NeuroSys 3D-Printed Prosthetic Hand Control

Technical Report

Adaptive Systems Laboratory
Division of Computer Engineering School
of Computer Science and Engineering
University of Aizu

PHEA Sinchhean, BEN ABDALLAH Abderazek
Contact: [m5251138, benab]@u-aizu.ac.jp
Edition: May 13, 2022

TODO Steps

1. Directory Structure
2. Setting up Myo Armband Sensor Driver & SDK
3. sEMG Data Collection
4. Deep Learning Model Training
5. Online Testing on Computer Software
6. Prosthetic Hand Control

Directory Structure

- sEMG_recog
  - data — Data used for training
  - train_model.py — Train the model
  - results — Accuracy results and model obtained from training
  - live_emg_test.py — Online testing with sensor using the trained model
  - convert_to_lite.py — Convert the trained model to tflite model for Raspberry Pi
  - converted_model.tflite — Converted tflite model

This directory is used for training a model on Windows

- PyoConnect_v2.0
  - motor_con.py — File used for motor control setting
  - test.py — File used for testing motors
  - myo_raw.py — File used for real-time inference and prosthetic hand controlling
  - converted_model.tflite — tflite model used for inference on Raspberry Pi
  - Other files — Many other files come with the library

This directory is used for controlling prosthetic hand using Raspberry Pi

PyoConnect is a library for bluetooth connection between Myo armband and Raspberry Pi

1. Setting up Myo Armband Sensor Driver & SDK

1. Download and install Myo connect on your system to use Myo armband sensor (Provided)
   - (windows version)
   https://github.com/NiklasRosenstein/myo-python/releases/download/v1.0.4/Myo+Connect+Installer.exe
   - Check if the sensor is connected to the computer.
2. Download the Myo sensor SDK (Provided)
   - https://github.com/NiklasRosenstein/myo-python/releases/download/v1.0.4/myo-sdk-win-0.9.0.zip

Other versions are available here: https://github.com/NiklasRosenstein/myo-python/releases

2. sEMG Data Collection (windows SDK)

1. On windows, make sure Visual Studio is installed.
2. Go to **myo-sdk-win-0.9.0** (the downloaded SDK) → **samples**
3. Open & run **emg-data-sample-VisualStudio2013** to test if the sEMG data is sent to the computer.
   - If array of number is shown on the screen, the data is sent successfully.
4. To save the captured data to a file, we want to modify the **emg-data-sample-VisualStudio2013** project
   - The file is provided. But please follow to understand.

~continue, sEMG Data Collection (windows SDK)

Within class DataCollector, print() function, modify the code as shown on the right.

- Create a directory where to save the data
- Name the file properly as we have to save multiple files. We used **.csv** format here.
- Include fstream library on top of the file **#include <fstream>**
- Instead of just displaying the data on the screen, we append the data to the file.

```
void print()
{
    // Clear the current line
    std::cout << '\r';

    std::ofstream  log;
    log.open("./test.csv", std::ofstream::app);

    // Print out the EMG data.
    for (size_t i = 0; i < emgSamples.size(); i++) {
        std::ostringstream oss;
        oss << static_cast<int>(emgSamples[i]);

        std::string emgString = oss.str();

        //std::cout << emgString << std::string(4 - emgString.size(), ' ') << ',';
        std::cout << emgString << std::string(4 - emgString.size(), ' ') << ',';
        log << emgString << std::string(4 - emgString.size(), ' ') << ',';

    }
    log << '\n';
    //std::cout << std::flush;
    std::cout << "\n";


}
```

~continue, sEMG Data Collection (windows SDK)

Under main() function, modify the infinite while loop to capture the signal as shown.

- We only capture 200 timesteps of data each time.
- The sample frequency is changed.

```
while (1) {
    if (i == 200) {
        return 1;
    }
    //auto stop = std::chrono::high_resolution_clock::now();
    //auto duration = std::chrono::duration_cast<std::chrono::microseconds>(stop - start);
    //if (duration.count() >= 1000000) return 1;

    // In each iteration of our main loop, we run the Myo event loop for a set number of milliseconds.
    // In this case, we wish to update our display 20 times a second, so we run for 1000/20 milliseconds.
    hub.run(1);
    /*
    1000 -> 1Hz (1/s)
    500 -> 2Hz
    200 ->  5Hz
    50  -> 20Hz
    5   -> 200Hz
    20 -> 50Hz
    1 -> 1000Hz
    */
    // After processing events, we call the print() member function we defined above to print out the values we've
    // obtained from any events that have occurred.
    collector.print();
    i++;

}
```

~continue, sEMG Data Collection (windows SDK)

Try to collect the sEMG data as much as possible for making better Deep learning Model.

- In the example, we collect 7 gestures (including rest), 20 times for each gesture.
- Collect the data in different sessions E.g. Today collect once, tomorrow collect once again.
- In the example, we have collected only 2 sessions of data.

close-01.csv	10/19/2021 11:38 AM	Microsoft Excel C...	9 KB
close-02.csv	10/19/2021 11:39 AM	Microsoft Excel C...	9 KB
close-03.csv	10/19/2021 11:39 AM	Microsoft Excel C...	9 KB
close-04.csv	10/19/2021 11:40 AM	Microsoft Excel C...	9 KB
close-05.csv	10/19/2021 11:40 AM	Microsoft Excel C...	9 KB
close-06.csv	10/19/2021 11:40 AM	Microsoft Excel C...	9 KB
close-07.csv	10/19/2021 11:40 AM	Microsoft Excel C...	9 KB
close-08.csv	10/19/2021 11:40 AM	Microsoft Excel C...	9 KB
close-09.csv	10/19/2021 11:40 AM	Microsoft Excel C...	9 KB
close-10.csv	10/19/2021 11:41 AM	Microsoft Excel C...	9 KB
close-11.csv	10/19/2021 11:41 AM	Microsoft Excel C...	9 KB
close-12.csv	10/19/2021 11:41 AM	Microsoft Excel C...	9 KB
close-13.csv	10/19/2021 11:42 AM	Microsoft Excel C...	9 KB
close-14.csv	10/19/2021 11:42 AM	Microsoft Excel C...	9 KB
close-15.csv	10/19/2021 11:42 AM	Microsoft Excel C...	9 KB
close-16.csv	10/19/2021 11:42 AM	Microsoft Excel C...	9 KB
close-17.csv	10/19/2021 11:42 AM	Microsoft Excel C...	9 KB
close-18.csv	10/19/2021 11:42 AM	Microsoft Excel C...	9 KB
close-19.csv	10/19/2021 11:43 AM	Microsoft Excel C...	9 KB
close-20.csv	10/19/2021 11:43 AM	Microsoft Excel C...	9 KB
hold-01.csv	10/19/2021 12:02 PM	Microsoft Excel C...	9 KB
hold-02.csv	10/19/2021 12:03 PM	Microsoft Excel C...	9 KB
hold-03.csv	10/19/2021 12:03 PM	Microsoft Excel C...	9 KB
hold-04.csv	10/19/2021 12:03 PM	Microsoft Excel C...	9 KB

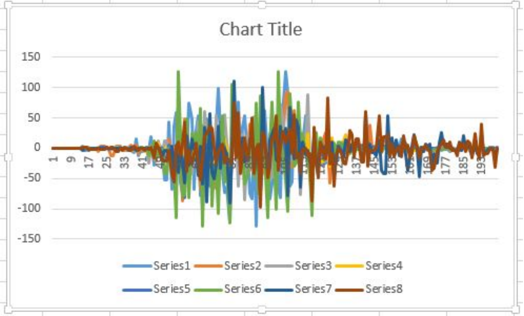
~continue, sEMG Data Collection (windows SDK)

There are many approaches to capture the sEMG data.

Here we captured starting from rest state to a gesture and back to rest state again.

You can check the saved sEMG data using Excel and create a graph as shown on the right.

	A	B	C	D	E	F	G	H
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0
14	3	1	0	0	-2	-2	-3	2
15	-1	0	0	-3	-3	0	1	1
16	1	0	-3	-1	-1	-1	2	2
17	-1	-2	-1	1	-1	-1	-3	-2
18	-2	-1	-1	0	-2	-1	-1	-2
19	-2	-1	-1	0	-2	-1	-1	-2
20	-1	-1	0	0	-1	0	-1	-2
21	-2	-5	-2	2	-1	0	0	-1
22	-2	3	-1	0	1	0	0	-2
23	-2	3	-1	0	1	0	0	-2
24	-2	3	-1	0	1	0	0	-2
25	0	1	-1	-1	1	-1	-1	1
26	-2	-2	-1	-1	-2	0	0	0
27	0	-12	-2	0	0	-1	0	1
28	0	-12	-2	0	0	-1	0	1
29	-2	3	1	0	-1	0	-1	-5
30	-2	3	1	0	-1	0	-1	-5
31	1	3	0	1	1	-1	0	-1



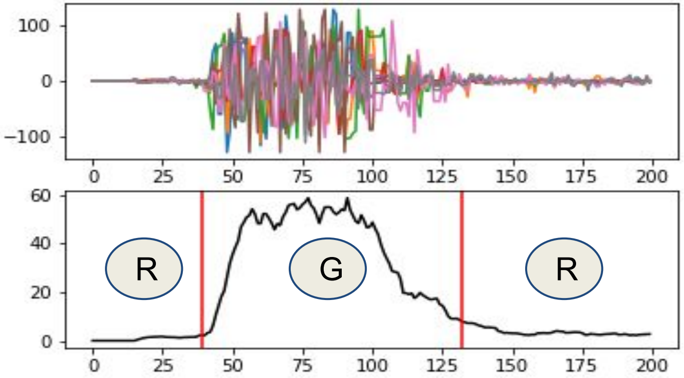
*It is assumed the data is stored in mydata.

3. Deep Learning Model Training

There are many approaches to train a deep learning AI model.

Here we used **standard deviation** and **RNN(LSTM, GRUs)**.

- Implementation using **Python, TensorFlow**.
- Standard deviation is used to detect the where gesture starts and stops.



- Divide into windows
  - Apply standard deviation
    - standard deviation of each channel

$$\sigma_c = \sqrt{\frac{\sum^W (x_c - \mu)^2}{W}}$$

    - average standard deviation

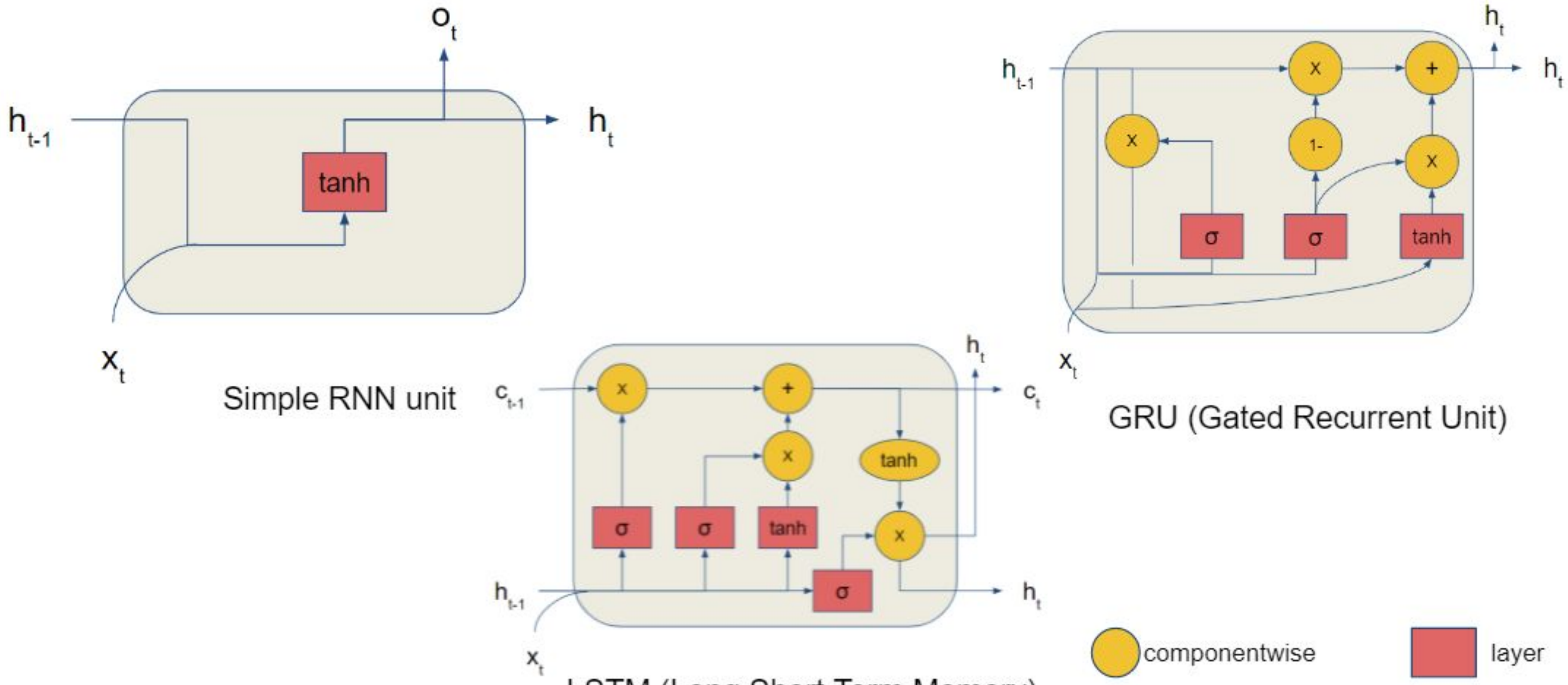
$$\overline{\sigma} = \sum^C \frac{\sigma_c}{C}$$

  - Thresholding
    - $\overline{\sigma}$ < threshold: Rest
    - Otherwise: Gesture

x : signal data point
W : window size
C : number of channels
μ : mean

~continue, Deep Learning Model Training

Comparing simple RNN, LSTM, and GRU units.

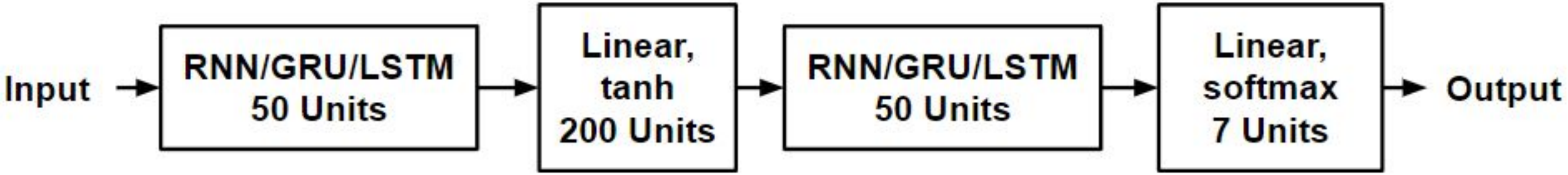


Simple RNN unit

LSTM (Long Short Term Memory)

GRU (Gated Recurrent Unit)

~continue, Deep Learning Model Training



We are using the above RNN,GRU,LSTM architecture. The last layer is softmax with 7 outputs to match with the 7 classes of gesture.

- 10% of data is used for offline evaluation
- Evaluation based on 5-fold cross validation
- Use different time window sizes to evaluate [10, 20, 30, 40, 50]
  The more data points we use, the better the accuracy but higher delay
- At the end the trained models are saved in a directory.

~continue, Deep Learning Model Training

Use train_model.py to train an RNN model

Specify the classes, cross-validation time, datapoint, and number of repetition, and number of clsses

```
class_list = ["close","open","one","two","three","hold","R"]

howmany = 5
datapoint = 200
repeat_num = 40
classnum = 7
```

Specify the data directory, and parameters used for labelling.

```
if __name__ == "__main__":
    create_model = ModelCreation(subj_name='phea', w1=10, w2=5, start_thresh=8, save_detection_plots=True,)
    create_model.train()
    create_model.test()
```

*After training you can convert the trained model to tflite model for Raspberry Pi

4. Online Testing on Computer Software

Before testing with physical 3D-printed hand, we can test on computer software simulation.

Python is used for this task.

1. Install myo-python in your python environment
   `pip install myo-python`
2. Download myo-python from github (provided)
   https://github.com/NiklasRosenstein/myo-python
3. We are using **live_emg_test.py** to run online testing

~continue, Online Testing on Computer Software

Modify the **live_emg_test.py** file to run online testing

Import tensorflow

```
import tensorflow as tf
from tensorflow.keras import models
```

Define classes and timesteps used to train the model

```
class_list = ["close","hold","one","open","R","three","two"]
timestep = 200
```

Using the trained model to predict the live emg data input

```
def display_data(self):
  emg_data = self.listener.get_emg_data()
  emg_data = np.array([x[1] for x in emg_data])


  if emg_data.size == 0:
    print("No data")
  else:
    data = np.zeros((1,timestep,8))
    data[0,:emg_data.size//8,:] = emg_data

    model_path = "D:\\ASL\\Masters\\NeuroSys\\results\\phea1\\CV_results\\cv_5\\model_cv_5.h5"
    model: models.Model
    model = models.load_model(model_path)

    predicted = model.predict(data)

    pre_class = class_list[int(np.argmax(predicted[0,emg_data.size//8-1,:]))]
    print(pre_class)
```

- Point to the SDK bin directory
- Define the window size to capture sEMG data

```
def main():
  myo.init(bin_path=r'D:\\ASL\\Masters\\myo\\myo-sdk-win-0.9.0\\bin')
  hub = myo.Hub()
  listener = EmgCollector(40)
  with hub.run_in_background(listener.on_event):
    Plot(listener).main()
```

5. Prosthetic Hand Control

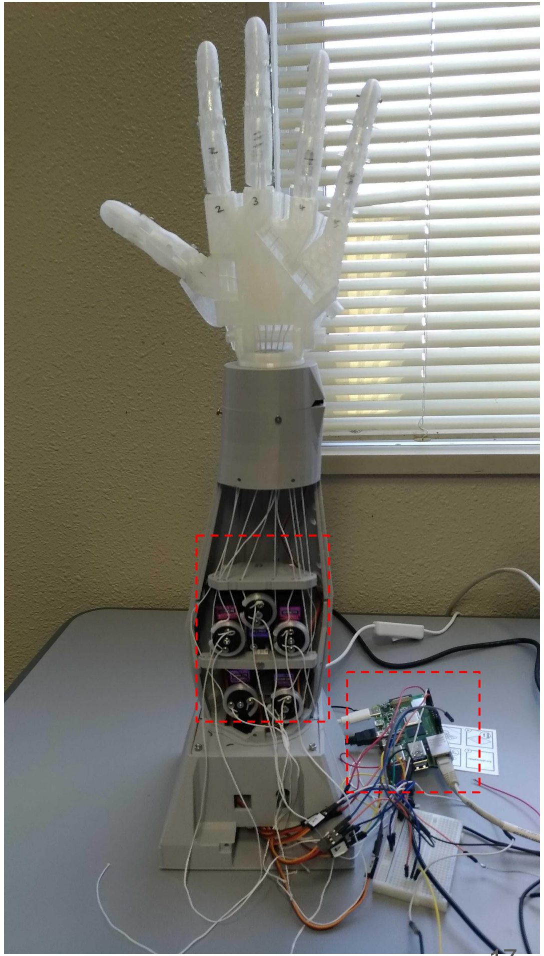
Raspberry Pi is used to control the 5 motors controlling the fingers of the hand.

Convert the trained tensorflow keras model (64-bit) to tensorflow lite model (32-bit)
- Raspberry Pi by default only supports 32-bit.

```
import tensorflow as tf
from tensorflow.keras import models

model_path = r"D:\ASL\Masters\NeuroSys\results\phea_left_combine\CV_results\cv_4\model_cv_4.h5"

model = tf.keras.models.load_model(model_path)
converter = tf.lite.TFLiteConverter.from_keras_model(model)

tflite_model = converter.convert()
open("converted_left_model.tflite", "wb").write(tflite_model)
```

~continue, Prosthetic Hand Control

1. Because the SDK and other libraries used on Windows are not supported on linux, we are using Pyoconnect 2.0. http://www.fernandocosentino.net/pyoconnect/PyoConnect_v2.0.zip
2. Modify to run and inference the sEMG signal using the trained model. (Provided)



~continue, Prosthetic Hand Control

myo_raw.py

Importing tflite libraries and made motor_con file

```
import tflite_runtime
from tflite_runtime.interpreter import Interpreter
import numpy as np

from motor_con import motor
```

```
##############################################################################
########################################Edit Here###########################
##############################################################################
    class_list = ["close", "hold", "one","open", "three", "two", "R"]
    emgdata = []
    avg = []
#      tmp_angle = None
    def proc_emg(emg, moving, times=[]):
        if HAVE_PYGAME:
            ## update pygame display
            plot(scr, [e / 500. for e in emg])
#             print(emg)
            emg = list(emg)
            #print(len(emg_data))
            emgdata.append(emg)
            if len(emgdata) == 60:
                #print(emg_data)
                emg_data = np.array(emgdata)
                timestep = 200
                #print(emg_data)
                data = np.zeros((1,timestep,8), dtype=np.float32)

                data[0,:emg_data.size//8,:] = emg_data
                #tt = data
                model_path = "/home/pi/model.tflite"
                interpreter = Interpreter(model_path)
                #allocate the tensors
                interpreter.allocate_tensors()
                #print(data.shape)
                #Get input and output tensors
                input_details = interpreter.get_input_details()
                output_details = interpreter.get_output_details()

                interpreter.set_tensor(input_details[0]['index'], data)

                interpreter.invoke()

                predicted = interpreter.get_tensor(output_details[0]['index'])
                tt = predicted
                print(predicted[0,emg_data.size//8-1,:])
                avg.append(predicted[0,emg_data.size//8-1,:])
                pre_class = class_list[int(np.argmax(predicted[0,emg_data.size//8-1,:]))]
```

The inference and motor controlling part is within the ##Edit Here## part

```
                #average of 2 predictions and then move the motors
                if (len(avg) == 2):
                    new_motor = motor()
                    ree = avg[0]
                    for i in range(len(avg)-1):
                        ree += avg[i+1]
                    pd_class = class_list[int(np.argmax(ree))]
                    print(pd_class)
                    if (pre_class == "open"):
                        new_motor.SetAngle([0, 0, 0, 0, 0])
                        time.sleep(1)
                    if (pre_class == "R"):
                        new_motor.SetAngle([90, 90, 90, 90, 90])
                        time.sleep(1)
                    elif (pre_class == "close"):
                        new_motor.SetAngle([180, 180, 180, 180, 180])
                        time.sleep(1)
                    elif (pre_class == "one"):
                        new_motor.SetAngle([180, 180, 180, 0, 90])
                        time.sleep(1)
                    elif (pre_class == "two"):
                        new_motor.SetAngle([180, 180, 0, 0, 90])
                        time.sleep(1)
                    elif (pre_class == "three"):
                        new_motor.SetAngle([180, 0, 0, 0, 90])
                        time.sleep(1)

                    new_motor.cleanup()
                    del avg[:]

                del emgdata[:]
        ## print framerate of received data
        times.append(time.time())
        if len(times) > 20:
            #print((len(times) - 1) / (times[-1] - times[0]))
            times.pop(0)

    m.add_emg_handler(proc_emg)
    m.connect()

    m.add_arm_handler(lambda arm, xdir: print('arm', arm, 'xdir', xdir))
    #m.add_pose_handler(lambda p: print('pose', p))

##############################################################################
##############################################################################
##############################################################################
```

Demo