Myo Gesture Control Armband for Medical Applications

16 October 2015

Mahmoud Abduo
`mta128@uclive.ac.nz`

Matthias Galster
`matthias.galster@canterbury.ac.nz`

Department of Computer Science and Software Engineering
University of Canterbury

Abstract. Through studying the relationship between surface electromyography and hand kinematics, hand-amputees may be able to recover a significant part of their lost functionality using noninvasive methods. By investigating the accuracy of the Myo gesture control armband, more people may be able to afford devices which help them recover said functionality. By setting up an experiment using a data acquisition interface to gather surface electromyography to classify movements, the accuracy of the Myo armband can be compared to other devices that have been used in similar benchmark studies. By analyzing the results obtained, the Myo armband may be a viable replacement for other, more expensive devices that can analyze surface electromyography. This project aims to provide an interface to record gestural data from the Myo gesture control armband relative to a scientific benchmark database as well as to provide an extendable platform through which other researchers may conduct further relevant experiments using other devices and sensors.

Keywords—Surface Electromyography; NinaPro; Prostheses; Rehabilitation Robotics; Myo Gesture Control Armband;

Table of contents

1 Introduction

1.1 Context

Prosthetic amputee rehabilitation involves using prosthesis, a man-made device which may replace a missing body part, to recover lost functionality. Research in rehabilitation robotics using myoelectric prostheses that allow for developing and testing movement recognition algorithms has been performed in the past. Results have been recorded to compare various methods on a benchmark scientific database [1-3]. The aim of these methods is to enable trans-radial hand-amputees to regain at least a significant part of their lost hand functionality using non-invasive methods.

Myoelectric prostheses research involves using surface electromyography (sEMG), a non-invasive technique that measures muscle activation potentials through the use of electrodes that can detect such sEMG signals through contact with the skin [4-7]. In the early stages of such research since the 1960s, patients have been limited to one or two degrees of freedom with prosthetic hands using one or two electrodes. These prosthetic hands behaved similar to a simple gripper and were unable to provide enough grasping functionality or the capabilities of a human hand [8].

1.2 Problem

By studying the relationship between sEMG and hand kinematics regarding the positions of fingers, hand and wrist joints, more advanced methods have been developed to enhance robotic hand prostheses [1]. More recently, the use of more electrodes, along with the application of machine learning algorithms has led to more sophisticated control recognizing approximately 10 hand postures [9, 10]. Furthermore, these more advanced methods allow for better determination of the required force, enabling patients with better grip and grasp functionality. Recent advances show that it is possible to detect movement-related potentials of finger movements in trans-radial amputees at an accuracy of up to 90% using 19 electrodes [11]. Precise positioning of the electrodes has also been proven unnecessary by applying pattern recognition techniques [2, 11]. However, it has been determined that the number of electrodes used does affect the overall accuracy of movement recognition [12].

1.3 Motivation

This project aims to expand on the Non-Invasive Adaptive Hand Prosthetics (NinaPro) project started in January 2011 with the aim of controlling prostheses through sEMG channels and to provide a public sEMG database suitable for analysis of sEMG and its application to hand prosthetics [2, 13]. The Myo gesture control armband is a commercially available product that is able to detect sEMG signals and spatial data regarding the orientation and movement of the user's arm. The Myo armband is relatively cheaper than other sEMG electrodes such as Ottobock and Delsys electrodes. By investigating the performance of the Myo relative to data that is currently available from the NinaPro database, the Myo may be a viable replacement for more expensive devices used previously.

In an attempt to further encourage research in this field that supports the motives of the NinaPro database, this project aims to provide an open source solution which various devices can be integrated into. This solution would take care of all experimental procedures required such as which gestures need to be recorded and how they would be stored. Ultimately, researchers and developers would need to integrate their respective drivers into this solution given an easy to use API and they would be ready to go.

In this report, some background information is provided describing the previous experiment which this project is based on and relevant information regarding the Myo gesture control armband. This is followed by a brief overview of the solutions before going into detail about each solution. For each solution, an outline of the software engineering process is described, including requirements engineering, technical design, implementation, testing and evaluation. For the first solution, an additional section about the experimental setup to record Myo gestural data is described. A retrospective regarding the overall project progress and some suggestions for future extensions are then provided.

2 Background

2.1 Related work

The highlighted research that this project is based on was directly involved with the initial setup of the NinaPro database [1]. As a result of their efforts, the NinaPro database currently contains gestural data from 67 intact subjects and 11 amputee subjects. Using a standardized set of gestures, their aim was to set up a scientific benchmark database where algorithms for movement recognition and force control could be tested.

The data acquisition setup consisted of several sensors that were integrated to record hand kinematics, dynamics and the corresponding muscular activity. These were recorded using a CyberGlove II dataglove, a Finger-Force Linear Sensor and either OttoBock or Delsys double-differential sEMG electrodes respectively. Intact subjects involved with the data acquisition experiments needed to wear these devices to record gestural data. During experimental set up, intact subjects were asked to mimic the gestures shown in a video as accurately as possible. On the other hand, amputee subjects were asked to mimic the movements with their missing limb as naturally as possible. Even though they could not see any physical movement, they were asked to not exert as much force possible to keep the movements natural.

Since the different devices published data at different frequencies, the data obtained from the experiments was then synchronized, relabelled and filtered. The data was then tested for gesture classification accuracy across 50 movements. For intact subjects, the highest average classification accuracy was approximately 75%. On the other, the highest average classification accuracy was at approximately 46%.

2.2 Dexterous Control

To expand on research performed previously, the hand movements intended to be performed in this experiment are based on the movements performed in the NinaPro acquisition protocol. These movements are based on movements from the robotics and the taxonomy literature and from the DASH (Disabilities of the Arm, Shoulder and Hand) protocol for functional movements [1-3, 13, 14].

These movements are divided into four main classes:

- 12 basic finger movements (exercise A)
- 8 static hand postures (exercise set B)
- 9 basic wrist movements (exercise set B)
- 23 grasp and functional movements (exercise set C)

The set of movements can be seen in Figure 1.

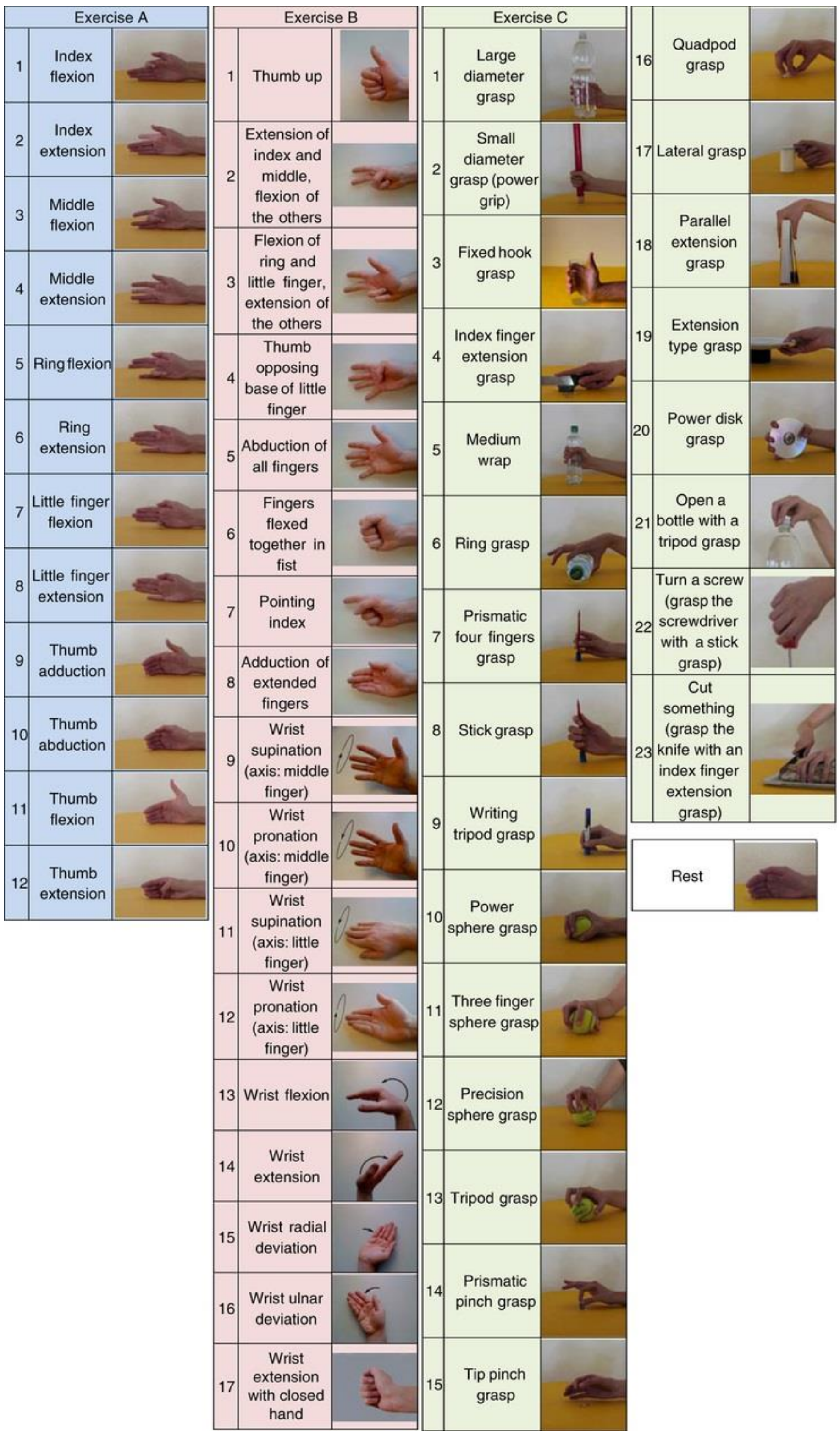

Exercise A
1 Index flexion
2 Index extension
3 Middle flexion
4 Middle extension
5 Ring flexion
6 Ring extension
7 Little finger flexion
8 Little finger extension
9 Thumb adduction
10 Thumb abduction
11 Thumb flexion
12 Thumb extension
Exercise B
1 Thumb up
2 Extension of index and middle, flexion of the others
3 Flexion of ring and little finger, extension of the others
4 Thumb opposing base of little finger
5 Abduction of all fingers
6 Fingers flexed together in fist
7 Pointing index
8 Adduction of extended fingers
9 Wrist supination (axis: middle finger)
10 Wrist pronation (axis: middle finger)
11 Wrist supination (axis: little finger)
12 Wrist pronation (axis: little finger)
13 Wrist flexion
14 Wrist extension
15 Wrist radial deviation
16 Wrist ulnar deviation
17 Wrist extension with closed hand
Exercise C
1 Large diameter grasp
2 Small diameter grasp (power grip)
3 Fixed hook grasp
4 Index finger extension grasp
5 Medium wrap
6 Ring grasp
7 Prismatic four fingers grasp
8 Stick grasp
9 Writing tripod grasp
10 Power sphere grasp
11 Three finger sphere grasp
12 Precision sphere grasp
13 Tripod grasp
14 Prismatic pinch grasp
15 Tip pinch grasp
16 Quadpod grasp
17 Lateral grasp
18 Parallel extension grasp
19 Extension type grasp
20 Power disk grasp
21 Open a bottle with a tripod grasp
22 Turn a screw (grasp the screwdriver with a stick grasp)
23 Cut something (grasp the knife with an index finger extension grasp)
Rest

Fig 1. 52 gestural movements based on the DASH protocol.

2.3 Myo Gesture Control Armband Overview

The Myo armband, Figure 2, has eight medical grade stainless steel EMG sensors. Similar to other surface electrodes, the EMG signals returned by the sensors represent the electric potential of the muscles as a result of muscle activation [15]. However, since the electric potential of muscle is small, in the range of sub millivolts, signals are sensitive to other sources of electric noise such as electric noise induced by wall-electricity. The range of potentials provided by the Myo armband is between -128 and 128 in units of activation [16]. These units of activation are integer values of the amplification of the potentials measured by the EMG sensors. The Myo armband is capable of pulling sEMG data at a sample rate of 200Hz.

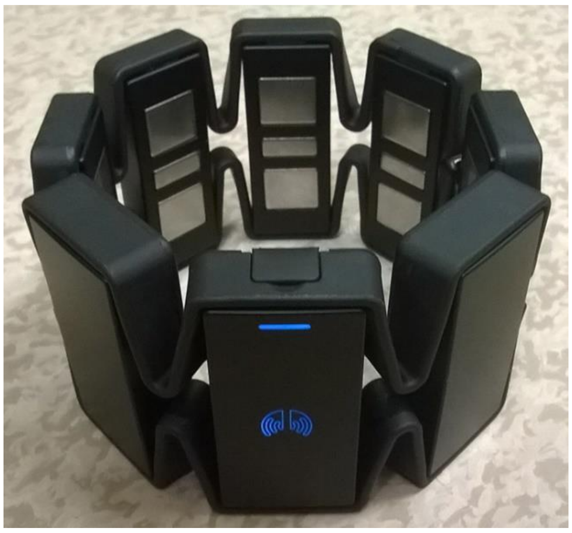


Fig 2. The Myo gesture control armband.

The Myo armband also has a nine axis inertial measurement unit (IMU) which contains a three axis gyroscope, three axis accelerometer and a three axis magnetometer [16]. From these units, the orientation and movement of a wearer's arm can be determined through analysing the spatial data provided. The orientation data indicates the positioning of the armband in terms of roll, pitch and yaw. The angular velocity of the armband is provided in a vector format and the accelerometer represents the acceleration the Myo armband is undergoing at a given time. However, the Myo armband is better suited for determining the relative positioning of the arm rather than the absolute position, a consideration to be aware of when applying pattern recognition algorithms. Currently, the Myo armband is able to pull IMU data at a sample rate of 50Hz.

The Myo armband has been made to work best at the widest part of the forearm, that is, the upper forearm [16]. Sizing clips are available which allow for a more constrained grip, better suited for smaller arms. Unlike other EMG sensors, the Myo armband does not require the wearer to shave the area around which the armband will be worn. This allows for easier setup procedures in experimental or real world environments.

3 Solutions Overview

3.1 Data Acquisition Interface using the Myo Gesture Control Armband

The main aim of this solution was to set up an interface as quickly as possible to gain gestural data from a number of participants. This data was intended to be passed on to separate research group that would use the data as a training set for a machine algorithm. The algorithm would then be tested with performing gestures and the accuracy of the Myo armband for detecting gestures could be measured based on how many gestures the algorithm successfully recognises. If the accuracy of the Myo armband was deemed accurate enough, further investigation into the hardware used to build the Myo armband to potentially build robotic hand prostheses which can be word by amputees. The advantage that the Myo armband technology would provide is a much lower cost of purchase compared to other sEMG detecting hardware.

3.2 Extendable sEMG Interface

Based on the limited amount of data that was gathered during the first phase of this project, the project scope shifted to focus on creating an extendable platform, i.e. an open source solution, which can be utilised by other researchers and developers who which to carry out similar experiments to gather sEMG and other types of data from gestures.

3.3 Technology Stack and Development Tools Overview

All solution development and automated testing was done in Visual Studio within the .NET 4.5 framework using C# and XAML. The choice of programming language was based on what the previous interface was built with and what the other developers are familiar with. This also means that the solutions developed will run only on Windows operating systems that support .NET 4.5, in other words, Windows Vista or higher [17]. Version control was accommodated for using Microsoft's Team Foundation Server.

Database output files containing recorded gestural data was analysed and validated using Matlab. For setups that aim to obtain gestural data using the Myo armband, the armband and a USB Bluetooth for connection with the computer are required. Other libraries used while developing included:

- MyoSharp, an open source C# wrapper for the Myo armband [18]. The Myo SDK by default does not contain support for C# development. The underlying Myo components are written in C++. Fortunately, C++ code can be invoked using C#. The MyoSharp wrapper provides access to Myo SDK functions for C# applications. The functions include identifying a connected Myo armband and intercepting events that include sEMG and IMU data.
- CSMatIO, a .NET library which allows developers to read and write Matlab files using C# [19]. This was used to store recorded gestural data in the same format that is provided in the Ninapro database.

4 Solution 1: Data Acquisition Interface using the Myo Gesture Control Armband

The data acquisition software that was built was based on a similar interface that was used in a previous experiment where sEMG, orientation, accelerometer, incline and force data was acquired using a setup that involved a combination of double differential sEMG electrodes, a motion capture data glove and a Finger-Force Linear Sensor [1, 20]. The initial attempt was to extend this software to include support for recording data available from the Myo armband. However, due to the complexity involved with the setup of the other machines' drivers and an unfamiliar codebase, it was deemed simpler and quicker to build a separate interface from scratch, dedicated to recording data from the Myo armband.

This interface was then used in an experimental set up to obtain gestural data from a number of participants that volunteered. The aim was to input the data into pattern recognition algorithms to classify movements as a result of sEMG records. The number of correct movement classifications from live sEMG data, input in real time, would have been used to determine the accuracy of the Myo armband compared to other sEMG sensors. Since this is outside of the scope of the project, the data will be provided to those involved with the NinaPro database for further analysis.

4.1 Requirements Engineering

The goal of this solution was to set up a data acquisition interface that would enable the collection of gestural data from the Myo armband based on the experimental set up from previous work in this area [1, 2, 13, 21]. The data acquisition interface that was used previously was acquired, including the source code and output files. Output files were necessary since there was no access to the original sEMG sensors that were used. For analysis of the interface during run time, screenshots and videos were requested. Figure 3 shows the original data acquisition interface with some of the sensors used along with labels that identify various parts of the user interface.

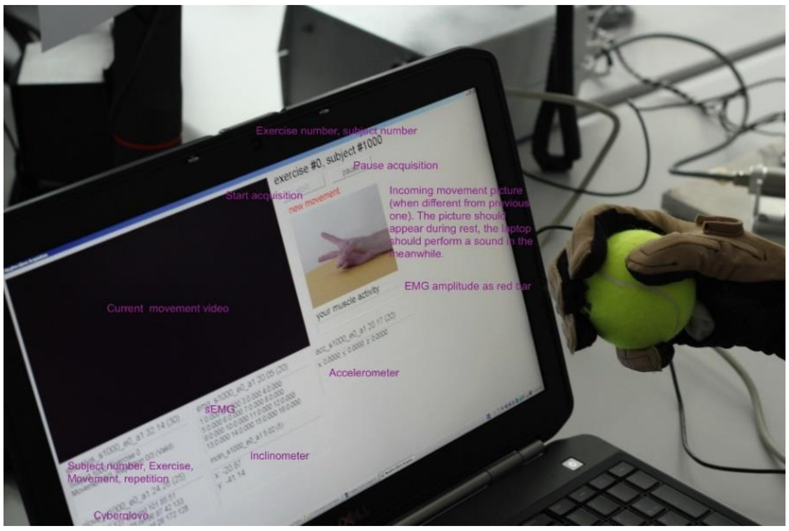

Exercise number, subject number
exercise #0, subject #1000
Pause acquisition
Start acquisition
new movement
Incoming movement picture (when different from previous one). The picture should appear during rest, the laptop should perform a sound in the meanwhile.
EMG amplitude as red bar
your muscle activity
Current movement video
Accelerometer
sEMG
Inclinometer
Subject number, Exercise, Movement, repetition
Cyberglove

Fig 3. The previous data acquisition software running with a user wearing a motion capture data glove.

One of the aims of obtaining the preview of the interface and output files was to gain an idea of the values measured by those machines, their respective units of measurement and how they were recorded. Multiple sensors were used in the original experiment. Since the Myo armband is unable to obtain the same amount of information, the interface implemented had to be adjusted to provide a similar feel to the other interface for consistency.

Interface aside, there was still the matter of implementing the required behavior of the interface. This was a challenge since there was no official documentation regarding the previous system requirements and the source code was poorly documented. The initial design of the interface was based on assumptions that allow for a participant to view the various gestures in a given order and to record the data output from the Myo armband. Given the knowledge at the time, the main highlights of the interface behavior can be summarized as follows:

- Participants should be shown one gesture at a time
- Participants should see an image and a video of the gesture that needed to be performed
- The video should play during recording time so that the user can mimic the gesture they see
- Participants should control the starting and stopping of each recording
- Participants should be able to record multiple repetitions of each gesture
- Participants should be able to navigate between various exercises

However, working with assumptions was not sufficient. To ensure that the interface behavior works as expected for experimental purposes, it was sent to be reviewed by the original creators of the interface. This included both the source code as well as a video that demonstrated the various features of the interface during use. As expected, a number of clarifications and corrections were provided by the end of the review and the interface needed to be adjusted as such.

It turned out that the interface should have provided a more automated experience where once a participant starts recording, the interface will take care of what gestures showed on the screen and it will keep track of the number of repetitions before moving to the next gesture. It also needed to allow for some rest time between each gesture. The functionality to pause recording was purely to stop recording if some unexpected event happened or if certain experimental variables needed to be adjusted, for example, which objects are involved with a particular grasping gesture.

It was also suggested that the interface should synchronize the different types of data obtained from the Myo armband. This was necessary since different types of data from the Myo armband were obtained at different frequencies, that is, the sEMG data was obtained at 200Hz and the remaining data, such as accelerometer, gyroscope and orientation data, was obtained at 50Hz. The recommended solution from the researchers involved in the previous experiment was to linearly interpolate the lower frequency data so that it matched the sample rate of the sEMG data.

4.2 Technical Design

4.2.1 Software Architecture

The main purpose behind the creation of this interface was to have a tool for recording gestural data in an experimental setup. For this reason, little though was put into setting up an extendable architecture and the focus was on implementing the minimum features necessary to start collecting data. To make use of data binding, a design option provided by the .NET framework that separates user interface code from the underlying models, the solution was built within a Model-View-ViewModel (MVVM) software architecture. A brief outline of the basic architecture can be seen in Figure 4.

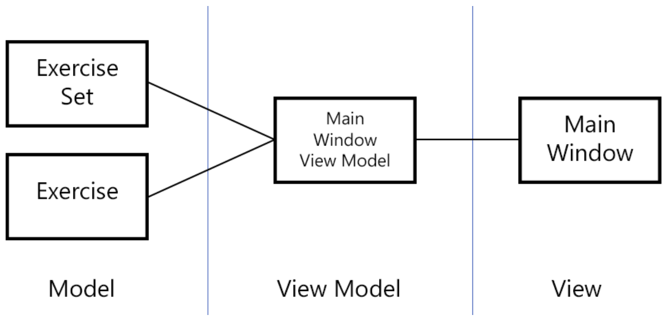


Fig 4. Model View View Model Architecture of the data acquisition software.

Initially, all of the interface' s logic, including the Myo data gathering, display of data on the interface, the start and pause functionality and the storing of data to disk functionality were inside a single view model that related to the single view. However, as the functionality of the interface began increasing with the increasing requirements, it was deemed better to extract some of the functionality into its own separate modules to avoid unnecessary complexity within a single class file. For example, this helped reduce complexity when implementing the linear interpolation of data mentioned previously since it was no longer grouped with the same functionality responsible for collecting data.

4.2.2 Asynchronous Design

In order to maintain responsiveness and constant activity of the interface, asynchronous functions were used to perform various background tasks such as resting and storing the data to disk. In other words, the interface display and its buttons needed to remain active during the rest period. Furthermore, since sEMG signals were also recorded during the rest period, the recording functionality needed to remain functional while the interface was resting or storing data to disk. This was also assisted by using an event based implementation where Myo data was stored to temporary memory as different Myo events produced sEMG and other data. Similar was the case where the temporary data recorded needed to be stored to disk. Because reading and writing files to disk, as well as linearly interpolating data before writing, could take some time, it was determined that this could be performed in a background thread such that the ongoing functionality of the interface could continue without interruption.

Another requirement of the interface was to play a beep sound when the interface automatically went to the next exercise during recording. Asynchronous functionality was used when implementing the beep sound functionality so that beep played in the background and did not stall the main thread in which the interface was running.

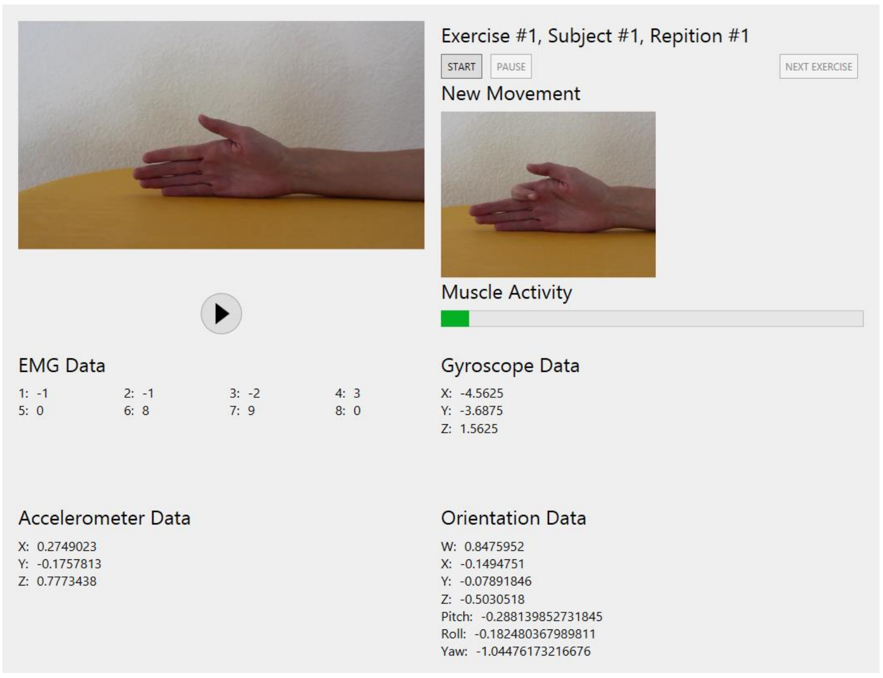
4.2.3 Database

Data was recorded straight into Matlab database files that are stored in the current in a Myo database folder created within the logged user's documents folder. Each participant has their own folder and within that folder, three Matlab database files are created as gestures are recorded for each of the three exercise sets.

The database file output was designed according to the output provided by previous experiments and the format suggested on the NinaPro website. Each database recorded contains the following variables:

- subject: the subject number
- exercise: the exercise set number
- acc (3 columns): the accelerometer data of the Myo armband in x,y,z vector form
- emg (8 columns): the sEMG signal of the eight EMG sensors
- gyroscope (3 columns): the angular velocity represented in x,y,z vector format
- inclin (3 columns): the orientation data represented as pitch, roll and yaw
- orientation (4 columns): the orientation data represented in a w,x,y,z format
- stimulus (1 column): the movement performed by the subject at a given time
- repetition (1 column): The repetition number of the movement performed by the subject at a given time
- emgTimestamp (1 column): The timestamp of the data row recorded relative to the emg variable

4.3 Implementation


Exercise #1, Subject #1, Repition #1

New Movement

Muscle Activity

EMG Data

```
1:  -1        2:  -1        3:  -2        4:  3
5:  0         6:  8         7:  9         8:  0
```

Gyroscope Data

```
X:  -4.5625
Y:  -3.6875
Z:  1.5625
```

Accelerometer Data

```
X:  0.2749023
Y:  -0.1757813
Z:  0.7773438
```

Orientation Data

```
W:  0.8475952
X:  -0.1494751
Y:  -0.07891846
Z:  -0.5030518
Pitch:  -0.288139852731845
Roll:  -0.182480367989811
Yaw:  -1.04476173216676
```

Fig 5. Myo armband data acquisition software.

As can be seen from Figure 5, the interface displays all the data that can be obtained from the Myo armband in the bottom half. In the upper half, the interface displays a video and image of the exercise that a participant will be required to perform. The software contains three predefined sets of exercises. Recording functionality is also provided so that sEMG and IMU data can be recorded against the current exercise and repetition performed. Once enough repetitions of an exercise have been recorded, the next exercise in the selected exercise set is automatically loaded and the recording procedure is repeated. Participants have a three second rest period between exercises. Data recorded is stored in a database file for each subject and exercise set performed. Various tables are used to record sEMG, accelerometer, orientation and gyroscope data.

4.4 Testing

4.4.1 Synchronization testing

Successful synchronization of the different Myo data was tested after some test data had been recorded. To be able to perform this testing, the interface was used to record a number of different exercises with resting recordings as well as recordings that have been paused and resumed. As mentioned in the previous progress report, the file that was stored to the disk was in a Matlab format. One of the advantages of storing data in this format is that Matlab can be used to perform data analysis and testing on recorded data.

To test the data, a simple Matlab script was prepared to automate the testing of the data. The script first checked that the number of signals matched between the 200Hz data and the interpolated 50Hz data. Also, to perform more intensive testing, the synchronization reference was also stored alongside the test data. This provided the capability to check whether the number of each synchronization reference matched between the high frequency data and the interpolated data, ensuring that data had been interpolated into the correct number of resulting data. It also allowed for checking the mathematical correctness of the data produced to fill in the gaps within the low frequency data.

4.4.2 Multiple Myo Comparison

For an early test of the accuracy and reliability of the Myo gesture control armband, a second Myo armband was added to this project. This was to ensure that data among the two armbands was consistent so that the data obtained from the upcoming experiments could be relied on for research purposes.

The initial testing of the two Myo armbands consisted of testing them across Myo applications that are available through the Myo website [16]. Both Myos were capable of recognizing the basic gestures built into the Myo SDK such as finger spreading, wave in, make fist and so on. This was followed by a basic method of testing to compare the signal ranges from each sEMG electrode on the Myo armbands. On average, the range difference between the recorded results was ±5 units on a scale of 256, in other words, approximately ±2% uncertainty between the two armbands for each of their corresponding sEMG electrodes. This allows for some confidence in the consistency of the Myo armband as the project proceeds.

4.4.3 Pilot Testing

This was carried out to ensure that the interface behaved as expected and did not crash unexpectedly. This included testing recording and data storage functionality, resting functionality, pausing functionality as well as checking that the exercises were shown in the correct order and that the correct number of repetitions were being performed.

After fixing a number of bugs and performance issues detected as a result of this testing, a few pilot tests were performed to ensure that data collection under normal circumstances would be successful. After pilot testing, the interface was tested in real world conditions by involving real participants.

4.5 Experiment

4.5.1 Procedure

To compare the performance of the Myo armband to other benchmark studies that attempt to channel sEMG to determine the intention of a user's movement, a sample of gestural data was gathered from a number of volunteers.

After signing a consent form, participants were asked to fill out a questionnaire that asked them for their first and last names, age, gender, height, weight, laterality, profession, hobbies related to their hand use and their self-reported health status at the time of acquisition.

Participants were also asked to perform a short test in the form of a questionnaire which provided a DASH score, a measurement which clinicians can use to assess joints in the upper extremity [22].

Participants were then requested to wear the Myo armband on the widest part of their right forearm. Sizing clips were applied as necessary to ensure that a firm and comfortable fit is achieved. The participant were then asked to perform a waving gesture to ensure that the armband is synchronized with them.

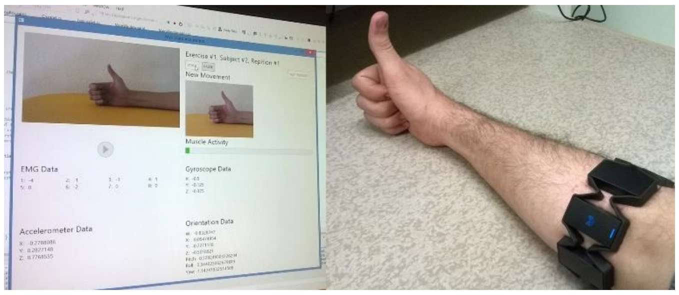


Fig 6. Mimicking of exercises presented by the data acquisition interface while wearing the Myo armband.

The Myo armband requires five minutes of warm up time to form a strong connection to the muscles in an arm for optimal results [16]. During this warm up time, participants were asked to perform a set of training exercises to better understand how the recording procedure is performed. The video of the gesture showed them how the movement has to be performed and consequently, they performed the movement with their arm as seen in Figure 6. After completing the training exercises, the participant were asked to perform the three sets of exercises as highlighted above. Participants were allowed short break periods between the training period and exercise sets to avoid muscle fatigue and to reduce its potential influence on the sEMG signal.

4.5.2 Observations & Adjustments

While carrying out the experiments, it was apparent there were a number of experiment instructions that needed to be clarified to the participants to ensure that the correct movements were being performed. For example, the rest position between various exercises needed to be explained to participants because some of them assumed various positions that were either different or uncomfortable for them. It was also a good idea to show a demonstration of the gestures that needed to be performed in the third exercise set where various external objects were used.

The user interface also contained a muscle activity bar which showed how much intensity a participant was applying with their gesture. For example, if the arm is at rest, the muscle activity is low. On the other hand, if the participant was clenching their fist tightly, the muscle activity would be very high. One problem observed was when participants noticed this and occasionally attempted to fill the bar to the max by intensifying their actions. This strongly influences the sEMG signal being recorded since there is a lot more noise as a result. As far as the accuracy of the Myo is concerned, this can be seen within applications that have been made to work with the Myo where a tensed finger spread gesture registers as a fist gesture. This issue was brought back to original researchers involved in the previous experiment and it was explained that the muscle activity bar is necessary, especially in the case of amputee subjects to give an indication that their actions are being registered accurately. As a result, a necessary instruction to the participants was to perform the gestures in a relaxed and natural manner.

The Myo armband requires a connector application to ensure that the Bluetooth connection is set up. This connector application also comes along with a number of other applications that allow a Myo user to control various aspects of the computer and applications installed on it. It is important that all of these extra features are disabled before carrying out the experiment. The connector program also requires that a user synchronize the Myo armband before continuing use. It is important to note that the Myo armband will stop recording data if it is not synchronised with the user in an attempt to conserve battery life.

4.6 Evaluation

For its intended purpose, the interface developed successfully retrieves sEMG and IMU data as intended. In total, four participants were involved with gathering gestural data. After removing any information that may lead to the identification of the participants involved, the data is ready to be published on the NinaPro database. The original target was at least ten sets of data, however, this was not a simple feat since it was not easy to persuade people to spend one hour performing gestures with nothing in return. It is recommended that anyone who wishes to further this research would provide some sort of motivator or compensation to the participants to obtain more data sets.

From a software engineering point of view, although the end result achieved in this stage of the project is fairly basic, it has proved to be very useful in the next stage where it is extended to support other sEMG sensors.

5 Solution 2: Extendable sEMG Interface

After halting the efforts to obtain gestural data from volunteers, the focus of the project shifted to creating an extendable solution that may be programmed to work with various sEMG sensors. The underlying purpose behind this is to create a platform that is available to other researchers who may wish to carry out this experiment with little effort required to set up a data acquisition interface. Based on current experience with the Myo gesture control armband and applications that have been developed to work with its small gesture set, it is unlikely that the technology used in the Myo armband will be sufficient to replace more expensive sensors. This was noticed when trying to perform simple tasks such as navigating between the presentation slides where the Myo armband failed to register simple gestures such as the tapping of fingers.

5.1 Requirements Engineering

From an experimental point of view, the behaviour of this interface needs to behave in a similar manner to that which was developed in the first solution. This includes the setup of the three exercise sets, the navigation between gestures and the recording of various gestural data. There is a higher emphasis on sEMG data since that is the main factor involved in determining gestures, but the solution should be extendable to support other types of data that may be available for collection.

By default, the data acquisition interface is not able to identify what kind of device is attached to a computer nor how to intercept signal events. To make it simple for extension, developers should only need to provide a mapping between whichever data types that are retrieved by a specific device and what data types the interface may accept. This suggests some form of API that that developers are able to utilize in order to notify the interface when a data signal is received.

Secondarily, developers should be able to extend the data types that can be recorded by the interface. The second solution currently provides an API for receiving sEMG signals of any number as well as accelerometer, orientation and gyroscope signals. These are no longer coupled as was the case in the first solution since devices may provide some or all of these signals. Since devices may also have provide other types of data, the design of the extendable interface should be modular with little to no coupling between various data type modules.

At the end of this, documentation is provided to guide future developers through the process of extending this interface for support with other devices.

5.2 Technical Design

5.2.1 Modular Design: Dependency Injection

A much higher emphasis was placed on ensuring that the software design was modular and loosely coupled so that developers could tweak various data recognition modules without worrying about any of the other behavioural aspects of the software. Compared to the first solution which consisted of approximately ten files, the second solution has approximately seventy files excluding resource files.

A number of frameworks were explored during the technical design phase including Microsoft's Prism framework which provided vast support for modular MVVM architectures. Eventually, the Unity Application Block as recommended by Microsoft's Patterns and Practices was used [23]. The software architecture was built around the infrastructure provided by the Unity Container, a lightweight, extensible dependency injection container. Dependency injection allows for simpler creation of objects by abstracting requirements. By defining a set of mappings between interfaces and concrete objects, objects only need to define which objects it needs and not how to construct them.

For example, if the exercises view model in the data acquisition interface need a recording service, it defines the recording service interface it needs in the constructor as seen in Figure 7. The Unity Container then takes care of setting up the exercises view model and passing it a concrete recording service. This becomes especially useful if the constructor of the concrete recording service required an interface to the file storage service. Similarly, when constructing the recording service, the Unity Container takes care of providing it with a concrete implementation of the file storage service.

```
public ExercisesUserControlViewModel(IEventAggregator eventAggregator,
    IExerciseService exerciseService,
    IRecordingService recordingService)
{
    _eventAggregator = eventAggregator;
    _exerciseService = exerciseService;
    _recordingService = recordingService;
    …
}
```

```
public class RecordingService : IRecordingService
{
    public RecordingService(IEventAggregator eventAggregator,
        IFileStorageService fileStorageService)
    {
        _eventAggregator = eventAggregator;
        _fileStorageService = fileStorageService;
        …
    }
}
```

Fig 7. The ExercisesUserControlViewModel constructor and the recording service interface it requires (top). The concrete RecordingService which implements the interface that the ExercisesUserControlViewModel is expecting.

This kind of construction can be consistently seen across all view models and services that have been implemented in this solution. As a result, concrete implementations of view models and services do not interact with each other directly, minimising coupling between components.

5.2.2 Modular Design: Publish/Subscribe

Because of the nature of event interception between the data acquisition interface and the various devices that it may be receiving signals from, a communication infrastructure needed to be implemented. One method to implement this is to use the observer pattern. The observer pattern typically consists of an observer object attaching itself to a subject and waiting for a notification from the subject when a change occurs to the subject. This is well suited in one to many relationships, but the observer pattern may lead to high coupling and complicated implementations when many to many relationships are required, as often is the case in application development.

The publish/subscribe pattern allows for the setup of communication channels between various senders of messages, called publishers, and receivers, called subscribers [24]. Unlike the observer pattern, publishers broadcast messages without knowledge of any subscribers, even if there are none. Similarly, subscribes listen to event classes of interest without knowledge of which publishers exist. This is facilitated by some form of an event aggregator. In the context of the solution that was implemented, Microsoft's PubSubEvents library was used to facilitate communication between driver mappings and various interface modules such as view models responsible for viewing the data on the screen and data recording services [25]. Because publishers and subscribers do not need to know about each other, this further supports the modular design of the extended sEMG recording interface.

5.2.3 Software Architecture

For the same reasons as the first solution, an MVVM architecture was set up. However, because of the modular design used, there are key aspects that need to be analysed. Figure 8 shows an initial design that provided a brief guideline as to how the application would be designed based on a pure focus of intercepting sEMG signals from various devices that should have been implemented in the DeviceDriverService.

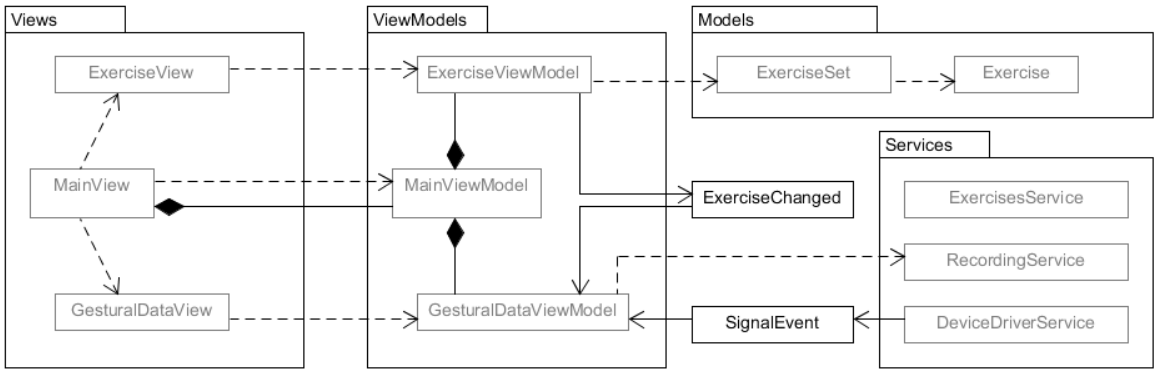
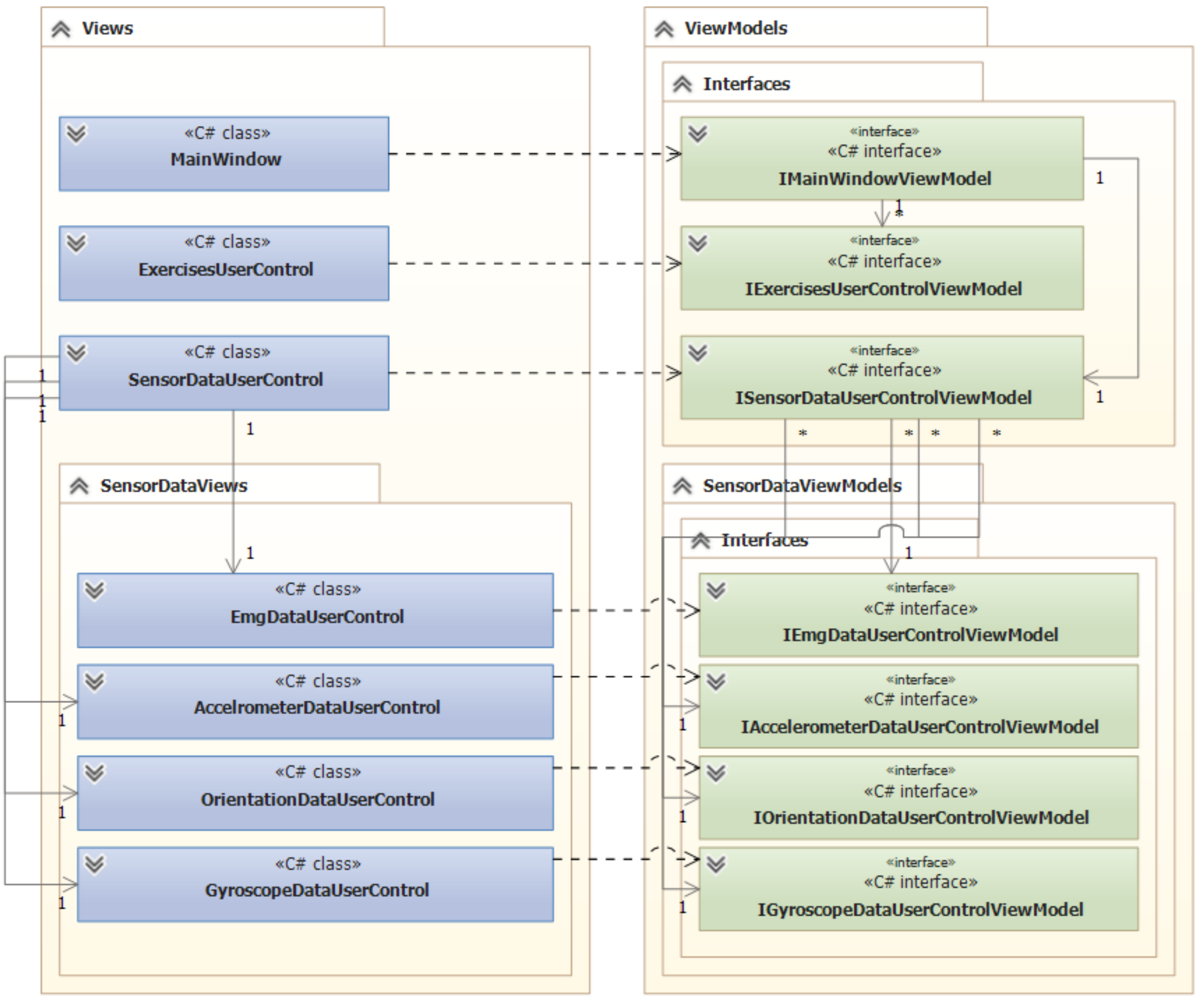


Fig 8. Initial UML class diagram for the second solution.

This was changed vastly in the later stages of implementation. Figure 9 highlights how the view has been separated into various sections responsible for different functions. At highest level is the MainWindow which includes two sub-views. The ExerciseUserControl is responsible for showing the current gesture as a video and image and contains buttons to navigate between gestures as well as to start and pause recording. The SensorDataUserControl is responsible for containing modules that are responsible for displaying different types of data from the device connected. At this stage, there are four implemented data type modules, developers who which to extend this can use them as an example and would have to build a module that contains the data they want to display and to reference it within the SensorDataUserControl.


Views
«C# class» MainWindow
«C# class» ExercisesUserControl
«C# class» SensorDataUserControl
SensorDataViews
«C# class» EmgDataUserControl
«C# class» AccelrometerDataUserControl
«C# class» OrientationDataUserControl
«C# class» GyroscopeDataUserControl
ViewModels
Interfaces
«interface» «C# interface» IMainWindowViewModel
«interface» «C# interface» IExercisesUserControlViewModel
«interface» «C# interface» ISensorDataUserControlViewModel
SensorDataViewModels
Interfaces
«interface» «C# interface» IEmgDataUserControlViewModel
«interface» «C# interface» IAccelerometerDataUserControlViewModel
«interface» «C# interface» IOrientationDataUserControlViewModel
«interface» «C# interface» IGyroscopeDataUserControlViewModel
1
*
Fig 9. View to view model relationships through interfaces.

The Unity Container described above takes care of mapping the MainWindow to the interface of the view model it requires. The sub-views that require sub-view models are then consequently taken care of as necessary with no need for explicit construction from the MainWindowViewModel.

User interface elements aside, the behavioural aspects of the software take place within the view models. Figure 10 shows a simplified relationship between the ExerciseUserControlViewModel and other services. The ExerciseService is responsible for loading one of three exercise sets defined above and the gestures within them. Navigation between gestures is defined within the ExerciseSet model.

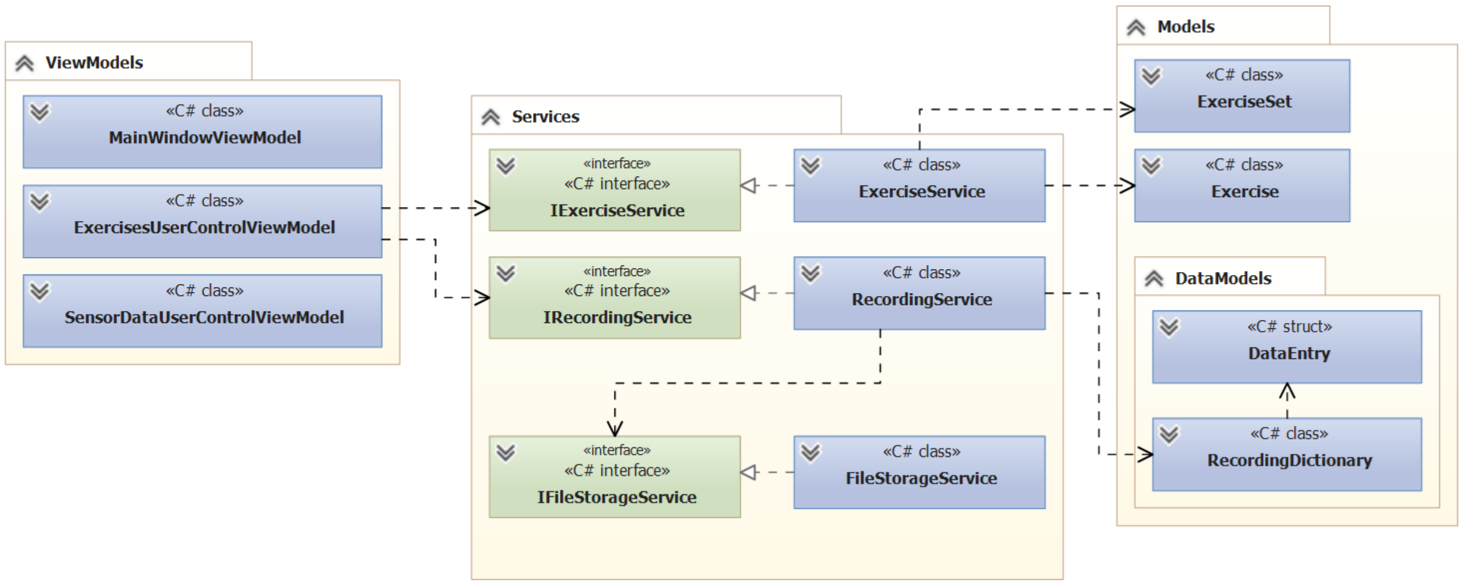


Fig 10. View model to services to models relationships.

The RecordingService is responsible for keeping track of all signals being received from an integrated driver. Because all sensor data is generally represented in numerical form, a DataEntry struct is defined for each signal received and a RecordingDictionary keeps track of the currently recorded gesture. Once enough repetitions are gathered, the RecordingService requests the FileStorageService to store the data recorded in the RecordingDictionary and all data entries are cleared to ensure that the application does not consume more memory than necessary.

5.2.4 Driver Mapping

Infrastructure of the interface aside, still remains the question of how various drivers will plug into the interface. This is where the infrastructure set up by the Unity Container and the publish/subscribe pattern are particularly useful.

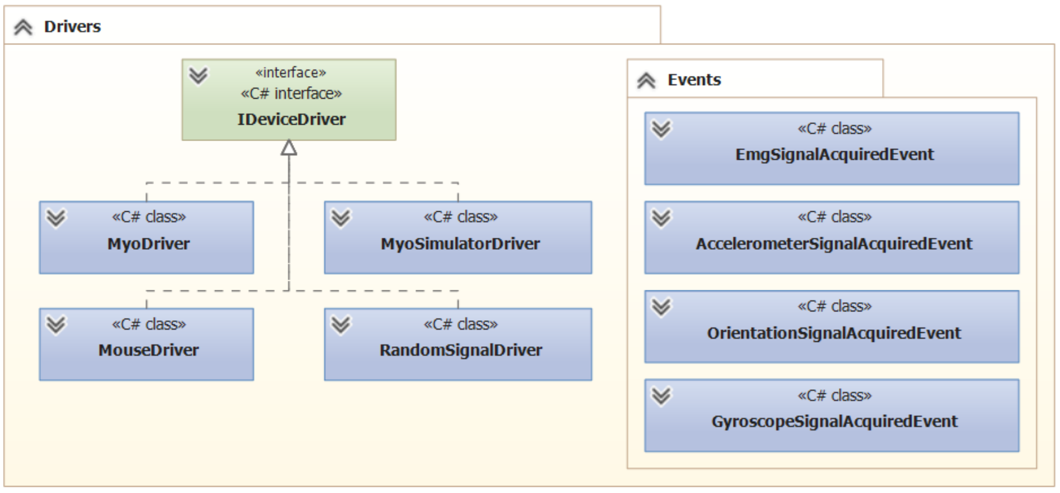
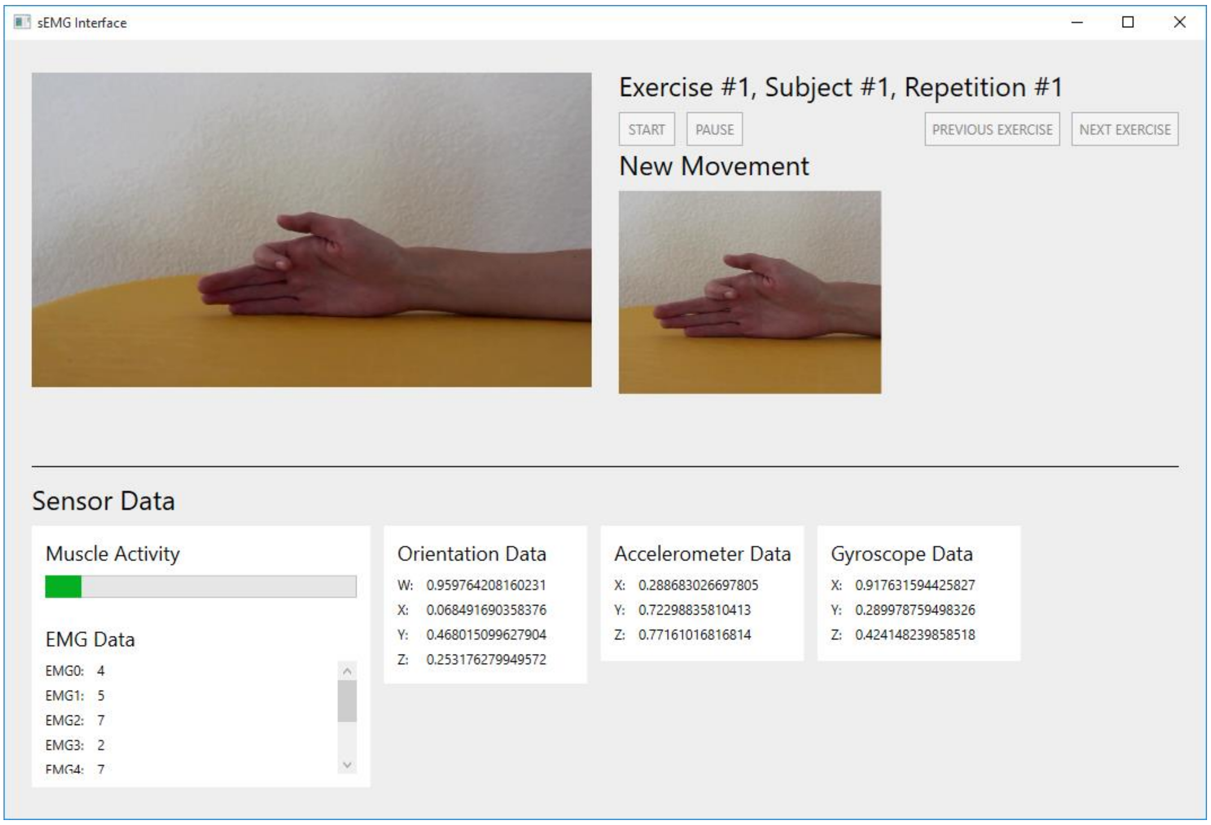


Fig 11. Device driver interface, sample implementations and signal events.

As seen in Figure 11, creation of a new driver mapping consists of inheriting from IDeviceDriver and creating the mappings within. For example, in the MyoDriver, connection to the Myo armband is setup via MyoSharp in the constructor of the driver. A hook is then made such that an EmgSignalAcquiredEvent is called every time an EMG signal is intercepted from the Myo. The EmgSignalAcquiredEvent is a PubSubEvent that can be subscribed to from other interface modules such as user interface elements or the recording service. Since the EmgSignalAcquiredEvent is not unique to the Myo Armband or any other sEMG sensor, a developer would be required to map whatever output the sensor provides to a double array that is then published.

Once a driver is ready, they just need to change the application driver that is referenced in the application entry point. This way, developers extending this solution need not worry about the remainder of the implementation. Multiple devices can also be mapped to the interface within a single driver file.

5.3 Implementation



```
Orientation Data
W:  0.959764208160231
X:  0.068491690358376
Y:  0.468015099627904
Z:  0.253176279949572

Accelerometer Data
X:  0.288683026697805
Y:  0.72298835810413
Z:  0.77161016816814

Gyroscope Data
X:  0.917631594425827
Y:  0.289978759498326
Z:  0.424148239858518

EMG Data
EMG0:  4
EMG1:  5
EMG2:  7
EMG3:  2
EMG4:  7
```

Fig 12. A screenshot of the extendable sEMG interface.

From an external point of view, the implementation of the extendable sEMG interface looks very similar to the first interface implemented as seen in Figure 12. The various data types being output in the sensor data section were boxed to make them represent separate modules better.

Since this is intended as an open source solution, user interface elements can be added, removed or adjusted as necessary.

5.4 Testing & Quality Assurance

Testing methods applied included methods that were applied in the first solution along with a couple of more methods necessary for ensuring the quality of this solution.

5.4.1 Driver Mapping Testing

Since a major feature of this solution was to support different types of drivers, a number of test drivers were created to ensure data representation and storage behaved as expected. Example test drivers included a driver that published the mouse coordinates every ten milliseconds and a simulator Myo driver which published the same data types using random numbers.

5.4.2 Unit testing

The first solution was lacking in unit tests. At the time it seemed appropriate to ignore them since the interface was not the main objective at the time, rather, it was the experiment to collect data from participants. This is not the case in the second solution. Since this solution will be available for extension by other developers, it is important that there is some form of automated testing to give developers some peace of mind when making changes or additions to the project files.

18 unit tests were written to cover the core functionality of the interface. As an overall solution, this contributes to 53.5% code coverage. However, considering the business logic of the solution, the coverage is as follows:

• Models:	82.9%
• View models:	84.2%
• Services:	89.7%
• Total of above:	85.5%

This is sufficient to ensure that the underlying state of the interface is as expected at any given time. User interface elements were not included in the unit testing code coverage since functionality such as button states, recording states and exercise states were managed by the view models and services.

5.5 Evaluation

The primary objective of this extendable solution was to provide simple capability of mapping drivers for use within the interface. This has been made simple enough with relevant examples and documentation to help developers make quick additions, given that they are familiar with the logic of whichever drivers they wish to integrate. As a secondary objective, developers should be able to add the capability of intercepting different signal types to the interface. Although this is slightly more complicated, with relevant documentation and the currently existing implementations of sEMG and other data signals, this should be a straightforward process for developers.

6 Discussion

6.1 Retrospective

Looking back, the initial aim of the project was more towards exploring the capabilities of sEMG signals from muscles to potentially aid amputees with recovering some of their lost functionality. While this is still a very worthwhile goal, this felt far from the purposes of what a software engineering research project would offer. This can be seen in how there was little emphasis towards the technical software engineering aspects in the first solution when trying to build an interface as quickly as possible. Nevertheless, there were non-technical software engineering techniques applied in the early phases of the project when eliciting the requirements for the data acquisition interface. This also involved a lot of back and forth between the original researchers who performed the original experiment that this project was based on.

On the other hand, there was a much higher technical focus in the second stage of the solution. This opened up many more opportunities to explore various techniques that were taught at the University of Canterbury throughout the last few years. This includes exploring various software architectures and comparing various design patterns and choosing the best suited for the task required. While building the second solution, there was a heavy focus on key software design principles that encouraged loosely coupled and highly cohesive components among other principles and heuristics. Although the end result provided a basic set of functionality, it was set up much more elegantly compared to previous university work.

As a bonus, C# knowledge and familiarity with Microsoft best practices were improved. This is useful for future work with the technologies used in this project and to provide guidelines for best practises with projects that involve other languages. It also helps with better understanding how asynchronous programming works in a multi-threaded language which may help in future projects where user interface responsiveness is a necessary requirement.

Working with the Myo armband has also provided some opportunity to explore various ways in which humans can interact with computers. Experience gained with developing with the Myo can be reused if there are particular systems that need to be developed where direct hand contact is not an option and basic gestural functionality is needed. Although some effort is needed to work with the Myo armband, it is still a much more subtle way of interacting with systems than vocal systems.

Although not directly software related, working on this project has helped build an appreciation of how much of the functionality provided by the human body is taken for granted. It is clear that technology is extremely far from replicating the functionality, dexterity and degrees of freedom that a human hand provides.

6.2 Potential for Future Work

There is still much room for additions and improvements that can be added to this project. Considering some of the feedback provided during the demo, one of the additions could be to limit the signal update frequency from a visual point of view so that data does not fluctuate as rapidly and provides some observable feedback to the observer. Some of the suggestions include making use of a moving average and some sort of filtering being applied to the values represented. Another concern within this is to limit the number of significant figures being displayed on the screen. This would be followed up with some research regarding how relevant some of the less significant figures and how much of it is accurate and how much is just noisy data. Following this, there may be some considerations as to how much visual data should be displayed on the screen since the majority of it is not human readable. A more impressive way of representing the signals obtained would be through the use of graphs to represent them, similar to what can be seen in visualization tools in the Myo Market Beta [16].

Another suggestion was to implement some sort of universal plug and play (UPnP) mechanism to make it simpler for researchers to integrate with various sensors. This was far beyond the scope of the project since this implies that there is some sort of standardization within the field of sensors. Other software such as National Instrument's LabVIEW was explored where various sensors could be integrated, however, some developer effort was still required.

Considering that there are four sets of data gathered from the experiments run in the first stage of this project, it would be interesting to share this with the original researchers involved to see how many gestures a trained machine algorithm could recognise based on these alone. Now that there is some familiarity with the research involved in helping amputees, it would interesting to keep up with research outside of the scope of the experimenting with the Myo armband.

7 Conclusion & Final Remarks

It is unlikely that the accuracy of the Myo armband will be sufficient to assist hand amputees with enough capability to recover significant lost hand functionality. It will definitely be interesting to see how research in this field grows since the current state of technology is still far from replicating the functionality of a real human hand. In conclusion, this project can be considered a success if even a small part of it helps benefit the advancement of this study in some way or another.

References

[1] M. Atzori, A. Gijsberts, C. Castellini, B. Caputo, A.-G. M. Hager, S. Elsig*, et al.*, "Electromyography data for non-invasive naturally-controlled robotic hand prostheses," vol. 1, 23 December 2014 2014.

[2] M. Atzori and H. Müller. (2012). *NINAPRO project first milestone: Set up of the data base*. Available: http://publications.hevs.ch/index.php/publications/show/1165

[3] P. L. Hudak, P. C. Amadio, and C. Bombardier, "Development of an upper extremity outcome measure: the DASH (disabilities of the arm, shoulder and hand) The Upper Extremity Collaborative Group (UECG)," *American Journal of Industrial Medicine,* vol. 29, pp. 602-608, 1996.

[4] L. Philipson, D. S. Childress, and J. Strysik, "Digital approaches to myoelectric state control of prostheses," *Bulletin of Prosthetics Research,* vol. 18, pp. 3-11, 1981.

[5] A. H. Bottomley, "Myoelectric control of powered prostheses," *J Bone Joint Surg,* vol. B47, pp. 411 - 415, 1965.

[6] D. S. Childress, "A myoelectric three-state controller using rate sensitivity," *Proceedings of 8th ICMBE, Chicago,* pp. 4-5, 1969.

[7] G. Li, A. E. Schultz, and T. A. Kuiken, "Quantifying Pattern Recognition-Based Myoelectric Control of Multifunctional Transradial Prostheses," *IEEE Transactions on Neural Systems and Rehabilitation Engineering,* vol. 15, pp. 185 - 192, 2010 2010.

[8] M. Zecca, S. Micera, M. C. Carrozza, and P. Dario, "Control of multifunctional prosthetic hands by processing the electromyographic signal," *Critical Reviews in Biomedical Engineering,* vol. 30, pp. 459-485, 2002.

[9] A. Chan and K. Englehart, "Continuous myoelectric control for powered prostheses using hidden Markov models," *IEEE Transactions on Biomedical Engineering,* vol. 52, pp. 121 - 124, 2005.

[10] S. Bitzer and P. v. d. Smagt, "Learning EMG control of a robotic hand: Towards active prostheses," *Proceedings of ICRA, International Conference on Robotics and Automation,* pp. 2819–2823, 2006.

[11] F. V. Tenore, A. R. A. Fahmy, S. Acharya, R. Etienne-Cummings, and N. V. Thakor, "Decoding of individuated finger movements using surface electromyography," *IEEE Transactions in Biomedical Engineering,* vol. 56, pp. 1427-1434, 2009.

[12] M. Atzori, C. Castellini, and H. Müller, "Spatial Registration of Hand Muscle Electromyography Signals," *7th International Workshop on Biosignal Interpretation,* 2012.

[13] M. Atzori, A. Gijsberts, S. Heynen, A.-G. M. Hager, O. Deriaz, P. v. d. Smagt*, et al.*, "Building the Ninapro database: A resource for the biorobotics community (submitted)," *Proceedings of IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob 2012),* 2012.

[14] I. Kuzborskij, A. Gijsberts, and B. Caputo, "On the challenge of classifying 52 hand movements from surface electromyography," *Engineering in Medicine and Biology Society (EMBC), 2012 Annual International Conference of the IEEE* 2012.

[15] P.-G. Jung, G. Lim, S. Kim, and K. Kong, "A Wearable Gesture Recognition Device for Detecting Muscular Activities Based on Air-Pressure Sensors," *IEEE Transactions on Industrial Informatics,* vol. 11, pp. 485 - 494, 2005.

[16] Thalmic Labs. (2015). *Myo Gesture Control Armband*. Available: https://www.thalmic.com/en/myo/

[17] Microsoft. (2015). *.NET Framework System Requirements*. Available: https://msdn.microsoft.com/en-us/library/8z6watww%28v=vs.110%29.aspx

[18] N. Cosentino and T. Uzun, "MyoSharp," ed, 2014.

[19] toaaot, "csmatio," ed. SourceForge, 2014.

[20] R. Kõiva, B. Hilsenbeck, and C. Castellini, "FFLS: an accurate linear device for measuring synergistic finger contractions," *Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. ,* pp. 531 - 534, 2012.

[21] M. Atzori, "Ninapro Repository ", ed. HES-SO Valais, 2014.

[22] O. Scores. (1996). *The Disabilities of the Arm, Shoulder and Hand (DASH) Score*

[23] Microsoft. (2013). *Unity Container*. Available: https://msdn.microsoft.com/en-us/library/ff647202.aspx

[24] Microsoft. (2015). *Publish/Subscribe*. Available: https://msdn.microsoft.com/en-us/library/ff649664.aspx

[25] Microsoft. (2014). *What's New in Prism Library 5.0 for WPF*. Available: https://msdn.microsoft.com/en-us/library/gg430871%28v=pandp.40%29.aspx