# Biointerfacing vMPL Lab

## Using the Virtual Modular Prosthetic Limb (vMPL) and Virtual Integration Environment (VIE) Systems

## Contents

**Biointerfacing Lab Introduction** ........ 2

**Part 1: Controlling a dexterous prosthetic limb** ........ 4

Establishing Communications with the virtual Modular Prosthetic Limb (vMPL) ........ 4

Setting up MATLAB for the Lab ........ 4

Sending limb commands to the vMPL from MATLAB ........ 4

Exploring vMPL Commands ........ 7

Continue Exploring (optional): ........ 7

Returning to / Rebooting the MATLAB Environment ........ 8

**Part 2: Using the Myo Armband EMG Biointerface** ........ 9

Getting Familiar with the MyoBand ........ 9

Generating and Measuring Simulated EMG Data ........ 11

**Part 3: Closing the Loop** ........ 13

Conventional ("Simple") Control Mode ........ 13

Detect intended motions ........ 13

Command joint velocity and position using EMG ........ 15

Reflection: Conventional ("Simple") Control Mode ........ 15

**Part 4: Complex Motions with Pattern Recognition** ........ 16

Starting the MiniVIE Pattern Recognition Interface ........ 16

Training Data ........ 18

Controlling the Arm with Pattern Recognition ........ 20

**Part 5: Putting all of the Pieces Together** ........ 23

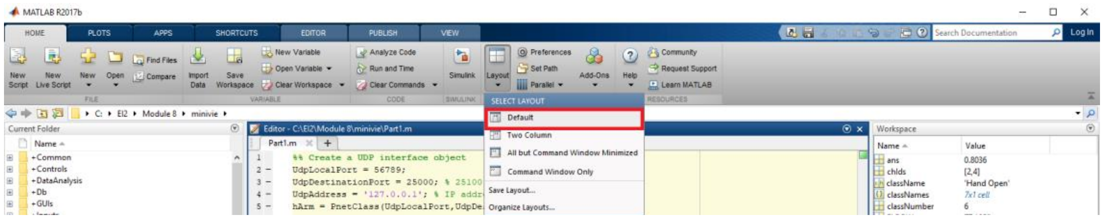
JOHNS HOPKINS
WHITING SCHOOL of ENGINEERING

## Biointerfacing Lab Introduction

The goal of this lab is to build a software based prosthetic controller that will collect and process real-time EMG signals, and then convert them into movement commands for a virtual prosthetic limb within a simulated environment. We will guide you through this multi-step process of developing a prosthetic controller by individually exploring the various components, and then assembling them in the final part of the lab.

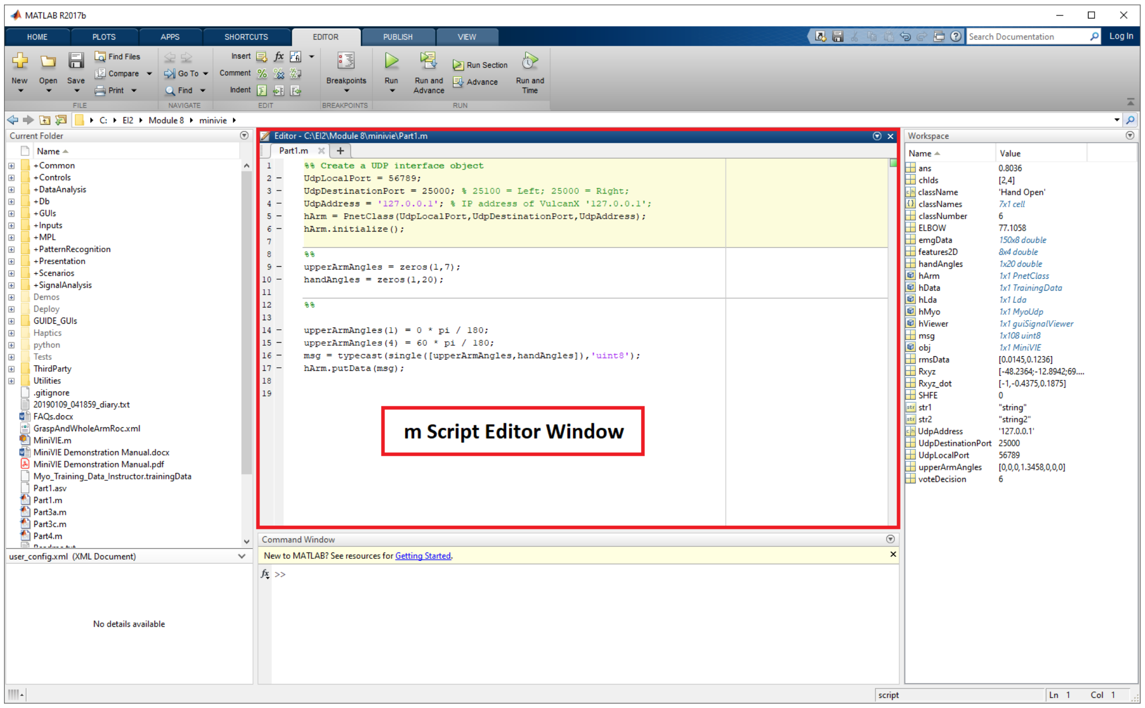
Most of the work for this lab will be performed within the MATLAB coding environment. Within the lab instructions, you will find MATLAB code shown in courier font that can be executed at the MATLAB command prompt or within a MATLAB script file or "m" file (e.g. "MATLABScript.m"). An "m" file can be created and edited within the Editor Window. The "Default" MATLAB window layout will show the "m" Script file Editor in the center at the top:

To show Default MATLAB Window layout:



```
%% Create a UDP interface object
UdpLocalPort = 56789;
UdpDestinationPort = 25000; % 25100
UdpAddress = '127.0.0.1'; % IP addr
hArm = PnetClass(UdpLocalPort,UdpDe
```

The "m" Script Editor Window:

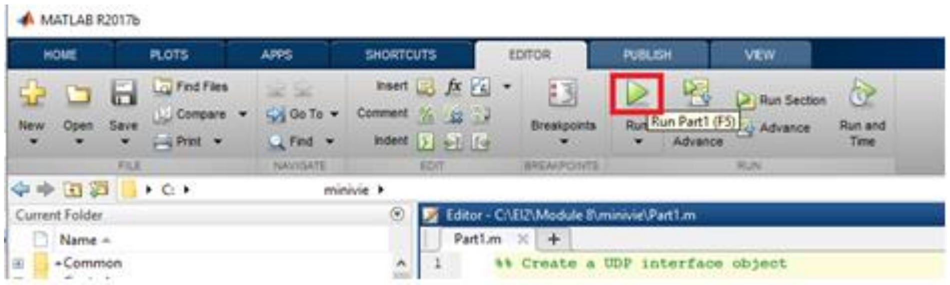


```
%% Create a UDP interface object
UdpLocalPort = 56789;
UdpDestinationPort = 25000; % 25100 = Left; 25000 = Right;
UdpAddress = '127.0.0.1'; % IP address of VulcanX '127.0.0.1';
hArm = PnetClass(UdpLocalPort,UdpDestinationPort,UdpAddress);
hArm.initialize();

%%
upperArmAngles = zeros(1,7);
handAngles = zeros(1,20);

%%

upperArmAngles(1) = 0 * pi / 180;
upperArmAngles(4) = 60 * pi / 180;
msg = typecast(single([upperArmAngles,handAngles]),'uint8');
hArm.putData(msg);
```

| Name | Value |
|---|---|
| ans | 0.8036 |
| chIds | [2,4] |
| className | 'Hand Open' |
| classNames | *7x1 cell* |
| classNumber | 6 |
| ELBOW | 77.1058 |
| emgData | *150x8 double* |
| features2D | *8x4 double* |
| handAngles | *1x20 double* |
| hArm | *1x1 PnetClass* |
| hData | *1x1 TrainingData* |
| hLda | *1x1 Lda* |
| hMyo | *1x1 MyoUdp* |
| hViewer | *1x1 guiSignalViewer* |
| msg | *1x108 uint8* |
| obj | *1x1 MiniVIE* |
| rmsData | [0.0145,0.1236] |
| Rxyz | [-48.2364;-12.8942;69.... |
| Rxyz_dot | [-1,-0.4375,0.1875] |
| SHFE | 0 |
| str1 | "string" |
| str2 | "string2" |
| UdpAddress | '127.0.0.1' |
| UdpDestinationPort | 25000 |
| UdpLocalPort | 56789 |
| upperArmAngles | [0,0,0,1.3458,0,0,0] |
| voteDecision | 6 |

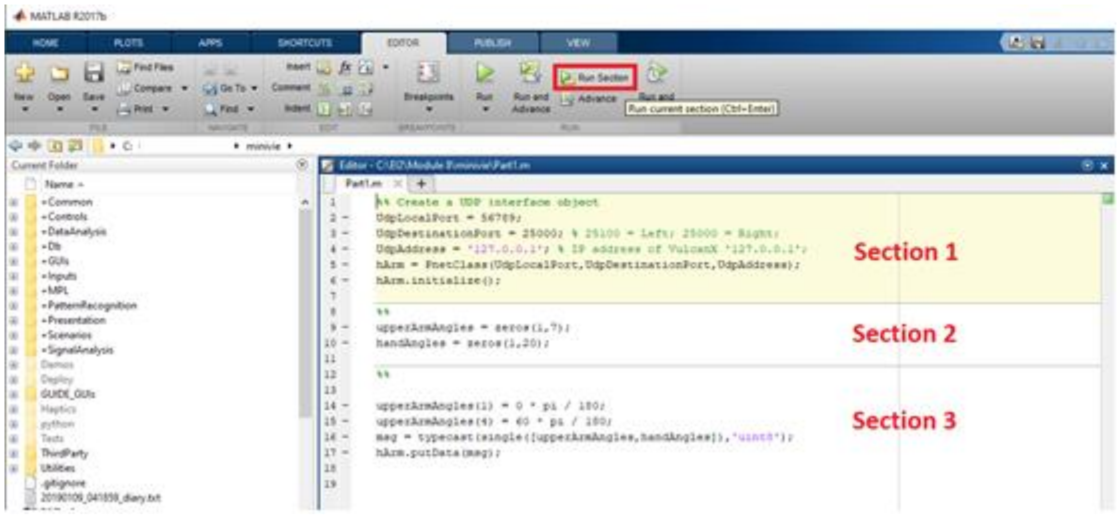
```
New to MATLAB? See resources for Getting Started.
>>
```

When presented with MATLAB code, please use the code to generate your own "m" file. The entire "m" file can be "run" or executed at the command prompt (e.g. ">>MATLABScript"), by pressing F5 in the MATLAB Editor, or by pressing the "Run" button (as shown below):



```
%% Create a UDP interface object
```

To run individual cells or "Sections" of code, which are denoted by **%%** before a code block, can be evaluated using CTRL+Enter, or by pressing "Run Section". Individually running all of the sections in an "m" file (Section 1, 2, and 3 below) in succession would be the same as running the entire "m" script file by pressing the "Run" button:


MATLAB R2017b
Run Section
Run current section (Ctrl+Enter)
Editor - C:\EI2\Module 8\minivie\Part1.m
Section 1
Section 2
Section 3

```
%% Create a UDP interface object
UdpLocalPort = 56789;
UdpDestinationPort = 25000; % 25100 = Left; 25000 = Right;
UdpAddress = '127.0.0.1'; % IP address of VulcanX '127.0.0.1';
hArm = PnetClass(UdpLocalPort,UdpDestinationPort,UdpAddress);
hArm.initialize();

%%
upperArmAngles = zeros(1,7);
handAngles = zeros(1,20);

%%

upperArmAngles(1) = 0 * pi / 180;
upperArmAngles(4) = 60 * pi / 180;
msg = typecast(single([upperArmAngles,handAngles]),'uint8');
hArm.putData(msg);
```

Look for sections in the lab instructions that contain this icon (to the right) – this specifies particular things that you should include into your lab report (data, input, written responses, etc.):

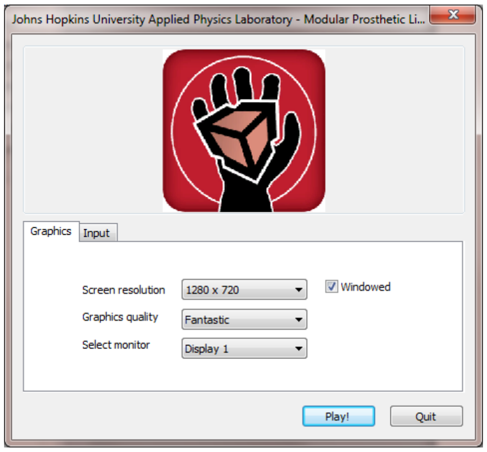
**<u>Before beginning the lab</u>**, make sure to have the following software installed and setup using the instruction guides on Blackboard:

- MATLAB and MiniVIE: MATLAB Installation and Setup Guide
- vMPL software: vMPL Setup Guide
- Ensuring that UDP Communication is working (more in the guide and this lab)

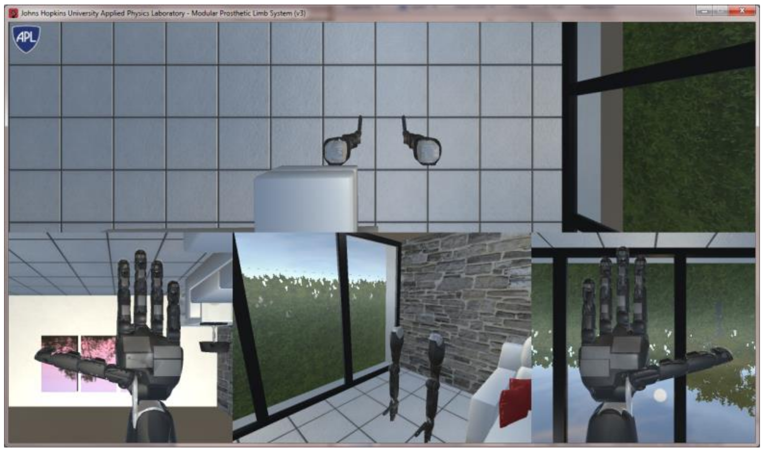
## Part 1: Controlling a dexterous prosthetic limb

### Establishing Communications with the virtual Modular Prosthetic Limb (vMPL)

1. Open the computer shortcut for *vMPL* application
    a. Select a Screen Resolution that will work with your computer (e.g. 1280 x 720), and **check** "Windowed"
    b. Select a Graphics Quality that will work on your computer (e.g. "Fastest" if you have an older computer)


Johns Hopkins University Applied Physics Laboratory - Modular Prosthetic Li...

Screen resolution: 1280 x 720
[✓ Windowed]
Graphics quality: Fantastic
Select monitor: Display 1


Johns Hopkins University Applied Physics Laboratory - Modular Prosthetic Limb System (v3)
APL

### Setting up MATLAB for the Lab

- Open MATLAB
- Use the shortcut generated in the set up guide to navigate to your MiniVIE folder

Note: If the **goto MiniVIE shortcut** does not exist **(the MATLAB Installation and Setup Guide document shows you how to set up MATLAB with the shortcuts)**, manually go to the *MiniVIE* folder in the MATLAB navigation bar

- Type the following in the console:

```
>> MiniVIE.configurePath % Make all of the MATLAB libraries available for use
```

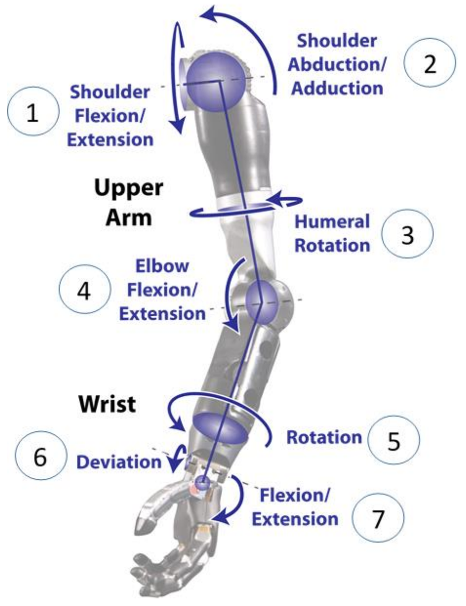
### Sending limb commands to the vMPL from MATLAB

This part of the lab will show you how to set up an interface to send movement commands from MATLAB to the vMPL. These movement commands will specify a specific joint angle for specific joints of the limbs in the vMPL to move to. Create a new "m" script, and enter the code below to set up this movement command interface. For those interested, the movement command messages are sent over the TCP/IP layer as UDP messages. Read the code comments in green text for more information about each line of code. It's ok if you don't fully understand each line of code – you will be directed to change/add to only a few specific lines of code to start (usually designated with a yellow or purple highlight):

```
%% Create a UDP interface object to send commands to the vMPL
UdpLocalPort = 56789;
UdpDestinationPort = 25000; % 25100 = Left limb; 25000 = Right limb;
UdpAddress = '127.0.0.1'; % IP address for sending commands to own computer
hArm = PnetClass(UdpLocalPort,UdpDestinationPort,UdpAddress);
hArm.initialize(); % hArm is the "Handle" for sending commands to the vMPL
```

Below will show you how to create a variable to store series of values that will contain the commanded joint angles for each of the 7 upper arm angles in the prosthetic limb (these are stored in radians).

| ----------Upper Arm Joint----------------------- | -------Joint #----- |
|---|---|
| Shoulder Flexion (+) / Shoulder Extension (-) | **1** |
| Shoulder Adduction (+) / Shoulder Abduction (-) | **2** |
| Humeral Internal Rotation (+) / Humeral External Rotation (-) | **3** |
| Elbow Flexion (+) / Elbow Extension (-) | **4** |
| Wrist Pronation (+) / Wrist Supination (-) | **5** |
| Ulnar Deviation (+) / Radial Deviation (-) | **6** |
| Wrist Flexion (+) / Wrist Extension (-) | **7** |

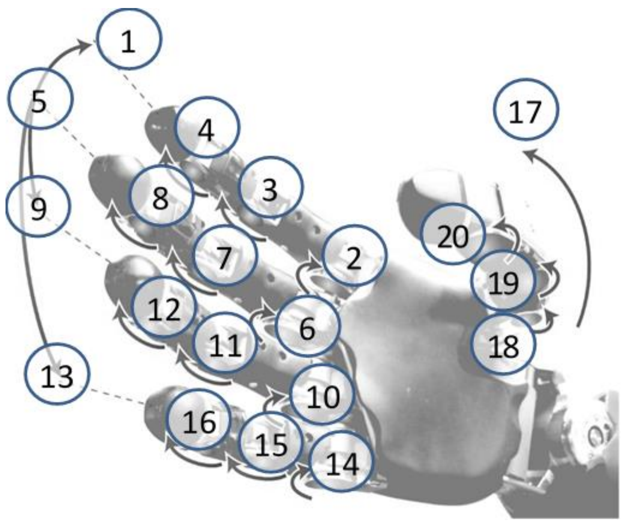


Place the following command in a new section within your "m" script file to create the variable for these 7 joint angles of the upper part of the limb:

```
upperArmAngles = zeros(1,7); % Create array of numbers for each arm joint in limb
```

Below we will create a variable to store the 20 hand angles, again using joint angles that are represented in radians (and not in degrees).

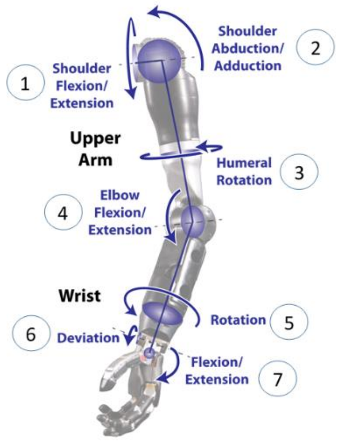
| ---Hand Joint----- | ---Joint #----- |
|---|---|
| INDEX_AB_AD: | **1** |
| INDEX_MCP: | **2** |
| INDEX_PIP: | **3** |
| INDEX_DIP: | **4** |
| MIDDLE_AB_AD: | **5** |
| MIDDLE_MCP: | **6** |
| MIDDLE_PIP: | **7** |
| MIDDLE_DIP: | **8** |
| RING_AB_AD: | **9** |
| RING_MCP: | **10** |
| RING_PIP: | **11** |
| RING_DIP: | **12** |
| LITTLE_AB_AD: | **13** |
| LITTLE_MCP: | **14** |
| LITTLE_PIP: | **15** |
| LITTLE_DIP: | **16** |
| THUMB_CMC_AD_AB: | **17** |
| THUMB_CMC: | **18** |
| THUMB_MCP: | **19** |
| THUMB_DIP: | **20** |



Place the following command in this same section of your "m" script file:

```
handAngles = zeros(1,20); % Create array of numbers for each hand joint in limb
```

JOHNS HOPKINS
WHITING SCHOOL *of* ENGINEERING



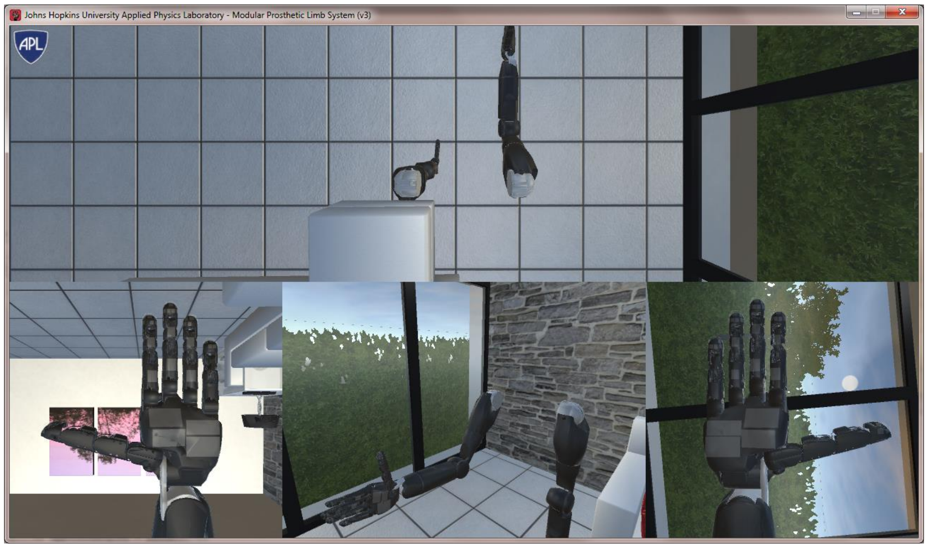
**<u>Sending Commands to the vMPL:</u>** Next, we will specify desired joint angles for each joint in the limb in the vMPL and transmit the command with these joint angles to the vMPL (typecast in byte-code format – conversion code provided for your below). This code below will first set angles for any particular joints that you want to move (we show commands for the Shoulder flexor and Elbow flexor below) – these angles will be in radians (the code below will take the degrees value and multiply by pi and divide by 180 degrees). After you've set the angle(s) in the variable for the desired joint(s), you will generate a command message ("msg" below) from both number array variables with desired limb joint angles (upperArmAngles and handAngles). This message will be generated in a specified format for UDP communication, and then send that message using your "hArm" UDP Handle – you can reuse the two lines of code whenever you need to send a command to the vMPL:

```
upperArmAngles(1) = 35 * pi / 180; % Set Shoulder Flexor to 35° converted to radians
upperArmAngles(4) = 60 * pi / 180; % Set Elbow Flexor to 60° converted to radians
msg = typecast(single([upperArmAngles,handAngles]),'uint8'); % Generate Command msg
hArm.putData(msg); % Send constructed command message to the vMPL using the Handle
```

**<u>Joint Angle Ranges:</u>** When sending commands to the vMPL, you must make sure that the angles in the commands sent do not go outside the allowable range of motion. For example, you can't send a negative value for the Elbow joint angle, as the elbow does not bend backward in the vMPL limb, just as the elbow does not bend backward for (most) humans. If you try to send a command with an invalid range, the vMPL will only travel to the max or min value. Here are the joint ranges that are permitted:

| **Joint of vMPL limb:** | **Minimum** | **Maximum** |
|---|---|---|
| Shoulder Flex / Extend | -40° | 175° |
| Shoulder Abduct / Adduct | -160° | 0° |
| Humeral Rotation | -45° | 90° |
| Elbow Flex / Extend | 0° | 150° |
| Wrist Rotation | -90° | 90° |
| Wrist Flex / Extend | -60° | 60° |
| Wrist Deviation (or Abduct / Adduct) | -15° | 45° |
| Index / Middle / Ring / Little MCP Flex / Extend | -30° | 110° |
| Index / Middle / Ring / Little PIP Flex / Extend | 0° | 100° |
| Index / Middle / Ring / Little DIP Flex / Extend | 0° | 80° |
| Index Abduct / Adduct | -20° | 0° |
| Middle Abduct / Adduct | 0° | 0° |
| Ring Abduct / Adduct | 0° | 30° |
| Little Abduct / Adduct | 0° | 20° |
| Thumb CMC Abduct / Adduct | 0° | 105° |
| Thumb CMC Flex / Extend | 0° | 55° |
| Thumb MCP Flex / Extent | 0° | 60° |
| Thumb IP Flex / Extend | 0° | 60° |

After you've sent the movement command to the vMPL, ensure that the limb in the vMPL has moved. If your hArm handle was set up with a UdpDestinationPort of 25000 it will send commands to the Right limb, and if the destination port for your hArm handle is 25100, it will send commands to the Left limb. You can also send commands to both limbs by setting up two UDP handles (e.g. hArmLeft and hArmRight) – this is optional!

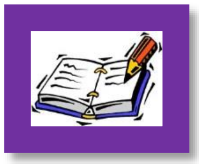

Johns Hopkins University Applied Physics Laboratory - Modular Prosthetic Limb System (v3)
APL

Congratulations, you have now issued commands to the virtual Modular Prosthetic Limb (vMPL). By prototyping within a virtual environment in this way, you can make a rapid transition from offline evaluation of your prosthetic controller to eventually controlling real hardware. Also, using the TCP/IP model (sending the UDP commands) you can also control the limbs on other computers or over the internet.

## Exploring vMPL Commands

Repeat the lines of code above to achieve the limb positions that are listed below. Take a screen shots ("Print Screen" for Windows or Mac, or use the Windows Snipping Tool) showing each position achieved, and save this screen shot and the joint angles command into a word document ("Ctrl-V" paste function) -- you will need these for your lab report submission for this week.

1) Asking a Question
2) Doing the Chicken-Dance
3) We're #1
4) Paper, Rock, or Scissors (choose 1)
5) Finally, add one position of your own choosing

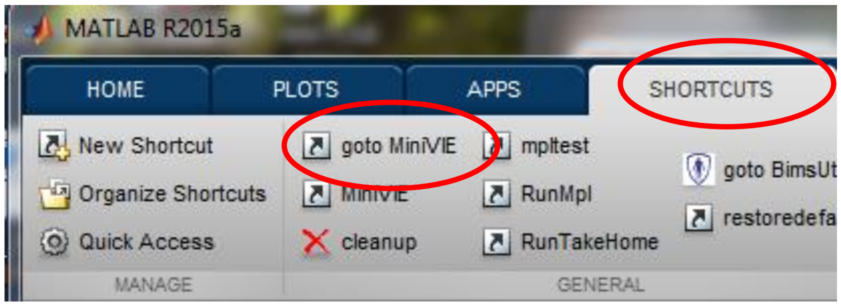


## Continue Exploring (optional):

Update your commands to control both left and right arms independently. This will require you to create two UDP socket handles, one for the left limb, and one for the right limb (e.g. hArmLeft, hArmRight).

## Returning to / Rebooting the MATLAB Environment

- If MATLAB crashes at any time, you will need to restart the program and reset the conditions that were set in Part 1. This will require you to open MATLAB, and click on the "goto MiniVIE" in the Shortcuts Tab again:



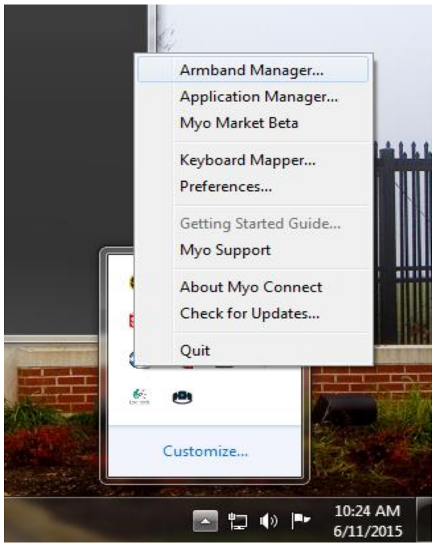
As before, setup the Command Interface with the vMPL. Enter at the command prompt, or create a script to execute the commands below (the command messages are sent over the TCP/IP layer as UDP messages). NOTE: If you rerun this code many times while debugging, you will open up too many UDP ports and you may get at PNet error. If this occurs, restart MATLAB and consider commenting out this code after the first time you run it.

```
%% Create a UDP interface object to send commands to the vMPL
UdpLocalPort = 56789;
UdpDestinationPort = 25000; % 25100 = Left limb; 25000 = Right limb;
UdpAddress = '127.0.0.1'; % IP address for sending commands to own computer
hArm = PnetClass(UdpLocalPort,UdpDestinationPort,UdpAddress);
hArm.initialize(); % hArm is the "Handle" for sending commands to the vMPL
upperArmAngles = zeros(1,7); % Create array of arm joint in limb
handAngles = zeros(1,20); % Create array for hand joints in limb
```

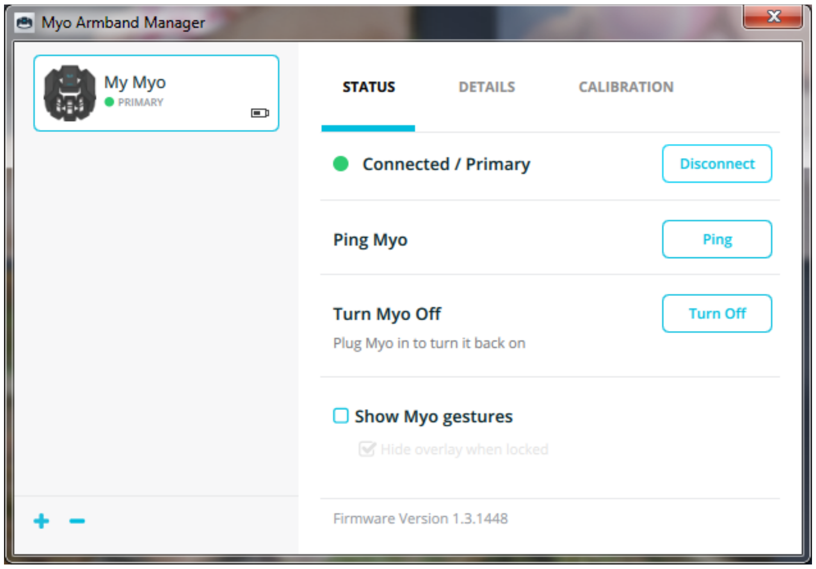
## Part 2: Using the Myo Armband EMG Biointerface

### Getting Familiar with the MyoBand

1) Launch the Myo Armband Manager. (Right-Click the tray icon and select "Armband Manager").
    a. If this is not present, may need to make sure that the MyoConnect software is installed from the internet.



2) Ensure the device is unplugged from USB and connected via Bluetooth (Press the "Connect" button).



Connected / Primary

Ping Myo

Turn Myo Off

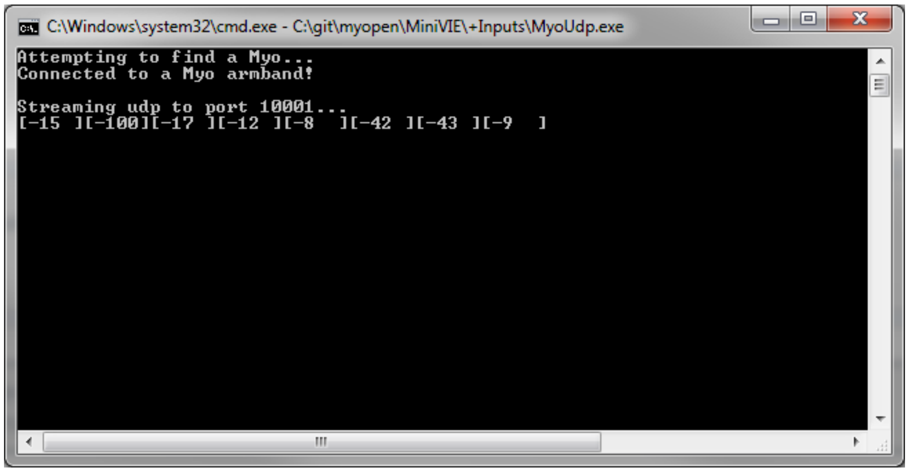
Plug Myo in to turn it back on

☐ Show Myo gestures

☑ Hide overlay when locked

Firmware Version 1.3.1448

3) Place the Myo on your arm with the logo facing up (I.e. if you bend your elbow, it will be in the crook of your arm). The Blue LED Stripe should be towards your hand.
4) Start the MyoUdp Shortcut om the +Inputs folder of MiniVIE and ensure data is streaming.
    a. The numbers should change when you flex your arm, as the numbers indicated values being read from the electrodes.


C:\Windows\system32\cmd.exe - C:\git\myopen\MiniVIE\+Inputs\MyoUdp.exe

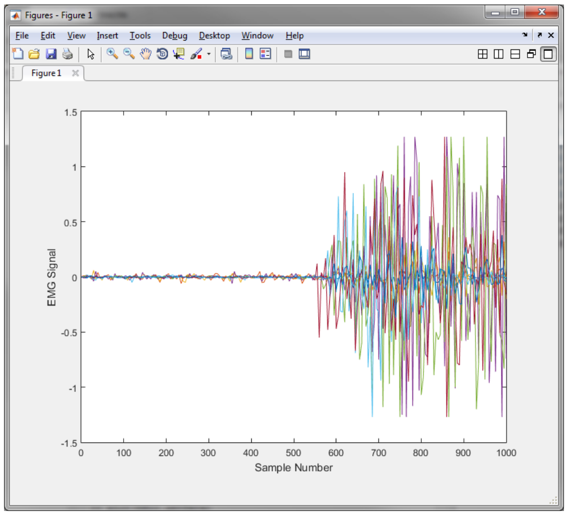
```
Attempting to find a Myo...
Connected to a Myo armband!

Streaming udp to port 10001...
[-15 ][-100][-17 ][-12 ][-8  ][-42 ][-43 ][-9  ]
```

5) Create the MATLAB Interface by creating a new script in the MATLAB editor, and enter the following commands. Note: Upon issuing these commands the first time, this will launch a small executable that handles the device communication.

```
%% Create an object for UDP interface to the Myo Armband
hMyo = Inputs.MyoUdp.getInstance();
hMyo.initialize();
```

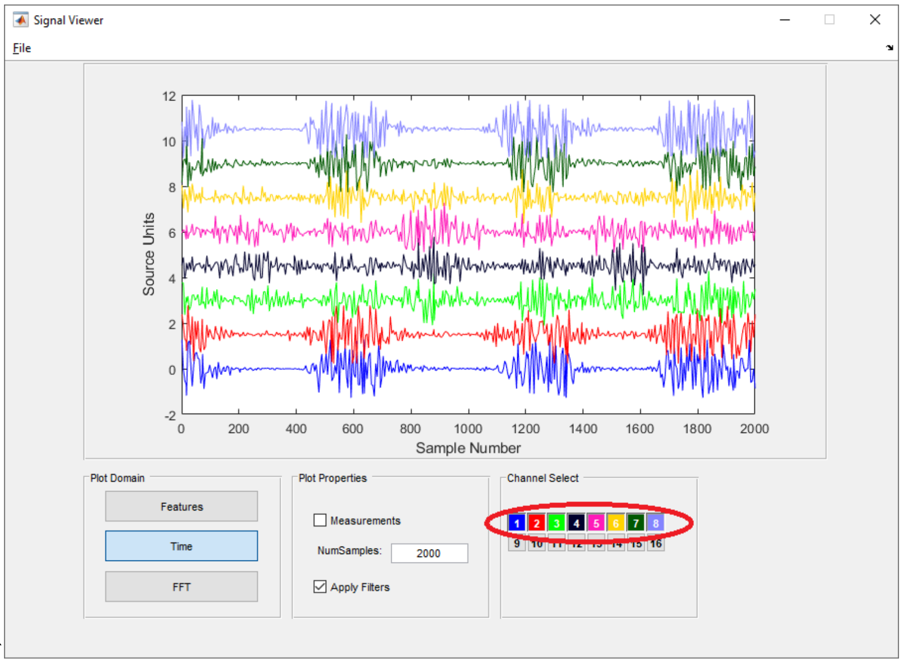
6) Get EMG data and plot. Note: You may have to run getData again if the first graph presents a flat line at zero due to the initialization sequence.

```
%% Get data and plot.  emgData size is [1000 samples x 8 channels]
emgData = hMyo.getData;
plot(emgData)
xlabel('Sample Number'); ylabel('EMG Signal')
```



7) Open the real-time Signal Viewer. (Note: the Signal Viewer uses a MATLAB timer() object to automatically update the display. While it is possible to run other commands, if performance is degraded, close the window when not in use.)

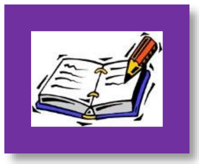
```
hViewer = GUIs.guiSignalViewer(hMyo); % View Myoband voltage signals in real time
```

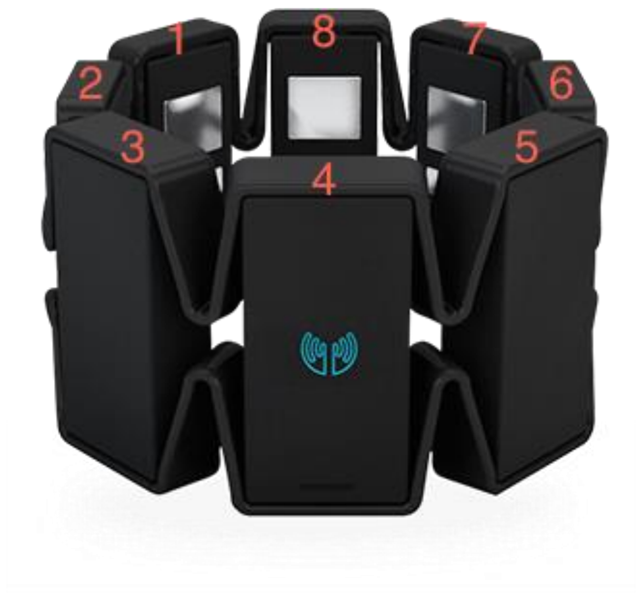


**Use the channel selector to enable all channels (1-8).** These channels are color coded.

## Generating and Measuring Simulated EMG Data

1) Use the **'Measurements'** checkbox in the "Plot Properties" box to observe the RMS in the SignalViewer for each channel.
2) Record the RMS channel values for each of the 8 electrode channels (red numbers in photo below) during the following movements (a-i) **using the table below**. Note: For EMG, you must flex muscles as if working against resistance so that the muscle contraction is consistent and measurable.
    a. No Movement (rest)
    b. Wrist Flexion
    c. Wrist Extension
    d. Wrist Pronation
    e. Wrist Supination
    f. Tip Grasp
    g. Power Grasp
    h. Hand Open
    i. No Movement (rest) (repeat no movement to ensure that the signals are consist from when you began)





Note: Alternatively, if you are proficient in MATLAB, it may save you time to write a set of commands to complete the table below using the previous hMyo.getData command, and then compute the root mean square amplitude from each of the channels of interest using:

```
rms(hMyo.getData(100))
```

or

```
sqrt(mean(hMyo.getData(100).^2))
```

Note: When executing these functions above, you should only get measured data in the first 8 values returned (for the 8 channels, the rest of the values should show as zeros) - Up to 16 channels are accommodated in a single streaming interface, but we are only generating and using the first 8 for the EMG simulator.

| | Channel 1 | Channel 2 | Channel 3 | Channel 4 | Channel 5 | Channel 6 | Channel 7 | Channel 8 |
|---|---|---|---|---|---|---|---|---|
| a. No Movement (rest) | | | | | | | | |
| b. Wrist Flexion | | | | | | | | |
| c. Wrist Extension | | | | | | | | |
| d. Wrist Pronation | | | | | | | | |
| e. Wrist Supination | | | | | | | | |
| f. Tip Grasp | | | | | | | | |
| g. Power Grasp | | | | | | | | |
| h. Hand Open | | | | | | | | |
| i. No Movement (rest, repeat) | | | | | | | | |

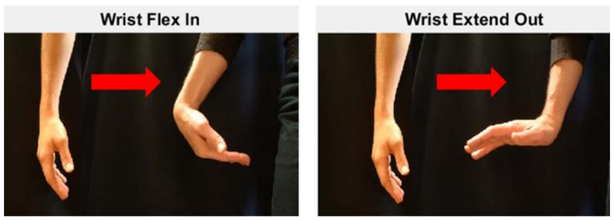
## Part 3: Closing the Loop

In this part of the lab you will use the biointerface to "close the loop" between the human operator and the device by controlling the MPL using EMG data. To plan out the control, you will need to identify what characteristic of the EMG signal you want to use as an input, and how to control the device.

### Conventional ("Simple") Control Mode

In a conventional ("simple") control mode for surface EMG, control will be achieved using the RMS value of 100ms of the EMG signal (sampled at 1kHz) for two channels, one used for each control movement (e.g. wrist flexion, wrist extension). Look at the table you made in Part 2 and determine the channel with the greatest difference in RMS value between wrist flexion and no movement, and note the RMS voltage values for these channels in row 1 of the table below. Do the same for wrist extension in row 2. (Note: Voltage values are normalized).

| | Best Myoband Channel # (from table above) | Channel RMS Value for: **"No Movement"** | Channel RMS Value for: **Flex / Extend** |
|---|---|---|---|
| Wrist Flexion | Channel #: | | |
| Wrist Extension | Channel #: | | |



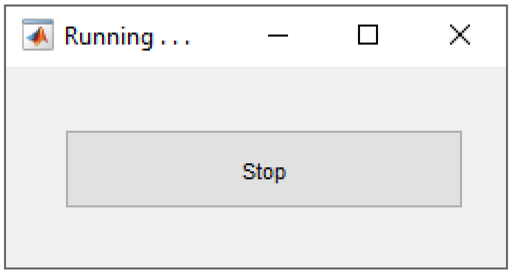
### Detect intended motions

In this section we will develop a script that will 'detect' and display the intended motion or your arm, in real-time. Use the channels and analysis from above to select the best channels for "Wrist Flex" and "Wrist Extend", then update a text display that will read out the current motion. Verify that the system displays "Rest" (No Movement), "Wrist Flex", and "Wrist Extend" when the appropriate motion is selected in the EMG Simulator GUI.

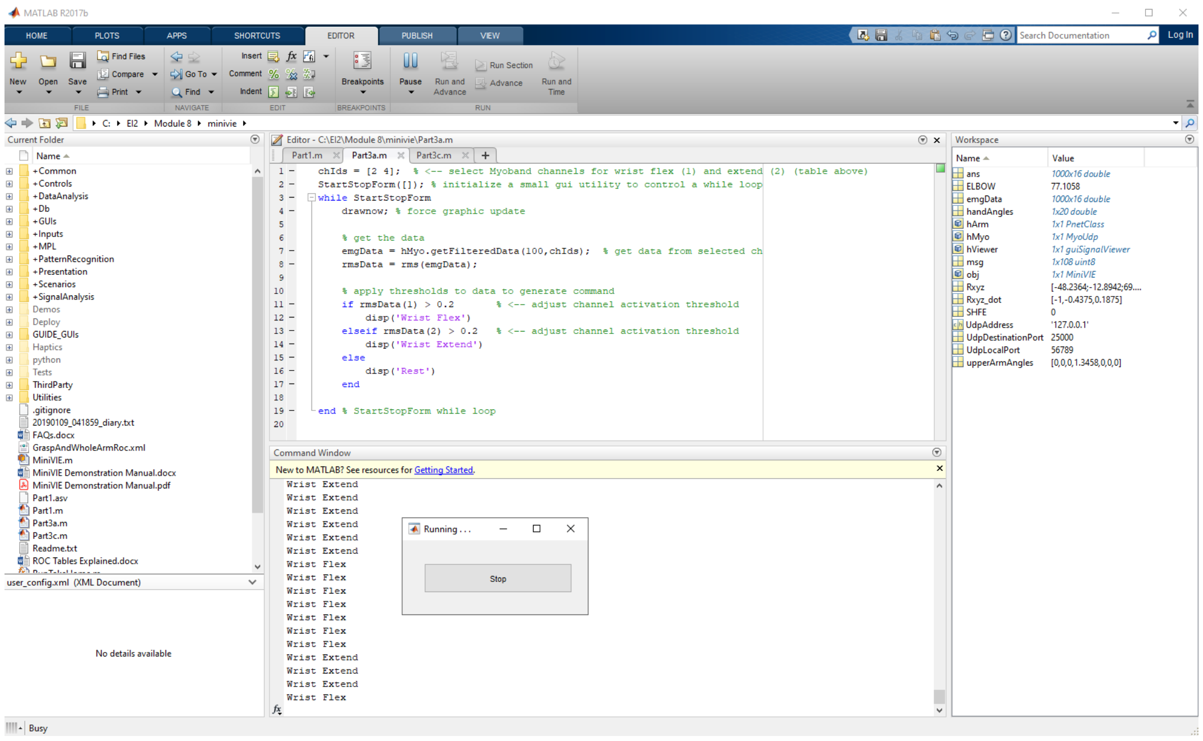
Note: In the example code below, the highlighted portion indicates the parameter to assign channels for wrist flexion and extension, based on which channels had the greatest RMS values. Also, set the activation threshold above the resting voltage but below the peak voltage so the arm only moves when commanded.

```
chIds = [2 4];  % <-- select EMG channels for wrist flex [X _] and extend [_ X] (table above)
StartStopForm([]); % initialize a small gui utility to control a while loop
while StartStopForm
    drawnow; % force graphic update

    % get the data
    emgData = hMyo.getFilteredData(100,chIds);  % get data from selected channel
    rmsData = rms(emgData);

    % apply thresholds to data to generate command
    if rmsData(1) > 0.2       % <-- adjust channel activation threshold value
        disp('Wrist Flex')
    elseif rmsData(2) > 0.2   % <-- adjust channel activation threshold value
        disp('Wrist Extend')
    else
        disp('Rest')
    end

end % StartStopForm while loop
```

Note: That when the “StopStartForm” is running, you will see a small window open with a “Stop” button:



As long as this window is open, the code between the line “while StartStopForm” and the line “end % StartStopForm while loop” will be executed over and over again, in a loop:

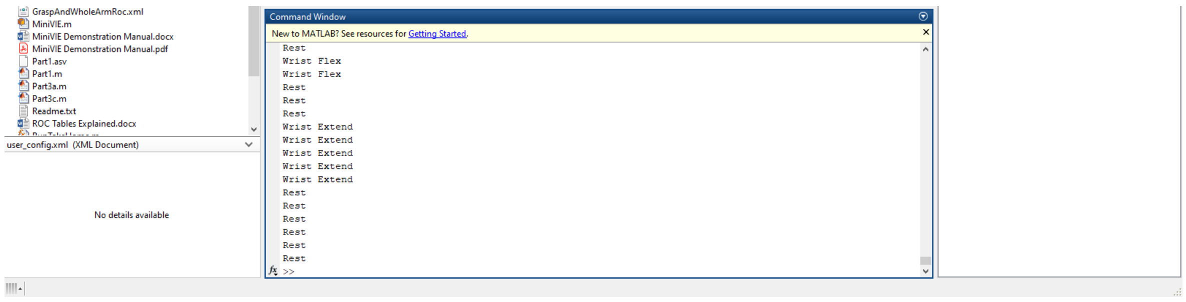


```
chIds = [2 4];  % <-- select Myoband channels for wrist flex (1) and extend (2) (table above)
StartStopForm([]); % initialize a small gui utility to control a while loop
while StartStopForm
    drawnow; % force graphic update

    % get the data
    emgData = hMyo.getFilteredData(100,chIds);  % get data from selected ch
    rmsData = rms(emgData);

    % apply thresholds to data to generate command
    if rmsData(1) > 0.2       % <-- adjust channel activation threshold
        disp('Wrist Flex')
    elseif rmsData(2) > 0.2   % <-- adjust channel activation threshold
        disp('Wrist Extend')
    else
        disp('Rest')
    end

end % StartStopForm while loop
```

```
New to MATLAB? See resources for Getting Started.
  Wrist Extend
  Wrist Extend
  Wrist Extend
  Wrist Extend
  Wrist Extend
  Wrist Extend
  Wrist Flex
  Wrist Flex
  Wrist Flex
  Wrist Flex
  Wrist Flex
  Wrist Flex
  Wrist Flex
  Wrist Extend
  Wrist Extend
  Wrist Extend
  Wrist Flex
```

| Name | Value |
|---|---|
| ans | 1000x16 double |
| ELBOW | 77.1058 |
| emgData | 1000x16 double |
| handAngles | 1x20 double |
| hArm | 1x1 PnetClass |
| hMyo | 1x1 MyoUdp |
| hViewer | 1x1 guiSignalViewer |
| msg | 1x108 uint8 |
| obj | 1x1 MiniVIE |
| Rxyz | [-48.2364;-12.8942;69.... |
| Rxyz_dot | [-1,-0.4375,0.1875] |
| SHFE | 0 |
| UdpAddress | '127.0.0.1' |
| UdpDestinationPort | 25000 |
| UdpLocalPort | 56789 |
| upperArmAngles | [0,0,0,1.3458,0,0,0] |

When you click on the “Stop” button on this open window, it will end the loop, and move onto the next line of code after “end % StartStopForm while loop” (which for this, given that there is no code after this line, means it will stop and the console prompt “>>” will return in the console:



```
New to MATLAB? See resources for Getting Started.
  Rest
  Wrist Flex
  Wrist Flex
  Rest
  Rest
  Rest
  Wrist Extend
  Wrist Extend
  Wrist Extend
  Wrist Extend
  Wrist Extend
  Rest
  Rest
  Rest
  Rest
  Rest
  Rest
  >>
```

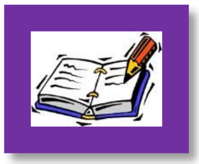
## Command joint velocity and position using EMG

Note: As in Part 3(a) above, the highlighted portions indicate parameters you will need to manually set.

```
chIds = [2 4];  % <-- select the same EMG channels for wrist flex (1) and extend (2)

tLast = tic;    % store the current time for real-time control
p = 0;          % store current position
direction = 1;  % specify a direction variable +/- 1

StartStopForm([]); % initialize a small gui utility to control a while loop
while StartStopForm
    drawnow; % force graphic update

    % create a dt (delta time) variable for velocity to position
    % integration.  Using dt will keep speed constant regardless of loop
    % time
    dt = toc(tLast);
    tLast = tic;

    % get the data
    emgData = hMyo.getFilteredData(100,chIds);  % get data from selected ch
    % handle 2 channels
    rmsData = rms(emgData);

    % threshold data to generate command of certain joint velocity/direction
    if rmsData(1) > 0.2        % <-- adjust channel activation threshold
        v = +1.2;  % fixed negative velocity
    elseif rmsData(2) > 0.2    % <-- adjust channel activation threshold
        v = -1.2;  % fixed positive velocity
    else
        v = 0;
    end

    % perform velocity integration to get position for joints
    p = p + (v * dt);

    % constrain the angle to the joint limits (between ±pi/4, or ±45deg)
    p = min(max(p,-pi/4),pi/4);

    % set the position of the wrist
    upperArmAngles(7) = p;
    msg = typecast(single([upperArmAngles,handAngles]),'uint8');
    hArm.putData(msg);

    % print the status
    fprintf('Channel #%d = %6.2f Channel #%d = %6.2f Wrist = %6.2f\n',...
        chIds(1), rmsData(1), chIds(2), rmsData(2),p);

end % StartStopForm while loop
```

## Reflection: Conventional ("Simple") Control Mode

Observing both the outputs to the MATLAB console window (e.g. "Wrist Extend", "Rest" or No Movement, "Wrist Flex"), and the movements of the vMPL, assess the performance of the conventional ("simple") EMG model. **How well it is able to discriminate between movement classes** when there is a wrist flexion EMG signal, when there is a wrist extension EMG signal, and when there is a no movement ("Rest") signal. Consider the original channel #s selected and their RMS values for movement and no movement ("Rest"). In considering how they might use their prosthesis, **describe how this simple control mode performance might impact a prosthetics user**.

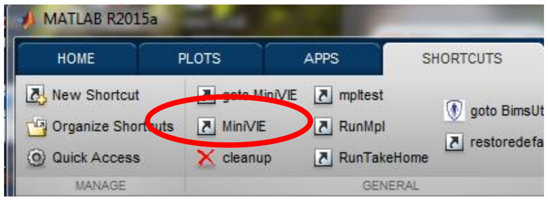


JOHNS HOPKINS
WHITING SCHOOL
*of* ENGINEERING

## Part 4: Complex Motions with Pattern Recognition

### Starting the MiniVIE Pattern Recognition Interface

1) Launch MiniVIE GUI using the "MiniVIE" link in the Shortcuts tab:

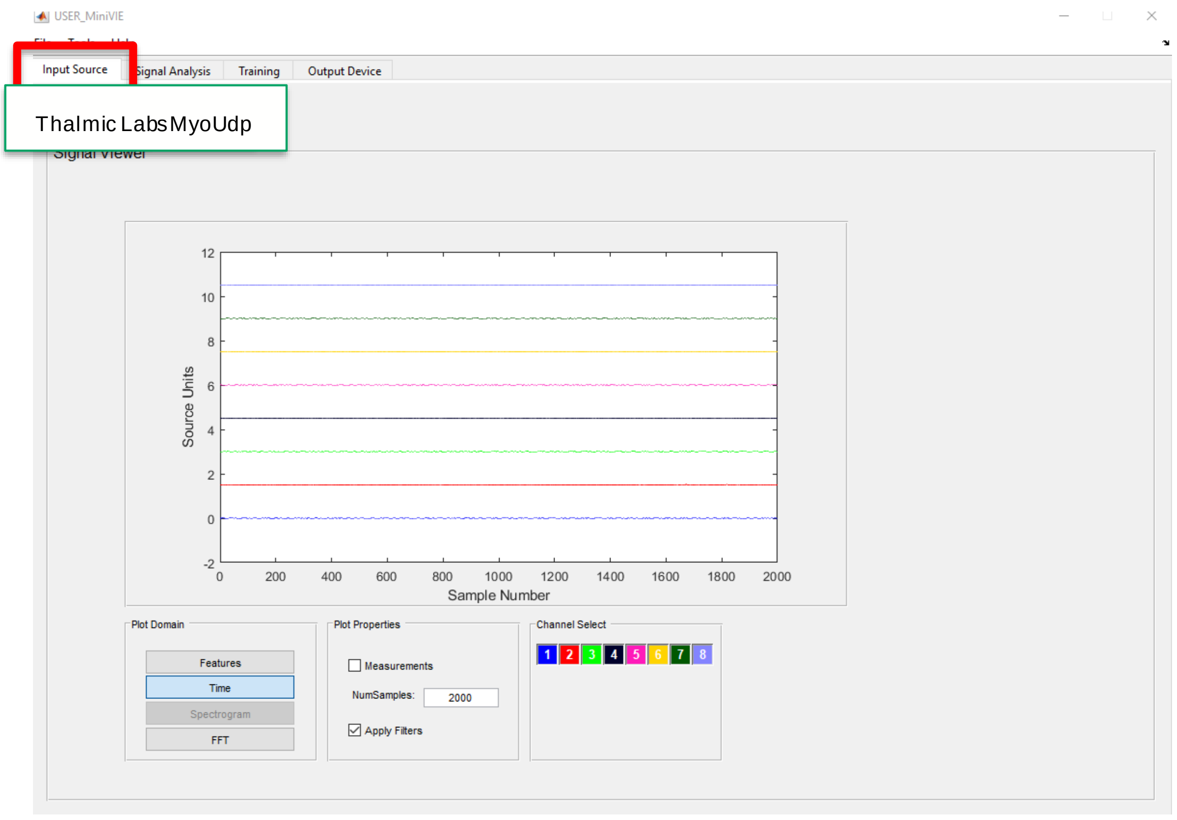


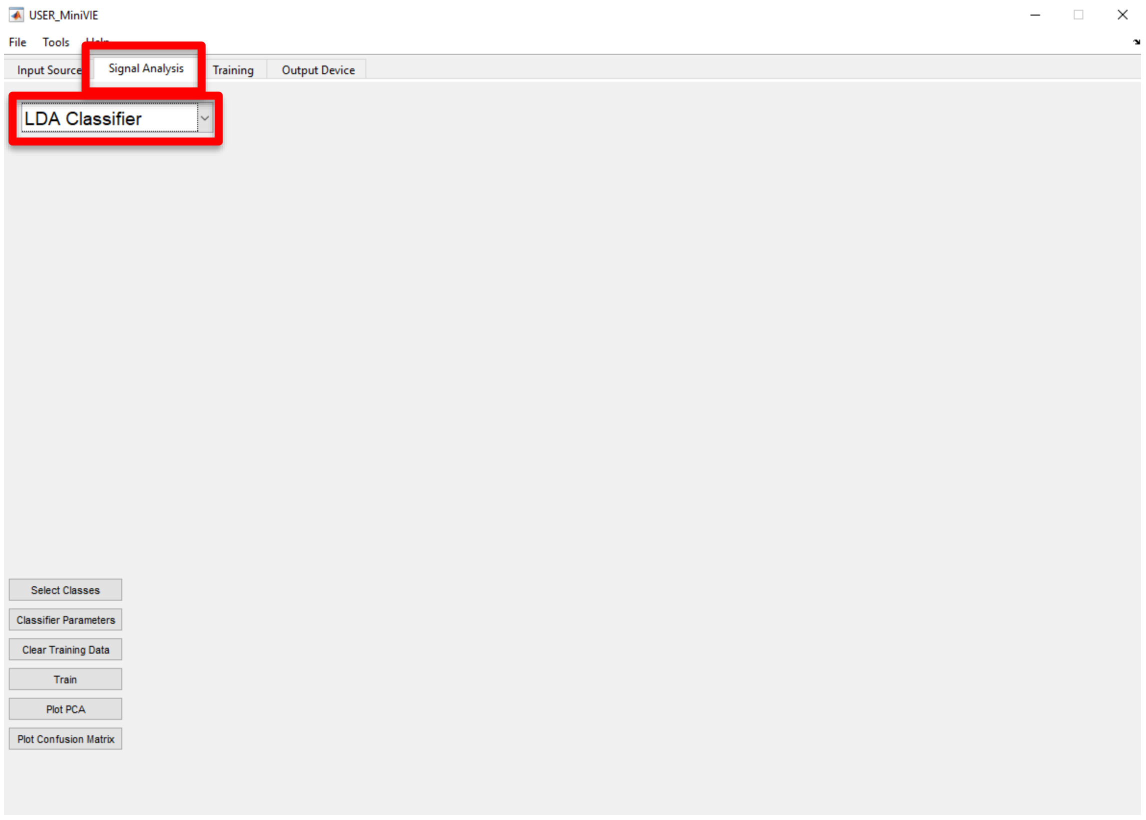
Note: If shortcuts don't exist, enter the following at the command prompt:

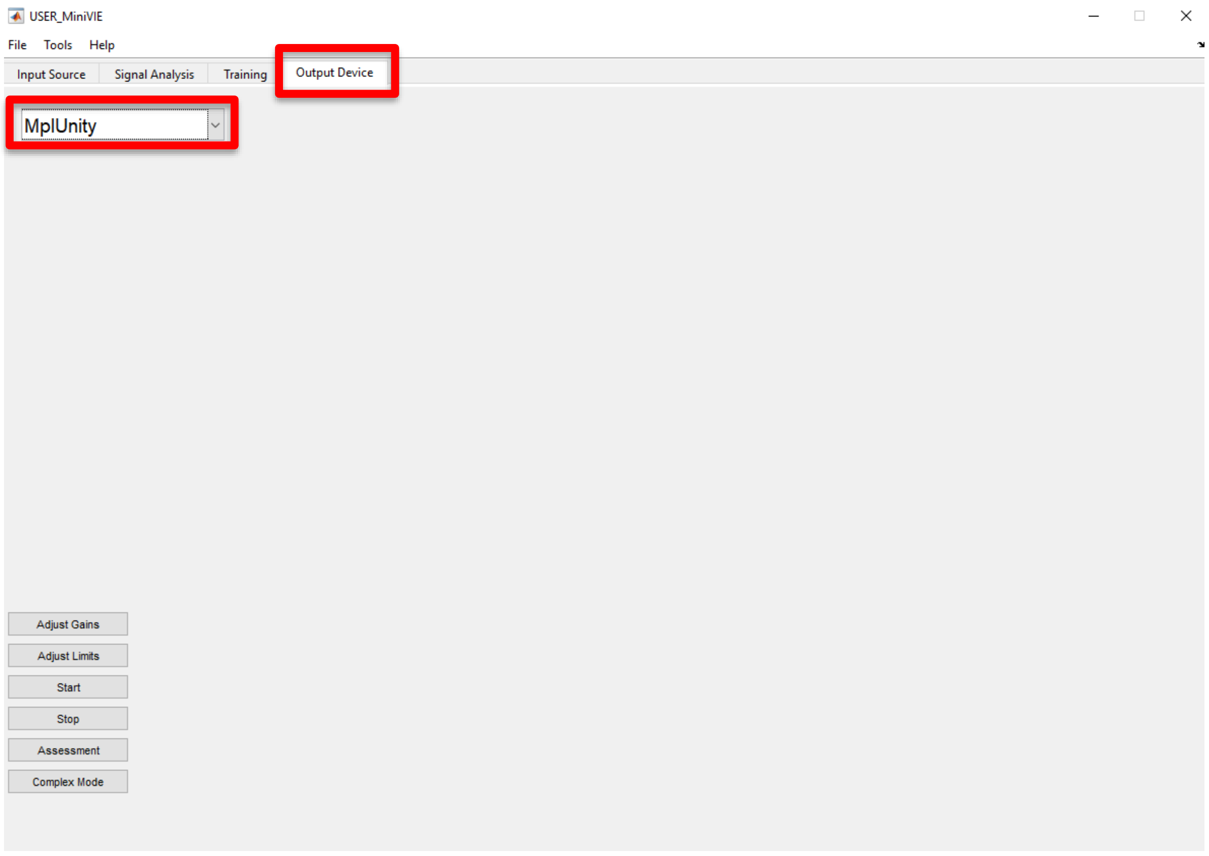
```
>> MiniVIE
```

2) If prompted for a configuration file, use the default "user_config.xml".

3) Set the drop-down menus within the MiniVIE window to those shown in the figure below. For Inputs, select "Thalmic Labs MyoUdp". For Signal Analysis, select "LDA Classifier" in the drop down box. For OutputDevice, select "MplUnity" in the drop down box.

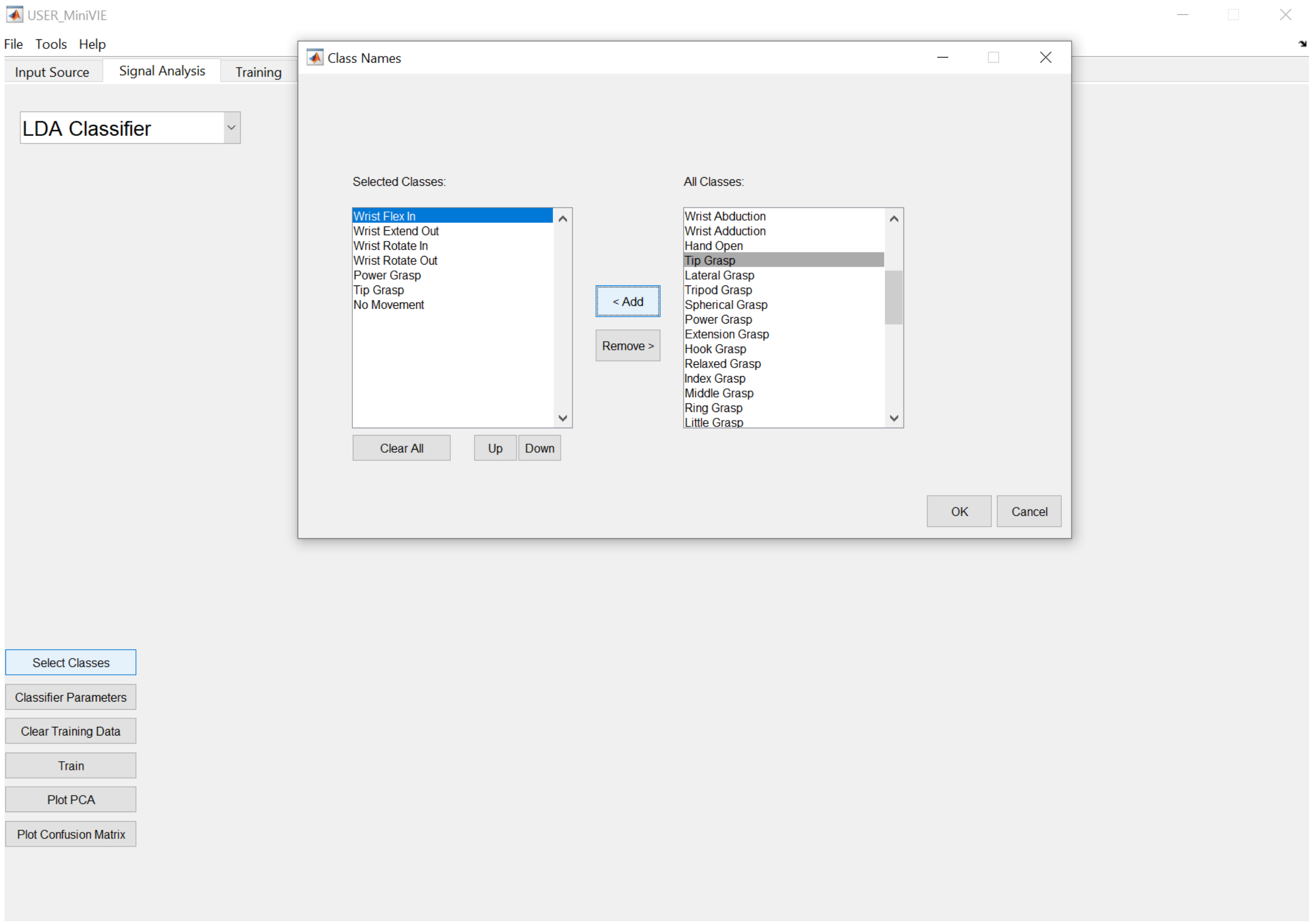



USER_MiniVIE
File
Tools
Help
Input Source
Signal Analysis
Training
Output Device
LDA Classifier
Select Classes
Classifier Parameters
Clear Training Data
Train
Plot PCA
Plot Confusion Matrix


USER_MiniVIE
File
Tools
Help
Input Source
Signal Analysis
Training
Output Device
MplUnity
Adjust Gains
Adjust Limits
Start
Stop
Assessment
Complex Mode

## Training Data

1) To train our interface to detect motions, we first need to select the "classes" of motions we'd like to discriminate between. In the Signal Analysis tab, click on Select Classes, and add all of the classes from the table you populated in Part 2, by clicking Add while having the corresponding classes highlighted.



Selected Classes:

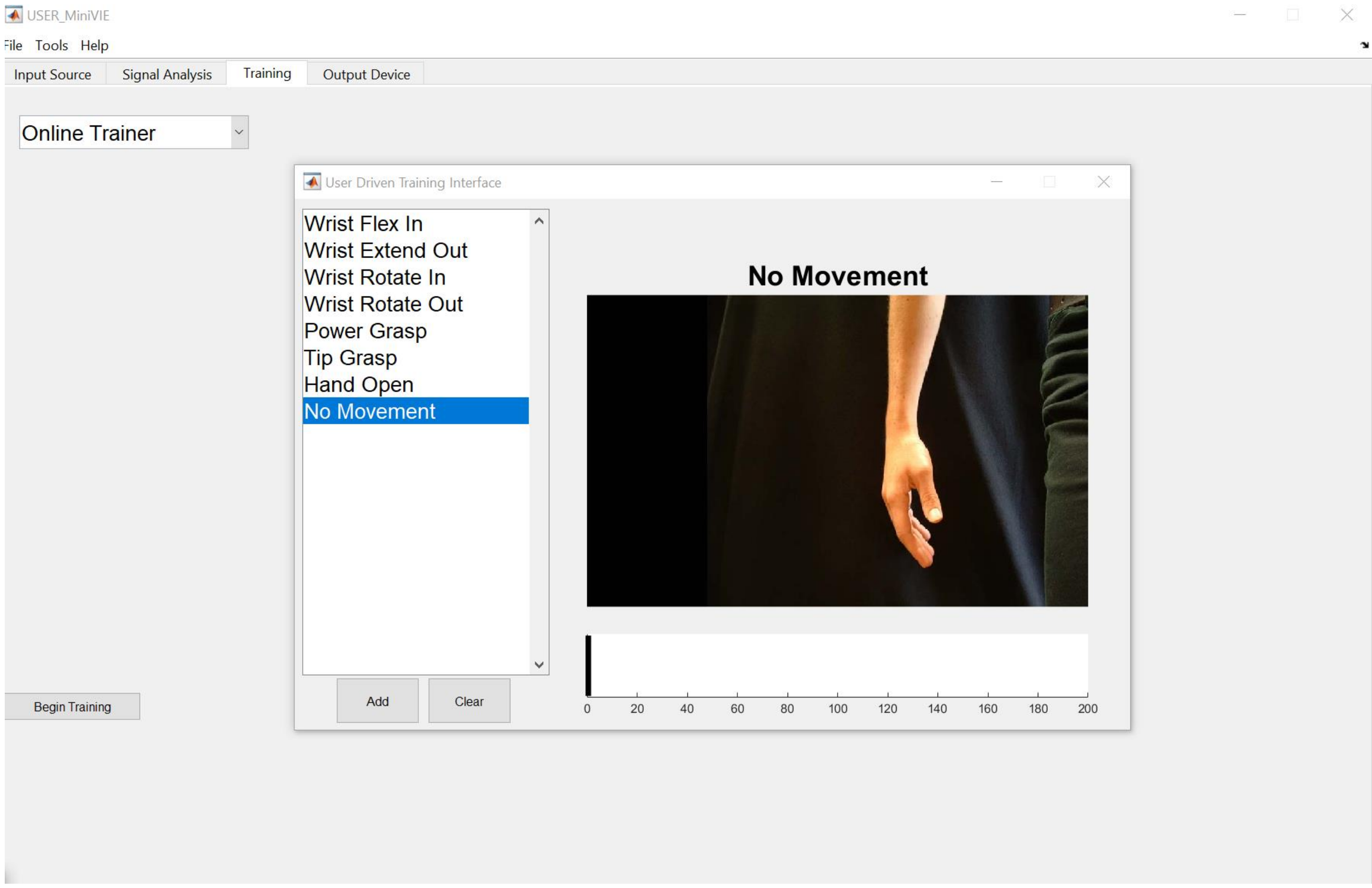
Wrist Flex In
Wrist Extend Out
Wrist Rotate In
Wrist Rotate Out
Power Grasp
Tip Grasp
No Movement

All Classes:

Wrist Abduction
Wrist Adduction
Hand Open
Tip Grasp
Lateral Grasp
Tripod Grasp
Spherical Grasp
Power Grasp
Extension Grasp
Hook Grasp
Relaxed Grasp
Index Grasp
Middle Grasp
Ring Grasp
Little Grasp

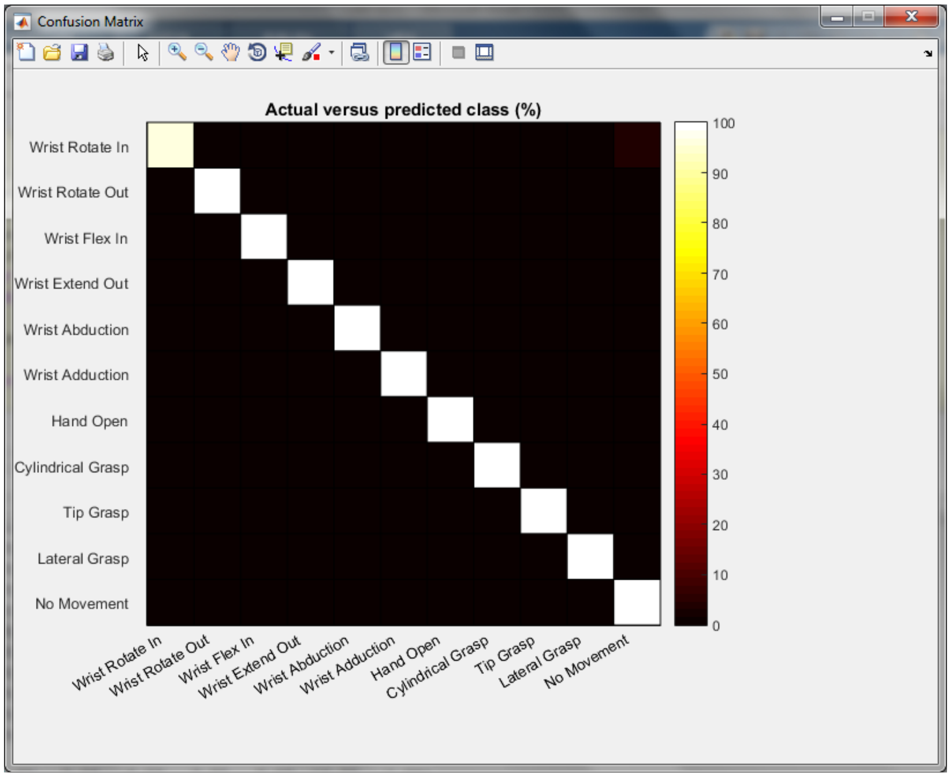
2) Note, if you are having odd behavior throughout the lab, try clicking "Clear Training Data" on the Signal Analysis tab.
3) Now, you will associate data to a motion you physically perform (which is known by you). In the world of machine learning, this is known as labeling our training data. To do this, go the TrainingTab, select the OnlineTrainer in the dropdown menu, and wait for the training interface window to pop up. (DON'T SELECT BEGIN TRAINING AS THIS BUTTON IS CURRENTLY BROKEN). Highlight each motion class, and **while you are performing the motion** (remember to activate muscles as if moving against resistance), click the "Add" button to add data. Click "Stop" to finish, trying to collect at least a second or two of data. If a class is behaving incorrectly, or you know you collected erroneous training data, click Clear to clear training data associated with just that class.
   a. NOTE: If the OnlineTrainer is not working (the progress bars aren't filling up), we recommend using simple trainer instead. You can use the default settings, or decrease the number of repetitions so you only need to do each movement once. This training style is different, in that

the GUI will prompt you through a set of motions. BE READY TO PERFORM THEM WHEN THE GUI SAYS SO. It may take a few tried to get this correct, so be sure to clear training data in between attempts, and **save your training data once you succeed**.



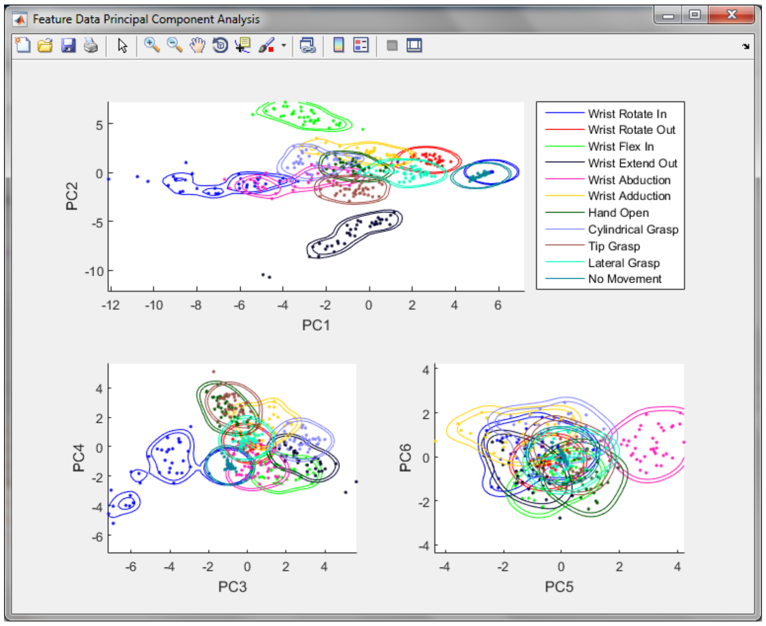
4) A ".trainingData" file is used to store EMG data and associated motion labels (the names of the motions or movement classes being performed during each frame of the EMG data). Please save this out once trained, by clicking File > Save Training Data in case of any software crashes, that way you can load it in at a later time. **Make sure you know where this is saved to as it will be needed later.** To load in the data, click File > Load Training Data and navigate to that file. However, the training will be invalid if you change the positioning of the MyoBand on your arm.

5) When we train these movement classes for our movement controller, we want to understand how easy it is for the controller to confuse one movement for another movement (i.e. how well the system is able to discriminate patterns in EMG activity associated with one movement with patterns of EMG activity associated with another). This can best be visualized with a confusion matrix. To see the confusion matrix for the movements that you trained, press the confusion matrix button (at the bottom of the "Signal Analysis" tab in the MiniVIE GUI). **Include a screen shot of the confusion matrix in your report**. **Write down the two movements with the lowest prediction accuracy.** Consider potential factors that contributed to error such as similarity of the movement, or improper training.



Sample Confusion Matrix output

Optional: In addition to the confusion matrix, you can also visualize the Principal Components of the EMG feature data to consider where data might overlap and lead to classification inaccuracies.

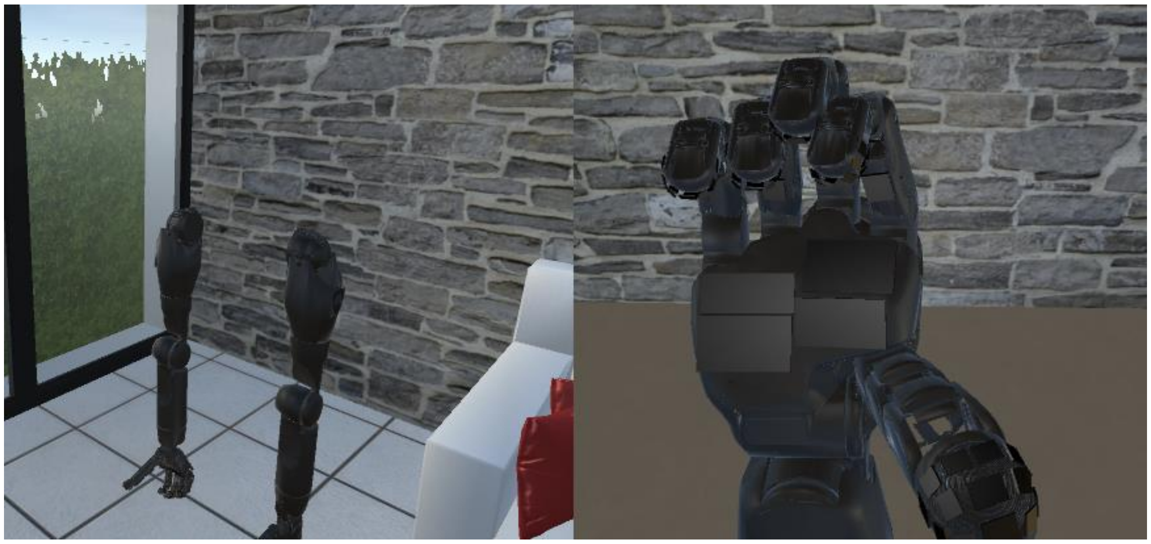


## Controlling the Arm with Pattern Recognition

1) With the MiniVIE GUI window still open, attempt to control the arm with the trained classifier by performing movements and looking at movements of trained joints in the vMPL window. Identify any movements that have confusion.

**Did these confused movements agree with the confusion matrix from earlier? Was the arm control less or more accurate than the confusion matrix implied? Comment on these in your report.**

vMPL Window during hand close movement:

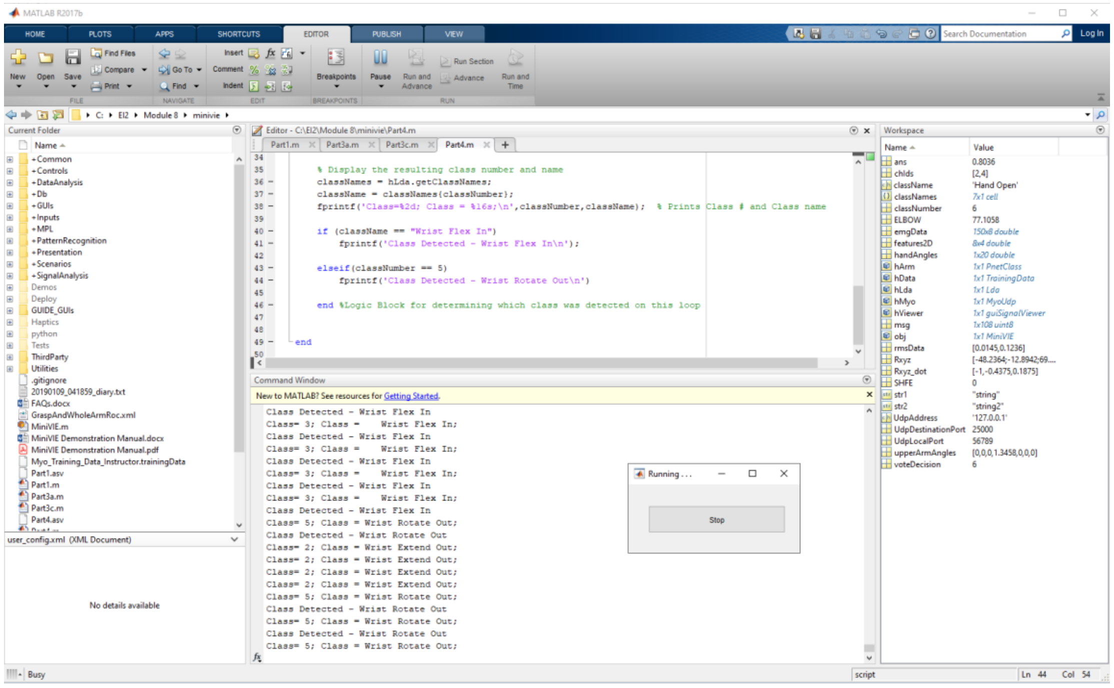


2) **<u>Close the MiniVIE GUI window</u>** – we will be using coding scripts moving forward.

3) Using the code below in a new m file script, we will start to develop our own controller by performing the same movement classification and vMPL commanding as in the MiniVIE GUI. Replace the highlighted training data filename in the code below with the name and location of the training data file you saved earlier.

```
% Perform pattern classification by loading training data and use a signal
% source and classifier object to derive intent based on the
% current signal state

% Load training data file
hData = PatternRecognition.TrainingData();
hData.loadTrainingData('C:\Temp\Myo_Training_Data.trainingData'); % <-- Your File Here
% If you get a prompt from MATLAB to open a '.xml' user configuration file,
% select the 'user_config.xml' file located in the miniVIE folder.

% Create EMG Myo Interface Object
hMyo = Inputs.MyoUdp.getInstance();
hMyo.initialize();

% Create LDA Classifier Object
hLda = SignalAnalysis.Lda;
hLda.initialize(hData);
hLda.train();
hLda.computeError();

StartStopForm([]); % initialize a small GUI utility to control a while loop
while StartStopForm

    % -------------------------------------------------------
    % -------- CLASSIFYING ARM AND HAND MOVEMENTS ----------
    % -------------------------------------------------------

    drawnow;

    % Get the appropriate number of EMG samples for the 8 Myo channels
    emgData = hMyo.getData(hLda.NumSamplesPerWindow,1:8);

    % Extract features and classify
    features2D = hLda.extractfeatures(emgData);
    [classNumber, voteDecision] = hLda.classify(reshape(features2D',[],1));

    % Display the resulting class number and name
    classNames = hLda.getClassNames;
    className = classNames{classNumber};
    fprintf('Class=%2d; Class = %16s;\n',classNumber,className);  % Prints Class # and Class name

    if (classNumber == 2) % Check for specific classNumber integer value

        fprintf('Class Detected - %16s\n', className); % Prints message to the console

    elseif(classNumber == 5) % Check for specific classNumber integer value

        fprintf('Class Detected - %16s\n', className) % Prints message to the console

    end %Logic Block for determining which class was detected on this loop, by Class Number


end % StartStopForm while loop
```

As you run this script, you can see in the console print-outs that the motion classes are detected as you make the same movements you used in when training the movement classes:



C: ▸ EI2 ▸ Module 8 ▸ minivie ▸

Current Folder

```
+Common
+Controls
+DataAnalysis
+Db
+GUIs
+Inputs
+MPL
+PatternRecognition
+Presentation
+Scenarios
+SignalAnalysis
Demos
Deploy
GUIDE_GUIs
Haptics
python
Tests
ThirdParty
Utilities
.gitignore
20190109_041859_diary.txt
FAQs.docx
GraspAndWholeArmRoc.xml
MiniVIE.m
MiniVIE Demonstration Manual.docx
MiniVIE Demonstration Manual.pdf
Myo_Training_Data_Instructor.trainingData
Part1.asv
Part1.m
Part3a.m
Part3c.m
Part4.asv
```

user_config.xml (XML Document)

No details available

Editor - C:\EI2\Module 8\minivie\Part4.m

Part1.m | Part3a.m | Part3c.m | Part4.m

```

        % Display the resulting class number and name
        classNames = hLda.getClassNames;
        className = classNames{classNumber};
        fprintf('Class=%2d; Class = %16s;\n',classNumber,className);  % Prints Class # and Class name

        if (className == "Wrist Flex In")
            fprintf('Class Detected - Wrist Flex In\n');

        elseif(classNumber == 5)
            fprintf('Class Detected - Wrist Rotate Out\n')

        end %Logic Block for determining which class was detected on this loop


    end

```

Command Window

New to MATLAB? See resources for Getting Started.

```
Class Detected - Wrist Flex In
Class= 3; Class =    Wrist Flex In;
Class Detected - Wrist Flex In
Class= 3; Class =    Wrist Flex In;
Class Detected - Wrist Flex In
Class= 3; Class =    Wrist Flex In;
Class Detected - Wrist Flex In
Class= 3; Class =    Wrist Flex In;
Class Detected - Wrist Flex In
Class= 5; Class = Wrist Rotate Out;
Class Detected - Wrist Rotate Out
Class= 2; Class = Wrist Extend Out;
Class= 2; Class = Wrist Extend Out;
Class= 2; Class = Wrist Extend Out;
Class= 2; Class = Wrist Extend Out;
Class= 5; Class = Wrist Rotate Out;
Class Detected - Wrist Rotate Out
Class= 5; Class = Wrist Rotate Out;
Class Detected - Wrist Rotate Out
Class= 5; Class = Wrist Rotate Out;
```

Workspace

| Name | Value |
|---|---|
| ans | 0.8036 |
| chIds | [2,4] |
| className | 'Hand Open' |
| classNames | 7x1 cell |
| classNumber | 6 |
| ELBOW | 77.1058 |
| emgData | 150x8 double |
| features2D | 8x4 double |
| handAngles | 1x20 double |
| hArm | 1x1 PnetClass |
| hData | 1x1 TrainingData |
| hLda | 1x1 Lda |
| hMyo | 1x1 MyoUdp |
| hViewer | 1x1 guiSignalViewer |
| msg | 1x108 uint8 |
| obj | 1x1 MiniVIE |
| rmsData | [0.0145,0.1236] |
| Rxyz | [-48.2364;-12.8942;69.... |
| Rxyz_dot | [-1,-0.4375,0.1875] |
| SHFE | 0 |
| str1 | "string" |
| str2 | "string2" |
| UdpAddress | '127.0.0.1' |
| UdpDestinationPort | 25000 |
| UdpLocalPort | 56789 |
| upperArmAngles | [0,0,0,1.3458,0,0,0] |
| voteDecision | 6 |

## Part 5: Putting all of the Pieces Together

This last part of the lab is focused on combining all of the various examples/capabilities that you've constructed or modified up to this point, and make a fully operational end-to-end EMG Based Motion Controller for the virtual prosthetic limb. These include:

1) The ability to send specific joint commands to the limb by first specifying a set of upper arm (`upperArmAngles`) and hand (`handAngles`) angles, and then sending a command to the virtual environment (Part 1 of lab)
2) The ability to generate and measure simulated EMG data using the EMG Simulator (Part 2 of lab)
3) The ability to process EMG data and classify which movement associated with that EMG data is being performed (Part 3 and 4 of lab)

For Part 5 you will combine/reuse the provided code blocks (e.g. from Part 1, and Part 4) within a single m script file to generate an end to end EMG Based Motion Controller. This Controller will process simulated EMG data captured from the MyoBand, determine the movement classes associated with that simulated EMG data, and then send the corresponding movement commands for those movement class to the vMPL system to move the virtual prosthetic limb. Your Controller must be able to perform each of these process steps, and will use the abilities described in the list above that you have already demonstrated for controlling wrist, hand positions, and elbow positions.

**Important Note: Your Controller will need to be able to <u>control at least *four* distinct movements</u>, <u>*two* of these being *hand* movement classes</u> (e.g. "Hand Open", "Power Grasp"). For the hand movements, you can just move one or two joints in the hand, as controlling all joints would take quite some time.**

Once you've implemented this Controller to move the vMPL using signals generated from the EMG Simulator, **take screen captures of two vMPL positions** to demonstrate arm control. **Save your control code script for Part 5 and include it in your lab report submission.**

**Note:** If you can't get your code working, then instead just describe how your system would work with written language.