1   Introduction

Researchers are currently working on ways to control devices using bioelectric signals. In this portion of the lab you will build an EMG signal classifier to implement myoeletric, or muscle, control in a game of Tetris in Matlab.

2   Software Setup

Streaming data from SpikerBox into Matlab requires a separate USB interfacing library that is implemented in Python. To get data in real time from SpikerBox, you will need to install Python and the Python USB library `hidapi`. The next several sections take you through how to install Python and the `hidapi` library.

2.1   Installation on Windows

Matlab Tetris requires Python to be installed to interface with the USB. This section walks through how to install Python, `pip`, and `hidapi`. If you already have any of these installed, jump to the appropriate section.

2.1.1   Install Python

To install Python, download the Python installer from Windows if you have a 64-bit computer. On the first screen, make sure to check the box that reads "Add Python 3.6 to PATH". Select "Install Now" and click "Yes" if your computer asks for administrator privileges:
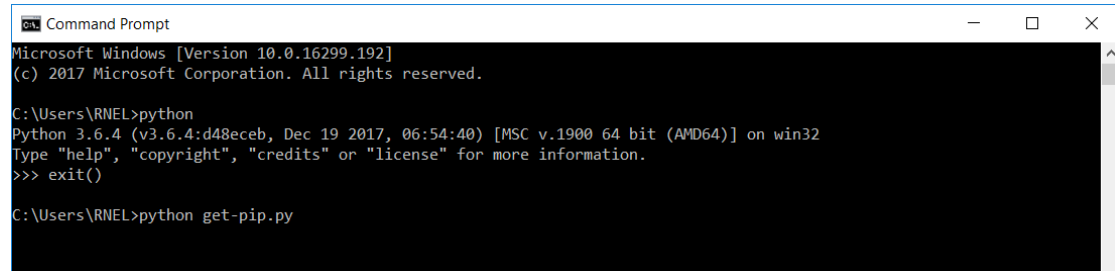

Once installation has completed, open "Command Prompt" in your start menu and ensure that Python is installed by typing `python` into the command line and pressing enter. If you see "Python 3.6.4" and other details, then Python has been correctly installed:
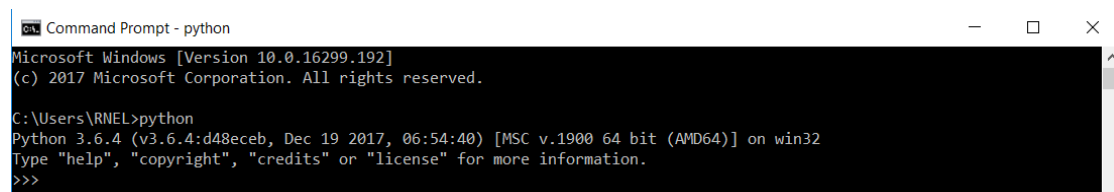
2.1.2   Install `pip`

Install `pip` by downloading the [script](#) **into your user directory** (`C:\Users\<USER>`). Exit the Python shell in the command line by typing `exit()` and install `pip` with the command:

```
python get-pip.py
```


Verify that `pip` is installed by typing `pip -version` into the command line and pressing `Enter`:

```
C:\Users\RNEL>pip -version
```

You should see a similar output:

```
pip 9.0.1 from c:\users\rnel\appdata\local\programs\python\python36\lib\site-packages
```

This means that `pip` has been successfully installed.

2.1.3   Install `hidapi`

The easiest way to install `hidapi` is through `pip` on Windows. In the command prompt, install `hidapi` with the following command and press enter:

```
C:\Users\RNEL>pip install hidapi
```

A quick way (optional) way to test your `hidapi` install is to run the following commands:

```
C:\Users\RNEL>python
```

Then, in the Python shell, try:

```
>>> import hid
```

If the import succeeds, `hidapi` has been successfully installed.

2.2   Download Myoelectric Tetris

Download the code for [Myoelectric Tetris](#) by downloading the `.zip` file and unzipping it into a directory of your choice. In the directory, there should be the following files:

1. `emg_buffer.py`: This is the Python script that gets EMG data from SpikerBox. **Do not change this file**.

2. `py_emg_buffer.m`: The Matlab connector to `emg_buffer.py`. **Do not change this file either.**

3. `test_emg_buffer.py`: A simple Python script to test the USB interface manager implemented in `emg_buffer.py`. Running this in a Python shell allows for debugging of the Python USB library, but should not be necessary for this portion of the lab.

4. `matlabtetris.m`: The main function that runs Tetris in Matlab, reads EMG data from SpikerBox, and calls the EMG classification function that you write. Run this function to start a game of Tetris.

5. `set_window_size.m`: Set the time window of EMG to buffer for classification, in milliseconds.

6. `emg_control.m`: The EMG signal classifier. This Matlab script takes in two channels of raw EMG input and outputs an array of length 3 corresponding to the direction the Tetris piece should move (left, right, or turn).

3   Implementing Myoelectric Control

In this portion of the lab, you will be adjusting settings on a root-mean-square-based EMG classifier to control pieces in a game of Tetris. Hook up the SpikerBox to electrodes on a forearm flexor-extensor muscle pair and turn SpikerBox on. Try running `matlabtetris.m`. You should see the Tetris game pop up in a new window. Press `Start`. The game should begin, and you will see a scatter plot of RMS values from each EMG channel.
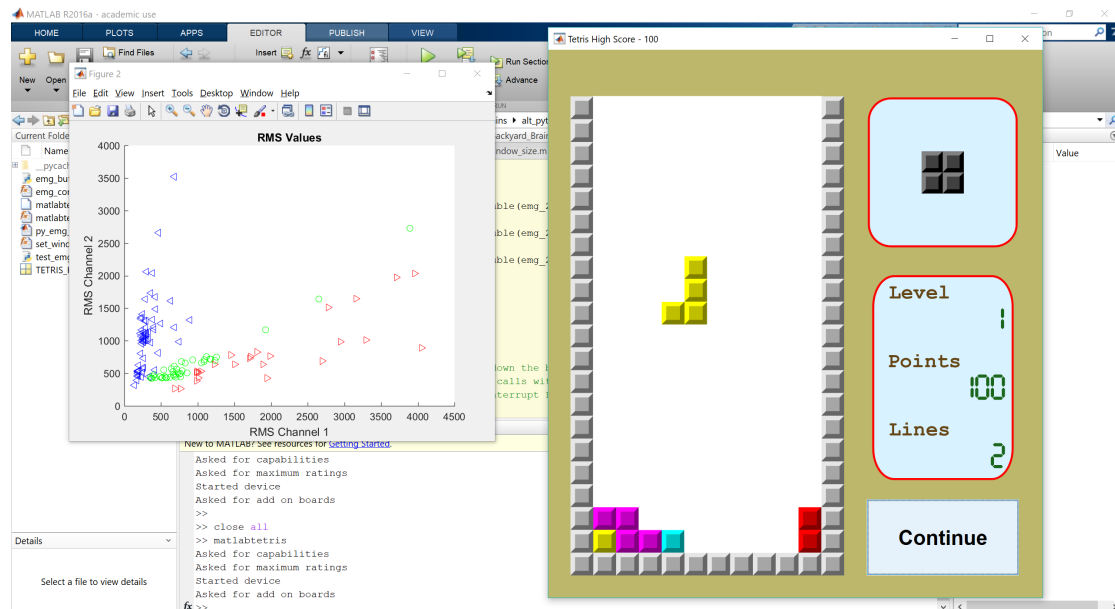
3.1   RMS Classification

Open `emg_control.m`. This Matlab function is where the classifier is implemented. It takes as input the raw signal of two EMG recording channels, which provide signal from your wrist flexor muscle and wrist extensor muscle. Using these raw EMG signals, `emg_control.m` computes the RMS of the signal on each channel and performs a simple thresholding based on the ratio of the RMS of the two channels. The function returns an array of length 3, where a value of 1 in the first element means the Tetris piece should move left, a value of 1 in the second element means the piece should move to the right, and a value of 1 in the last element indicates the Tetris piece should turn in the clockwise direction. Since the Tetris piece can only make one movement per timestep, only one element in the vector is nonzero. This is called a **one-hot encoding** scheme.

```
function [left, right, turn] = emg_control(emg_1, emg_2)
left = 0;
right = 0;
turn = 0;


rms_1 = rms(emg_1);
rms_2 = rms(emg_2);
```

Currently, the classifier is a rough threshold on the ratio of RMS of channel 1 to RMS of channel 2. Think about how you should set the thresholds on the RMS ratio based on which muscles your electrodes are hooked up to. For example, if channel 1 is recording the wrist flexors in your forearm, then a RMS ratio $\frac{RMS(Channel1)}{RMS(Channel2)} > 1$ would indicate that the wrist flexor muscles are contracting more strongly than the extensor muscles, so the piece should move to the left. Similarly, if the RMS ratio $\frac{RMS(Channel1)}{RMS(Channel2)} = 1$, then your flexors and extensors are co-contracting approximately equally, which would mean you should turn your Tetris piece.

```
if rms_1/rms_2 > 1.8
    right = 1;
elseif rms_1/rms_2 < 0.8
    left = 1;
```

```
elseif rms_1/rms_2 > 0.8 && rms_1/rms_2 < 1.8
    turn = 1;
end
```

Think about what thresholds or aspects of the EMG signal you would want to use for classification. The scatter plot of RMS values of each channel may help you.

The scatter plot in the figure above shows the classification of each point. As you classify EMG signals during the game, a point corresponding to (RMS(Channel1), RMS(Channel2)) will be plotted in the scatter plot with the color and shape indicating what your classifier labeled that point as. Blue arrows pointing left indicate that the classifier predicted that the Tetris piece should move left, red arrows pointing right indicate the Tetris piece moved right, and green circles indicate that the Tetris piece rotated.

Try playing Tetris a couple times and look at where the RMS values fall. Move the Tetris piece with some flexion, extension, and co-contraction movements. If your electrodes are on your left arm, then a wrist flexion indicates that the Tetris piece should move to the right. Look at where the point shows up–does the classifier indeed plot a red rightward arrow for that point? Try a couple of other movements. Where do these points fall? Are there decision boundaries you can draw in this two-dimensional RMS space that will help you classify the movement the Tetris piece should do? Change the `if` statements in `emg_control.m` to RMS ratios or threshold values that you think will classify the EMG signal the best.

3.2   EMG Window Size

Another factor in classification accuracy is the time window of EMG signal that we use. In `set_window_size.m`, set the length of the EMG signal (in milliseconds) you want to use for classification. Try different values ranging from 5ms to 500ms. What are the advantages of using a shorter time window? A longer time window?

```
function window_size = set_window_size()
window_size = 100; % in ms
end
```