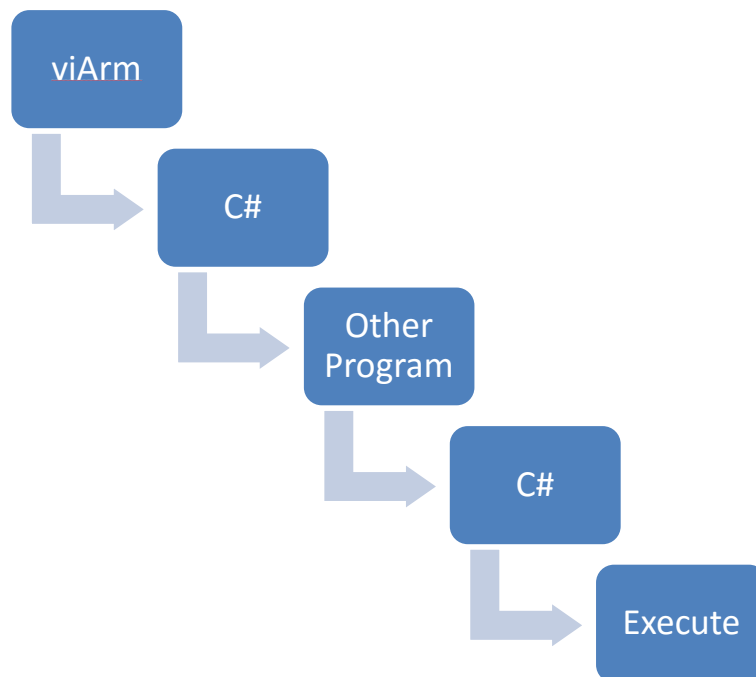


Figure 12: High level flowchart of I/O Loop design concept

The second design concept involved using another program as a kinematic equation solver. Following Figure 12, I/O Loop will have a C# script within the Unity interface that would take in an iArm instruction, parse the information, and then send the data via UDP to an independent program such as ROS or RoKiSim (as seen in Figure 11). The program would then compute the iArm kinematics with its kinematic equation solver and send the results back to the C# script attached to Unity. Then this script would execute as in Unified Unity, where it would emulate the viArm limb movement.

The calculations needed to do the forward and inverse kinematic equations involve the use of the Denavit-Hartenberg (D-H) parameters. These parameters characterize the dimensions and joint types of a robot and are typically displayed in a table. The four parameters include: “d” – the offset length along the axis perpendicular to the common normal, “ θ ” – the angle around the perpendicular axis, “a” – the length in the normal axis, and “ α ” – the angle around the normal. Table 6 below is the D-H table for the iArm and Figure 13 is a visual example of the parameters labeled on a sample link configuration. The variable “i” refers to the specific joint.

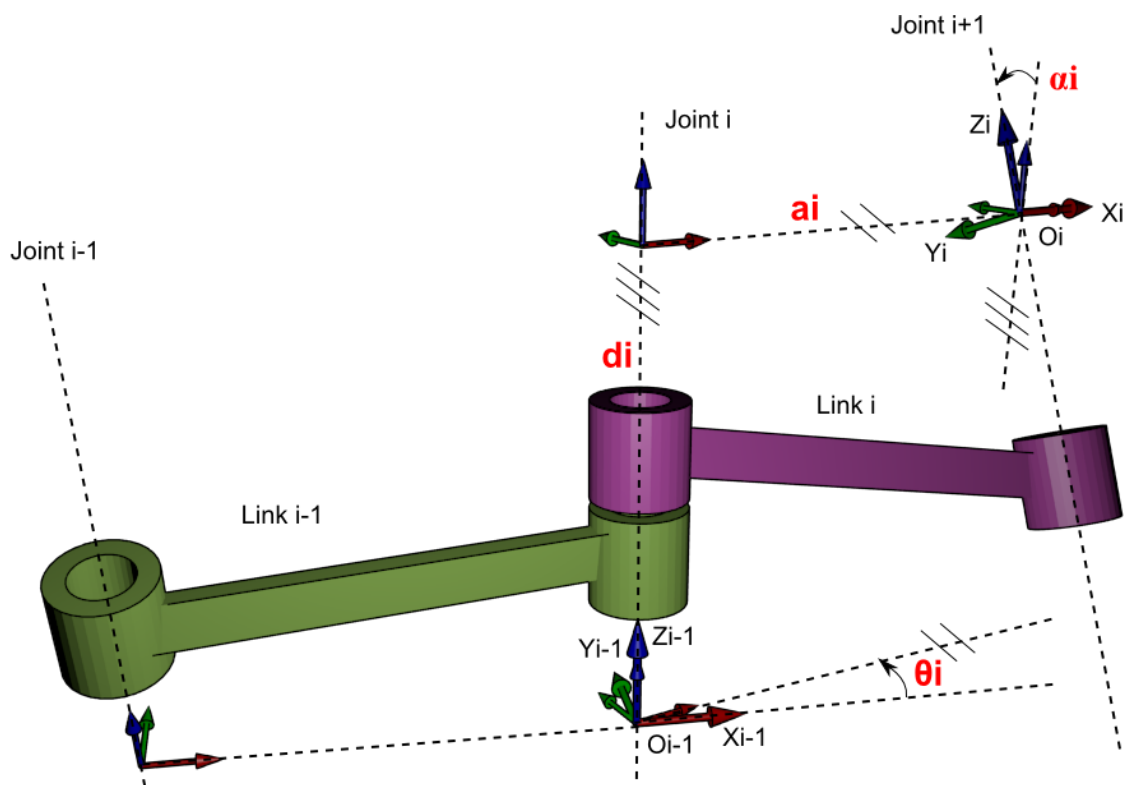


Figure 13: Visualization of Denavit-Hartenberg Parameters on sample link configuration.

http://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters#/media/File:Classic-DHparameters.png

Table 6: Denavit-Hartenburg Parameters for Exact Dynamics' iArm

i	α (in radians)	a (in meters)	θ (in radians)	d (in meters)
1	$\pi/2$	0.391	θ_1	0
2	0	0.400	θ_2	-0.175
3	$\pi/2$	0	θ_3	0.075
4	$\pi/2$	0	θ_4	0.330
5	$-\pi/2$	0	θ_5	0
6	0	0	θ_6	0

The D-H Table would be converted into homogeneous transforms and would serve as a tool to find the kinematic equations in order to calculate position, rotation, and joint angles. The advantage of this design concept is that it computes the exact kinematics of the iArm rather than relying on the PhysX physics engine.

An independent process also has drawbacks. With this concept, the user must run 2-4 different processes to run the VIE. In addition, the communication between the processes would add transmission delay to the system and consume more power because of the need for a script that is constantly listening and transmitting to other processes. Overall, the VIE program size must account for both Unity and another program.

Table 7: Predicted I/O Loop EC's

Engineering Characteristic	Units	Value
Precision Error	cm	0
Velocity Error	cm/s	0
Delay	ms	>10
Program Size	GB	1.3
Complexity	# of programs	2-4

Table 7 shows the predicted engineering characteristics of I/O Loop. Starting from the bottom, the complexity of I/O Loop would be relatively high with 2-4 programs, depending on the communication capabilities of the independent program. The program size would only be slightly larger than the original program. The slight increase is from the additional program for adding in speech commands. For example, if RoKiSim is chosen to make the calculations, then only 51 Megabytes would be added to the original program size. The delay for this concept could be more than ten milliseconds. According to CodeProject, the average roundtrip latency of UDP is 10 ms [5]. Adding on the latency of the kinematic equation solver, the delay could potentially be much larger. Precision and Velocity Error, can be assumed to be precise and negligible.

The I/O Loop should theoretically meet all requirements of the project. The only concern is real time emulation. There could be enough delay to cause a visual lag. However, with a more powerful computer, the delay could in theory be minimized and ignored.

Concept Evaluation and Selection

After examining the design concepts and evaluating them using the Weighted Decision Matrix shown in Table 8, design concept 1, Unified Unity, was selected as the final design concept.

Table 8: Weighted Decision Matrix

EC	Weight Factor	Unified Unity			I/O Loop		
		Value	Score	Rating	Value	Score	Rating
Precision Error (cm)	0.28	0.0	4	1.12	0.0	4	1.12
Velocity Error (cm/s)	0.28	0.0	4	1.12	0.0	4	1.12
Delay (ms)	0.14	0.0	4	0.56	>10	2	0.28
Program Size (GB)	0.09	1.2	3	0.27	1.3	3	0.27
Complexity (#)	0.21	1	4	0.84	2	2	0.42
Total				3.91	3.21		

Delay and complexity were the deciding factors in concept selection. Having a packaged kinematic equation algorithm embedded in Unity provided more optimal engineering characteristics. While many values had the same ranking score, the wide margin between complexity made the difference.

Unified Unity should be able to meet all customer requirements. With this, it can provide ease of use without needing to keep track of programs other than Unity. Transmission time would be negligible and communications do not have to be set up between processes.

Embodiment Design

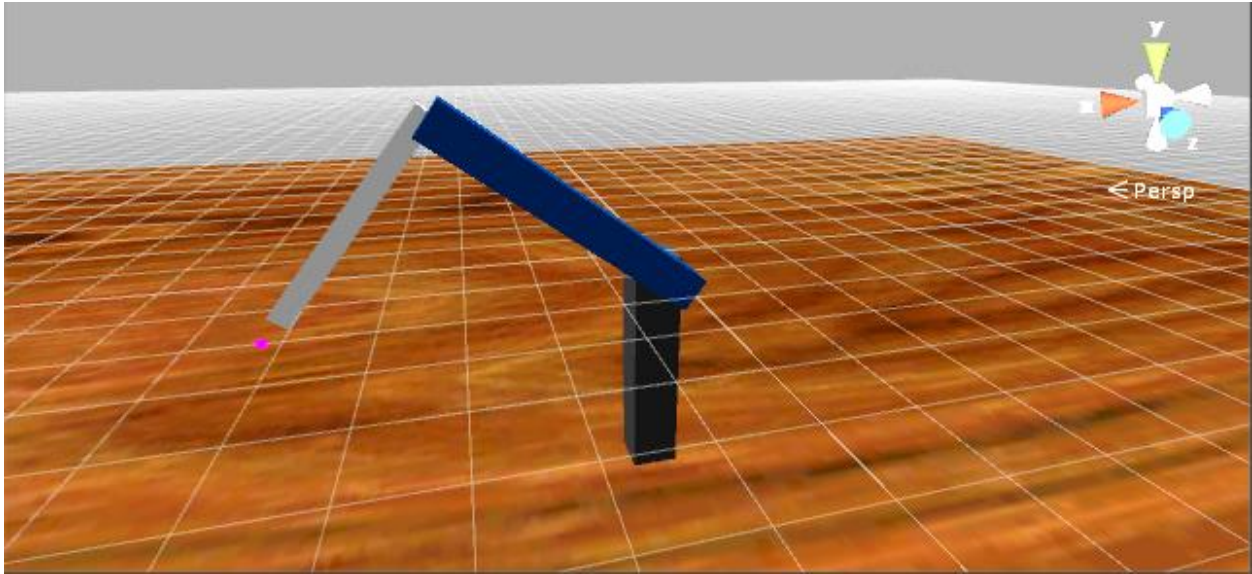


Figure 14: Prototype model of viArm in Unity

Figure 14 above shows Team WALL-E World's first created prototype of the virtual iArm. Each rectangular block represents a segment of the iArm, and the pink dot at the endpoint represents the wrist position. Each of the rectangular blocks corresponded to a link in the actual iArm, while the wrist position was used to mimic the actual position of the iArm's wrist. Within Unity, hinge joints are attached to the blocks. Hinge joints are special PhysX physics objects that allow for rotations relative to other objects. For example, rotating the blue block with the hinge joint would move the rotation axis from the center of the block to the point where it attaches to the black block (simulating moving the arm up and down). The Exact Dynamics iArm STL (Stereolithography) files were used to develop a 3-D CAD model of the actual iArm as seen in Figure 15.



Figure 15: Stereolithographic Objects of viArm in Unity

The design for the VIE incorporated the steps in Figure 16. The user could move the viArm with directional buttons on the display, type a command into a textbox, or run a series of pre-programmed trajectories that interact with objects that are randomly positioned in the virtual workspace. An attached behavioral C# script took in the inputs, parsed them, and determined the appropriate trajectory. The results will be input to an emulator function that moves the viArm from its old pose to its new pose. In addition, with specialized limb movement functions, there can be intermediate arm positions.



Figure 16: High level overview of VIE

Figure 17 is the Project Architecture that includes the interactions of each component of the design.

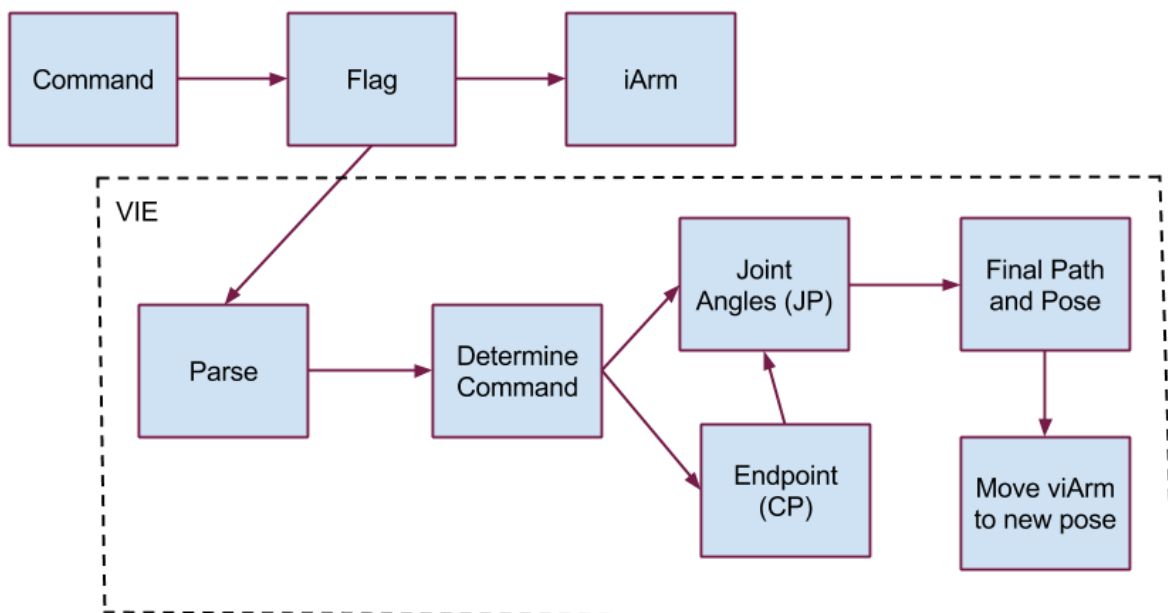


Figure 17: Project Architecture

The “Command” block, as stated earlier, took in different iArm commands. The commands were inputted via the Unity interface, which is shown in Figure 18.

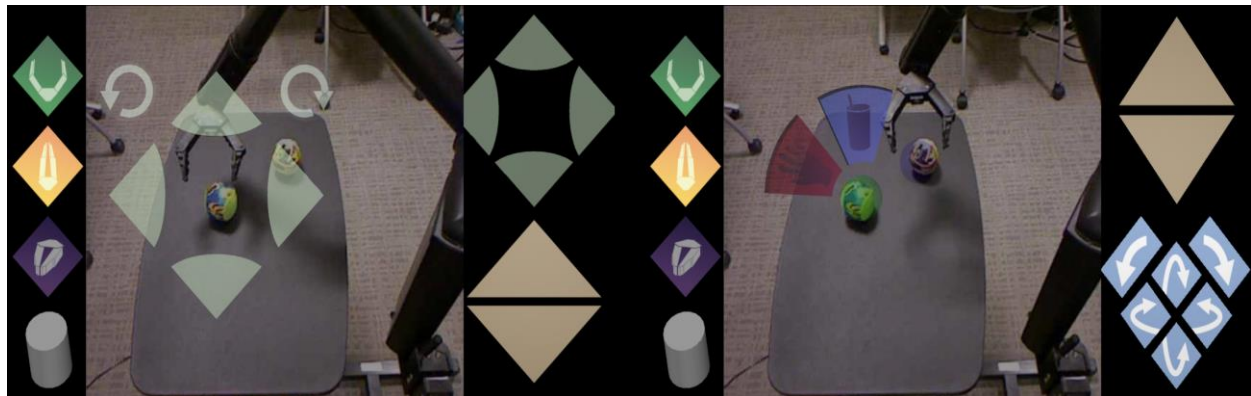


Figure 18: Two Screenshots of the Unity interface in action

The viArm model would be working behind this interface and could be seen by the software engineers. Multiple cameras within Unity allowed the developers to see the interface and the VIE simultaneously.

The “Flag” block controlled a switch to determine if the command is going to the iArm or the VIE. An example pseudo-code for this is:

```

if(flag == 1)
{
    System = “iArm”;
}
else
{
    System = “VIE”;
    Parse(command);
}

```

When the flag is true, the command was sent to the iArm. When the flag is false the string command was sent into a parsing function that took the instruction data and stored them into public variables. Using the C# script, the instruction name would go into a switch that determined the given command and then called either the forward kinematics function or the

inverse kinematics function resulting in a final pose. An example of the inverse kinematic code to determine the position of the wrist is shown in Figure 19.

```
//Finding theta0////////////////////////////////
float r=Mathf.Sqrt(Mathf.Pow(Px,2f) + Mathf.Pow(Py,2f));
//theta(1)= atan2(Py,Px) + asin(d3/r);
theta[0]= Mathf.Atan2(Py,Px) + Mathf.Asin(d[2]/r);

//Finding theta1////////////////////////////////
float n2 = 1f;
float V114= Px*Mathf.Cos(theta[0]) + Py*Mathf.Sin(theta[0]);
r=Mathf.Sqrt(Mathf.Pow(V114,2f) + Mathf.Pow(Pz,2f));
//Psi = acos((a2^2-d4^2-a3^2+V114^2+Pz^2)/(2.0*a2*r));
float Psi = Mathf.Acos((Mathf.Pow(a[1],2f)-Mathf.Pow(d[3],2f)-Mathf.Pow(a[2],2f)+Mathf.Pow(V114,2f)+Mathf.Pow(Pz,2f))/(2.0f*a[1]*r));

theta[1] = Mathf.Atan2(Pz,V114) + n2*Psi;

//Finding theta2////////////////////////////////
float num = Mathf.Cos(theta[1])*V114+Mathf.Sin(theta[1])*Pz-a[1];
float den = Mathf.Cos(theta[1])*Pz - Mathf.Sin(theta[1])*V114;
theta[2] = Mathf.Atan2(a[2],d[3]) - Mathf.Atan2(num, den);
```

Figure 19: Inverse Kinematic Equations to determine Cartesian Position

This data was be processed by a Unity C# script that would determine a trajectory of the viArm. Using incremented joint angles at specific velocities, Unity would emulate the movement to the screen.

Speech recognition software was also implemented using C# .NET 4.0. We were unable to make it part of Unity because the version of C# interpreted by the Unity compiler was incompatible with the newest version. The program, RecoServeur, that converted the speech to text is open-source software found on the Unity3d Forums. RecoServeur used the Windows Speech Recognition SDK and transmitted the interpreted speech as strings through UDP. All the configurations could be set up through a grammar.txt file that included an IP address, UDP port number, a keyword that terminates the script, a list of recognizable commands, and a validity percentage of the speech to the predicted text. Figure 20 is a sample of the grammar.txt file. The grammar file could be thought of as a dictionary for commands.

```
# comment
#E complete
#I 127.0.0.1
#P 26000
#V 70
complete
color to blue
color to red
color to green
move left
go home
move right
move down
move up
move forward
move backward
arm fold
arm unfold
stop movement
grab the ball
select ball
bring food to mouth
bring drink to mouth
```

Figure 20: Sample Grammer.txt file (dictionary for commands)

We attached this script to Unity by calling the executable program, RecoServeurX86.exe, inside of a function within the main Unity C# script. The main script calls the program and hides this process within the background. This is pertinent in that it does not disrupt the end user (the patient) with unnecessary windows. The C# method is shown below in Figure 21.

```

3819 private void initSpeechCommands()
3820 {
3821     receiveThread = new Thread(new ThreadStart(ReceiveData));
3822     receiveThread.IsBackground = true;
3823     receiveThread.Start();
3824     hostname = Dns.GetHostName();
3825     IPAddress[] ips = Dns.GetHostAddresses(hostname);
3826     if (ips.Length > 0)
3827     {
3828         LocalIP = ips[0].ToString();
3829         UnityEngine.Debug.Log(" MY IP : " + LocalIP);
3830     }
3831
3832
3833     //moved to init
3834     try {
3835         Process myProcess = new Process();
3836         myProcess.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;
3837         myProcess.StartInfo.CreateNoWindow = true;
3838         myProcess.StartInfo.FileName = "C:\\Windows\\system32\\cmd.exe";
3839
3840         string path = "C:\\Users\\ANIntern\\Desktop\\Projects\\HARMONIE_iArm_v4\\Assets\\Assets\\Scripts\\Experimental\\speech.bat";
3841         myProcess.StartInfo.Arguments = "/c" + path;
3842         myProcess.EnableRaisingEvents = true;
3843         myProcess.Start();
3844     } catch (Exception e) {
3845         print(e);
3846     }
3847 }

```

Figure 21: C# Method that calls executable files into the background

Prototype Test Plan

The Prototype Test Plan for the VIE design explains how the 4 highest priority engineering characteristics – precision error, velocity error, delay, and program size – would be measured.

Precision Error

Precision error is the difference between the pose given by the iArm and the pose generated by the VIE. Figure 22 below was implemented to measure the error.

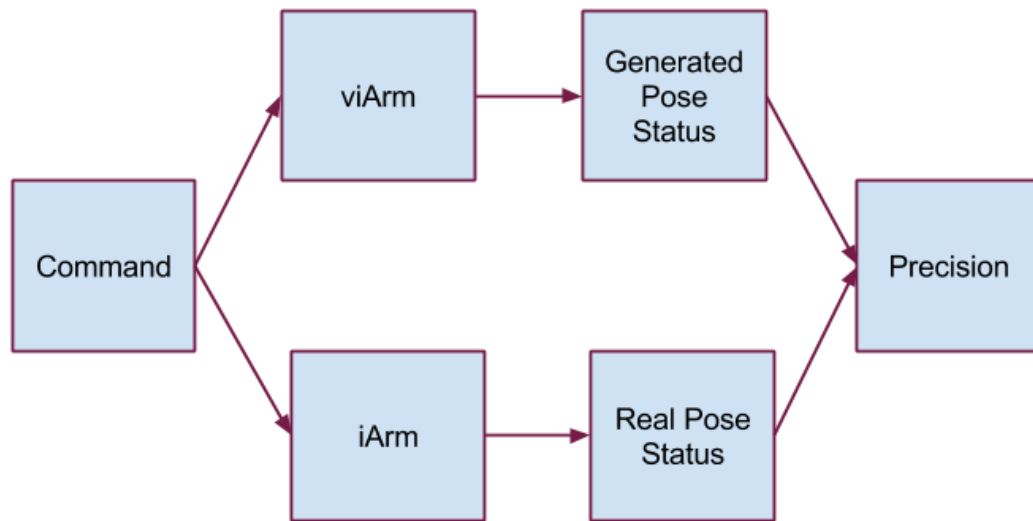


Figure 22: Flowchart to determine EC#1 Precision Error

Both the viArm and iArm started from the same position, known as the home position or unfold pose. In testing, the viArm and iArm were issued the same command, called the “CP” command. The “CP” (Cartesian position) command took in a desired Cartesian position for the end-effector position. Different positions were input into the systems, one for each quadrant in 3-D space for a total of 8. Both systems executed the command, and then responded with their status after the command was executed. These reported values would then be subtracted to determine the Euclidean distance between the physical position and virtual position.

Table 9: End-Effector Position Test Points Joint Position Commands

Test #	iArm End-Effector Position [mm]		
	X	Y	Z
1	100.00	200.00	300.00
2	125.00	150.00	-150.00
3	350.00	-175.00	0.00
4	350.00	-175.00	-150.00
5	-100.00	100.00	125.00
6	-150.00	350.00	-350.00
7	-125.00	-350.00	100.00
8	-200.00	-100.00	-200.00

Velocity Error

Velocity error is the difference between the joint speeds of the iArm and the corresponding speeds emulated by the VIE. Figure 23 below illustrates how it was computed.

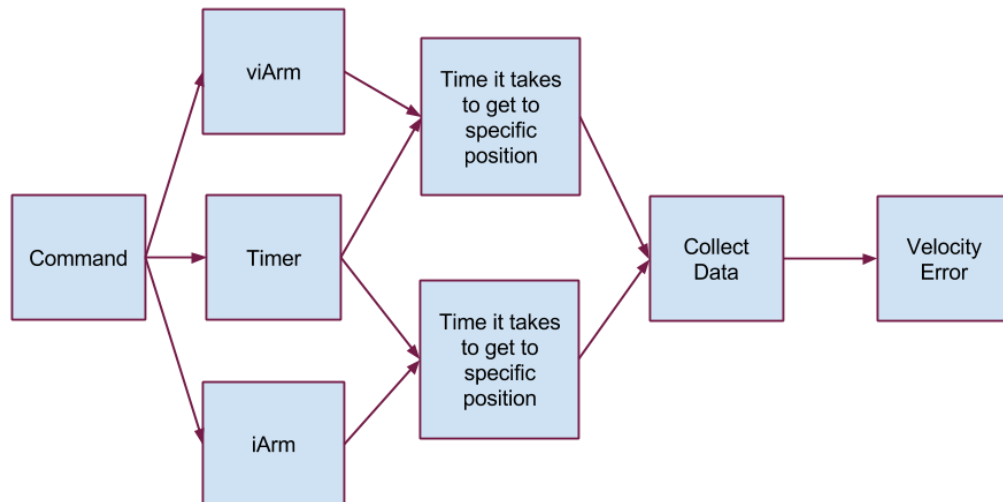


Figure 23: Flowchart to determine EC#2 Velocity Error

In testing, the viArm and the iArm were issued the same command with one timer connected to both systems. Both systems started from the same initial position of (300, 300, 300 mm) for the (X, Y, Z). The viArm and iArm were then both commanded to move 500 mm in a certain direction (both positive and negative X, Y, and Z for a total of 6 different new positions). At specific time intervals, a status command was executed to determine the speeds of all the joints for the iArm and the viArm. These speeds were subtracted and calculate the error from the results.

The reason why the velocity was so accurate was because the velocity value was determined by testing the physical iArm. In a similar testing plan to the one mentioned above, the velocity of the iArm was determined by starting the iArm at the initial position of (300, 300, 300 mm) and commanded to move in each direction. Each direction was repeated five times to reduce variability in the human error when timing the physical iArm with the stopwatch. These five trials averaged together to create the velocity value for the velocity value used in the viArm.

Delay

Engineering Characteristic #3, Delay, is computed as the difference in time between when the iArm achieves its final pose and when the viArm generates it final pose. Figure 24 shows the delay measurement plan as a flowchart.

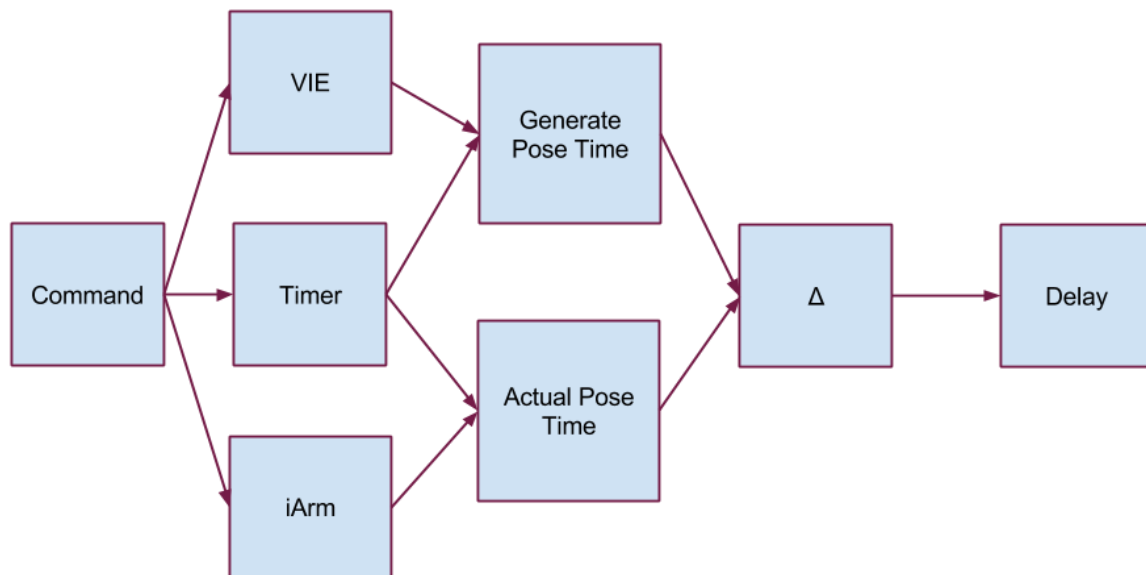


Figure 24: Flowchart to determine EC#3 Delay

Like the test plan for velocity error, this test required a shared timer that would check the time it takes to achieve the final pose from each system and find the difference. This involved using the velocity computed as mentioned on the previous page. In the program a co-routine was run to act

as a stopwatch for the VIE. The physical iArm's movement was timed with a stopwatch, and the time for the physical iArm was compared with the VIE's time to determine delay.

Program Size

Engineering Characteristic #4, Program Size, was essential in determining if the VIE was portable enough to easily transfer from computer to computer. Program size involved opening the Windows Explorer Properties GUI and locating the file size. Figure 25 illustrates an example GUI pointing to the file size information.

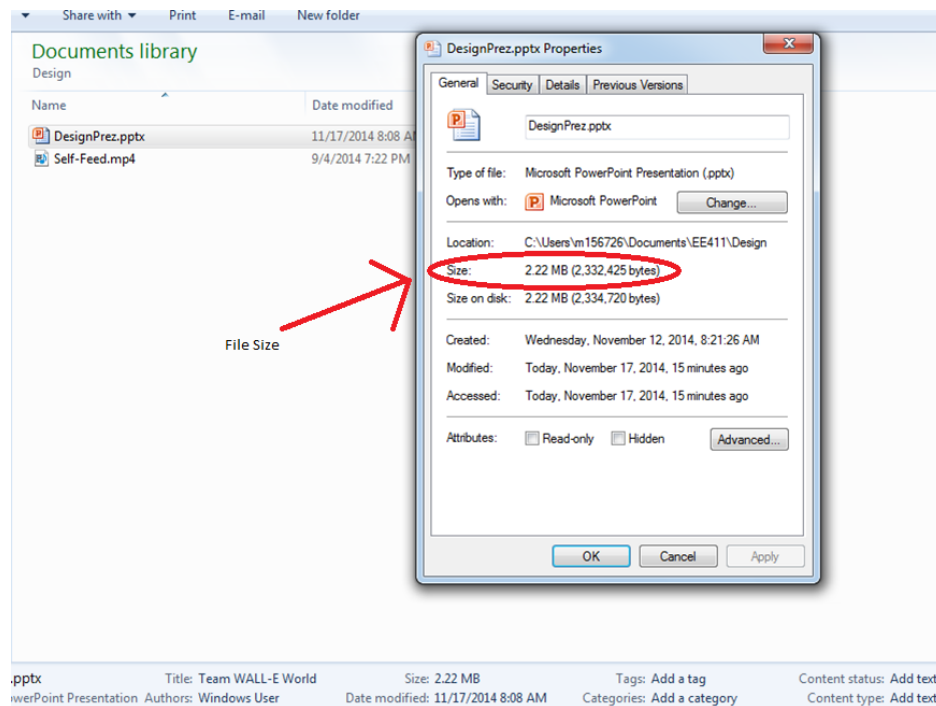


Figure 25: Method to determine EC#4 Program Size

Final Design

The first problem we addressed this semester involved the rotational axis for the arm segments. The rotational axis was incorrect, resulting in awkward rotations in which the arm would rotate away from the joints. We used different methods to try to piece the segments together, and we were finally able to achieve an accurate connection from the base of the arm to the bicep of the arm by using the Unity physics object, “HingeJoint”. This allows different objects to be connected together and for one to swing around the other. A good example would be a door swinging around the frame. In Unity, we are able to set multiple variables that allowed us to use a motor with a target velocity and force, a spring motor, minimum and maximum limits on angles, and other properties we did not use. In our case, we used motors to drive the joints to specific velocities. The default GUI set up for a HingeJoint within Unity is shown in Figure 26 below.

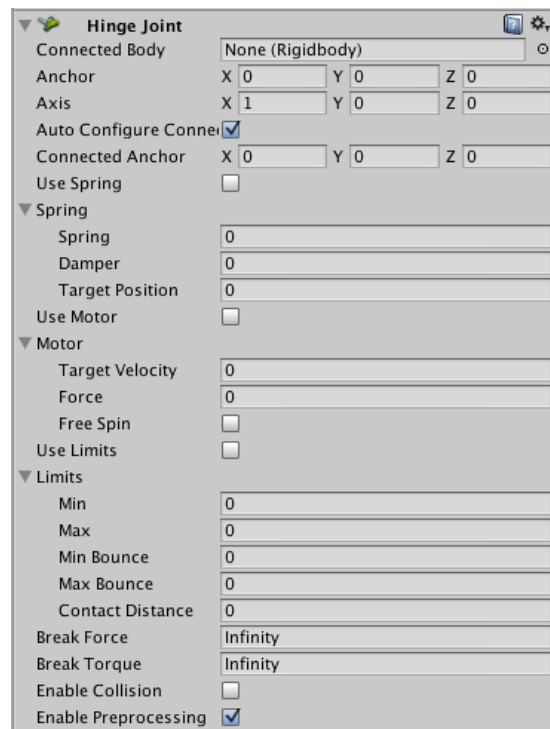


Figure 26: Unity HingeJoint Default GUI

After determining how we were going to emulate the commands with the VIE, we received the STL files from the vendor, Assistive Innovations. With this 3-D model, we were able to define GameObjects (or segments) to our VIE to construct the viArm, the camera, and light source, as seen in Figure 27.



Figure 27: VIE Hierarchy of GameObjects

Next we started working on emulating the arm movement in order to move the virtual arm with joint angle commands, which we achieved very accurately after a few weeks of constant debugging. More data points were gathered from the physical iArm to figure out how the actual iArm moved. It was then found out what the positive and negative direction of rotation was for each joint. After talking with our technical expert Dr. Brock Wester, the model was fine-tuned to make the viArm as similar as possible to the virtual MPL, which Johns Hopkins APL also uses. We made a parent class that incorporated the entire model and made each individual viArm segment as a child as seen in Figure 26. We then attached the HingeJoints and Rigidbodies to all of the children objects. This will allow APL to write more general scripts in the future that could apply to both systems. An issue present with both the vMPL (virtual MPL) and the viArm is extraneous movement when the motor was running from other joints. To combat this, first limits were set on joint angles, but this attempt failed. After debugging, by setting the motors of nonmoving joints to a high force value (1000 units) and the target velocity to 0 [degrees/second], the joints were held stable. An example of this would be during isometric exercising, where the muscle is exerting a force, but staying in a constant position.

With the extraneous movement settled, we were able to incorporate both the HingeJoint's motor and limits to achieve Joint Position (JP) commands. This command controls each of the individual joints and moves them to specific angles. The code to rotate an individual motor can be seen in Figure 28.

```

324 void rotMotor(HingeJoint hj, float velocity, float force, bool freeSpin, float limit, bool b_JP){
325     float newAngle = limit;
326     float presentAngle = hj.angle;
327     JointMotor tempMotor = new JointMotor ();
328     tempMotor.force = force;
329     tempMotor.freeSpin = freeSpin;
330     JointLimits tempLimits = new JointLimits ();
331     float rotateDirection = (((newAngle - presentAngle) + 360f) % 360f) > 180.0f ? -1 : 1;
332     //Make the angle between -180 and +180
333     newAngle = decreaseAngleSize(newAngle);
334     presentAngle = decreaseAngleSize(presentAngle);
335     //Moves Negative
336     if (rotateDirection == 1){
337         //Makes sure min limit is larger
338         if(newAngle < presentAngle){
339             presentAngle -= 360;
340         }
341         tempLimits.max = newAngle;
342         tempLimits.min = presentAngle;
343         tempMotor.targetVelocity = velocity;
344     }
345     //Moves Positive
346     if (rotateDirection == -1){
347         //Makes sure max limit is larger
348         if(newAngle > presentAngle){
349             presentAngle += 360;
350         }
351         tempLimits.max = presentAngle;
352         tempLimits.min = newAngle;
353         tempMotor.targetVelocity = -velocity;
354     }
355     //Give limits and motor variables to HingeJoint
356     hj.limits = tempLimits;
357     hj.motor = tempMotor;
358     HingeJoint tempHJ = hj;
359     tempHJ.useMotor = true;
360     tempHJ.useLimits = true;
361     hj = tempHJ;
362 } //ROTMOTOR

```

Figure 28: Code to move individual virtual joints

Along with the code, we ran several experiments to check the virtual end-effectors position against that of the physical iArm. Table 10 shows the positions of the iArm and viArm, and the difference in magnitude.

Table 10: Data for Precision Error in Joint Position Commands

	iArm End-Effector Position [mm]			viArm End-Effector Position [mm]			Euclidean Distance
Test #	X	Y	Z	X	Y	Z	Magnitude Error [mm]
1	100.00	200.00	300.00	100.20	200.14	300.00	0.24
2	125.00	150.00	-150.00	124.98	150.03	-149.73	0.28
3	350.00	-175.00	0.00	349.98	-174.97	0.00	0.04
4	350.00	-175.00	-150.00	350.00	-175.00	-149.44	0.56
5	-100.00	100.00	125.00	-99.10	99.45	124.45	1.19
6	-150.00	350.00	-350.00	-149.86	349.37	-348.79	1.37
7	-125.00	-350.00	100.00	-125.03	-349.45	99.62	0.67
8	-200.00	-100.00	-200.00	-199.85	-100.48	-199.40	0.78
Average							0.64

Referring back to the project's engineering requirements, we desired a target error of less than 3 centimeters. From our data, it was easily shown that we meet this requirement with more precision than previously expected. The average Euclidean distance between our actual and targeted positions was 0.64 mm. The reason for the small amount of error could have been from small differences in size between the viArm CAD model and the iArm that was used for testing.

Similar to the Joint Position commands, we developed Cartesian Position (CP) commands for the viArm. The CP command takes the viArm's end-effector and moves it to a desired position and orientation linearly. Work is still being done to test position and velocity errors for this command.

Another capability that we added to our VIE was to allow I/O inputs from the keyboard. We can now execute Cartesian Velocity (CV) commands by using the directional keypad. The CV commands allow the use to move the viArm end-effector in relative directions (left, up, forward,

etc.). To move the viArm up and down, we programmed the “Page Up” and “Page Down” to execute these commands respectively.

We developed a GUI that allows the users to run Joint and Cartesian position commands. To do this, the user must click on the “JP” or “CP” button, fill out the six text boxes to specify either a joint or Cartesian position and orientation, and finally click an execute button that will emulate the command to the screen. Our current model is shown on the next page in Figure 29. Future models will show text next to the text boxes to show what to place in each box (ex. Joint 1, 2, 3, 4, 5, 6 or Cartesian X, Y, Z, Roll, Pitch, Yaw).

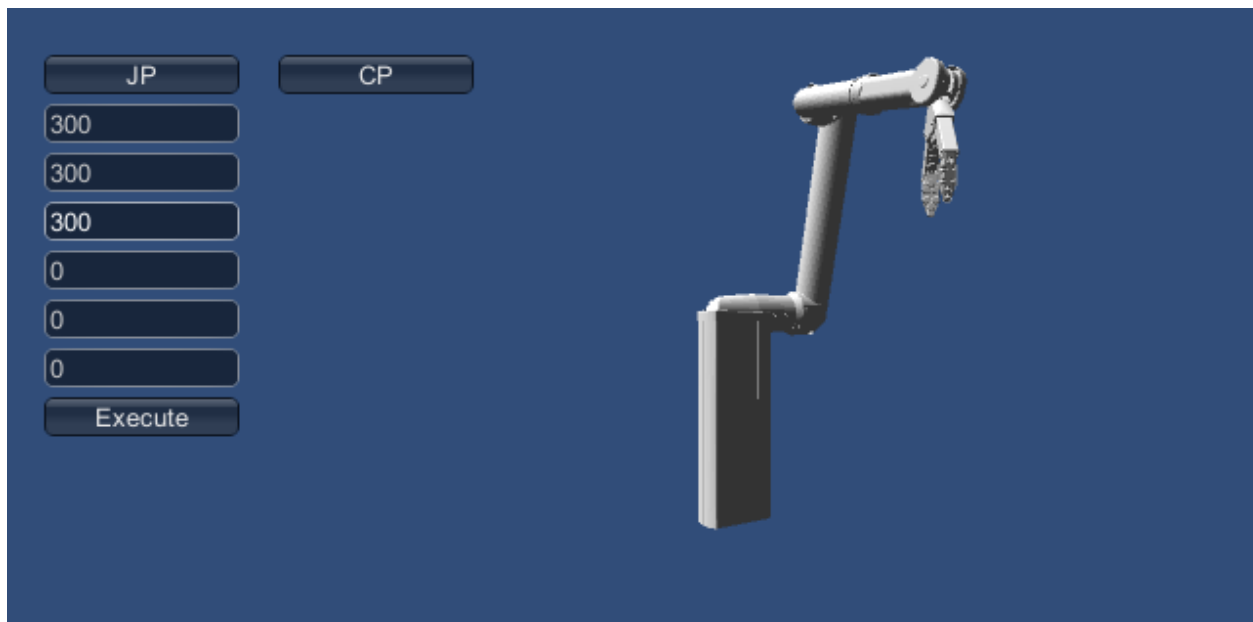


Figure 29: Current VIE Model

Currently, we have figured out the majority of the commands and are able to emulate them on the screen with the exception of the Joint Velocity (JV) command. In the upcoming weeks we need to collect more data from the iArm to receive more reliable errors and test to better understand the rotation of the real iArm’s end-effector. In addition, we will connect our VIE to the complete HARMONIE project by sharing a script that will continuously give the VIE status reports on the actual robotic arm’s position and orientation.

Project Deliverables

The deliverables for this project were a fully functional VIE for current and future iArm implementations, a speech recognition system, and a report of all design communication materials as desired by APL at the end of this project. The following is a list of these materials:

1. Virtual Integrated Environment
2. Speech/Voice Command Capability Script
3. Practical Testing of VIE with Training Time
4. Final Project Presentation
5. VIE Manual for the Developers
6. Documentation of all functions and algorithms implemented and tested

Project Management

Customer Feedback

After meeting with our customers at the conclusion of this project, they were pleased about the accuracy of our VIE to achieve such low errors for velocity and especially, position. In addition, the added speech capability has been transferred over to other aspects of the overall HARMONIE project. Future ideas for this project would be implementing “keep out” regions, collision detection and avoidance, and simulated object manipulation.

Team Composition

Table 11: Contact Information

Name	Company	Phone Number	E-Mail
Fred Tidwell	4 th	352-302-2050	m156726@usna.edu
Philip Song	22 nd	571-331-2806	m156324@usna.edu

Team Roles

- Fred Tidwell : Team Leader and TSD Liaison
- Philip Song : Purchaser, Team Safety Officer, and Design Communication Editor

Milestones

Appendix B shows a Gantt Chart for the project. The chart is in calendar timeline format starting at the beginning of the fall semester and ending on the projected completion date. Table 12 below lists significant milestone dates from the Gantt Chart.

Table 12: Milestones

Major Milestones for Prototype	Date
Implement Speech Control in HARMONIE	11/14/14
Determine Forward and Inverse Kinematic Equations	11/18/14
Implement Kinematic Equations	12/19/14
Function with and without Hardware	01/21/14
Prototype Demonstration	01/26/15
Compile Project	04/10/15
Capstone Day Presentation	04/23/15

Budget

Table 13 shows the projected cost of all components for the project. All of these items were provided by APL.

Table 13: Budget Breakdown


Item	Description	Approximate Price
Unity4 Pro	Game Engine	\$75.00 (Student fee)
Exact Dynamics' iArm	Robotic Arm	\$0.00 (Gift from vendor)
XBOX360 Kinect	Video feed with Depth Perception (Infrared)	\$110.00
Tobii X2-60	Eye-Tracker	\$0.00 (Gift from vendor)
Total Budgeted Cost		\$185.00

Appreciation of Context

The purpose of the VIE is to increase the functionality for operating a prosthetic limb. As the software continues to develop, the technology used in the limb will allow for feedback for the system, which then leads to better results for the limb. For example, when the physical iArm is run, the virtual system can be run beforehand to test the command in the environment. It can then detect and warn the physical iArm for any collisions with the surroundings or unintended movements of the iArm. Through the use of our VIE, more patients will be able to benefit from a safer, more accurate limb. This will assist them in their daily activities and allow them to return to their ordinary lives and make their lives easier.

The ethical duty of this project is ensuring the safety of the overall HARMONIE system. There is undoubtedly the question of making sure that the limb behaves as intended to and not uncontrollable. As each movement input is programmed into the system, it will be checked within the program itself to see that the limb does not move to extreme ranges, collide with the nearby environment, collide with itself, or move with too much force. With these safety checks in place, it will limit the chance for the users to cause harm to themselves as well as the people around them.

Quad Chart

	<p>Project Description:</p> <ul style="list-style-type: none"> The objective of this project is to create a Virtual Integrated Environment (VIE) with voice control that will emulate the motions and surroundings of Exact Dynamic's iArm without the use of any hardware.
<p>Project Impact:</p> <ol style="list-style-type: none"> 1. The software that will be developed will increase safety and functionality for prosthetic users. 2. The software will allow developers to debug and plan advanced trajectories. 3. The software has the capability of being implemented with different prosthetic limb systems. 	<p>Deliverables</p> <ol style="list-style-type: none"> 1. Virtual Integrated Environment 2. Speech/Voice Command Capability Script 3. Practical Testing of VIE with Training Time 4. Final Project Presentation 5. VIE Manual for the Developers 6. Documentation of all functions and algorithms implemented and tested

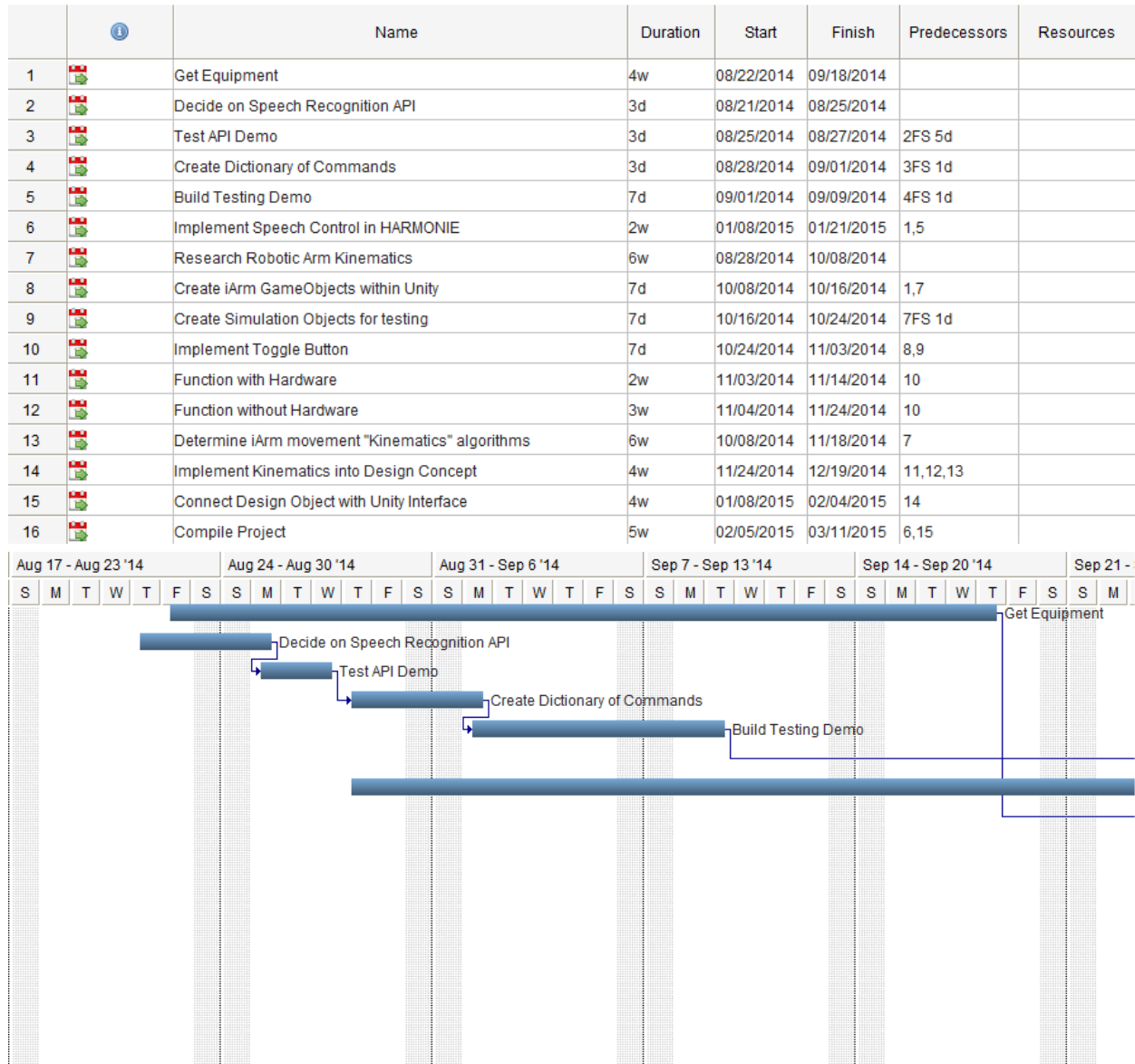
References

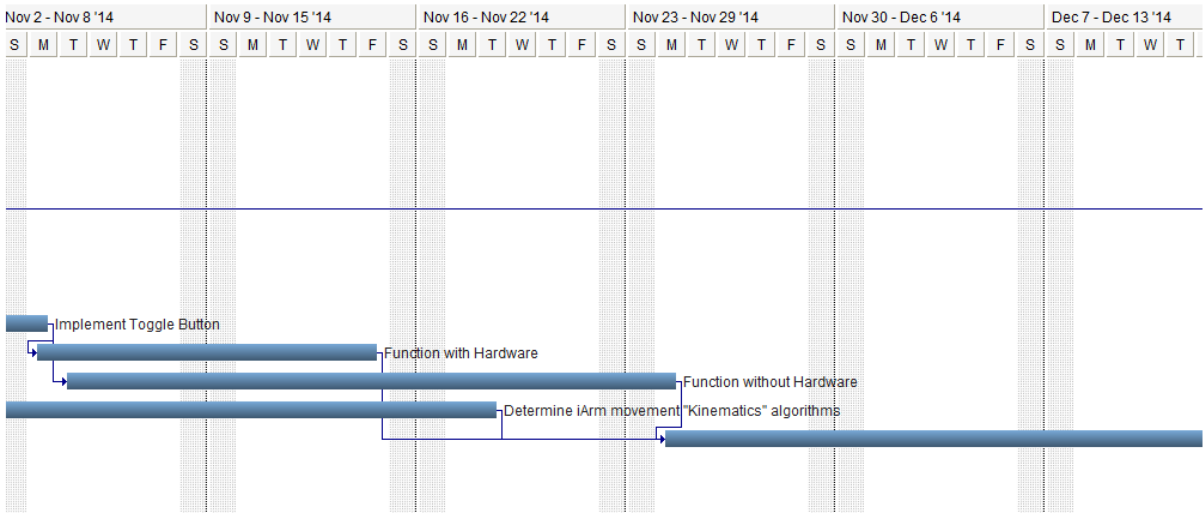
- [1] I. Bonev. (2015 January). *RoKiSim 1.7 :: Robot Kinematics Simulator* [Online]. Available <http://www.parallemic.org/RoKiSim.html>
- [2] T. Smith. *ROS.org / About ROS* [Online]. Available <http://www.ros.org/about-ros/>.
- [3] R. Armiger *et al.*, “A Real-Time Virtual Integration Environment for Neuroprosthetics and Rehabilitation,” *Johns Hopkins APL Tech Digest* vol.30, no. 3, pp. 198-206, December 2011.
- [4] K. M. Tsui and H.A. Yanco, “Simplifying Wheelchair Mounted Robotic Arm Control with a Visual Interface.” in *AAAI Spring Symp.: Multidisciplinary Collaboration for Socially Assistive Robotics*, Palo Alto, CA , 2007 pp. 97-102.
- [5] M. Pan. (2011 Nov 15). *Real time communications over UDP protocol*. [Online]. Available: <http://www.codeproject.com/Articles/275715/Real-time-communications-over-UDP-protocol-UDP-RT>

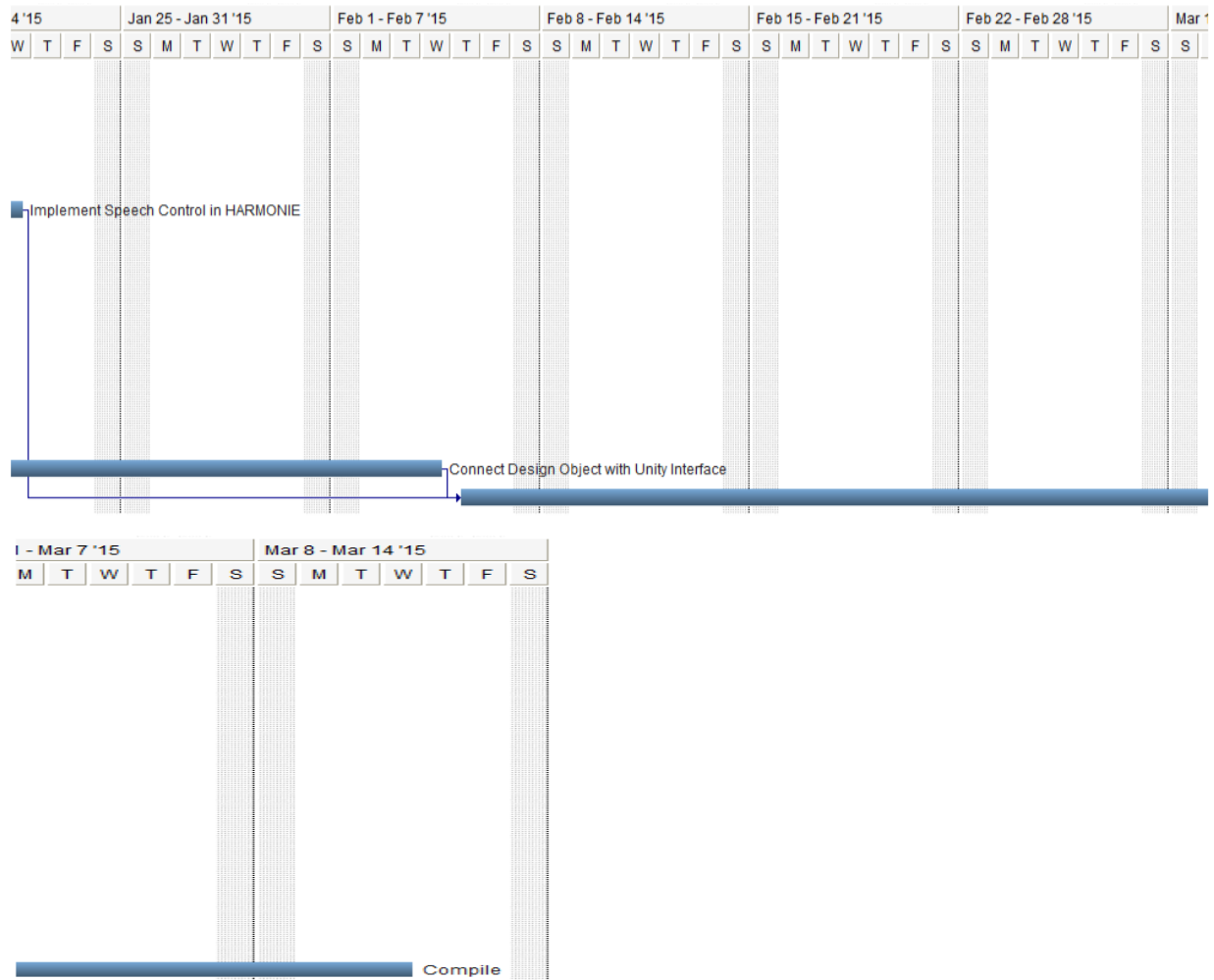
Appendix A: House of Quality

[illegible]

Appendix B: Gantt Chart



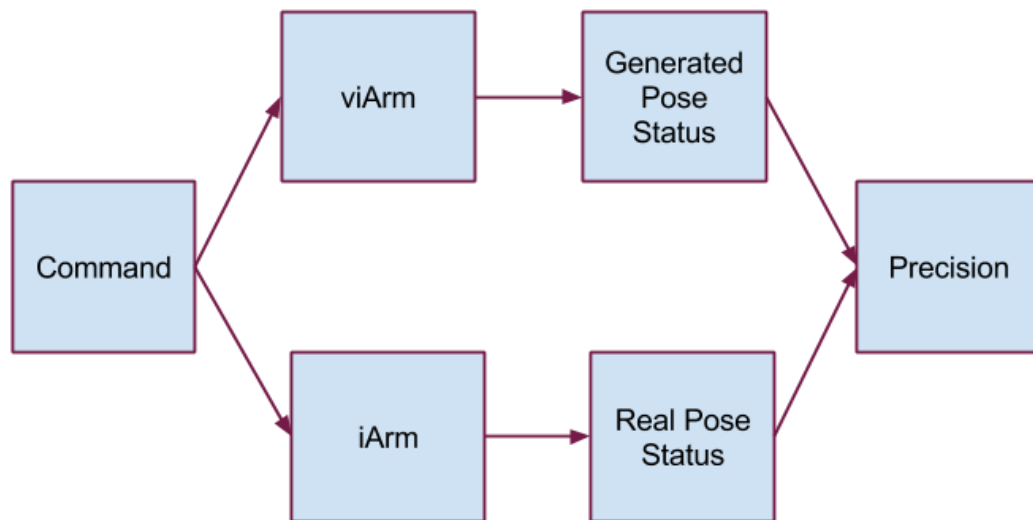




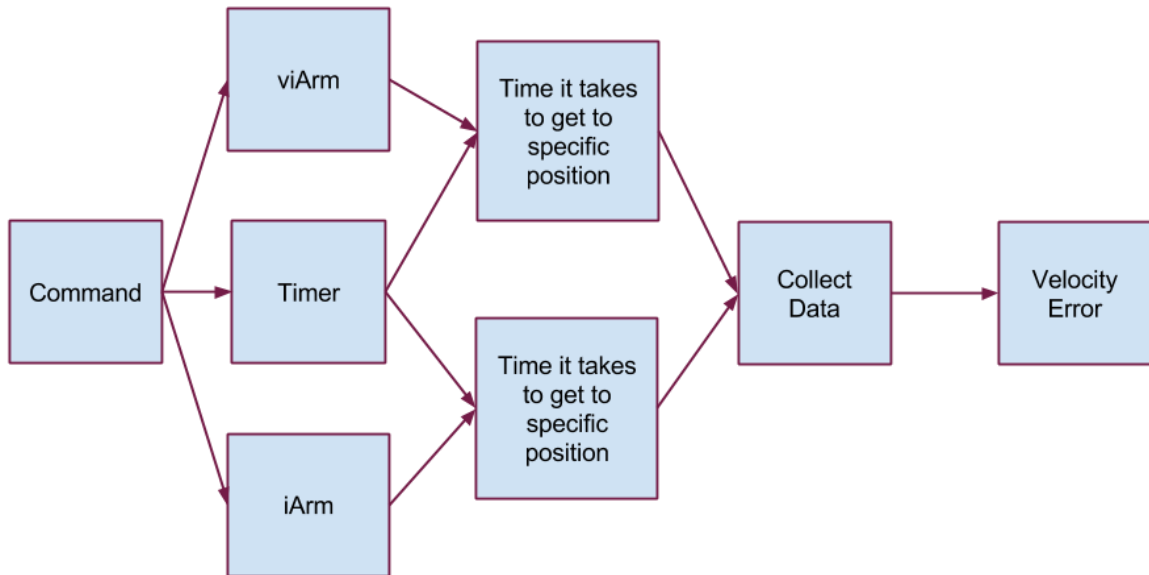
Appendix C: Prototype Test Plan

Engineering Characteristic	Target Values	Value
Precision Error (mm)	3	0.64
Velocity Error (m/s)	20	0.08
Delay (ms)	10	~0
Program Size (GB)	1.5	431 MB

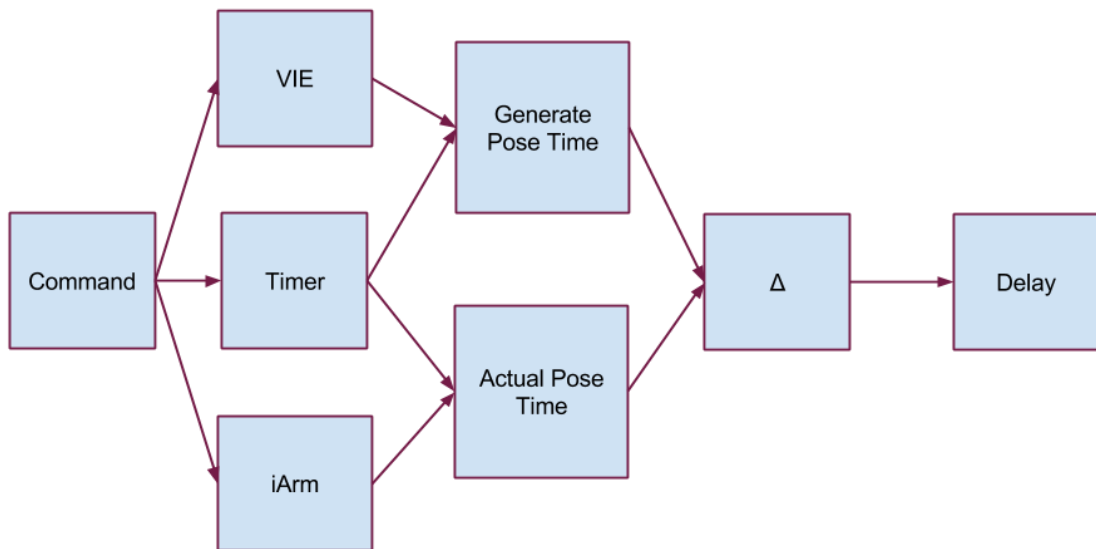
Flow Chart for Precision Error



Flow Chart for Velocity Error



Flow Chart for Delay



Appendix D: Team Charter

Team Name: WALL-E World

Logo:



Goal: To learn more about robotic communication and simulating real world applications in a virtual environment. In addition, we hope to assist in bring project HARMONIE into the consumer market.

Faculty/Staff Information

Name	Phone Number	Room	E-mail
Currie Wooten	410-293-6116	MU349	cwooten@usna.edu
Justin Blanco	410-293-6184	MU217	blanco@usna.edu

Meeting Time: Wednesday at 1430

Personal Statement:

Fred Tidwell:

I have a personal attachment to this project. Before going to the Academy, I planned on majoring in Biomedical Engineering and specializing in robotic prosthetics. Because of these past aspirations, I am finally able to apply my studies at the Academy to a field that I always wanted to be part of.

Philip Song:

I chose this project in order to have the opportunity to work with another institution outside of the Naval Academy. Being able to use the resources at Hopkins APL was an eye-opening experience. Learning how to work with a diverse group of people as part of this project will greatly benefit me in my Naval career.