Improving Tactile Feedback and Motive Efficiency of a Modular Prosthetic Limb

---

A Research Paper

Presented to the

Center for Science and Technology

Eleanor Roosevelt High School

In Partial Fulfillment

Of the Requirements for

Research Practicum

Dayann D'almeida

May 2021

ELEANOR ROOSEVELT HIGH SCHOOL APPROVAL PAGE

Improving Tactile Feedback and Motive Efficiency of a Modular Prosthetic Limb

By:

Dayann D'almeida

Has been approved on
May 14, 2021

APPROVED:

Jane Hemelt, Science and Technology Program Coordinator

Linda Watson, Computer Labs Teacher

Susan Peterson, Library Media Specialist

Rocco Mennella, Statistics Consultant

Sean Brady, Biology Research Practicum Teacher

Destinee Cooper, Physical Science Research Practicum Teacher

David Eisenberg, Engineering Research Practicum Teacher

Mark Oram, Biology Research Practicum Teacher

Sharla Peters-Smith, WISP Coordinator/Research Practicum Teacher

Laurent Rigal, Physical Science Research Practicum Teacher

Brian Stagg, Forestry Coordinator/Research Practicum Teacher

Yau-Jong Twu, Internship Coordinator/Research Practicum Teacher

ACCEPTED AND SIGNED: _____

Yau-Jong Twu,
Internship Coordinator/RP Teacher
Approving Committee Member

_____

Jane Hemelt
Science and Technology Program Coordinator
Approving Committee Member

Abstract

The primary goal of advanced prosthesis design is to provide the wearer with technology capable of matching healthy body parts in both form and function, with a state-of-the-art device being able to match the appearance of a human limb, in addition to matching the perceptive abilities of a healthy limb. This research sought to design software capable of generating perception data from a prosthesis. Using a virtual model of the Modular Prosthetic Limb, a highly advanced upper-arm prosthetic, code was written to provide the model with full movement capabilities in a virtual environment and obtain information on collisions between the limb and the objects modeled in the environment. This code was able to record instances of contact from the environment and extract force data as packets. These data packets were displayed as datasets and used in a classification-based machine learning algorithm. The five objects modeled were distinct in shape and size and were grasped by the limb in the most realistic manner possible. The data they provided differed due to processing lag of less than 20 ms. A K-Nearest Neighbors algorithm was chosen for classification due to the low hyperparameter nature of the data. Each object differed significantly in force output, showing that the code written was able to provide the limb with the means to differentiate between each object based on grip alone, meaning that the limb's physical counterpart could do the same. Using the KNN algorithm, classification of the objects yielded accuracies between 81% and 100%.

Acknowledgements

I would like to thank my mentor, Dr. Luke Osborn, for his help and presence through this research. His humor, optimism, and wisdom made our meetings something I always looked forward to and made this internship one of the most rewarding experiences I have ever been a part of. I would also like my internship coordinator, Dr. Yau Jong Twu, for always having an answer to any problems that arose. Furthermore, I would like to thank Ms. Susan Peterson, Ms. Linda Watson, and Ms. Jane Hemelt for providing me with the input and information I needed to build both my presentation and paper. In addition, I'd like to thank my friends, Minh Nguyen, Jennifer Taylor, and Jonah Valverde for answering my last-minute, late-night questions and coming to my rescue several times. Finally, I'd like to thank my mom, Ms. Marjorie Bedjamin, for fostering the curiosity that led me down this path.

Biographical Outline

Name: Dayann D'almeida

Intended College: Johns Hopkins University

Intended Major: Biomedical Engineering

Academic Achievements

- National African American Recognition, AP Scholar
- Principal's Honor Roll

Activities

- National Honor Society
- Latin Honor Society
- Key Club
- ERHS Cross Country, ERHS Track, ERHS Swimming

Work/Volunteer Experience

- Johns Hopkins Applied Physics Lab
- Intern

Advice to Class of 2021

- Don't limit yourself. If you're passionate about something, follow it and develop it as much as you can.

Table of Contents

List of Tables and Figures

Chapter One

The Problem and its Setting

Upper-limb amputations make up 70% of total amputations in the U.S., in order to restore greater Quality of Life (QOL), numerous advancements have been made within the field of prosthetic design, in order to enhance aspects such as dexterity, motion accuracy and overall proprioception. In recent years, advancement in prosthetic design and technology has allowed for the development of limbs that can provide tactile and sensory information synchronously, allowing for limbs that are able to both perceive and react to pain (Osborn, et al., 2018). This advancement, which made use of the somatosensory cortex and a neuromorphic model, brings forth the possibility for much more proprioceptive and efficient prostheses, more effectively able to behave as healthy limbs do.

This study seeks to use and modify a simulated version of the Modular Prosthetic Limb (MPL), a highly advanced prosthetic limb (Perry, et al., 2018), through a Unity-based modeling environment and the computing environment MATLAB. This is done with the intent to create a prosthetic that is more capable of recognizing and reacting to stimuli as a healthy limb would. The development of the limb focuses on the use of its many native sensors, as well as new decoding algorithms meant to create a limb that can achieve high levels of range and dexterity whilst simultaneously reacting accordingly to exteroceptive phenomena. The development of this more efficient limb could have a profound effect on the QOL of amputees and could affect robotics as a whole. Robots equipped with highly dexterous, tactile sensing limbs could have the ability to perceive

and process data much more efficiently than a human and could maneuver with much higher degrees of freedom.

Background of the Study

The usage of native sensors on prosthetics alongside neuromorphic modeling techniques such as spike rate adaptation has been done before (Osborn, et al., 2017) with success. Researchers have been able to develop limbs that respond to their environments with almost the same speed a healthy human limb does. Techniques such as high-density electrocorticography (ECoG) have also been used to restore high levels of dexterity to amputees through prosthetic limbs (Hotson, et al., 2016). The basis of the MPL allows it to achieve this high level of dexterity and range of motion naturally, making optimizing its performance the primary focus.

Being able to optimize the motion of the prosthetic, as well as its reactions to different objects is best done through object and limb modification, most of which will be achieved through different code and more specifically tailored materials. The data collected from the modified prosthetic will add to a limb which lacks highly extensive operation data.

Statement of the Problem

The MPL has a high level of dexterity and sensors that can detect various stimuli such as force, pressure, and heat. However, it has not been fully integrated with the neuromorphic model, which allows for pain detection and reaction. To remedy this lack of integration, the Unity-based virtual MPL will have to be modified to better respond to new tactile exteroceptive feedback. Through new programming, this modification would allow the most advanced prosthetic yet designed to have a new method of receiving

feedback, and could allow an upper-limb amputee to receive the dexterity benefits that the limb offers alongside fully integrated somatosensory feedback. This limb could also make use of its onboard sensors to detect and classify external stimuli from touch only. The study requires modifications to be performed on the MPL through Unity and MATLAB, which will further be tested within the Virtual Environment (VE). A K-Nearest Neighbors (KNN) algorithm would then be used in order to classify objects based on similarity and provide the limb with the ability to differentiate between them.

Engineering Goals

The primary goal of this project is to successfully modify the virtual MPL to work on a neuromorphic model, allowing in to receive tactile feedback and accurately perceive and react to various stimuli. Secondarily, the efficacy of the VE in modeling the hand and successfully employing its ability to accurately model the physics and capabilities of the limb will be integral in the limb's overall function and the accuracy of the algorithm.

Specifications

Software. MATLAB, Anaconda IDE, Spyder IDE, Windows (64-bit), JHUAPL vMPL Environment (Single, Double, and Palm Camera), Nvidia/AMD Integrated 3D Graphics

Hardware. Computer Workstation, Ethernet Switch, CPU (Intel Xeon E5-1560 or above), GPU (Nvidia Quadro K4200 or above).

Assumptions and Limitations

Assumptions.

1. MPL is operable, joints and sensors are working at optimal levels.
2. Defined user input (from an amputee) is not necessary.

3. The KNN algorithm can be integrated into the full course of programs and work simultaneously with the simulated limb.

Limitations.

1. The modified MPL cannot be used by an amputee.
2. A physical model of the MPL cannot be created to serve as motion capture data for the physics engine.

Statistical Analysis

The KNN algorithm serves as the primary vehicle for statistical analysis in this research, with the algorithm classifying quantitative values (force data from each finger of the limb) into quantitative, discrete values. The f1 scores and accuracies of the varies iterations provide the most information on the strength of the algorithm, with the f1 score representing the overall accuracy of the algorithm at identifying each class in the dataset and the accuracy representing the overall accuracy for the given iteration. The f1 score used in this research is especially crucial due to the slightly imbalanced nature of the labels, as the algorithm did not undergo stratified sampling, only standardization. All of these will be calculated via the Scikit-learn package in Python.

Summary

The physics engine and modeling platform of the JHUAPL vMPL allows for the use of a virtual version of the highly advanced MPL. This limb, which has several joints and sensors meant to replicate the high level of dexterity and tactile feedback capable in the human hand, does not contain an integrated neuromorphic system. The goal of this project is to effectively integrate a quasi-neuromorphic system when the limb, most likely through an Artificial Neural Network or Machine Learning Algorithm, which would then

allow it to not only detect external stimuli but react to it dynamically. As of now, both motion tracking and offline models of development are being proposed and optimized. The JHUAPL vMPL environment stands to be the basis by which the bulk of the project will be built upon.

Definitions of Terms and Abbreviations

1. ANN-Artificial Neural Network: Methods of computing based off biological models such as the human brain.
2. API-Application Programming Interface: Serves as an intermediary linked user code to external systems.
3. Classification Task: A type of machine learning that focuses on providing discrete values as outputs (ex. differentiating between a car and train).
4. ECoG-High Density Electrocorticography: Monitors brain activity through electrodes.
5. Hyperparameter: Factors that decide the learning process in a machine learning algorithm, such a K-Neighbors in a KNN algorithm.
6. IDE-Integrated Development Environment: A platform meant to help programmers develop software, generally comes with a debugger and editor.
7. KNN-K-Nearest Neighbors Algorithm: A supervised machine learning algorithm that can be used for both classification and regression tasks. Uses distance in order to classify objects, based on the idea that similar features lie closer together. Relies on discrete values, but trains easily and maintains high accuracy with low amounts of data and lower hyperparameter numbers.

8. MPL-Modular Prosthetic Limb: A highly advanced prosthetic limb developed by the Johns Hopkins University Applied Physics Lab, which has achieve a high level of dexterity and feedback through a mix of sensors and joints.
9. Neuromorphic: Computing and circuitry meant to emulate both the structure and operation of the human brain.
10. Precision Score: How often the algorithm is correct when it predicts a "positive result" for any object. Down a column on the confusion matrix.
11. Proprioception: Awareness and overall perception of one's own body in relation to the world.
12. Recall Score: How often the algorithm predicts correctly when it is a given object. Across a row on the confusion matrix.
13. QOL-Quality of Life: General ease of living.
14. Spike Rate Adaptation: A neuromorphic modeling technique that takes its basis from the voltage differences in membrane potentials of normal cells.
15. Upper-limb amputation: Amputation involving the area from the hand to the shoulder.
16. VE-Virtual Environment: An environment such as a physics engine in which real phenomena can be modeled virtually.

Chapter Two

The Review of Related Literature

As time passed and new technologies arose to supplement the field of biomechanics, prosthetic development shifted from focusing on simply creating systems that could work efficiently when in use to the creation of systems that could both work efficiently and deliver greater Quality of Life (QOL) to their users. This shift was spurred on by advancements in fields such as neuroscience and computer science, allowing for a synergistic mix with biomechanics that allowed for more advanced prostheses and prosthetic technologies to be designed and put into use. Taking advantage of these new advancements, the Defense Advanced Research Projects Agency (DARPA) requested several institutions to create a highly advanced prosthesis capable of matching the speed, weight, dexterity, and overall form of a healthy human limb (Burck et al., 2011). The result of the project was the peak of modern prosthetic technology, the Johns Hopkins Applied Physics Lab's Modular Prosthetic Limb (MPL), an incredibly advanced prosthesis designed to be a near perfect replication of a healthy human limb, in both form and function. The limb's wide array of sensors and motors, which combined numbered over one hundred, allowed for over 20 Degrees of Freedom, which surpassed all protheses that had been created at the time and provided it movement akin to that of healthy limbs.

The first version of the final Modular Prosthetic Limb was completed four years after the beginning of the project (Harris et al., 2011), with the technologies and programs that had been created to test and operate the limb being used by the Applied Physics Lab (APL) in other projects. The Virtual Integration Environment (VIE), which

had been used to test the MPL's capabilities in a virtual space was transitioned into being used for testing integration with users and prototyping new limbs while the physical MPL was used for further testing. As testing continued, so did advancements, leading to the creation of the e-dermis. Created by APL researcher, the e-dermis was able to be integrated into a limb, providing real-time tactile stimulus to the user through a combination of direct neural connections and myoelectric sensing (Osborn et al., 2018). While this advancement allowed for new possibilities in proprioception and overall QOL improvements, the e-dermis was used with a less advanced limb than the MPL and was only tested in specific, limited conditions, simply grasping objects designed to invoke a specific reaction through their curvature. Due to these circumstances, this incredibly advanced limb with several dozen native sensors is unable to process tactile information, meaning that the e-dermis must be added to the MPL and tested in a wide variety of situations. This research seeks to integrate the functionality of the e-dermis into the Virtual Modular Prosthetic Limb (vMPL), test its viability in a wide array of situations and expand on it with the addition of reflexive motion aided by a neuromorphic algorithm.

Prosthesis Control

Body-powered prostheses made up the majority of prostheses available before the development and improvement of most electronic methods. These prostheses connected limbs to harnesses via cables and allowed for limited proprioceptive movement (Huinink et al., 2016). In these early days of prosthesis development, body-powered prostheses were used for their low learning curve, comfort, and durability. Body-powered prostheses relied only on the physical movements of the wearer and required little upkeep and were

still widely used even when electrically controlled myoelectric prostheses entered the mainstream (Millstein et al., 1986). Early myoelectric prostheses still possessed the more human-like form factor that modern prostheses possess, but were less reliable, needing charging while being unable to perform labor-intensive tasks that could damage the limb-wide sensors that allowed them to operate. During these times, both control methods sought to provide a lightweight, durable prosthesis that could offer precise control while providing a lifelike form factor, with body-powered prostheses being very conspicuous yet allowing its user to perform a wide range of strenuous activity and myoelectric prosthesis balancing their often-heavy weight and need for constant maintenance with much greater proprioception and a form factor suited for social situations.

As myoelectric control grew out of its initial formative stages, the sensors necessary to provide movement became smaller and more precise, allowing for many of the pitfalls of the method to become less relevant. The smaller sensors and more efficient power draw of modern myoelectric prostheses left only their imprecise movements as drawbacks, with upper-arm amputations requiring incredibly precise sensors processing for the limb to accurately perceive the intended motion and execute it in a way that was natural and efficient for the wearer. Due to the difficult nature of classifying sensor data for movements in a system that was often affected heavily by the tendency of electromyographic systems to contain data completely unrelated to movement, creating a functional and reliable myoelectric limb often required large amounts of training for the user and the limb. This need for constant tests led to the adoption of several methods of motion classification, as well as virtual environments meant to help with training without incurring the stress associated with long term prosthesis use. As myoelectric protheses

became more and more advanced, so too did neural networks, leading to the rise of Brain-Machine Interfaces (BMIs) in prosthesis control. Invasive Electrocorticography (ECoG) and non-invasive Electroencephalography (EEG) allowed for the operation of this new form of prosthesis control (Fifer et al., 2014), which used the differences in voltage naturally found when neurons fire to determine movement, allowing for less substantially less noise when compared to electromyography (Hill et al., 2012) and much more precise control (Hotson, et al., 2016).

Virtual Modeling

With the advent of myoelectric prostheses and BMIs, it became more necessary for the integrated algorithms and systems present in the limbs to discriminate datasets for relevant sensor data, and users needed training sessions to accommodate their limbs with their specific movements. However, the stress associated with long periods of testing grew as limbs required more precise data, and users found themselves wearing their limbs for long periods of time. To accommodate this, researchers found ways of collecting data that would involve the user more in the process, rather than having them be idle while data was being collected (Oppenheim et al., 2010).

Some of these methods involved complete virtual modeling of the tests, with the sensors the amputee would wear only being used for mapping in a virtual environment. This made it so that the amputee was able to attempt phantom limb control and match it up to virtual movement, where the data would then be encoded and sent to a physical limb for further alteration. Researchers would be able to introduce tasks, change the control method and implement wide-scale algorithmic changes at a rate physical limbs would not be able to keep up with. Problems with myoelectric limbs such as the varying

amount of information the limb needed to process at once were resolved by implementing time-dependent neural network connections that would be able to discriminate the type of movement attempted and relay that to the virtual limb, training the physical limb to do so concurrently (Blana et al., 2016). This virtual method depended on both the virtual and physical limb being able to actuate several motors and sensors, needing them to be able to record accurate data and accomplish the complicated functions necessary for proprioceptive operation, with an entire virtual operation program being developed by Kumar and Todorov (2015). In operation, it allowed for accurate testing and alteration, allowing the limb to respond to the user's movement more accurately while simultaneously giving the user experience on the specific control scheme they would be using (Perry et al., 2018) in their Activities of Daily Life (ADL).

Motion Classification

Tol offset the imprecise nature of myoelectric control, neural networks and neuromorphic models began to be heavily used to classify movement and allow amputees to operate more responsive and dexterous prostheses. The Linear Discriminant Analysis (LDA) classifier is the most commonly used method of motion classification in myoelectric and Brain-Machine prostheses. This classification method depends on the use of neuromorphic pattern-recognition based algorithms, meant to be used in both supervised and unsupervised learning. The LDA is useful in that it is capable of dimensionality reduction, essentially reducing a large number of inputs repeatedly until they are able to consistently return corresponding movement outputs. While powerful in its ability to separate movement classes from a wide array of inputs, it is when combined with an Artificial Neural Network (ANN) that its full capabilities are shown, with

movement classes being repeatedly obtained from the LDA before being processed by the neural algorithm (Prahm et al., 2016). In the case of LDA classification, the pattern-recognition capabilities of the system are used, with the output being found by the algorithm and trained by error, classifying data with increasing accuracy, reaching p values of less than 0.05, until it is effectively capable of predicting movement from simple sensor data. This method proved to be incredibly efficient when dealing with myoelectric prostheses and neuroprostheses, as the higher number of degrees of freedom possessed by modern prostheses would be able to be used, as individual motors would actuate depending on the predicted data.

While the LDA classifier was the most widely used classifier based on ANNs, other methods of classification were used, mostly focusing on reducing the variance found in the LDA and achieving highly accurate classification in a shorter period. The pattern recognition method combined with linearization allowed for limbs to effectively control multiple degrees of freedom simultaneously, actuating several aspects of the limb at once and possessing some problems with variance in position. Using several degrees of freedom led to more complex arm positions and issues with the classification algorithm, which led to inaccurate motion predictions while in use. Several forms of positional classification were attempted in order to resolve this issue and resulted in various levels of success (Geng et al., 2017). Due to the inconclusive results of these new methods, LDA has been implemented to resolve the noise and extraneous predictions created by the position variance (Betthauser et al., 2017) using cluster-based unsupervised learning, providing the algorithms non-ideal data without inputs, separating it into clusters that the algorithm can then utilize to make predictions.

Prosthetic Kinematics

The MPL was unique in its ability to simultaneously operate more Degrees of Freedom (DOFs) than any other limb, with the vMPL possessing 27 total. These degrees of freedom were not limited to prosthetic movement, but also formed an integral part in the motion of robotic limbs when completing tasks. While operating in task-space, a robot or robotic appendage that can make use of a higher number of DOFs is more efficient in completing its tasks than one who cannot operate in several dimensions (Menon et al., 2014). When designing prosthetics, this same principle applies to the joints that the limbs have, with the number of effective joints corresponding to the DOFs that the limb possesses. Force data, angular velocity, and other physics-based data were essential in demonstrating that the MPL could replicate a healthy limb, but without effective joint mapping and control capable of integrating into a sensor array (Hunt et al., 2018), the limb would be unable to move with the freedom it currently possesses.

Tactile Neuromorphic Integration

Neuromorphic Computing. Neuromorphic Computing and neural networks were adopted by the field of prosthesis development as they served as a great aid to the development of prosthetics that better mirrored healthy limbs. Due to the rise of BMIs in prosthetic control, the integration of algorithms capable of accurately replicating the neuronal and synaptic behavior of the human brain (Upadhyay et al., 2019) when classifying and predicting motion has been explored as it could allow for better wholesale integration into users' bodies. These technologies, alongside ANNs, allow for less power consumption than conventional machine learning algorithms and process in a way that

heavily emphasizes adaptability, mirroring the pattern recognition and motion classification needs that mandated the widespread use of the LDA classifier.

Analogous neuromorphic techniques to ANNs such as Spiking Neuron Networks (SNNs) were also used to improve neuromorphic computing, based on the biological membrane voltage differences that occur when a neuron normally fires. These abstract models, when integrated into the noisy data associated with prosthesis control, have the capability to rapidly adapt and reconfigure their processes to best respond to variance.

Tactile Encoding. Although neuromorphic computing was first conceptualized for use in general computing paradigms, neuromorphic techniques such as neuron spiking expanded and grew into the development of new tactile feedback techniques. Recent studies have utilized neuromorphic algorithms, classification algorithms and spiking neurons simultaneously to detect and classify texture data (Iskarous et al., 2018), using these techniques to highlight the system's high adaptability and push it to its extremes.

These new technologies and those such as the e-dermis were developed within similar time frames, and both sought to provide tactile feedback to prostheses. The use of the same technique to achieve similar goals shows that, when considering the need for tactile feedback, the neuromorphic model was better suited to providing direct feedback to users.

Summary

Many of the aforementioned capabilities and elements necessary to the operation of the limb are integrated aspects of the MPL, such as the sensors and motors, as well as the different methods of control being simulated via the miniVIE. The use of a Virtual Integration Environment does not eliminate these aspects from consideration, simply

streamlining them and allowing them to be viewed as data rather than being extensively developed and personalized for use by a singular user. The unique aspect of the research is the incorporation of tactile encoding techniques into the operation of the virtual limb. This would require a specific algorithm to be created and formatted in a way that the miniVIE would recognize while also being programmed to replicate the reflexive tactile abilities of neuron spiking models. If successful, through the union of these various techniques in a highly modular environment, a virtual prototype of the MPL could be created, capable of recognizing and reacting to tactile stimuli and allowing for immense proprioceptive increases to its users.

Chapter Three

Method

The physical MPL underwent several tests meant to ensure its efficiency and means of operation with respect to DARPA's specifications. The vMPL has not been used for offline or online testing in the same capacity as the physical limb, and it first needs a control method that will allow it to respond to commands in the same way the physical limb does before automation or physics modeling is introduced. The novelty of creating a limb capable of reacting to stimuli virtually is what defines the steps that will be taken to develop the project, mainly finding a way to connect the model of the vMPL and the environment it operates in with the miniVIE, and to provide a way for code written through MATLAB and Python to modify the limb in real time. In order to accomplish this, the preliminary steps taken in the project must be completely focused on back-end work, that being the creation of various connection methods capable of linking all of these disparate elements.

The structure of the project exhibited a clear shift between working with modeling and connecting different programs to creating a full algorithm meant to analyze and identify contact data and match it to different objects, mainly through the change from MATLAB to Anaconda and Python. Through MATLAB, the miniVIE was able to serve as a virtual repository from which all the dependent functions were found and added to scripts to perform highly specific tasks. Python was used due to its ability to process high amounts of data in a highly efficient manner and the need for an algorithm to be developed that could classify objects based only on sensor data. In this sense, both

programming tools served equally invaluable roles in creating a fully-modeled vMPL capable of automatically detecting and reacting to different objects in its environment.

Instrumentation

I. Computer

a. Hardware

i. All files and code run on an APL-provided Dell Latitude 14 7000 Series E7450 with an Intel i7-5600U CPU.

b. General Software

i. Windows 10 Enterprise

ii. VulcanX v1.4.8.0 (Nodes communicate between programming language and vMPL as well as where percept data is collected)

iii. miniVIE (serves as version of Virtual Integration Environment, used to collect and process sensor data. Contains 24 main folders.)

c. Software for Modeling and Control

i. vMPL Environment (only executables, no Data folders)

1. vMPLShapeFeeling-PalmView
2. vMPLShapeFeeling-Backview
3. JHUAPL vMPL (singles camera)

ii. MATLAB R2020a (main modeling and control scripts/functions only)

1. move_vMPL_example.m (contains the most basic program responsible for creating the command encoder needed to

convert joint angles to movement as well as creating the UDP port that allows for TCP/IP communication.

2. vMPLPlayground.m (script uses world interface object to allow objects to work with vMPL, using packets in conjunction with the normal vMPL percepts to model object, object collisions, and manipulate their appearance)
3. Various dependencies: WifCommandEncoder(create WIF messages), VulcanXsink(uses pnet to open udp ports, send joint data and convert to bytes), MudCommandEncoder_Pitt(meant to create MUD messages, 772 lines for finger movements), Enum Finger/Arm (finger and joint control list), and DataSink.

d. Software for Sensor Data and Algorithms.

i. MATLAB R2020a (main sensor scripts only)

1. example_sensor_read.m (uses the wif commander and VulcanX percept capabilities to get force data and plot it.)
2. extract_sensor_data.m (obtains sensor data from the vMPL, can be modified to perceive more data such as torque, acceleration, etc..)

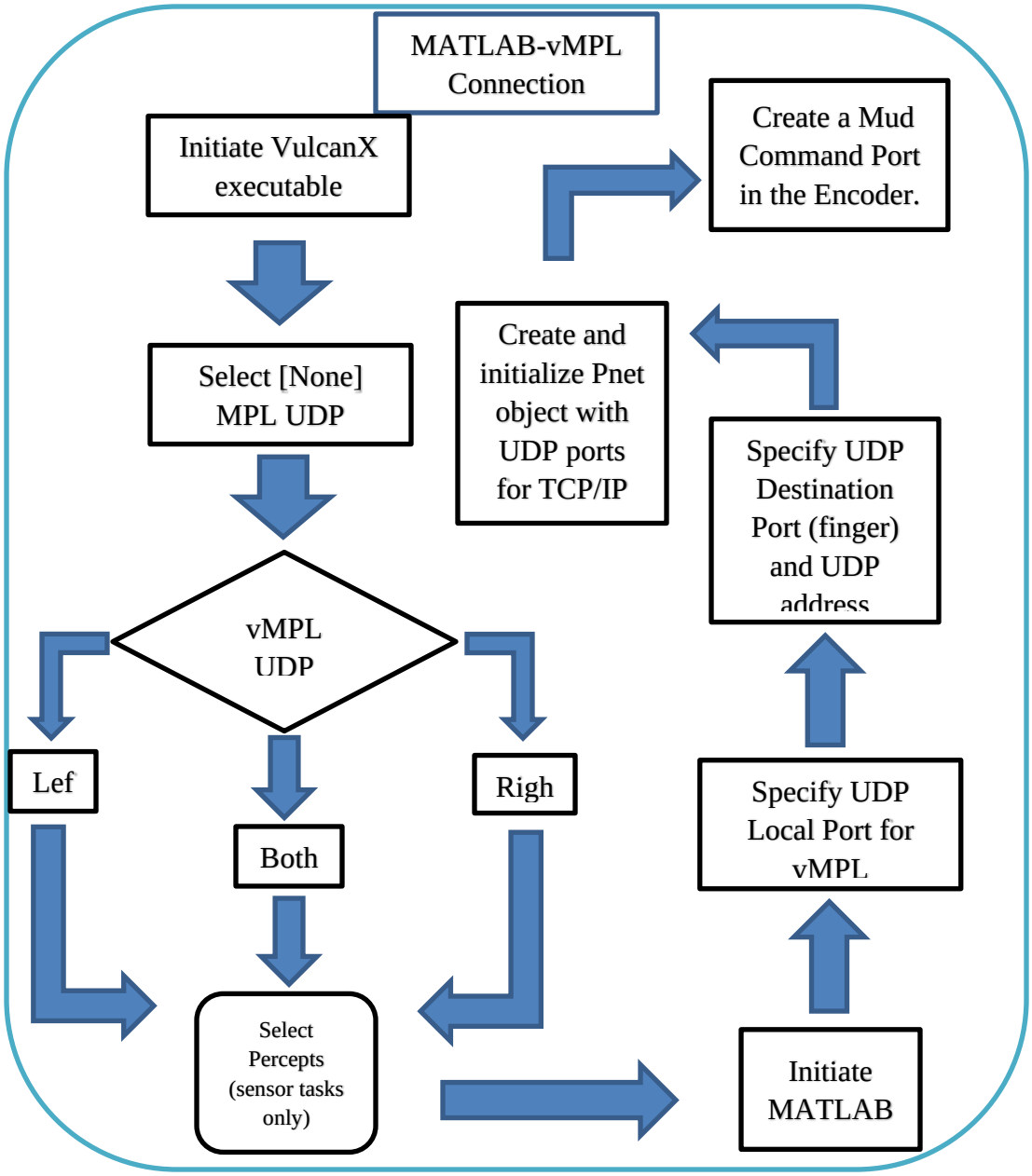
ii. Anaconda IDE

1. Spyder 4.1.4
    a. classification_algorithm_iterations (KNN algorithm separated into x and y before being trained)

Procedures

1. All research was conducted virtually in coordination with the JHUAPL ASPIRE program.
2. Anaconda IDE downloaded with Spyder install for future algorithm generation and data analysis.
3. VulcanX and vMPL environment added to general Box Folder along with MATLAB files, combining all necessary files in one virtual file.

Figure 3.1: UDP Connection



4. Began working with the system by controlling an entire arm (right and left, never simultaneously) in its environment. First script controlled the entire arm and fingers.
    a. Made use of every joint and finger DOF available to the limb, to make sure they were able to perform the full range afforded by human motion. Compared normal motion and attempted to replicate with limb.
    b. Wrote a while-loop capable of evaluating the hand's position using joint byte data, creating a perpetually moving limb. Stopped with use of tic and toc functions.
5. Controlled the arm with a more streamlined MATLAB script, capable of moving several limbs to the same angle with one line of code (fundamental connections still present) as well as timing grasps and modifying objects.
    a. Utilized both the streamlined and original methods of grasp several WIF objects such as spheres and cylinders, as well as more abstract shapes such as oblate spheroids.
    b. Added to the script the ability to copy the finger flexion call with the individual joint call code. Tested with the THUMP_DIP (Distal Interphalangeal).
6. Used another MATLAB script to extract sensor data from the modeled force sensors in the limb. Included a line of code capable of iterating the script 5 times, saving the raw data and images of the graphs generated.

   a. Saved this data as .mat files and matched it up with the graphs generated.

7. Used Python in order to generate a KNN-algorithm and classify the data obtained from MATLAB. Maintained 40% test data and 60% train data for each iteration, with the datasets from the window of 50-100 being chosen (50 Hz = 1-1.5 secs).
   a. Generated a classification report for each iteration of the algorithm, with each object being measured on precision, recall, an f-1 score, and support. This report also displayed the accuracy of the algorithm as a percentage. over each iteration.
   b. Generated a confusion matrix for each iteration of the algorithm, where the number of True Positives, True Negatives, False Positives and False Negatives were mapped.
   c. Generated a classification report for the algorithm, showing, amongst other values, precision, modeled by the equation.

$$\frac{TP}{TP + FP} = \frac{sucessful\ classifications}{all\ classifications}$$

   and recall, which could be modeled by the equation.

$$\frac{TP}{TP + FN} = \frac{successful\ classifications}{all\ posiible\ sucessful\ classifications}$$

Solutions Selection

The vMPL is used to due to its high level of control in relation to DOFs and overall consistent integration within MATLAB. To be able to collect force data from MATLAB, several of the previous methods needed to be combined and streamlined, which created a large number of dependency conflicts, especially due to the continued use of the more streamlined controller capable of interfacing with the WIF and modifying

objects. Despite this, it was chosen since it could collect the percept data in Packets already monitored by VulcanX and make use of MATLAB's native plotting capabilities to move the limb, modify objects, and obtain graphs in an incredibly short amount of time. A K-Nearest Neighbors algorithm was chosen due to its ability to randomize and train sets of data with very little time and overall lag, and due to the fact that the "neuron" spike effect upon contact with the objects only lasted for .5 seconds, a set of time that could easily be monitored through the KNN. Both a DTW and Euclidean distance method could be used, but a Euclidean distance metric was ultimately chosen in order to make use of the dataset's potential for clustering.

Data Analysis and Evaluation

Graphs plotted with the MATLAB script appear as graphs of every finger involved, with the time serving as the x-axis and the observed sensorValue serving as the y-axis. These observed values can range from torque to temperature, and the differences serve as the data that will be provided to the Python-based pattern recognition algorithm, where it will analyze the differences between observed sensorValue and the individual fingers present to identify what object the limb is grasping, providing an exteroceptive method of interaction to the limb. Any significant statistical analysis will require the testing of several groups (more than two) in order to determine whether there is any significant difference between them, however, any two data which provide similar sensorValue graphs could be analyzed with an ANOVA test. The KNN algorithm is designed to classify data in discrete values based on similarity, so most in-depth data analysis will be conducted by the y_pred = classifier.predict(X_test) function on line 34. The data contained in the confusion matrix can be classified as the following: a True

Positive (TP), where the algorithmic model predicts force data as corresponding to one of the 5 objects and does so correctly, a True Negative (TN), where the algorithm predicts force data doesn't correspond to a specific object and does so correctly, a False Positive (FP) also known as a *Type 1 errror*, where the model predicts force data as matching a spefic object when it fact it does not, and a False Negative (FN) or *Type II error*, where the model predicts force data as not matching a specific object when it in fact does.

Summary

The primary goal of this research is to develop a limb capable of recognizing the shapes of objects in its environment and reacting to them appropriately. However, the relative novelty of the means of control and the complexity of the vMPL model required the development of software and a full system capable of controlling the limb and modeling its interactions in an environment. This required an alteration of the miniVIE MATLAB repository and TCP/IP connections through the VulcanX program, as well as several files of new code meant to streamline controlling the limb. The research now moves onto the collection of data from various objects before generating a KNN algorithm through the Anaconda IDE and Spyder.

Chapter Four

The Findings

The preliminary research into the vMPL environment consisted of two distinct phases. The first was meant to prove that precise, remote control of the limb as possible, and the second was meant to model various objects and gather force and torque data from them. After the first phase had been completed and the virtual environment's limitations had been found, the focus shifted to creating the most varied and complex objects capable in order to train the algorithm to better recognize what it was interacting with. There were 4 basic objects modeled first, with more complex objects needed more precise control within the inherent programming limitations of the Unity-based vMPL environment. These first 4 objects demonstrated consistent grasping using only one of the aforementioned control mechanisms (timed/constant), while the more complex objects, such as the oblate spheroid and pyramid, required a mix of control mechanisms and a separate script in order to grasp correctly. However, the shift between control mechanisms also changed the consistent timing of the timed code, changing the consistent 50Hz sampling rate of the sensors and making them remain active for as long as the data buffer remained.

However, due to the fact that the overall goal of the research was to train an artificial intelligence (A.I.) with a data-classification algorithm, such as a Linear Discriminant Analysis (LDA), the inconsistency of the varying grasp types served as more of a positive, as a consistent “time” x-axis could serve as a pattern by which the algorithm would base all its predictions, requiring a level of natural inconsistency in order to train it to respond solely to the shape of the graph. Therefore, two datasets were

taken for the simple objects, and the more complex objects would be recorded with an inconsistent “time” axis. Due to the algorithm used being a K-Nearest Neighbors classifier, and therefore not being time dependent, a Scikit-learn scaler.transform was utilized to fit the training and test data to a standard, Normal form before the classifier was used.

Data

The primary medium through which this preliminary phase of the research was conducted was MATLAB. After control was established through the use of a designated Universal Datagram Protocol (UDP) port, the joint data was converted into bytes through an encoder. At the same time, the objects were modeled with a World Interface (WIF) sub-program. The vMPL was able to mirror the sensors of the MPL, save for the temperature, and was, therefore, able to detect force and torque natively when coming into contact with objects in the environment. The data was held in a data buffer with a 50Hz sampling rate, and the percept packets were matched with the approxTime sub-function.

The program as a whole was iterated 5 times using a for-loop, with each iteration providing a distinct set of data values corresponding to the object the hand was grasping. The script automatically saved a picture of the resultant graph to provide a visual representation of the dataset. Due to the general processing variability inherent in MATLAB’s software, Tables 4.1 and 4.2 possess a different number of data points, which proves to be inconsequential to the algorithm's overall design due to it being programmed to only detect shape. The time was normalized using the MATLAB function linspace, with the full command calling arguments linspace(x1,x2,n) with the exact

spacing being measured by the equation and forming a consistent x-axis alongside the 7 seconds outlined in the approxTime sub-function. Both the percept values ftsnForce and position were recorded, although only ftsnForce was used, as it relied directly on sensor information from the limb. For the algorithm, all data collected over the 5 iterations was saved in a .mat file before being converted into a larger .csv file. Every object had its data from point 100-150 (1-1.5 seconds or the moment of the initial "spike") collected and placed into the larger csv file, adding up to 255 datapoints, with one .csv file containing the full datapoints for Cylinder 1-Iteration 3, Cylinder 2-Iteration-2, Cylinder-Iteration 5, Pyramid-Iteration 4, and Sphere-Iteration-1. The data points for each finger were recorded, alongside the labels for each. This data was then split, with 60% serving as training data and 40% serving as testing data. The data was then standardized, removing the mean and scaling to unit variance through the formula

$$z = (x - u)s$$

where u was the mean of the training samples and s was the standard deviation. These values were stored within the X_train and X_test variables, which held the finger force data. Standardization was required in order to reduce any possible bad behavior within the machine learning algorithm by altering the data to appear Normally distributed. Due to the sheer number of data points resulting from a 50Hz sampling rate, all other iterations are displayed in the appendix of this paper, as well as the source code for the modeling of the limb and objects, alongside the data of the two disk-shaped cylindrical objects and the pyramid. The confusion matrix, classification report, code, .csv file, and shown represent the full datapoint, "combined" dataset, with all other iterations being displayed in the appendix of this paper.

Combined KNN Dataset

LITTLE	RING	MIDDLE	INDEX	THUMB	CLASS	
0	0	10.68878	0	0	Sphere	
0	0	19.15576	0	0	Sphere	
0	0	24.59873	0	0	Sphere	
0	0	24.59873	0	0	Sphere	
0	0	24.49026	0	0	Sphere	
0	25.1868	24.56696	0	0	Sphere	
0	29.93056	24.48956	25.34046	0	Sphere	
0	29.225	24.34087	29.60206	0	Sphere	
0	29.225	24.34087	29.60206	0	Sphere	
19.49917	29.07267	24.29094	29.27452	0	Sphere	
30.94319	28.95115	24.31042	29.0697	0	Sphere	
34.90309	29.11067	24.31824	28.93043	0	Sphere	
34.59429	28.81735	24.12446	28.73343	0	Sphere	
34.43597	28.63647	24.01661	28.61123	0	Sphere	
33.8797	28.48746	23.97804	28.50362	0	Sphere	
33.80809	28.41378	24.01906	28.44665	0	Sphere	
33.76525	28.37431	24.06669	28.41605	0	Sphere	

Table 4.1: ftsnForce values for KNN algorithm, data from row 1-28 displayed out of 1100, showing detected percept packets of arbitrary force value.

Code for Combined KNN Dataset

```
# -*- coding: utf-8 -*-
"""
Created on Sat May  1 19:08:09 2021

@author: kouakda1
"""

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import accuracy_score
import itertools

dataset = pd.read_csv(r'C:\Users\kouakda1\Box\2021_virtual_prosthesis_control\documents\machine_learning_set_all.csv

X = dataset.iloc[:, 0:4].values
y = dataset.iloc[:, 5].values

labels = ["Cyl1", "Cyl2", "Cylinder", "Pyramid", "Sphere"]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 8)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
```

Figure 4.1: Code from lines 1-34 of KNN Algorithm, modules for object manipulation and file editing, value and label separation, standardization with scaler meant to standardize order of magnitude in variance, classifier variable.

```

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
result = confusion_matrix(y_test, y_pred, labels=labels)
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_test, y_pred)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_test,y_pred)
print("Accuracy:",result2)

cmap=plt.cm.Oranges
plt.imshow(result, interpolation='nearest', cmap=cmap)
plt.title('Confusion matrix - number of predictions')
tick_marks = np.arange(len(labels))
plt.xticks(tick_marks, labels, rotation=45)
plt.yticks(tick_marks, labels)

thresh = result.max() / 2.
for i, j in itertools.product(range(result.shape[0]), range(result.shape[1])):
    plt.text(j, i, format(result[i, j], '.2f'),
        horizontalalignment="center",
        color="white" if result[i, j] > thresh else "black"
    )

plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.tight_layout()
```

Figure 4.2: Code from lines 35-63 of KNN Algorithm, confusion matrix generation and formatting, accuracy based on true labels and predicted labels, color generation script to distinguish positive classifications.

Confusion Matrix: Combined Dataset

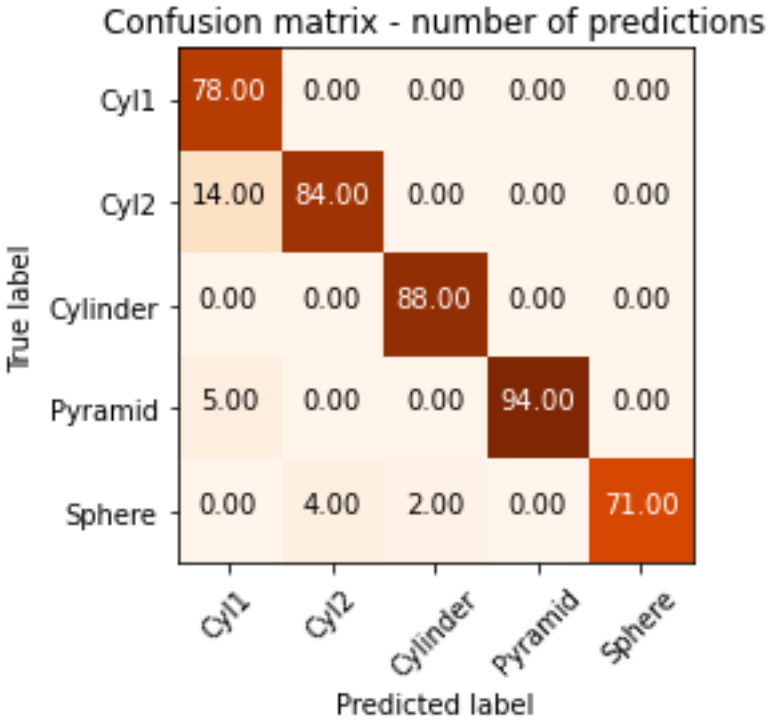


Figure 4.3: Confusion Matrix for combined dataset, with labels predicted by the algorithm on the x-axis and the true labels on the y-axis.

Classification Report: Combined Dataset

```
Classification Report:
              precision    recall  f1-score   support

        Cyl1       0.80      1.00      0.89        78
        Cyl2       0.95      0.86      0.90        98
    Cylinder       0.98      1.00      0.99        88
     Pyramid       1.00      0.95      0.97        99
      Sphere       1.00      0.92      0.96        77

    accuracy                           0.94       440
   macro avg       0.95      0.95      0.94       440
weighted avg       0.95      0.94      0.94       440

Accuracy: 0.9431818181818182
```

Figure 4.4: Classification report on combined dataset, showing precision and recall values for each object, as well as an f1 score and total accuracy. Also collects macro average (average of precision, recall, and f1 without weighting for the unbalanced distribution of values) and weighted average (average of precision, recall, and f1 with weighting for the unbalanced distribution of values).

Data Analysis

The differences between the datasets appeared more egregious than they truly were. When only based on the data between the sphere and the cylinder, there was a 44-point gap between the total amount of force sensor values measured (329-285); however, after accounting for how much of the time was spent in contact with the objects, the gap closed to only 14 points of data. This is most likely due to the variation in MATLAB's processing speed and the need to clear the buffer before recording new percept data. The 50Hz sampling rate meant that 50 data points were measured per second, making the gap in data representative of a 0.28s delay in capture between the objects.

The two objects were gripped in a very similar manner, as shown by Figures 4.4 and 4.5, showing a consistent lack of force data on the thumb. The movements outlined were precisely the same, with only the angle of grip being changed between the two. This constant zero is not implicative of a lack of sensory data capture from the thumb sensor but shows how the vMPL environment and the limb itself perceives force data. Constant forces are marked as zero, with limbs having noticeable collisions marked as increases or decreases. The graphs of the cylindrical disks corroborated this claim, as the plotting script did not recognize fingers that did not come into contact with the object. This constant occurrence indicated that the native graphing capabilities of MATLAB were able to consistently provide an accurate sensor reading of the thumb as serving mainly as support. The choice to shift to running the script as a large iterable was motivated by the algorithm's need to possess a large dataset from which it could make comparisons. To do this, the code was modified with a simple for-loop, which repeated the loop until 5 distinct sensor values were found, after which the graphs were built off. The main error resulting from this was that the program could not run a function that contained a way through which the data buffer could be cleared, allowing for new percept packets to fill it. This logic error was resolved by calling the function alongside the index before running in the script in the loop, allowing it to be "manually" reset.

The algorithm portrayed a completely different image than what would be normally expected from the grasp task. Despite possessing identical grasp angles, the cylinder and sphere possessed precision, recall and f1 scores all above 90%, with the cylinder possessing 0 False Negative cases and never being misidentified and the algorithm only mistaking the sphere for the cylinder twice. The precision score for the

index disk “Cyl1”, showed that the algorithm possessed the most False Positives for the object, mistaking it for the middle disk “Cyl2” 14 times and the Pyramid 5 times. The f1 score, the harmonic weighted mean of the objects’ precision and recall scores, used to obtain the accuracy of the model for each object and modeled by the equation

$$f1 = 2 \times \frac{precision \times recall}{precision + recal}$$

showed that the algorithm was only 89% accurate when classifying the index cylinder and 90% accurate when classifying the middle cylinder compared to a 96% and 99% accuracy when classifying the sphere and cylinder, respectively. The algorithm for the combined dataset possessed an accuracy of 94.3% , modeled by the equation.

$$\frac{TP + TN}{TP + FP + FN + TN}$$

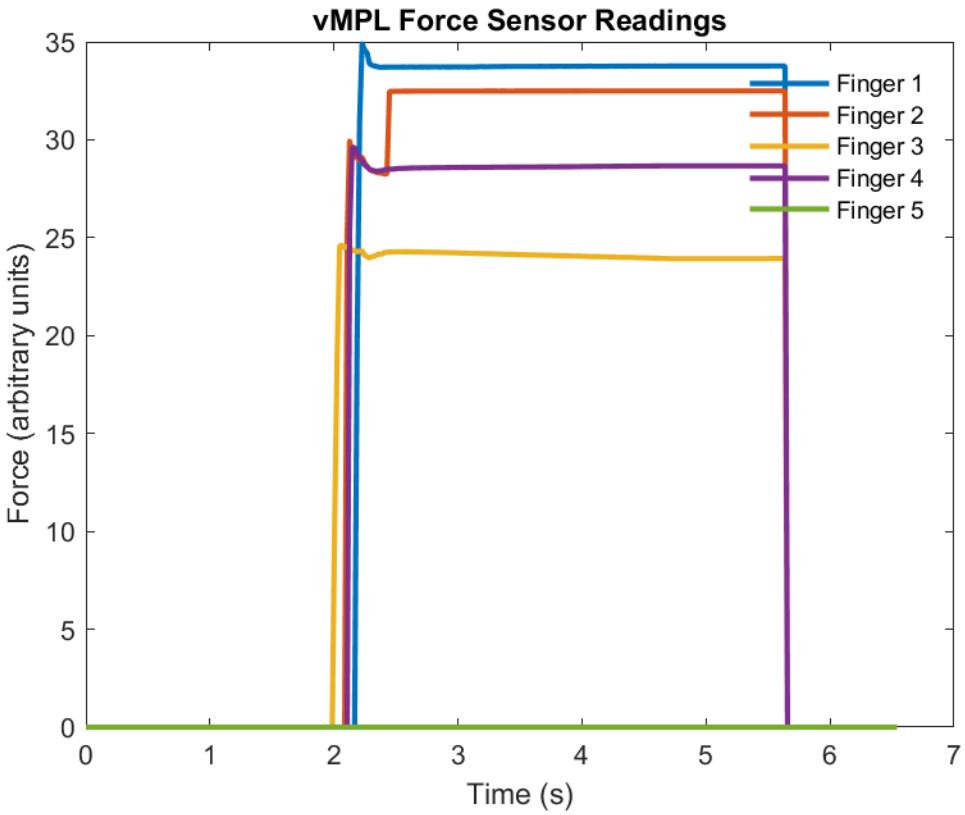


Figure 4.5: vMPL Force Sensor Readings of ftsnForce for Sphere, 1st Iteration, 2-second move-in time, 3-second hold, 2-second move-out time.

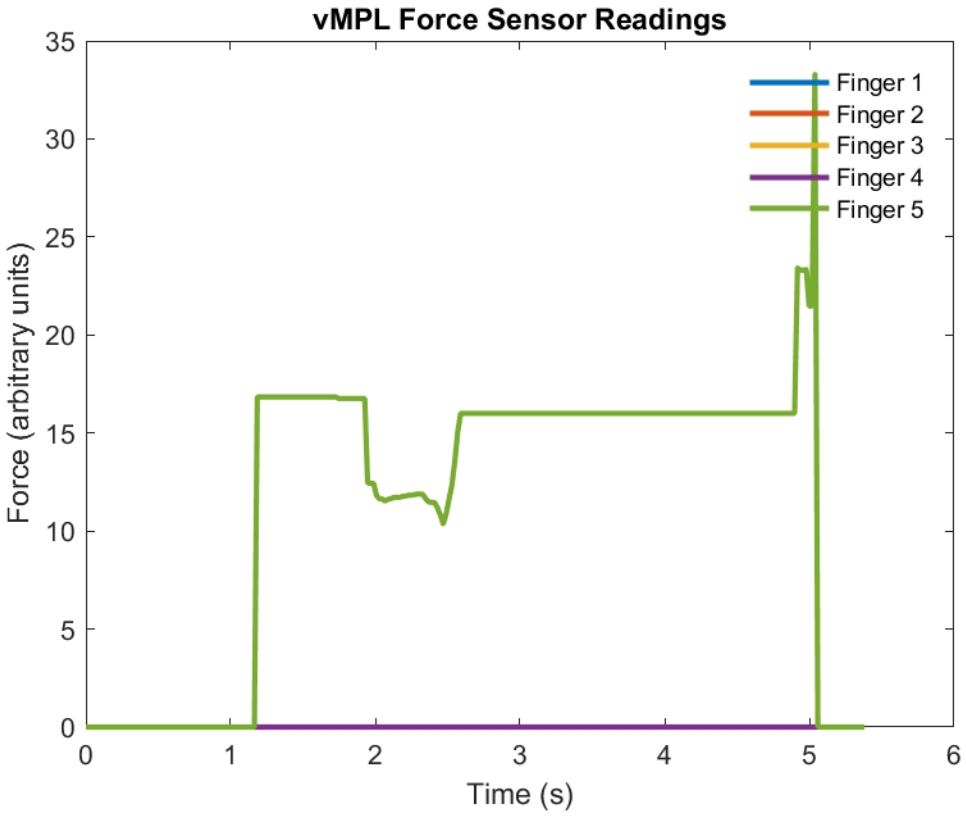


Figure 4.6: vMPL Force Sensor Readings of ftsnForce for Index Cylinder, 3rd Iteration, 2-second move-in time, 3-second hold, 2-second move-out time.

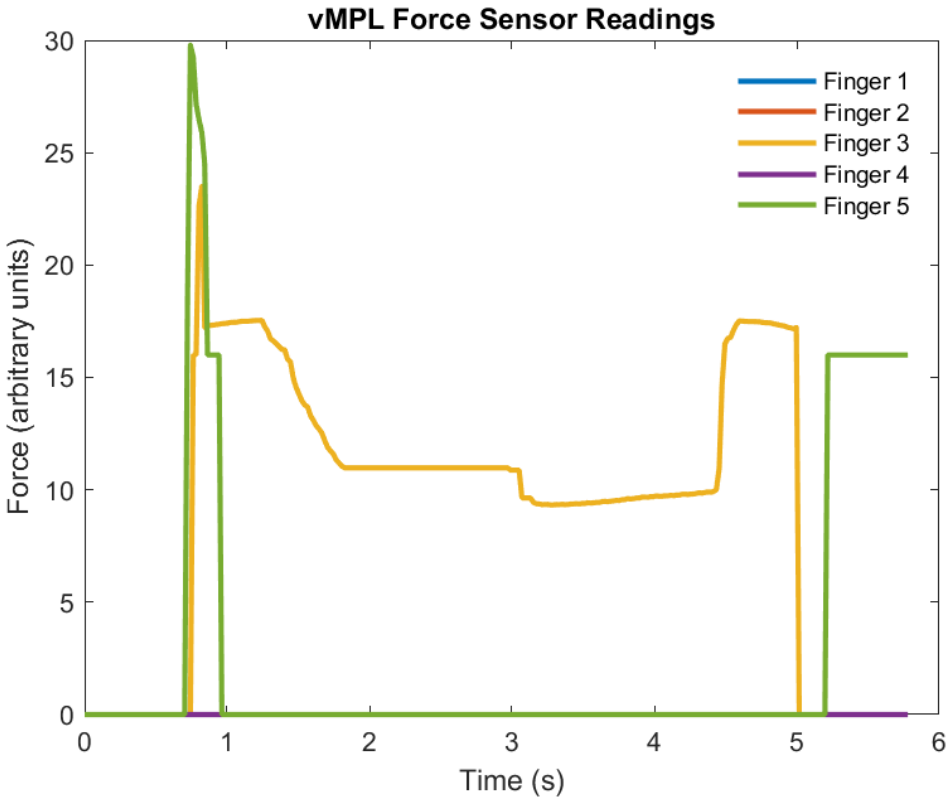


Figure 4.7: vMPL Force Sensor Readings of ftsnForce for Middle Cylinder, 2nd Iteration, 2-second move-in time, 3-second hold, 2-second move-out time.

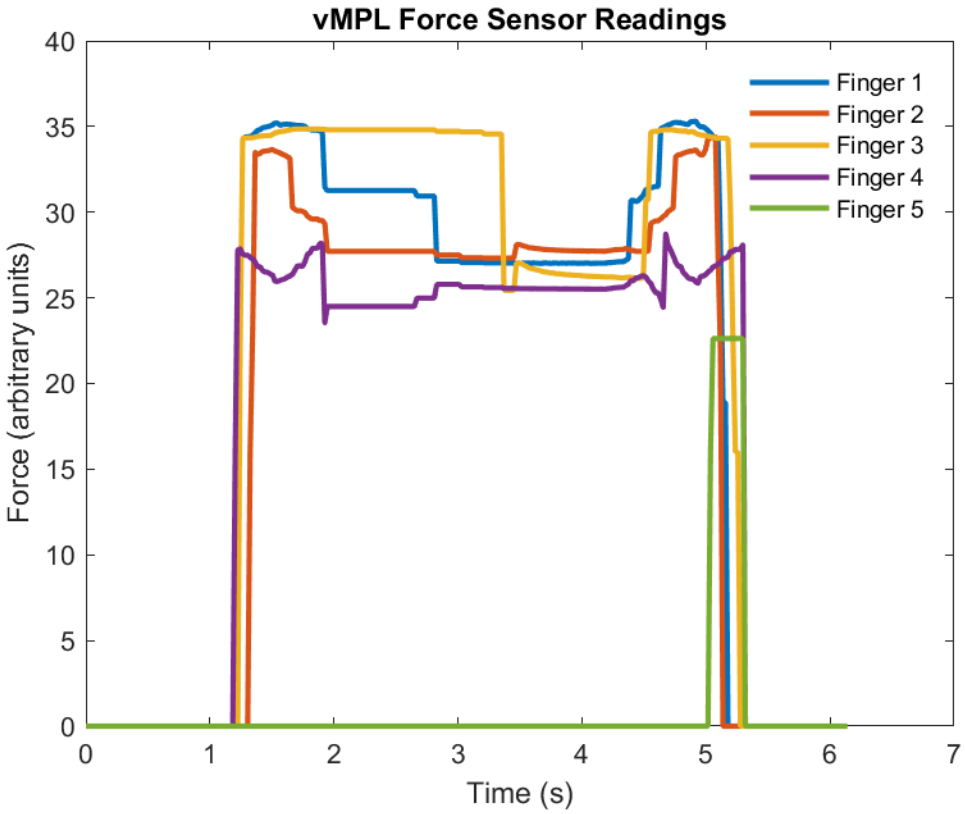


Figure 4.8: vMPL Force Sensor Readings of ftsnForce for Cylinder, 5th Iteration, 2-second move-in time, 3-second hold, 2-second move-out time.

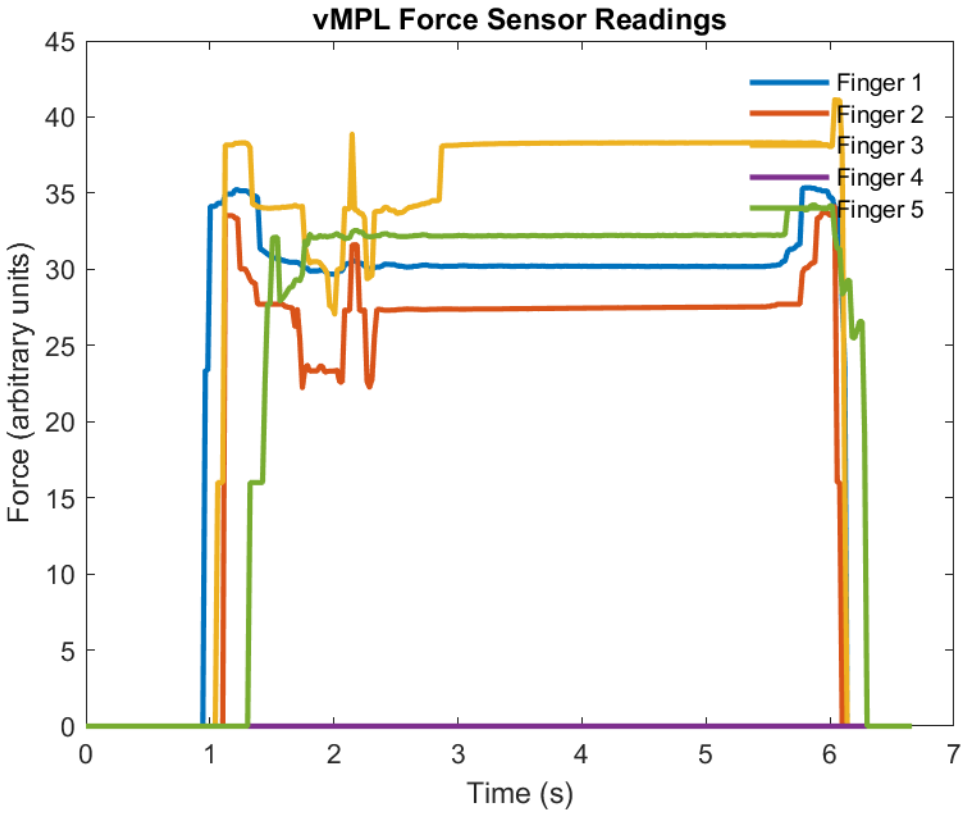


Figure 4.9: vMPL Force Sensor Readings of ftsnForce for Pyramid 4th Iteration, 2-second move-in time, 3-second hold, 2-second move-out time.

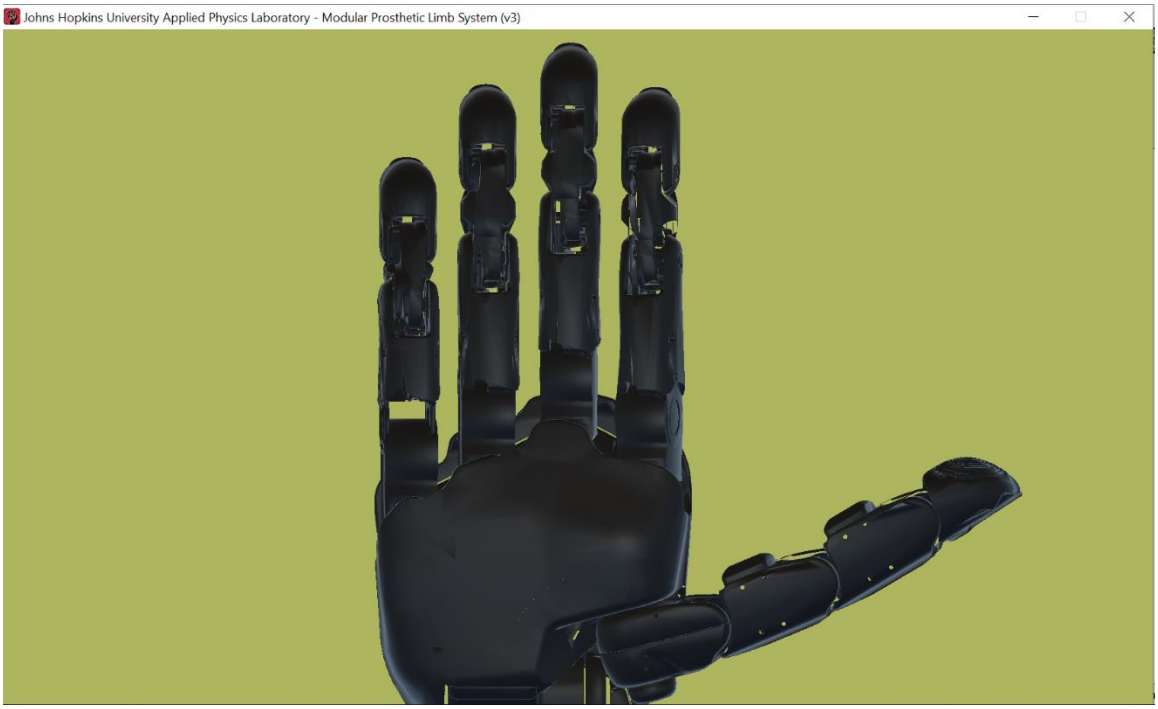


Figure 4.10: vMPL Palm View open hand state, no wif object present, all angles at 0.

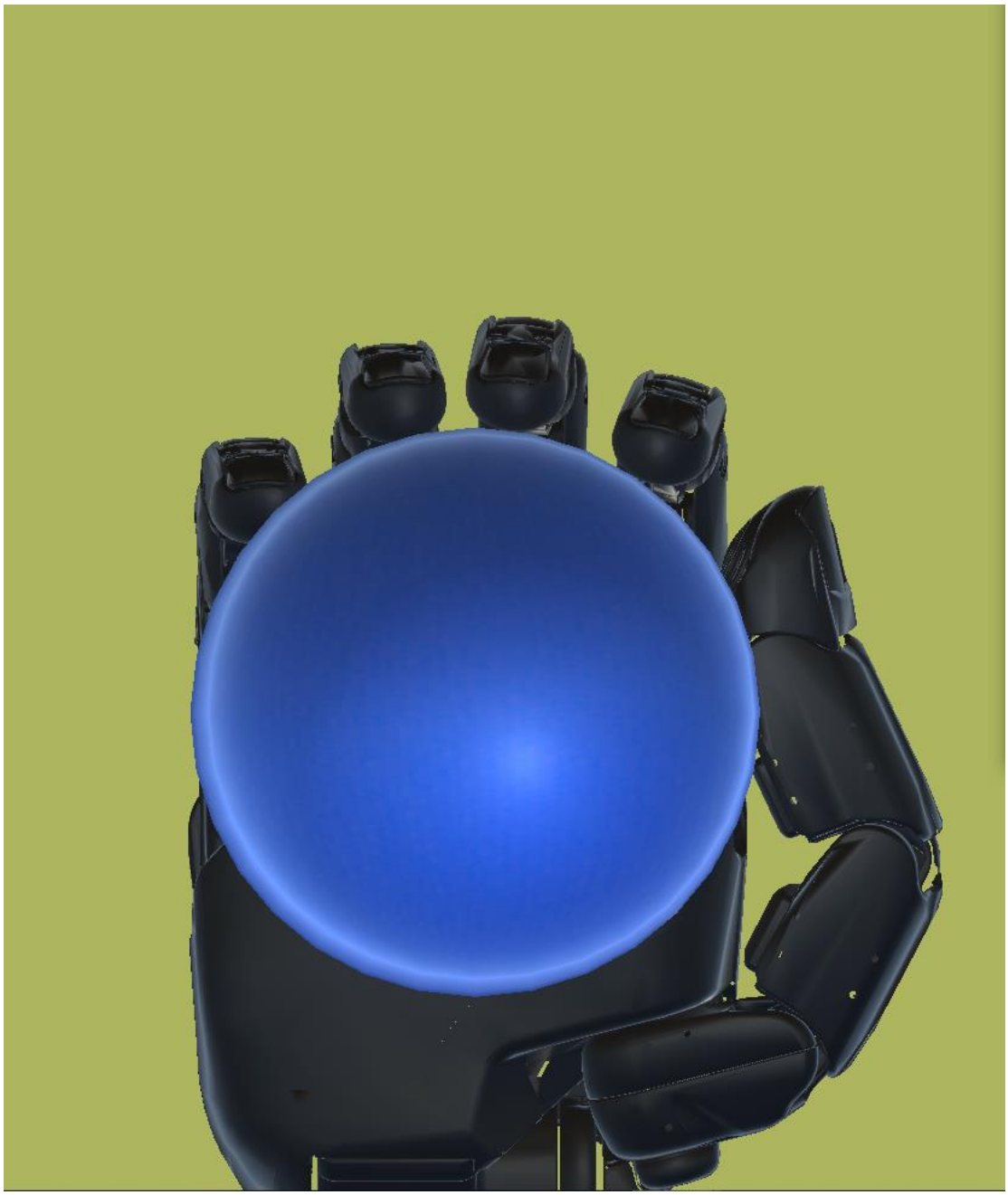


Figure 4.11: vMPL Palm View Sphere grip state: Thumb Metacarpal (THUMB_MCP), Thumb Carpometacarpal Flexion (THUMB_CMC_FE), Index Metacarpal (INDEX_MCP), Middle, Ring, and Little Metacarpal (MIDDLE_MCP, RING_MCP, LITTLE_MCP) all at 60°.

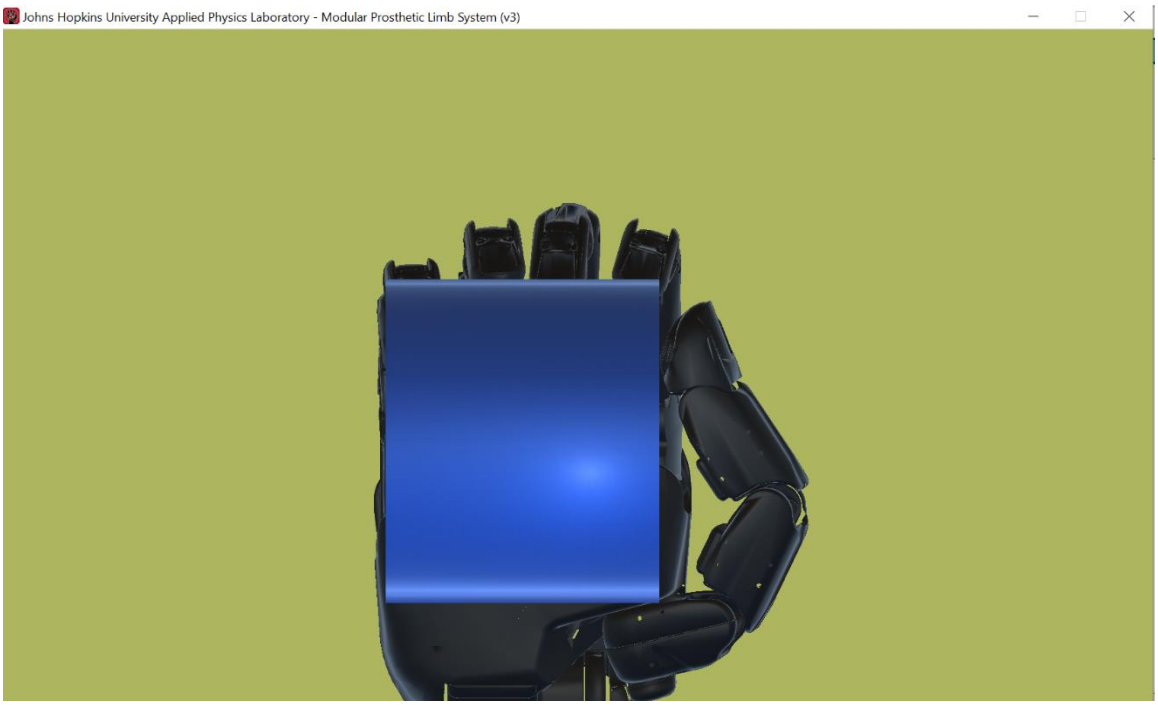


Figure 4.12: vMPL Palm View Sphere grip state: Thumb Metacarpal (THUMB_MCP), Thumb Carpometacarpal Flexion (THUMB_CMC_FE), Index Metacarpal (INDEX_MCP), Middle, Ring, and Little Metacarpal (MIDDLE_MCP, RING_MCP, LITTLE_MCP) all at 90°.

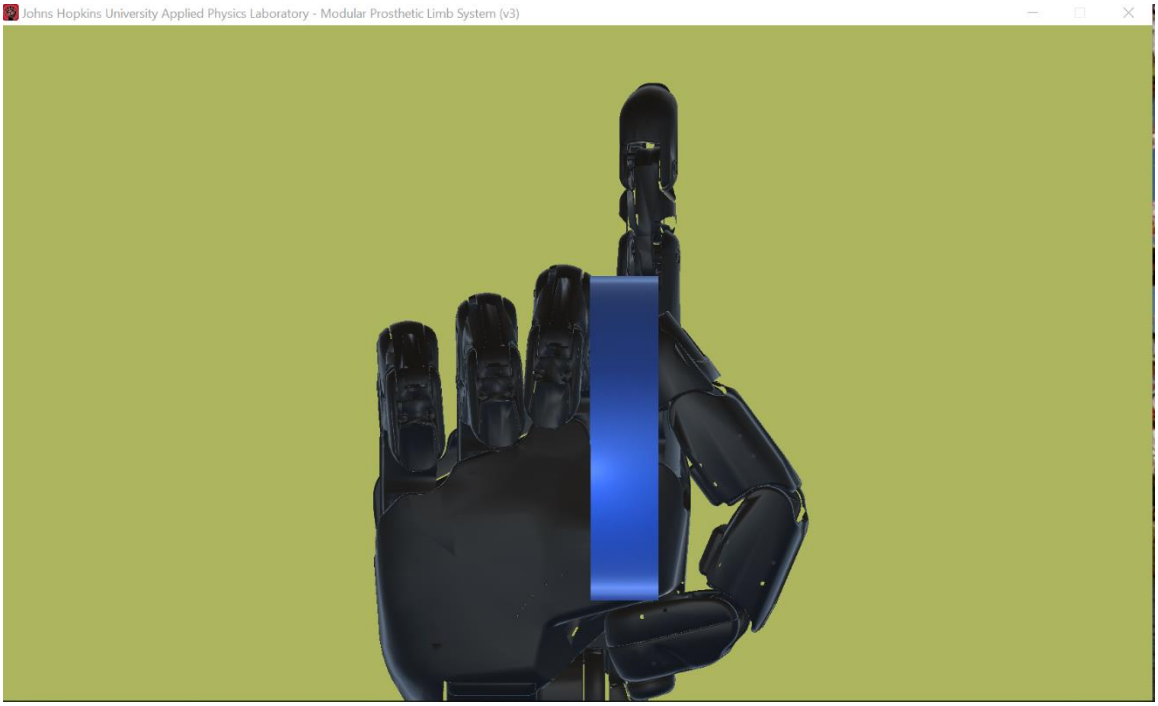


Figure 4.13: vMPL Palm View Sphere grip state: Middle Abduction/Adduction and Ring Abduction/Adduction (MIDDLE_AB_AD, RING_AB_AD), Middle, Ring, and Little Metacarpal (MIDDLE_MCP, RING_MCP, LITTLE_MCP) all at 100°. Thumb Metacarpal and Thumb Carpometacarpal Flexion (THUMB_MCP_FE, THUMB_MCP) at 180°. Little Proximal Interphalangeal (LITTLE_PIP) at 20°.

Figure 4.14: vMPL Palm View Sphere grip state: Middle Metacarpal (MIDDLE_MCP) at 45°, Ring Abduction/Adduction (RING_AB_AD) at -100°, Middle Proximal Interphalangeal (MIDDLE_PIP) at 50°, Ring Metacarpal (RING_MCP) at 100°, and Little Metacarpal (LITTLE_MCP) at 180°.

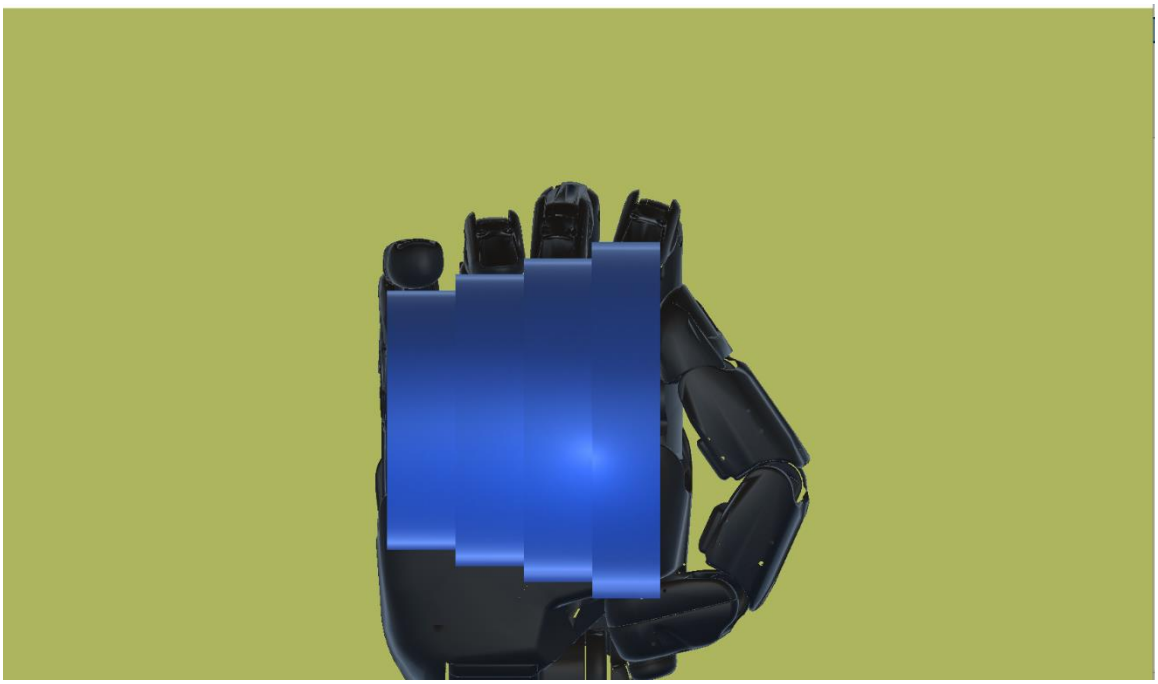


Figure 4.15: vMPL Palm View Sphere grip state: Little Proximal Interphalangeal (LITTLE_PIP), Little Distal Interphalangeal (LITTLE_DIP), Ring Metacarpal (RING_MCP), Middle Metacarpal (MIDDLE_MCP), Index Metacarpal (INDEX_MCP), Thumb Carpometacarpal Flexion (THUMB_CMC_FE), Thumb Metacarpal (THUMB_MCP), Thumb Distal Interphalangeal (THUMB_DIP) all at 180°.

Summary

This preliminary phase of the research served the goal of setting up an environment in which a virtual limb could collect data consistently from objects modeled within its environment, creating a virtual system capable of replicating real-world object interaction. After the control and modeling tasks had been completed, also through the use of VulcanX, the sensor collection built what had previously been accomplished. The link between MATLAB and the vMPL sensors was accomplished through VulcanX, with the vMPL percepts (sensors) serving as the primary means of data acquisition. The data presented served as the dataset used to train the object-recognition algorithm.

Upon completing the object graphing tasks, 20 individual data tables were created, with 20 accompanying graphs over 4 main code layouts, the data being collected through a script iteration. There was substantial variance in the number of data points collected, although the amount of pertinent data proved to be much less varied than the move-in and move-out data. The thumb's force data was measured at 0 for both objects at their first iteration, and both objects possessed very similar source control code. A KNN algorithm was generated, with 40% of the data being used as testing and classification data. A k-

neighbors number was chosen as 8, with the eight closest values determining the classification of any given dataset.

Chapter Five

Conclusion

Summary

The creation of the MPL was the culmination of years of specialized research and development, leading to the creation of a limb that significantly increased proprioception and excelled in both clinical uses and research, furthering advancements in the development of bionic systems. Although the limb stood as the most advanced prosthesis ever designed, due to the rapid continuation of technological innovation in the field and the scarcity of the physical limb, it became harder to use as a platform from which these new technologies could be tested and implemented. With the creation of the e-dermis, a way opened for amputees to react to objects in their grasp in the same manner as those with healthy limbs. Artificial limbs could now perceive differences between objects and potentially send that information back to the nerves of the amputee, further blurring the line between prosthesis and natural limb. Therefore, it was reasoned that the MPL, as the most advanced prosthesis designed, would undoubtedly be able to make full use of the object classification technology were it to possess it.

Created to allow testing and modification of the MPL without possession of the physical limb, the Unity-based vMPL allowed for the use of the full model of the limb, alongside object modeling capabilities and force data extraction. To validate the use of the vMPL as a platform before the classification algorithm could be created, preliminary research on the movement of the limb and the ability of the environment to simulate object-limb interaction was needed. This preliminary phase was accomplished through MATLAB, where a primary script was created to turn joint angle code into bytes,

allowing for control over all 27 DOFs the limb had access to. Subsequently, 6 unique objects were modeled within the vMPL's environment. Data were extracted as the limb grasped the objects, serving as raw, labeled time-force data meant to serve as training data for the algorithm. A KNN (K-Nearest Neighbors) machine learning algorithm was then implemented, where the labeled data was used as training and unlabeled data presented to the completed algorithm for labeling and classification.

Conclusion and Discussion

When tasked with moving to a specific joint angle, the limb moved with constant efficiency, the time between execution of the MATLAB code and subsequent movement often being <1s. The limb displayed full movement with all indicated angles, fully utilizing the range of motion allowed for the 7 arm joints and the 20 allowed for the finger joints. A program made the limb move continuously through the use of a conditional for-loop (Figure A.2). This continuous movement was especially of note, as it meant that the limb was capable of rapid movement and that the VulcanX system only viewed new joint angle signals as packets (Figure 3.1), with existing ones requiring multiple movements and essentially serving as the same signal. At the end of the joint movement phase, the limb was proven to not only be able to react to complex commands in a short time but was able to process MATLAB syntax and translate it into active movement directly.

This same logic powered the new "timed" code of the hand-object interface (Figure A.3), with the same joint angle being sent to several joints of the limb, allowing for more flexible timing in the movement of the limb and the creation of a consistent 7-sec measurement interval when implemented alongside sensor code. This program also

possessed the ability to generate objects in the vMPL interface through a wif commander (Figure A.1). Collisions between the hand and object were fully modeled in this interface natively, with fingers being rendered unable to physically move further when met with an object and dependent on the object's shape (Figures 4.9-4.14). The MATLAB script allowed for the movement of the arm to switch between the timed code and the joint angle code, as long as specific steps were followed (Figure 3.1). Both methods were used when performing grasping tasks for the cylinder, sphere, index disc, and middle disc. However, the environment exhibited modeling conflicts when introduced to shapes such as the oblate spheroid when modeled in a [1 2 1] length/width/depth structure (Figure A.4). The oblate spheroid's abstract shape created an error within the modeling of the vMPL, leading to an object that appeared to be much larger than what it presented itself as. A shift to the UDP joint angle control program allowed for more precise movements but still blocked collision between the hand and the object. Preliminary grasping data was then obtained using only the first four modeled objects, but the UDP control program's success allowed for the use of the oblate spheroid and pyramidal training data. However, they possessed inconsistent time measurements compared to the other objects. This result meant that the algorithm, which used KNN with DTW (Dynamic Time Warping), would have to account for timing differences when attempting classification tasks.

The differing appearances of the graphs of the objects demonstrated the efficacy of the modeled force sensors, as the cylinder and sphere were grasped using the same time joint angle code but resulted in consistently different graphs of force-time data (Figure 4.1). This dynamic reaction to objects modeled within the interface dispels the notion that the force sensors were wholly dependent on the joint angle's location, instead

measuring actual modeled force within the interface. Differences in the objects' graphs also highlighted differences in the shapes of the objects themselves, as sphere sensor data appeared more "constant" and dipped less often than cylinder sensor data, which possessed frequent spikes and dips as the fingers twitched. The two dimensions (force and time) of the data gave reason for the use of a low hyperparameter, distance-based learning algorithm such as KNN, as the relatively small dataset and variable time measurements could be classified with high accuracy nonetheless. Further proof that the modeling and control of the limb led to viable force data came from the range of accuracy displayed by the algorithm come from the classification reports, with full accuracy ratings ranging from 81% to 100%. The index cylinder constantly appeared as the least accurate, at one point possessing an f1-score of 70%, but never possessed anything less than a 100% recall score, implying a consistency in the spread of its force data that led to other objects being misidentified as it. However, when shown the graphs of the objects in the 4^{th} iteration, where the cylinder was misidentified as the index cylinder 10 times, a clear difference is shown in the appearance of the initial spike. This phenomenon means that any model that displays >90% accuracy cannot be ruled as trustworthy, as seen with iteration model 2, which possessed an overall accuracy of 83% yet a 0.48 f1-score and 0.32 recall score for the cylinder, meaning that when the object shown to it was the cylinder, it predicted incorrectly 68% of the time. In order to have the algorithm be of the most use, it needs to be able to predict positive results for objects with high certainty and classify objects with high accuracy, making the f1 score the most important metric by which the algorithm is judged and any object exhibiting below >90% f1 accuracy poorly

classified. Therefore, the 3rd iteration stands as the ideal dataset on which to train the algorithm.

Recommendations

Though fully capable of classifying the objects based solely on force-time data, it could be valuable to use other classification methods in both supervised learning and deep learning. An SVM (Support Vector Machine) algorithm, while not as well suited for time-based datasets as KNN DTW, could still classify the objects into types. However, it would be limited to two classifications due to its limits as a binary classifier. The data could also be repeatedly iterated and then used for training as a dataset for an ANN, most likely a CNN (Convolutional Neural Network) or LSTM (Long-Short Term Memory) as an RNN (Recurrent Neural Network) algorithm, since they are both able to analyze long series, despite their need for hyperparameters. Both could be achieved through the use of the Scikit-learn Python module or the Scikit-time-dl module. Torque and velocity data could also be collected and held in the same graph, necessitating the use of an algorithm capable of breaking down series or the use of a KNN DTW with a Fourier Transform. In order to increase the accuracy of the index cylinder and cylinder objects, a lower number of neighbors could be used for the algorithm, which each point being classified based on the 3 points closest to them, with k=3. The training data of the 3rd Iteration could also be used to train the other iterations, as it was able to demonstrate 100% classification accuracy.

Future Implications

The vMPL interface objects were rigid and unmoving, with grasp data collected from the fingers' collisions with the objects. Using an environment capable of replicating

more complex physical interactions could lead to a wealth of new sensor information being measured, providing the algorithm with a much larger dataset capable of being used in a neural network. This neural network could also detect correlation between values if force data is captured alongside torque and velocity data, allowing for much more insight than given by the KNN algorithm (due to the algorithm not providing logic for its classifications). Since the object control, modeling, and sensor programs have been created and only require minor changes to respond to different situations, only the vMPL interface's source code would have to be changed, allowing for the MPL's velocity sensors to come online.

More extensive overhauls to the vMPL interfaces could be achieved as well, as the modeling of non-rigid objects could create data incredibly divergent from the original data, necessitating the use of an algorithm capable of detecting and reacting to slight correlations between series. This would introduce much greater variability in the sensor data that would allow a dynamic temporal system to improve its classification with every added iteration.

To further push the vMPL as a training alternative for the MPL, a direct link between the two could be established, with Python-based classification data being relayed to the MATLAB control scheme, creating "reflexes" in the limb that would arise from touching a virtual object. With classification powered by a DL (Deep Learning) algorithm and a physical TENS unit, the vMPL could be used as a training tool or a therapy tool for upper-limb amputees, allowing them to react to touching objects of various shapes and comfort levels in an entirely virtual manner, lowering the need for a physical MPL or any physical limb in non-control training situations.

References

Betthauser, J. L., Hunt, C. L., Osborn, L. E., Masters, M. R., Levay, G., Kaliki, R. R., & Thakor, N. V. (2018). Limb position tolerant pattern recognition for myoelectric prosthesis control with adaptive sparse representations from extreme learning. *IEEE Transactions on Biomedical Engineering*, *65*(4), 770-778. https://doi.org/10.1109/TBME.2017.2719400

Blana, D., Kyriacou, T., Lambrecht, J. M., & Chadwick, E. K. (2016). Feasibility of using combined EMG and kinematic signals for prosthesis control: A simulation study using a virtual reality environment. *Journal of electromyography and kinesiology*, *29*, 21–27. https://doi.org/10.1016/j.jelekin.2015.06.010

Burck, J.M., Bigelow, J.D., & Harshbarger, S.D. (2011). Revolutionizing prosthetics: systems engineering challenges and opportunities. *Johns Hopkins APL Technical Digest, 30*(3), 186-197.

Fifer, M. S., Hotson, G., Wester, B. A., McMullen, D. P., Wang, Y., Johannes, M. S., Katyal, K. D., Helder, J. B., Para, M. P., Vogelstein, R. J., Anderson, W. S., Thakor, N. V., & Crone, N. E. (2014). Simultaneous neural control of simple reaching and grasping with the modular prosthetic limb using intracranial EEG. *IEEE transactions on neural systems and rehabilitation engineering: a publication of the IEEE Engineering in Medicine and Biology Society*, *22*(3), 695–705. https://doi.org/10.1109/TNSRE.2013.2286955

Geng, Y., Samuel, O. W., Wei, Y., & Li, G. (2017). Improving the robustness of real-time myoelectric pattern recognition against arm position changes in transradial amputees. *BioMed Research International, 2017*, 1-10. https://doi.org/10.1155/2017/5090454

Harris, A., Katyal, K., Para, M., & Thomas, J. (2011). Revolutionizing prosthetics software technology. *2011 IEEE International Conference on Systems, Man, and Cybernetics*, 2877-2884. https://doi.org/10.1109/ICSMC.2011.6084102

Hill, N. J., Gupta, D., Brunner, P., Gunduz, A., Adamo, M. A., Ritaccio, A., & Schalk, G. (2012). Recording human electrocorticographic (ECoG) signals for neuroscientific research and real-time functional cortical mapping. *Journal of visualized experiments : JoVE*, (64), 3993. https://doi.org/10.3791/3993

Hotson, G., McMullen, D. P., Fifer, M. S., Johannes, M. S., Katyal, K. D., Para, M. P., Armiger, R., Anderson, W. S., Thakor, N. V., Wester, B. A., & Crone, N. E. (2016). Individual finger control of a modular prosthetic limb using high-density electrocorticography in a human subject. *Journal of neural engineering*, *13*(2), 026017–26017. https://doi.org/10.1088/1741-2560/13/2/026017

Huinink, L. H., Bouwsema, H., Plettenburg, D. H., van der Sluis, C. K., & Bongers, R. M. (2016). Learning to use a body-powered prosthesis: changes in functionality and kinematics. *Journal of neuroengineering and rehabilitation*, *13*(1), 90. https://doi.org/10.1186/s12984-016-0197-7

Hunt, C. L., Sharma, A., Osborn, L. E., Kaliki, R. R., & Thakor, N. V. (2018). Predictive trajectory estimation during rehabilitative tasks in augmented reality using inertial

sensors. *2018 IEEE Biomedical Circuits and Systems Conference, (BioCAS).* https://doi.org/10.1109/BIOCAS.2018.8584805

Iskarous, M. M., Nguyen, H. H., Osborn, L. E., Betthauser, J. L., & Thakor, N. V. (2018). Unsupervised learning and adaptive classification of neuromorphic tactile encoding of textures. *2018 IEEE Biomedical Circuits and Systems Conference, (BioCAS), 1-4.* https://doi.org/10.1109/BIOCAS.2018.8584702

Kumar, V., & Todorov, E. (2015). MuJoCo HAPTIX: A virtual reality system for hand manipulation. *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, 657-663. https://doi.org/10.1109/HUMANOIDS.2015.7363441

Menon, S., Fok, S., Neckar, A., Khatib, O., & Boahen, K. (2014). Controlling articulated robots in task-space with spiking silicon neurons. *5th IEEE RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics (BioROB)*, 181-186. https://doi.org/10.1109/biorob.2014.6913773

Millstein, S. G., Heger, H., & Hunter, G. A. (1986). Prosthetic use in adult upper limb amputees: a comparison of the body powered and electrically powered prostheses. Prosthetics and Orthotics International, 10(1), 27–34. https://doi.org/10.3109/03093648609103076

Oppenheim, H., Armiger, R., & Vogelstein, J. (2010). WiiEMG: A real-time environment for control of the Wii with surface electromyography. *ISCAS 2010 - 2010 IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems*, 957-960. https://doi.org/ 10.1109/ISCAS.2010.5537390.

Osborn, L. E., Dragomir, A., Betthauser, J. L., Hunt, C. L., Nguyen, H. H., Kaliki, R. R., & Thakor, N. V. (2018). Prosthesis with neuromorphic multilayered e-dermis perceives touch and pain. *Science Robotics*, *3*(19), [eaat3818]. https://doi.org/10.1126/scirobotics.aat3818

Perry, B. N., Armiger, R. S., Yu, K. E., Alattar, A. A., Moran, C. W., Wolde, M., McFarland, K., Pasquina, P. F., & Tsao, J. W. (2018). Virtual integration environment as an advanced prosthetic limb training platform. *Frontiers in Neurology*, *9*(785), 1-8. https://doi.org/10.3389/fneur.2018.00785

Prahm, C., Eckstein, K., Ortiz-Catalan, M., Dorffner, G., Kaniusas, E., & Aszmann, O. C. (2016). Combining two open source tools for neural computation (BioPatRec and Netlab) improves movement classification for prosthetic control. *BMC research notes*, *9*(1), 429. https://doi.org/10.1186/s13104-016-2232-y

Appendix A

Figure A.1: Control, Modeling, and Sensing Code (MATLAB)

```
% This script shows how to extract sensor data from the vMPL

% make sure the minivie and dependencies folders are on your path
addpath(genpath('../dependencies'));
addpath(genpath('../minivie'));

This section sets up the connection to the vMPL through VulcanX
% initialize the vulcanX obj and wif commander
[mud, perceptSocket, vxSink] =
setupVulcanXObjects('255.255.255.255', 'right', true);
vxSink.putbytes(mud.DOMPositionCmd(zeros(1, 27)));
jointAngles = zeros(1, 27);
wif = WifCommander();

This section initializes some variables
% initialize the variables to hold the percept packets (i.e., the
sensor
% information from the vMPL)
perceptPackets = {};
clearPerceptBuffer_All(perceptSocket, perceptPackets);  %clear
the percept buffer (removes old sensor readings)
sensorVal.time = [];
sensorVal.ftsnForce = [];
sensorVal.position = [];
```

This section moves the vMPL fingers and grabs the "Percept Packets" from VulcanX.
Basically what happens is the vMPL sends out packets of information that contain sensor information. In MATLAB, we can read in those packets and extract the data we want from it

```
% move the fingers to grab an object (this section can be
customized as
% needed)
wif.setVisibleOnlyCylinders();

joint_names = {'THUMB_CMC_FE','INDEX_MCP', 'MIDDLE_MCP',
'RING_MCP', 'LITTLE_MCP', 'THUMB_MCP'};
moveDelta = pi/2;
moveSteps = 15;
approxTime = [2 3 2];

% jointAngles = MoveFingerInOut(vxSink, mud, jointAngles,
joint_names, joint_angles, steps, approxTime);
moveStepDelta = moveDelta ./ moveSteps;
dt = approxTime(1) / moveSteps;

%move the fingers in
for loop = 1 : moveSteps
    [jointAngles] = MoveFinger(vxSink, mud, jointAngles,
joint_names, moveStepDelta);
    pause(dt);
    perceptPackets = clearPerceptBuffer_All(perceptSocket,
perceptPackets);     %get all the new percept packets and add it
to the existing data
end

%hold the fingers in place
wait_timer = tic;   %timer to keep track of how long to wait
if approxTime(2) - dt > 0
    while toc(wait_timer) < (approxTime(2) - dt)
        pause(dt);
        perceptPackets = clearPerceptBuffer_All(perceptSocket,
perceptPackets);     %get all the new percept packets and add it
to the existing data
    end
end
dt = approxTime(3) / moveSteps;

%move the fingers out
for loop = 1 : moveSteps
    [jointAngles] = MoveFinger(vxSink, mud, jointAngles,
joint_names, -1 * moveStepDelta);
    pause(dt);
    perceptPackets = clearPerceptBuffer_All(perceptSocket,
perceptPackets);     %get all the new percept packets and add it
to the existing data
end
```

```
This section takes all the Percept Packets and pulls out the
fingertip force sensor data as well as the joint positions
% extract the sensor data from the the percept packet
sensor_data = extract_sensor_data(perceptPackets);
sensorVal.ftsnForce = [sensorVal.ftsnForce;
sensor_data.ftsnForce];
sensorVal.position = [sensorVal.position; sensor_data.position];
save('cylinder_data.mat','sensorVal')

This section plots all the data
% plot the force sensor values for each finger
figure(1)
time =
linspace(0,length(sensorVal.ftsnForce)/50,length(sensorVal.ftsnFo
rce));
p = plot(time,sensorVal.ftsnForce,'LineWidth',2);
ax = gca;   %update the plot axes with labels
ax.XLabel.String = 'Time (s)';
ax.YLabel.String = 'Force (arbitrary units)';
ax.Title.String = 'vMPL Force Sensor Readings';
L = legend('Finger 1','Finger 2','Finger 3','Finger 4','Finger
5');
L.Box = 'off';
```

Figure A.2: Rapid Arm Movement Code (MATLAB)

```
upperArmAngles (4) = 90 * pi / 180;
msg = mce.DOMPositionCmd([upperArmAngles,handAngles])
hArm.putData(msg)       %this line sends the message (you should see
the arm move)
while  msg == mce.DOMPositionCmd([upperArmAngles,handAngles])
      upperArmAngles(4)= 180 * pi / 180;
      hArm.putData(msg);
      msg_2 = mce.DOMPositionCmd([upperArmAngles,handAngles)]
      while  msg_2 == mce.DOMPositionCmd ([upperArmAngles,handAngles])
            upperArmAngles(4) = 90 * pi / 180;
            msg_2 = mce.DOMPositionCmd([upperArmAngles,handAngles])
            hArm.putData(msg_2)
            hArm.putData(msg)
      end
end
```

Figure A.3: MoveFingerInOut Code (MATLAB)

```
function [jointAngles] = MoveFingerInOut (vxSink, mudEncoder,
jointAngles, finger, moveDelta, moveSteps, approxTime)
```

Figure A.4: Oblate Spheroid Code (MATLAB)

```
wif.setVisibleOnly('rProbeBigSphere')
wif.rescale('rProbeBigSphere', 5 .* [1 2 1]);
```

Figure A.5: Sphere, 2nd Iteration

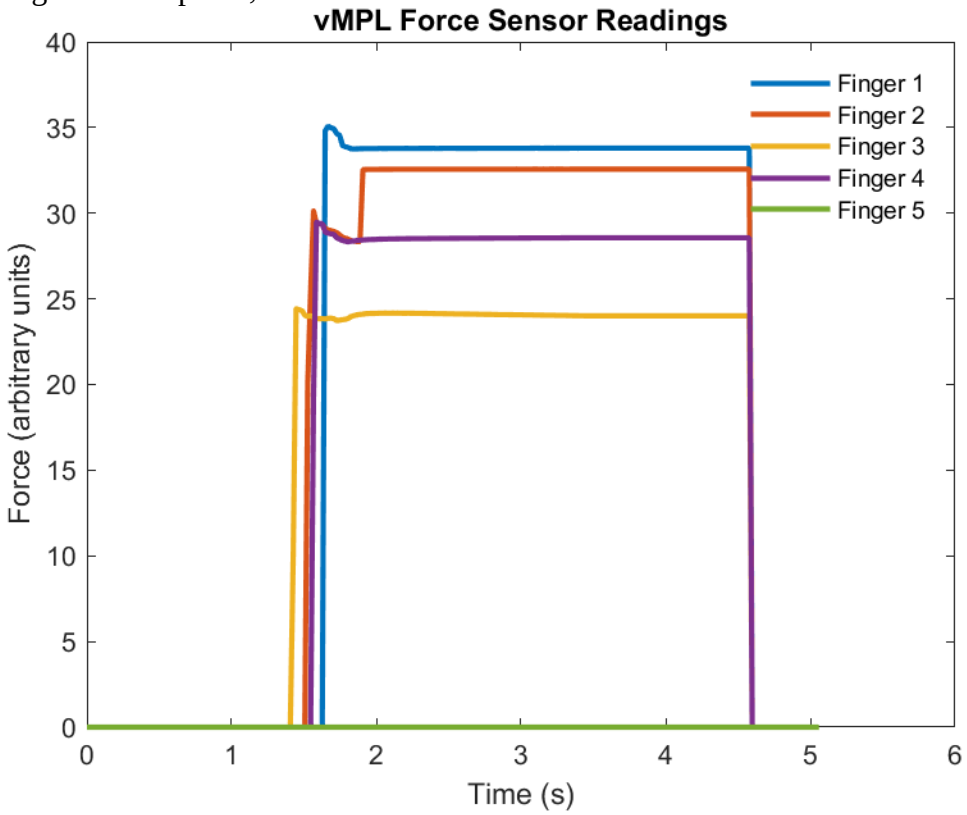


Figure A.6: Sphere, 3rd Iteration

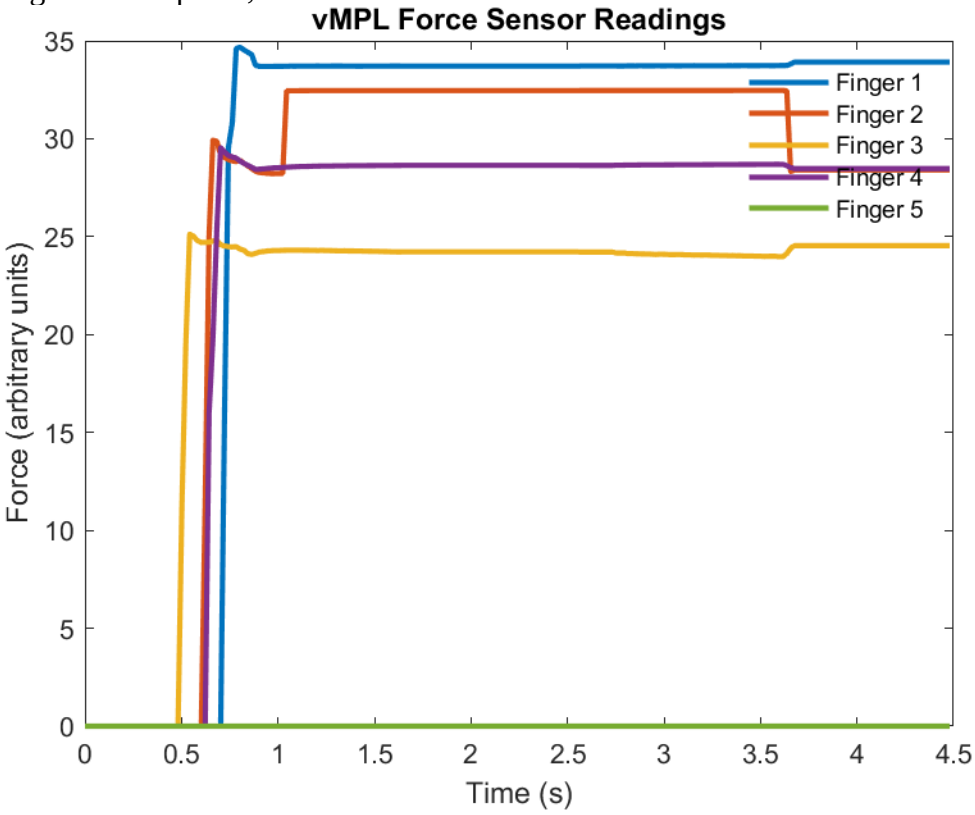


Figure A.7: Sphere, 4th Iteration

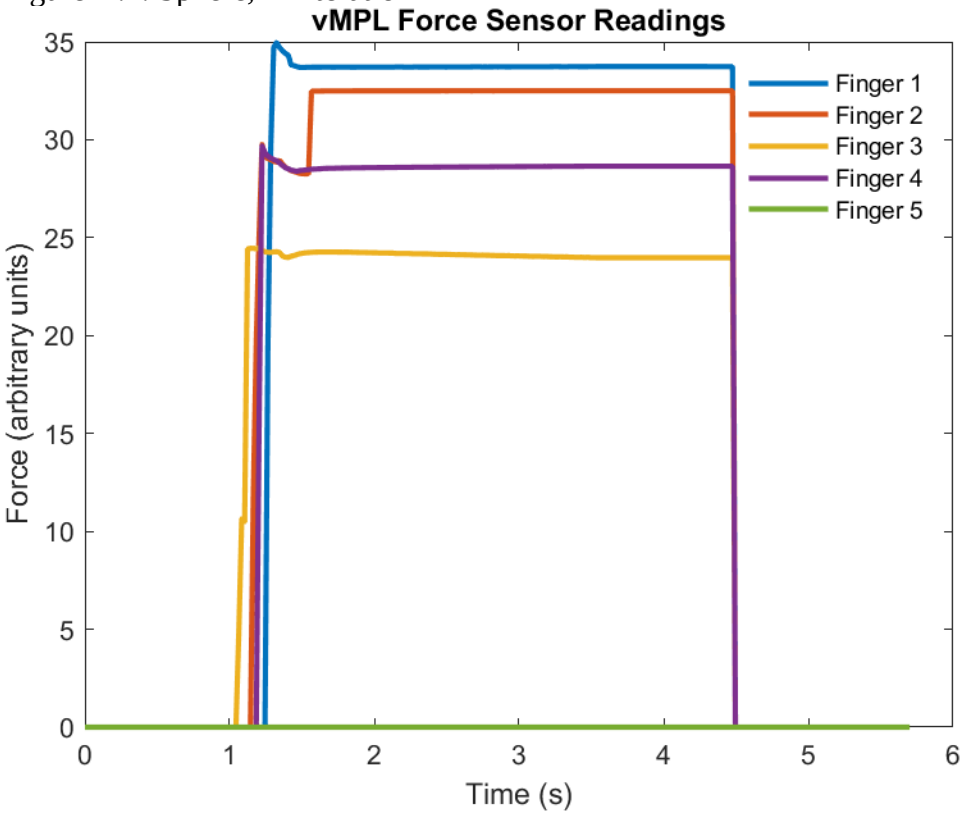


Figure A.8: Sphere, 5th Iteration

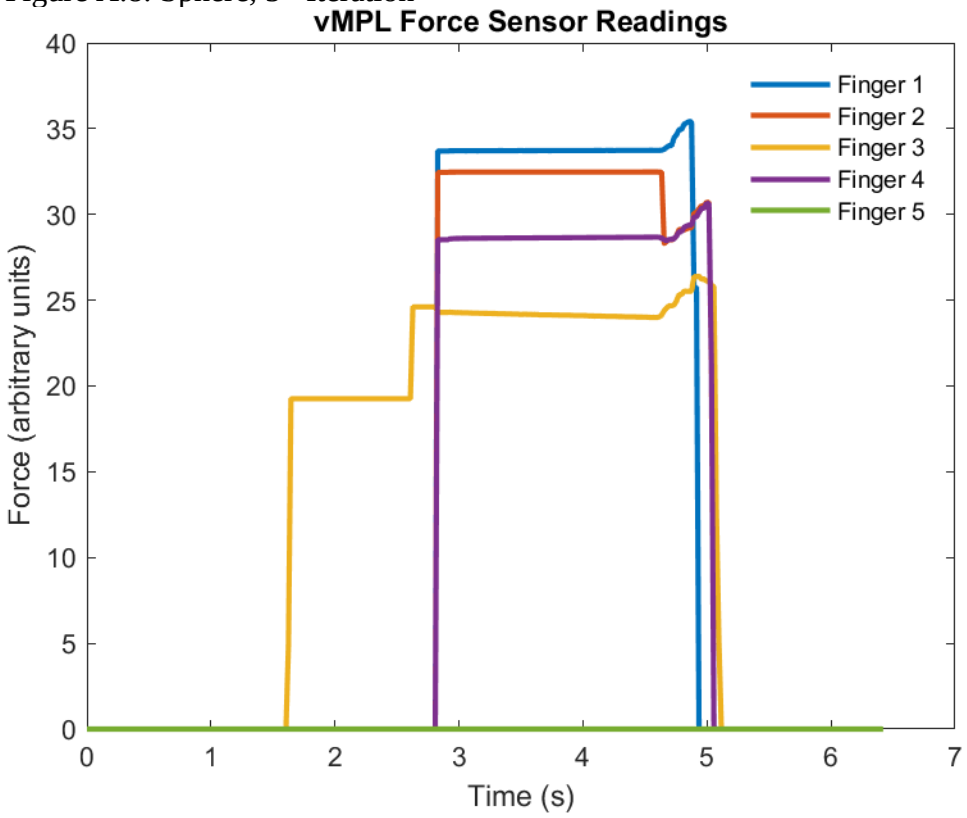


Figure A.9: Index Cylinder, 1st Iteration

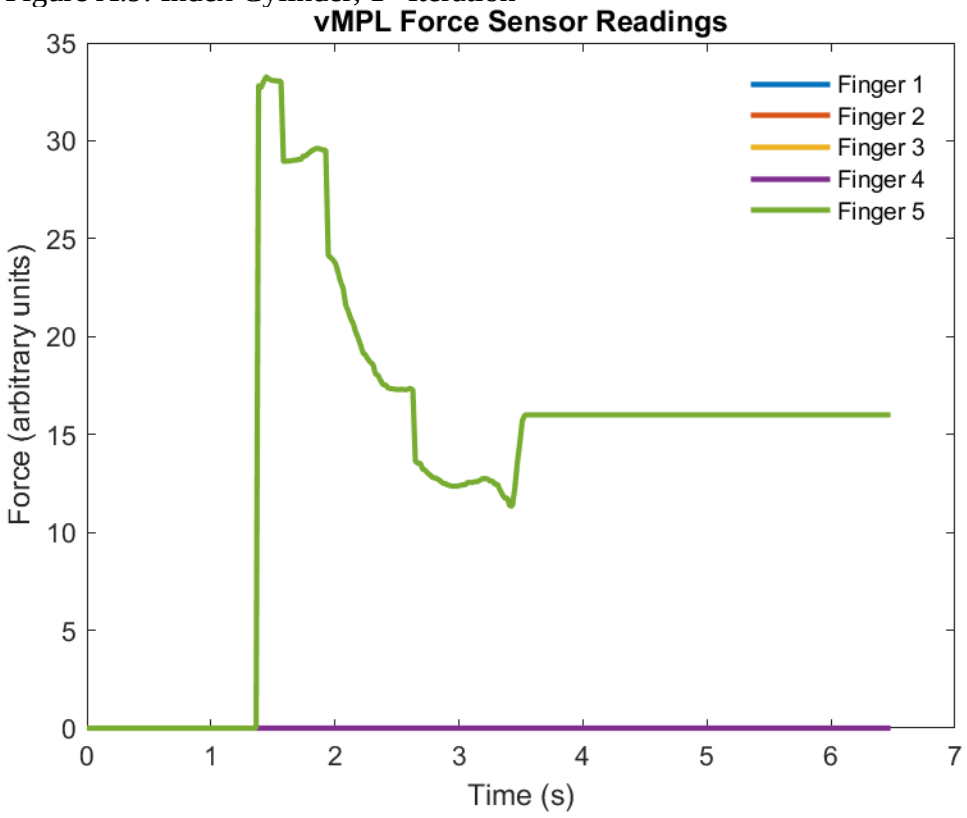


Figure A.10: Index Cylinder, 2nd Iteration

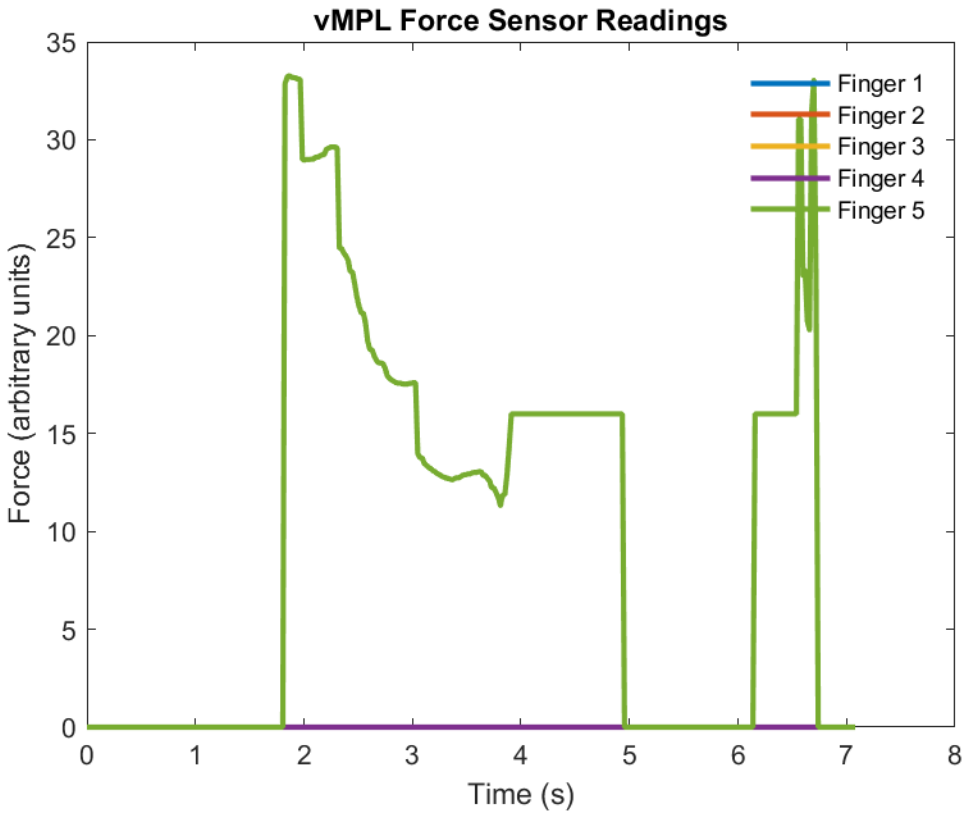


Figure A.11: Index Cylinder, 4th Iteration

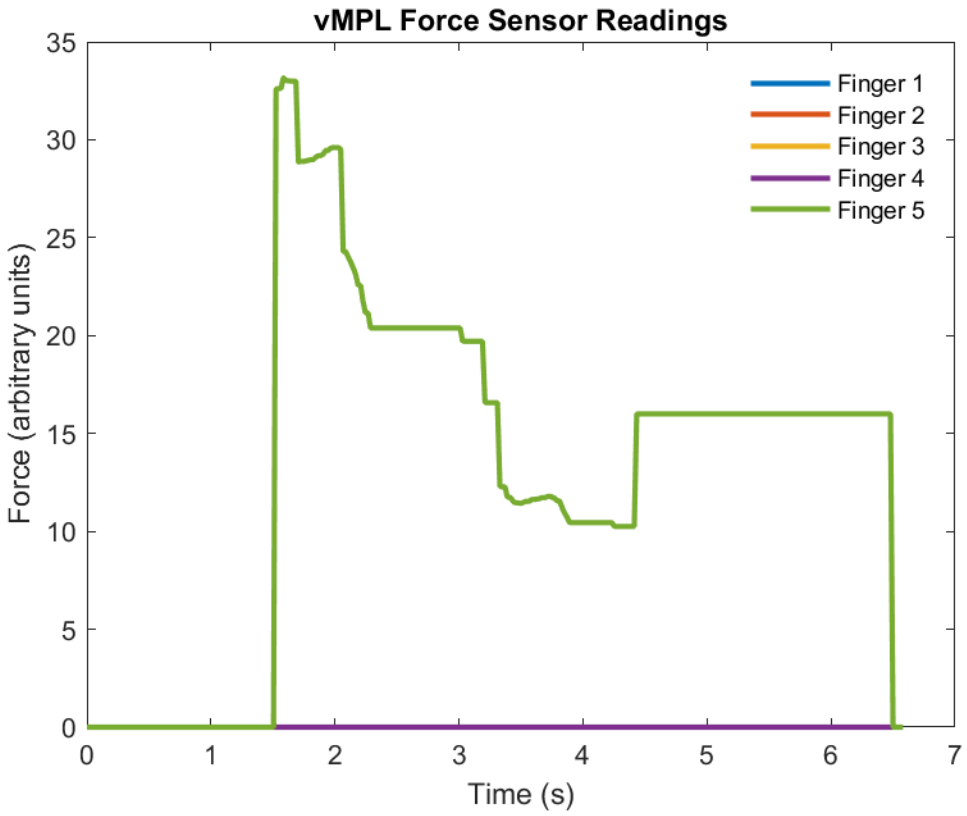


Figure A.12: Index Cylinder, 5th Iteration

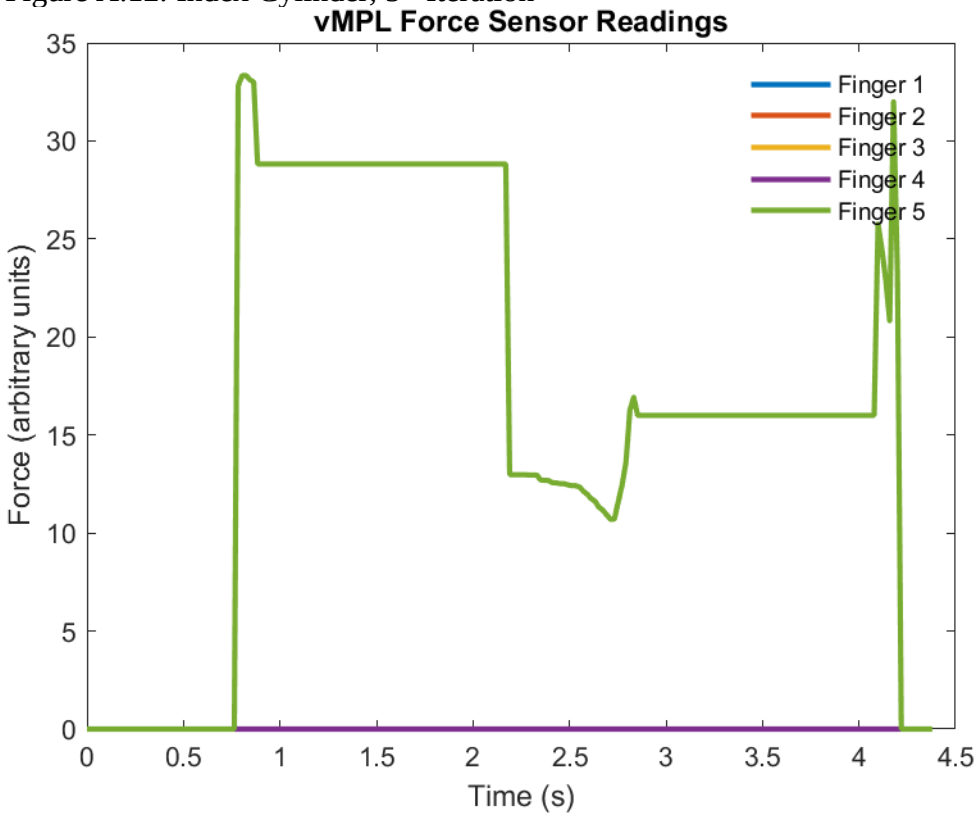


Figure A.13: Middle Cylinder, 1st Iteration

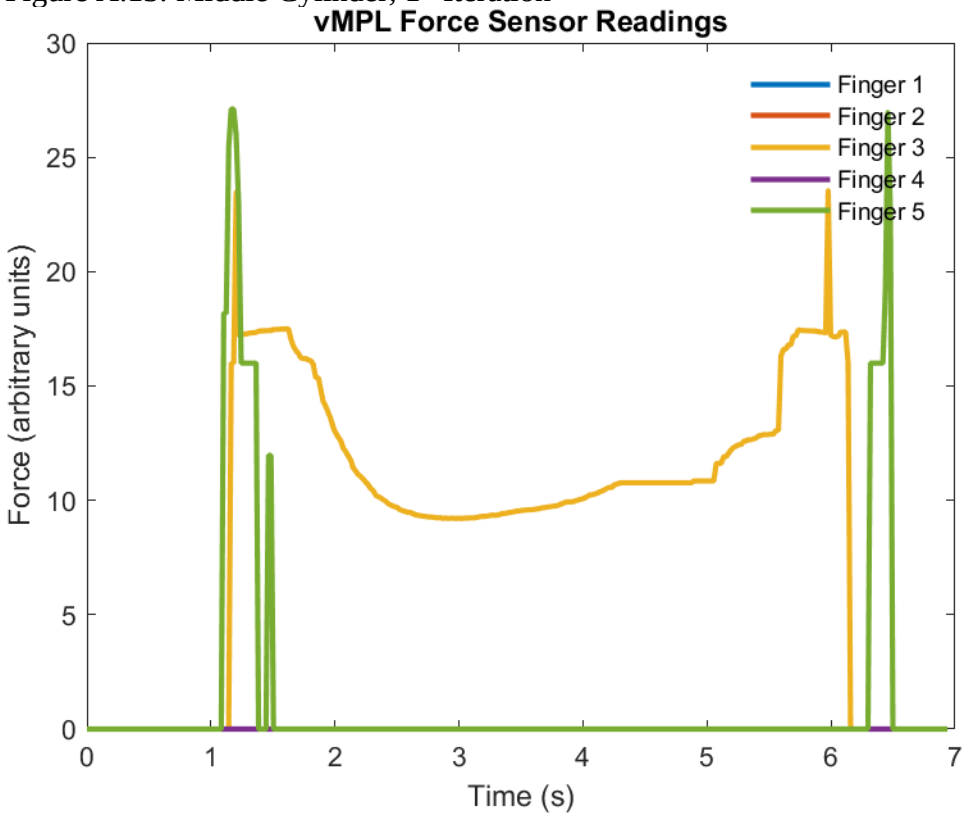


Figure A.14: Middle Cylinder, 3rd Iteration

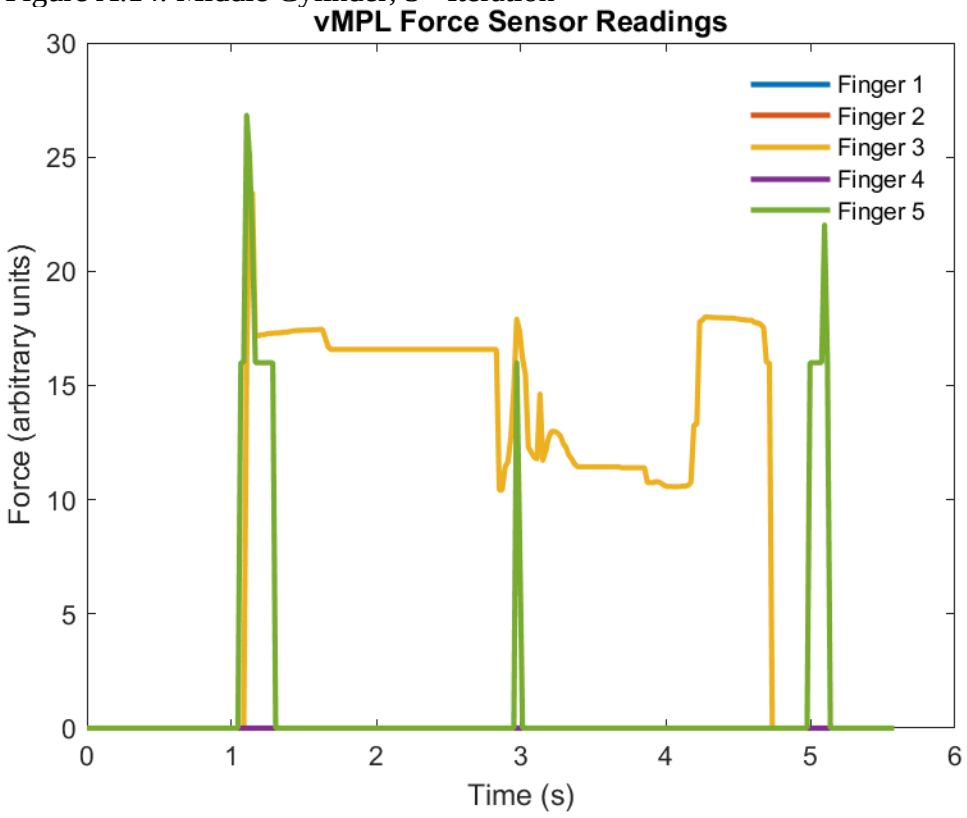


Figure A.15: Middle Cylinder, 4th Iteration

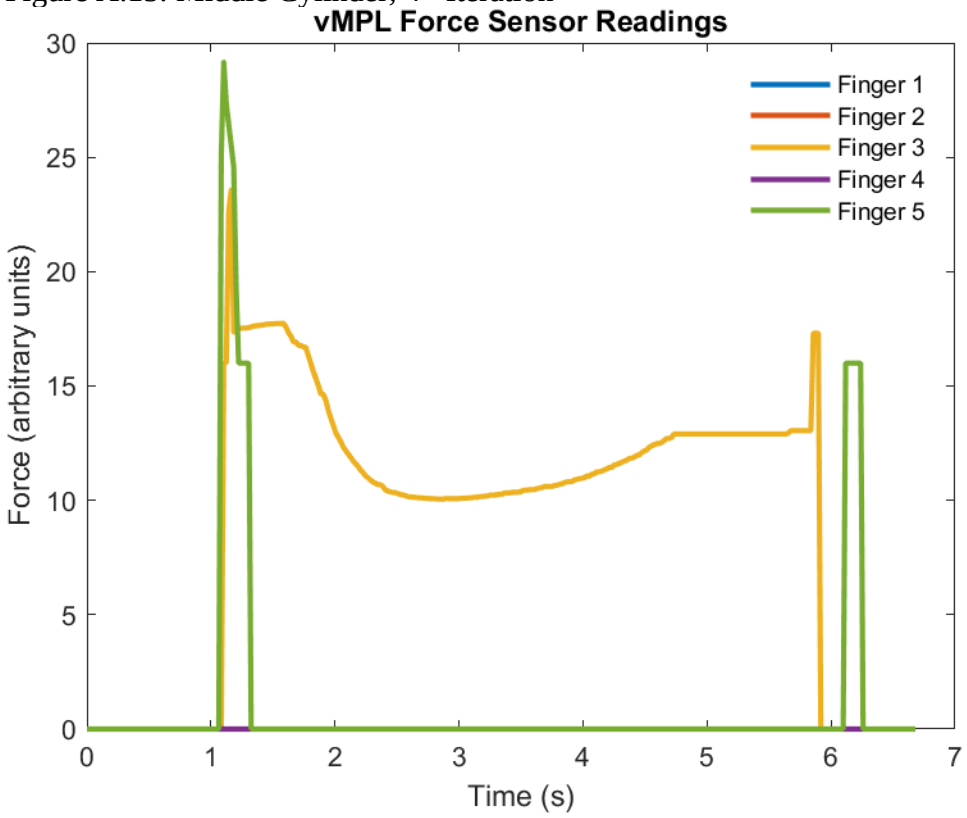


Figure A.16: Middle Cylinder, 5th Iteration

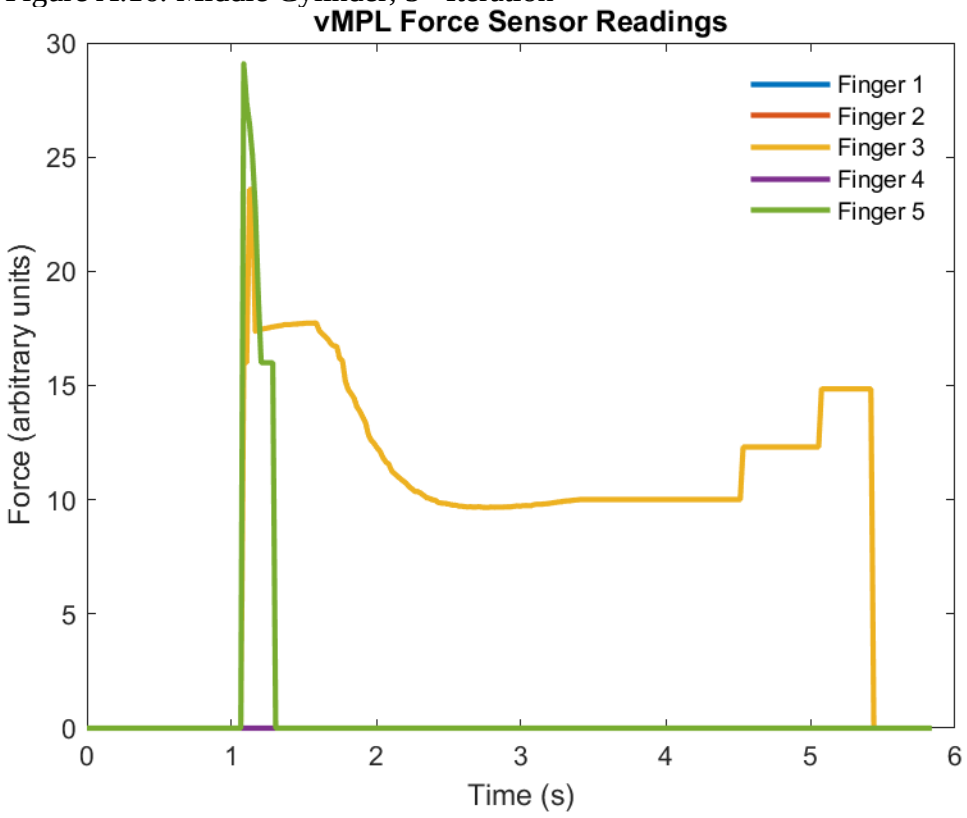


Figure A.17: Cylinder, 1st Iteration

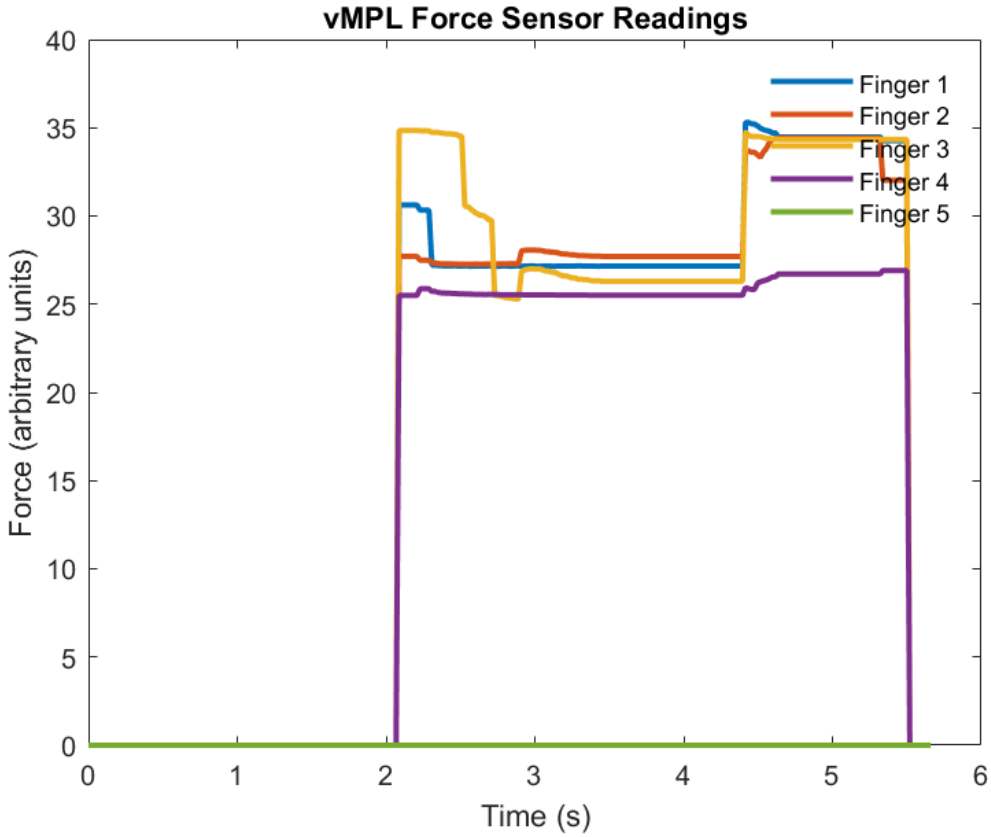


Figure A.18: Cylinder, 2nd Iteration

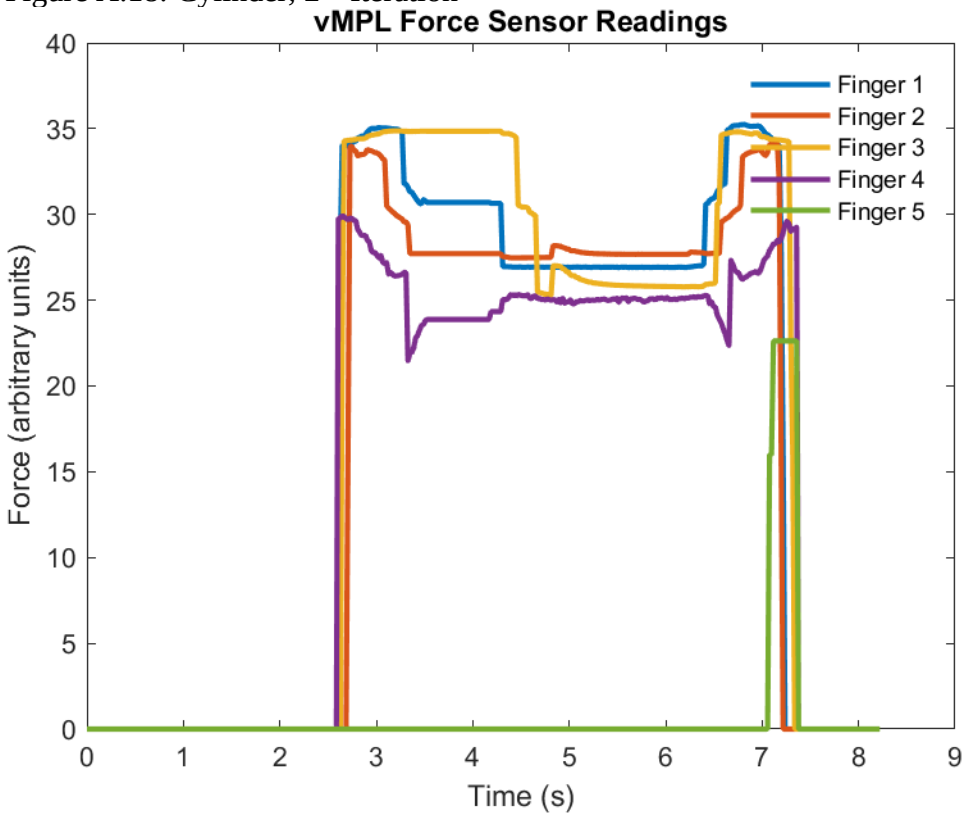


Figure A.19: Cylinder, 3rd Iteration

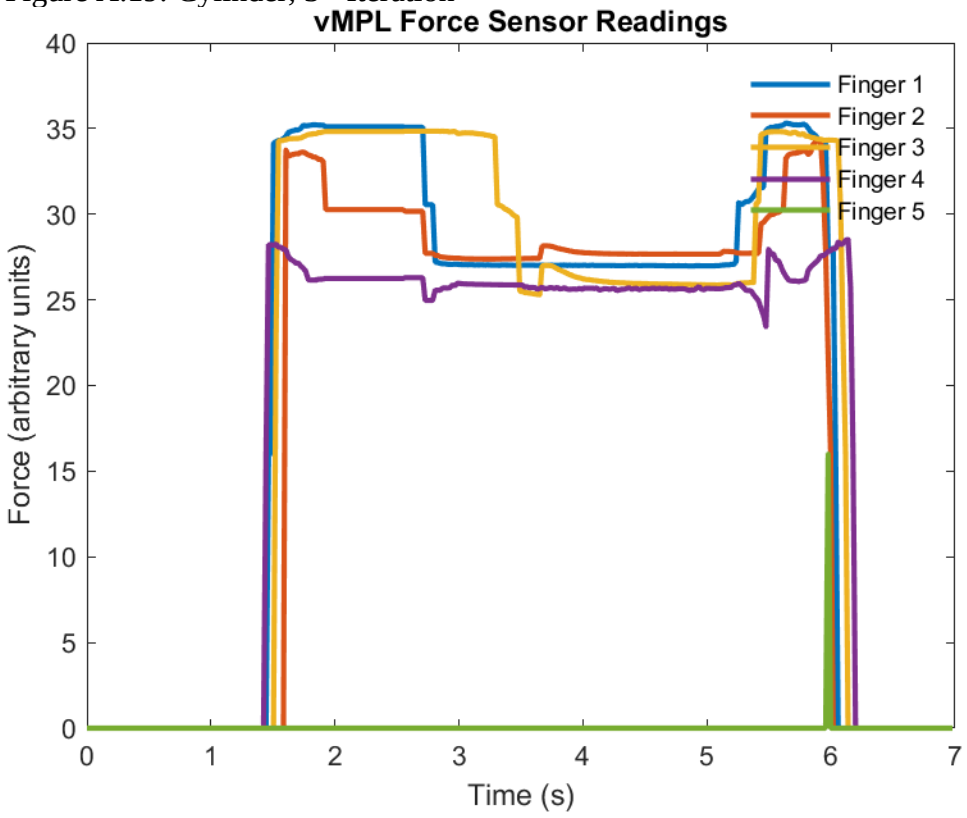


Figure A.20: Cylinder, 4th Iteration

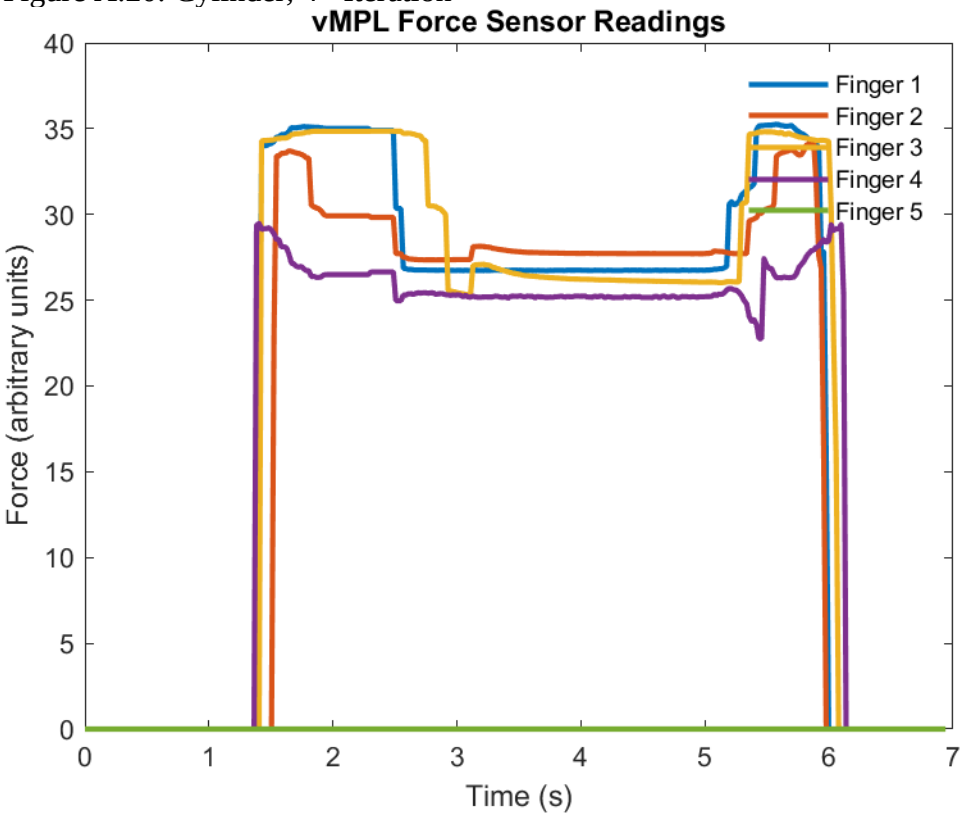


Figure A.21: Pyramid, 1st Iteration

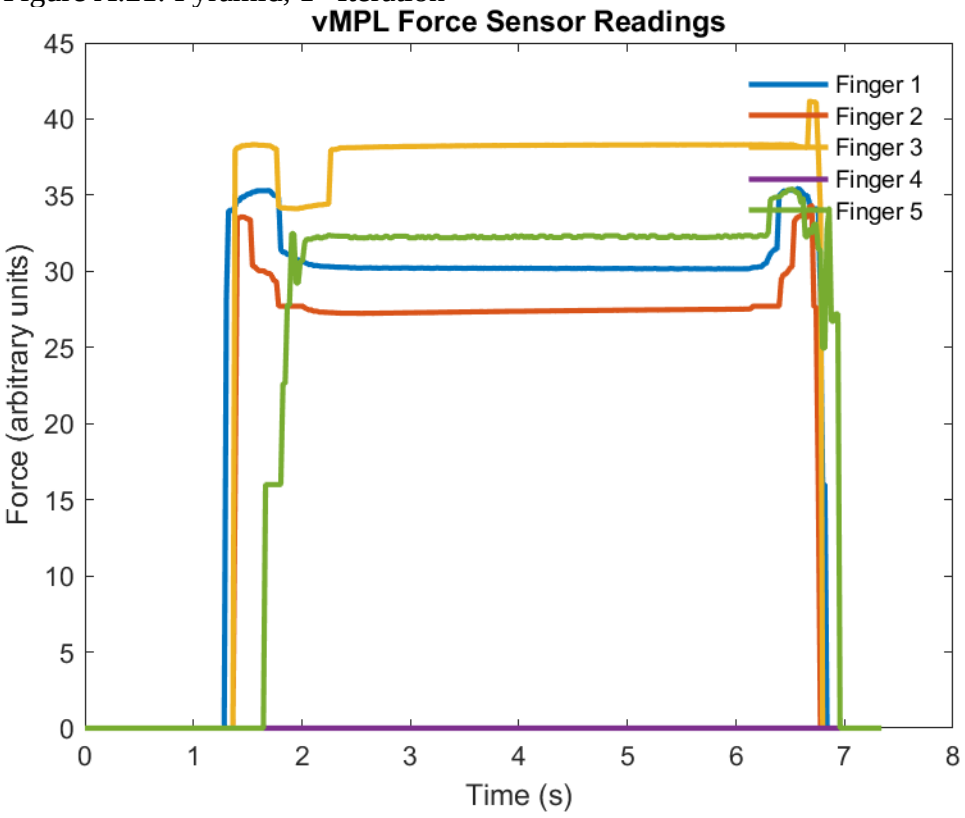


Figure A.22: Pyramid, 2nd Iteration

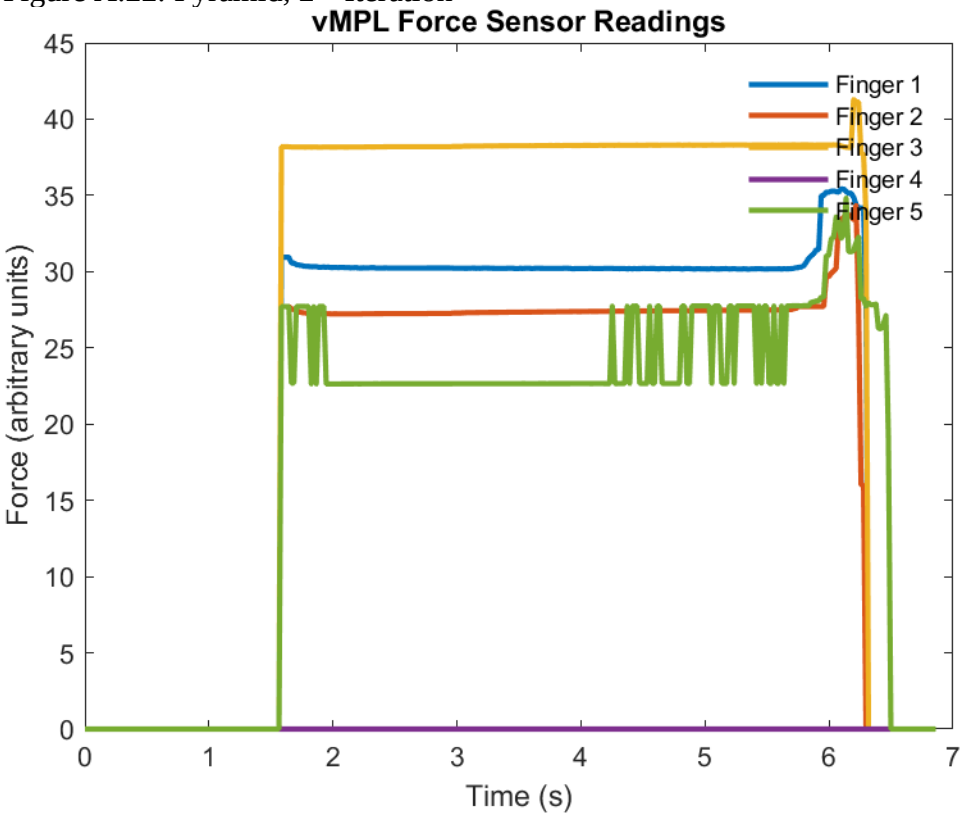
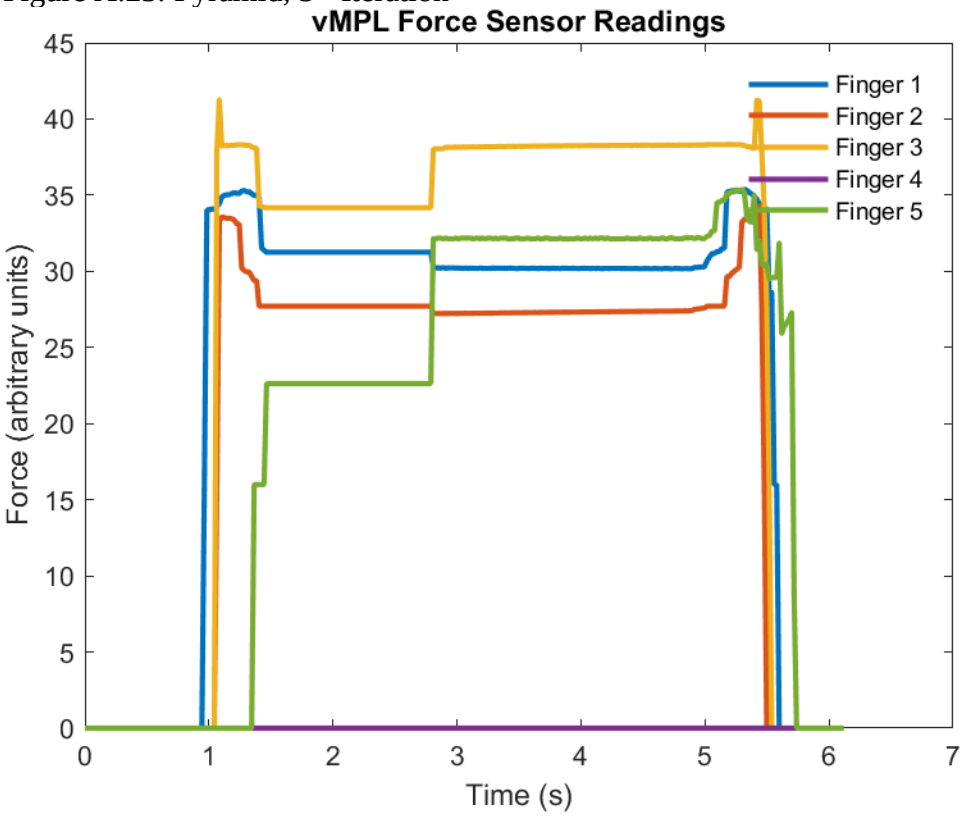
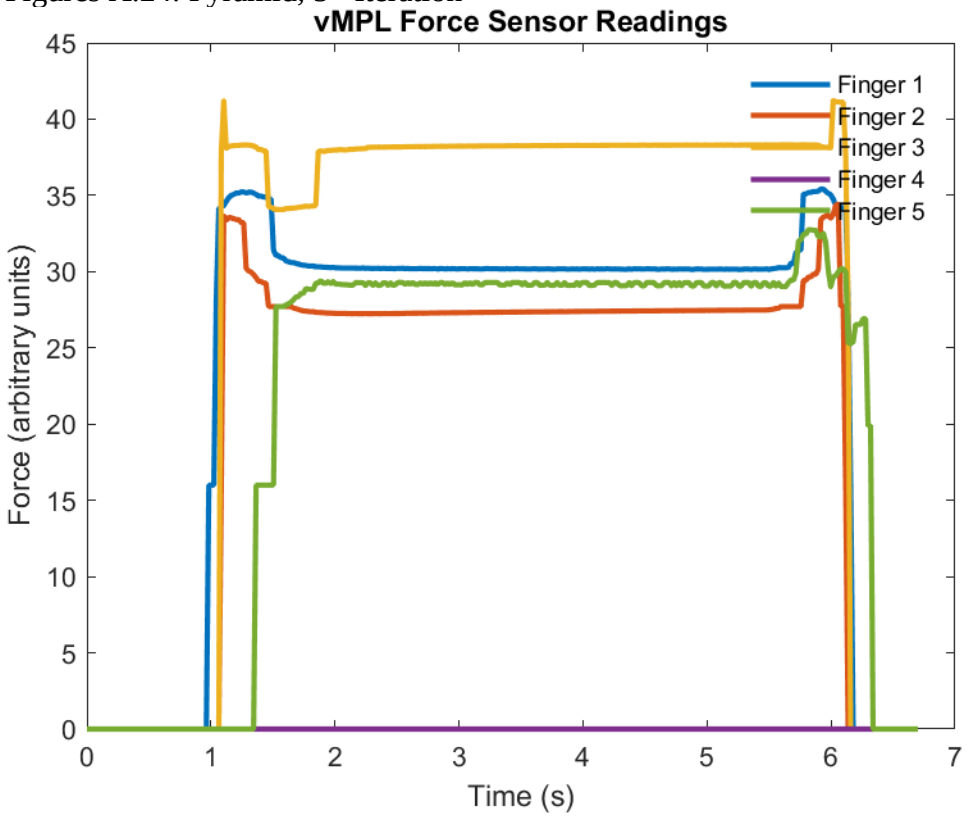


Figure A.23: Pyramid, 3rd Iteration



Figures A.24: Pyramid, 5th Iteration



Appendix B

Table B.1: Machine Learning Set, 1st Iteration (256 Entries)

0	29.225	24.340868	29.60206	0	Sphere
19.49917	29.07267	24.290945	29.27452	0	Sphere
30.94319	28.95115	24.310421	29.0697	0	Sphere
34.90309	29.11067	24.318235	28.93043	0	Sphere
34.59429	28.81735	24.124461	28.73343	0	Sphere
34.43597	28.63647	24.01661	28.61123	0	Sphere
33.8797	28.48746	23.978036	28.50362	0	Sphere
33.80809	28.41378	24.019059	28.44665	0	Sphere
33.76525	28.37431	24.066694	28.41605	0	Sphere
33.73369	28.33228	24.1224	28.39608	0	Sphere
33.70526	28.3016	24.161836	28.42188	0	Sphere
33.70526	28.3016	24.161836	28.42188	0	Sphere
33.70762	28.25071	24.241173	28.46822	0	Sphere
33.70815	28.25074	24.255939	28.48724	0	Sphere
33.71152	32.47154	24.266577	28.49813	0	Sphere
33.71152	32.47154	24.266577	28.49813	0	Sphere
33.71248	32.47336	24.272388	28.51013	0	Sphere
33.71107	32.47786	24.278547	28.52789	0	Sphere
33.71421	32.47799	24.276893	28.53339	0	Sphere
33.71567	32.47928	24.277219	28.53698	0	Sphere
33.71434	32.48068	24.276859	28.54161	0	Sphere
33.71225	32.48162	24.280097	28.54563	0	Sphere
33.71649	32.48216	24.278253	28.55417	0	Sphere

Table B.2: Machine Learning Set, 2nd Iteration (256 Entries)

LITTLE	RING	MIDDLE	INDEX	THUMB	CLASS
33.77032	32.55321	24.143569	28.45986	0	Sphere
33.77168	32.55546	24.149618	28.46651	0	Sphere
33.77168	32.55546	24.149618	28.46651	0	Sphere
33.7743	32.5583	24.157134	28.48145	0	Sphere
33.77954	32.5599	24.163872	28.48595	0	Sphere
33.77484	32.56027	24.165381	28.48822	0	Sphere
33.77708	32.56036	24.167794	28.49199	0	Sphere
33.77785	32.56205	24.163981	28.4976	0	Sphere
33.78251	32.56233	24.167153	28.49987	0	Sphere

Table B.3: Machine Learning Set, 3rd Iteration (256 Entries)

LITTLE	RING	MIDDLE	INDEX	THUMB	CLASS
33.71248	32.46302	24.223504	28.63962	0	Sphere
33.71248	32.46302	24.223504	28.63962	0	Sphere
33.71248	32.46302	24.223504	28.63962	0	Sphere
33.71248	32.46302	24.223504	28.63962	0	Sphere
33.71248	32.46302	24.223504	28.63962	0	Sphere
33.71248	32.46302	24.223504	28.63962	0	Sphere
33.71248	32.46302	24.223504	28.63962	0	Sphere
33.71248	32.46302	24.223504	28.63962	0	Sphere
33.71248	32.46302	24.223504	28.63962	0	Sphere

Table B.4: Machine Learning Set, 4th Iteration (256 Entries)

LITTLE	RING	MIDDLE	INDEX	THUMB	CLASS
0	0	0	0	0	Sphere
0	0	0	0	0	Sphere
0	0	0	0	0	Sphere
0	0	0	0	0	Sphere
0	0	0	0	0	Sphere
0	0	0	0	0	Sphere
0	0	4.8653913	0	0	Sphere
0	0	10.645223	0	0	Sphere
0	0	10.498421	0	0	Sphere

Table B.5: Machine Learning Set, 5th Iteration (256 Entries)

LITTLE	RING	MIDDLE	INDEX	THUMB	CLASS
0	0	19.257619	0	0	Sphere
0	0	19.257619	0	0	Sphere
0	0	19.257619	0	0	Sphere
0	0	19.257619	0	0	Sphere
0	0	19.257619	0	0	Sphere
0	0	19.257619	0	0	Sphere
0	0	19.257619	0	0	Sphere
0	0	19.257619	0	0	Sphere
0	0	19.257619	0	0	Sphere

Appendix C

Figure C.1: Confusion Matrix, 1st Iteration

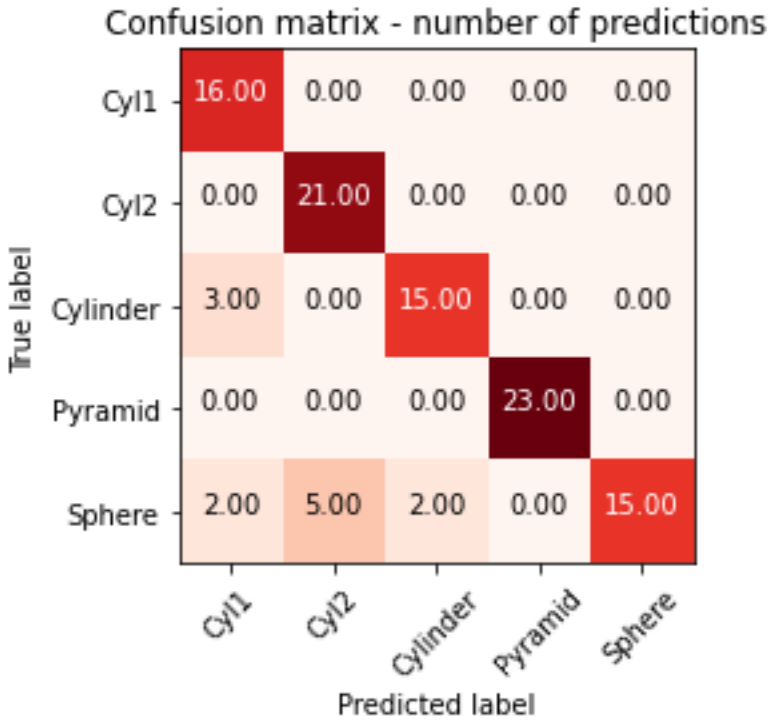


Figure C.2: Classification Report, 1st Iteration

```
Classification Report:
              precision    recall  f1-score   support

        Cyl1       0.76      1.00      0.86        16
        Cyl2       0.81      1.00      0.89        21
    Cylinder       0.88      0.83      0.86        18
     Pyramid       1.00      1.00      1.00        23
      Sphere       1.00      0.62      0.77        24

    accuracy                           0.88       102
   macro avg       0.89      0.89      0.88       102
weighted avg       0.90      0.88      0.88       102

Accuracy: 0.8823529411764706
```

Figure C.3: Confusion Matrix, 2nd Iteration

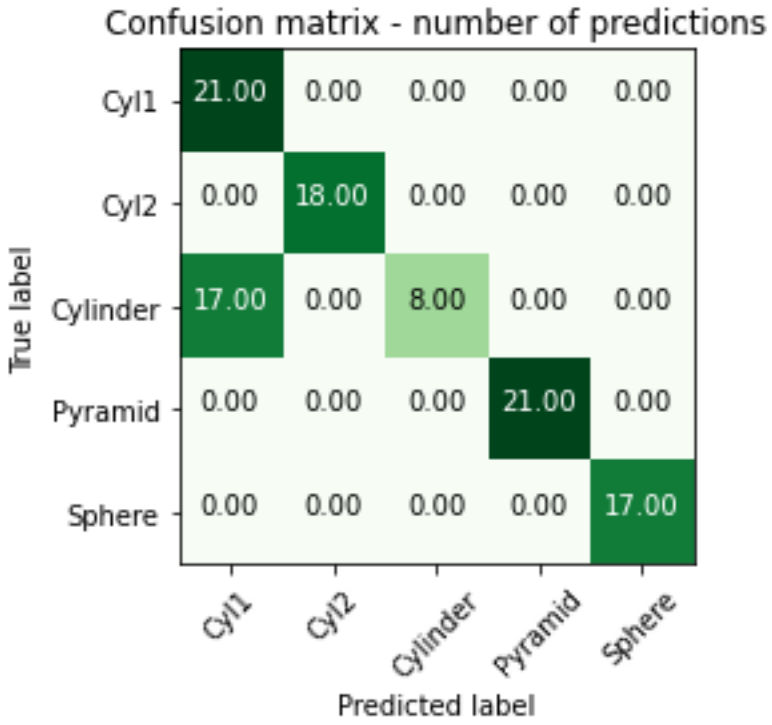


Figure C.4: Classification Report, 2nd Iteration

```
Classification Report:
              precision    recall  f1-score   support

        Cyl1       0.55      1.00      0.71        21
        Cyl2       1.00      1.00      1.00        18
    Cylinder       1.00      0.32      0.48        25
     Pyramid       1.00      1.00      1.00        21
      Sphere       1.00      1.00      1.00        17

    accuracy                           0.83       102
   macro avg       0.91      0.86      0.84       102
weighted avg       0.91      0.83      0.81       102

Accuracy: 0.8333333333333334
```

Figure C.5: Confusion Matrix, 3rd Iteration

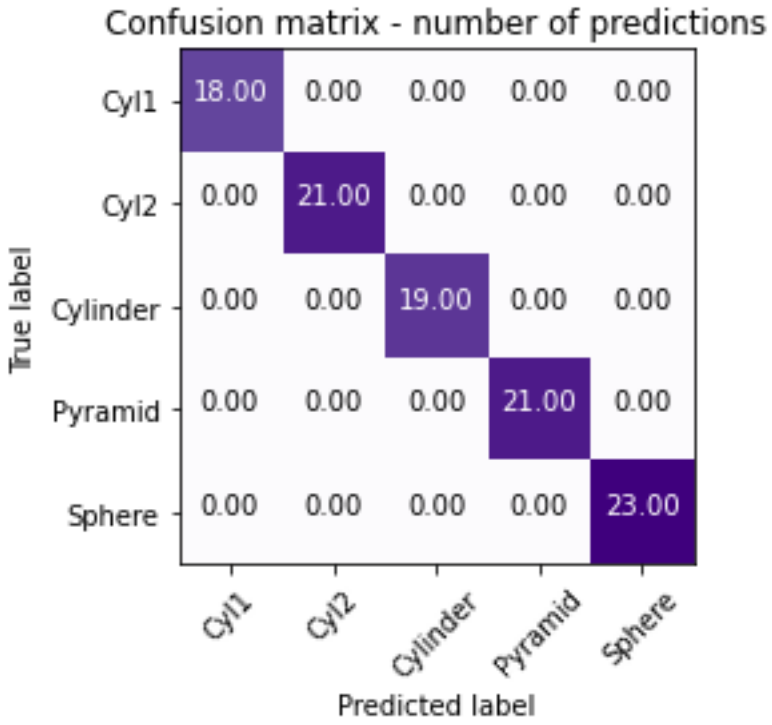


Figure C.6: Classification Report, 3rd Iteration

```
Classification Report:
              precision    recall  f1-score   support

        Cyl1       1.00      1.00      1.00        18
        Cyl2       1.00      1.00      1.00        21
    Cylinder       1.00      1.00      1.00        19
     Pyramid       1.00      1.00      1.00        21
      Sphere       1.00      1.00      1.00        23

    accuracy                           1.00       102
   macro avg       1.00      1.00      1.00       102
weighted avg       1.00      1.00      1.00       102

Accuracy: 1.0
```

Figure C.7: Confusion Matrix, 4th Iteration

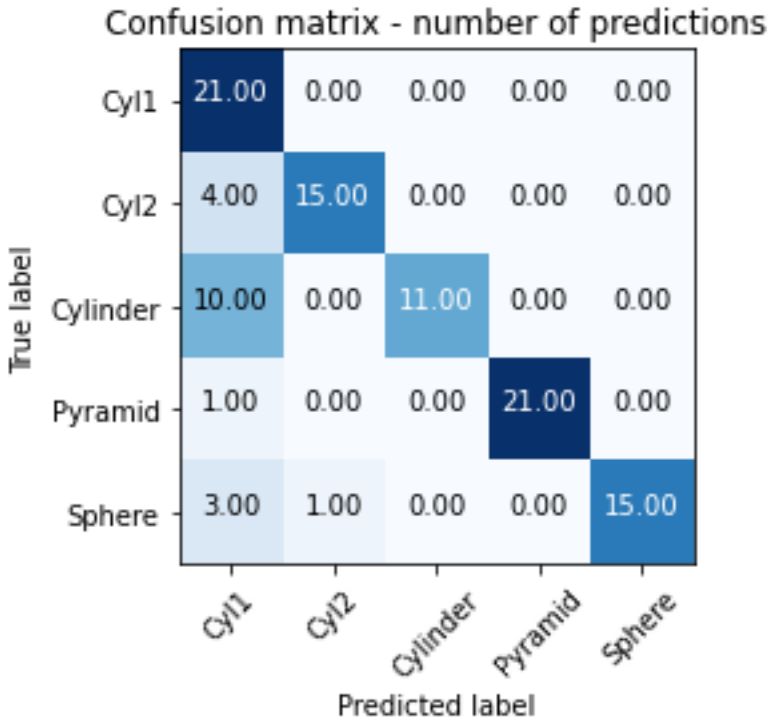


Figure C.8: Classification Report, 4th Iteration

```
Classification Report:
              precision    recall  f1-score   support

        Cyl1       0.54      1.00      0.70        21
        Cyl2       0.94      0.79      0.86        19
    Cylinder       1.00      0.52      0.69        21
     Pyramid       1.00      0.95      0.98        22
      Sphere       1.00      0.79      0.88        19

    accuracy                           0.81       102
   macro avg       0.90      0.81      0.82       102
weighted avg       0.89      0.81      0.82       102

Accuracy: 0.8137254901960784
```

Figure C.9: Confusion Matrix, 5th Iteration

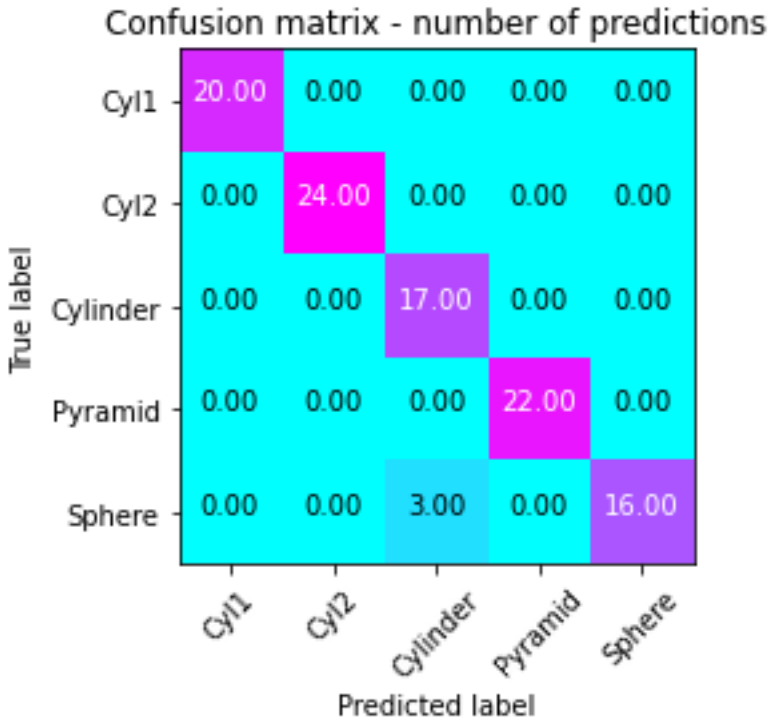


Figure C.10: Classification Report, 5th Iteration

```
Classification Report:
              precision    recall  f1-score   support

        Cyl1       1.00      1.00      1.00        20
        Cyl2       1.00      1.00      1.00        24
    Cylinder       0.85      1.00      0.92        17
     Pyramid       1.00      1.00      1.00        22
      Sphere       1.00      0.84      0.91        19

    accuracy                           0.97       102
   macro avg       0.97      0.97      0.97       102
weighted avg       0.97      0.97      0.97       102

Accuracy: 0.9705882352941176
```