**Abstract:**

The ability to perform activities of daily living (ADL) is essential to preserving functional independence and quality of life[1]. Hence, monitoring and using body-worn sensors provides an objective alternative to current assessment tools [1]. In the case of hand and upper arm movements, Surface Electromyography based armband sensors are used to develop motion-based gesture controllers which; for instance; are viable in the development of prosthetic limbs [2]. The purpose of this study is to develop an interface using armband SEMG sensors to generate a 3D representation of a hand within the Unity. This will necessitate a thorough grasp of SEMG (Surface Electromyography), a deep exploration into the functionality of armband sensors designed for SEMG data collection, and the acquisition of proficiency in utilizing SEMG datasets to create a realistic 3D hand model within the Unity platform.

1. **Introduction: Theoretical Background**

Electromyography (EMG) is the study of muscle function through analysis of the electrical signal associated with the activation of the muscle. This may be voluntary or involuntary muscle contraction. The EMG activity of voluntary muscle contractions is related to tension. The functional unit of the muscle contraction is a motor unit, which is comprised of a single alpha motor neuron and all the fibers it enervates. This muscle fiber contracts when the action potentials (impulse) of the motor nerve which supplies it reaches a depolarization threshold. The depolarization generates an electromagnetic field and the potential is measured as a voltage. The depolarization, which spreads along the membrane of the muscle, is a muscle action potential. The motor unit action potential is the spatio and temporal summation of the individual muscle action potentials for all the fibers of a single motor unit. Therefore, the EMG signal is the algebraic summation of the motor unit action potentials within the pick-up area of the electrode being used. The pick-up area of an electrode will almost always include more than one motor unit because muscle fibers of different motor units are intermingled throughout the entire muscle. Any portion of the muscle may contain fibers belonging to as many as 20-50 motor units [3]. Figure 1 shows an example of an EMG signal.

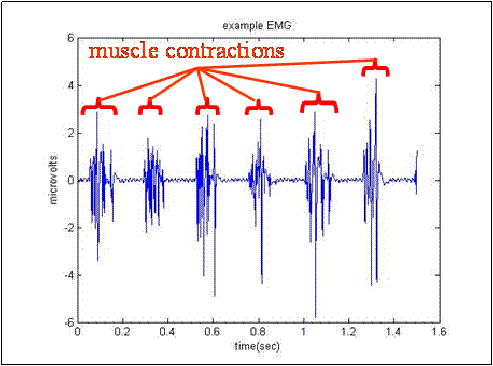


Figure 1

Currently, EMG signals can be measured by invasive methods, which use needle electrodes, and non-invasive techniques that consider surface electrodes placed on the person’s skin.[4]

Mainly, surface electromyography (sEMG) signals have been extensively used in developing prosthetic devices[5] and technology based on human–machine interaction [6]. Therefore, elements such as bracelets or armbands have been developed to obtain sEMG signals that provide relevant information in a non-invasive way. These armbands generally have several EMG sensors located radially on a flexible band, which is easy to place and can be used in several applications [7], notably the recognition of hand gestures [8]

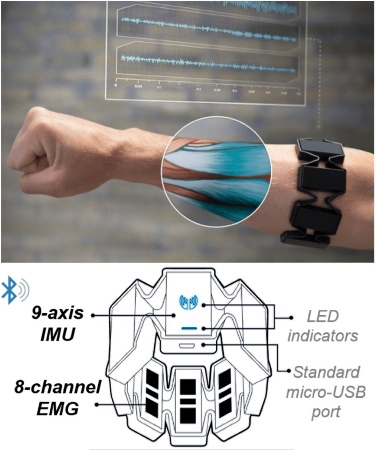
One of the most important developments of this type of device was the Myo band created by Thalmic Labs, which consists of 8 electrodes, a 9-axis inertial measurement unit, and a Bluetooth Low Energy module. Moreover, the Myo armband weighs 93 g and has a thickness of 1.15 cm [9]

Figure 2 : Myo Band by Thalmic Labs

The 8 channel sensors shown in Myo Band (Figure 2) indicates that the EMG device has the capability to record electrical activity from eight different muscle locations simultaneously. In other words, it can measure and record muscle activity from eight distinct muscles or muscle groups at the same time.

The 9 axis IMU, on the other part, is a sensor module that combines three different types of sensors to measure motion and orientation in three-dimensional space:

* Accelerometer (3 axes): Measures acceleration along three orthogonal axes (typically X, Y, and Z). It can detect changes in velocity and is often used to determine the orientation of an object relative to the Earth's gravitational field.
* Gyroscope (3 axes): Measures angular velocity or the rate of rotation around the same three axes. Gyroscopes are used to track changes in orientation or rotation of an object.
* Magnetometer (3 axes): Measures the strength and direction of the magnetic field. It is used to determine the orientation of an object relative to the Earth's magnetic field, which can be useful for compass-like applications or for understanding an object's absolute orientation in space.

Despite the various applications in which Myo Band was implemented, this armband has disadvantages, such as a sampling frequency of 200 Hz, which leads to the loss of relevant information since the dominant range of the sEMG signals is from 10 to 500 Hz [9], [10].

In 2018, Mahmoud Tavakoli et al. [11] developed a minimalist band with a sampling frequency of 1000 Hz, which used only two sensors for the recognition of four gestures (open, close, wave in, wave out). Nevertheless, the application of said bracelet is reduced due to the number of movements considered. Additionally, the optimal positioning of the sensors varies for each subject, generating relevant deviations in the obtained results. Moreover, even though the system only had two channels, the design was not sufficiently compact and did not operate wirelessly, which makes it challenging to implement in real scenarios.[9]

In 2022, Manuela Gomez-Correa and David Cruz-Ortiz published a paper titled “Low-Cost Wearable Band Sensors of Surface Electromyography for Detecting Hand Movements”, in which they described the design of a wearable electromyographic system (WES), which consists of two armbands, called WyoFlex, to acquire four sEMG signals. The WyoFlex band consists of four sEMG sensors measured by using a 12-bit ADC module (a 12 bit analogue-digital converter) of a FireBeetle ESP32-E microcontroller. The WyoFlex band considers a data transmission system through Wi-Fi to send and visualize the rebuilt signals online. Then, a maximum sampling frequency of 1600 Hz is guaranteed to avoid the loss of relevant information in the acquired sEMG signals. Furthermore, as one of the main characteristics, the proposed device has a mechanical design completely developed in three-dimensional (3D) printing, thus achieving a functional and affordable device.

## State of the job at the beginning of the semester

I have got ready the following equipment, files and programs:

* Access to Unity3d Software
* Access to MindRove Armband
* Access to Reallusion Character Creator Demo
* Mindrove SDK and Software

## Methodology

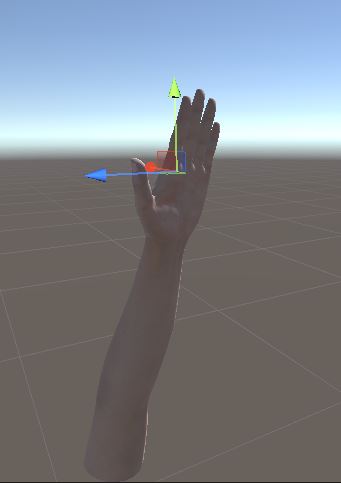
1. **Intialising the hand model in Unity:**

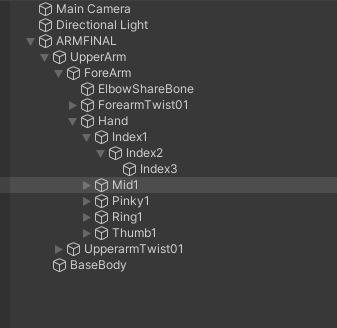
**Unity3d:**

Unity3D is a powerful and versatile game engine and development platform used for creating video games and interactive experiences. It supports both 2D and 3D graphics, making it a popular choice among developers for its ease of use and ability to deploy across multiple platforms, including Windows, macOS, Linux, iOS, Android, and various game consoles. Unity3D is known for its intuitive user interface and a comprehensive set of features that cater to both beginners and experienced developers. It includes a rich set of tools for asset creation, scripting, physics simulation, animation, and more, along with a vast marketplace of assets and plugins. Unity3D is widely used in the gaming industry, as well as in other areas like virtual reality, augmented reality, and architectural visualization.[12]

**The Arm Model Model:**

Choosing a proper and realistic arm model is a very important step that insures the clear representation of the real life hand movement in a 3d software environment; for this task, Reallusion Character Creator was used, which is a 3D character design software that allows for the creation and customization of detailed characters. It is particularly noted for its ability to customize individual body parts, like arms. The software supports the FBX format, which is key for its interoperability with other platforms like Unity3D. This feature enables users to easily export specific body parts, such as an arm model, in FBX format and integrate them into Unity projects, ensuring a seamless workflow for game and animation development. The software does not only allow for detailed customization of body parts but also includes a full rig with the exported models. This means that the arm model comes with a pre-built skeletal structure, enabling easy animation within Unity3D or other 3D software. [13]





**Figure shows the hand Model with its skeleton structure, also called “Rig”**

**The Joints Movement Script:**

Unity scripting, primarily done in C#, is essential for creating game mechanics, behaviors, and interactions in Unity projects. Scripts, attached to GameObjects, enable developers to control movements, handle user input, manage game states, and more. Unity's scripting API offers access to numerous game development functions, including physics, animations, and UI, making it a key tool for bringing interactivity and complexity to games.[12]

In real life, finger and hand movements are limited by the anatomical structure and joint mechanics. Each finger, primarily has two types of joints: the proximal interphalangeal (PIP) and distal interphalangeal (DIP) joints, along with the metacarpophalangeal (MCP) joint at the base of the fingers [14]. These joints mainly allow bending and straightening motions, which are akin to rotation around the finger's Z-axis in a 3D space.

The understanding of real-life finger and hand movements, particularly their limitations and primary movement along the Z-axis, directly influenced the design of the script attached to the fingers GameObjects, In this code, the rotation of joints around the Z-axis reflects the natural flexion and extension movements of fingers. Given that fingers primarily bend and straighten in a way that aligns with Z-axis rotation in 3D modeling, this approach in the script is aimed at replicating realistic hand and finger movements. By focusing on Z-axis rotation, the code effectively mimics the natural mechanics of finger joints, excluding the thumb which has a more complex range of motion. This decision to rotate joints around the Z-axis in the script aligns with the anatomical constraints and capabilities of real human hands, ensuring that the digit movements in the game or application are as lifelike as possible.

**RotateJointZ Code:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class RotateJointZ : MonoBehaviour

{

public Transform jointZ1;

public Transform jointZ2;

public Transform jointZ3;

public float targetZAngleJoint1 = 90f;

public float targetZAngleJoint2 = 180f;

public float targetZAngleJoint3 = 270f;

public float rotationSpeed = 10f;

private Vector3 originalAngleJointZ1;

private Vector3 originalAngleJointZ2;

private Vector3 originalAngleJointZ3;

private void Start()

{

originalAngleJointZ1 = jointZ1.localEulerAngles;

originalAngleJointZ2 = jointZ2.localEulerAngles;

originalAngleJointZ3 = jointZ3.localEulerAngles;

}

private void Update()

{

if (Input.GetKey(KeyCode.R))

{

RotateJointsMethod(targetZAngleJoint1, targetZAngleJoint2, targetZAngleJoint3);

}

else if (Input.GetKey(KeyCode.F))

{

RotateJointsMethod(originalAngleJointZ1.z, originalAngleJointZ2.z, originalAngleJointZ3.z);

}

}

private void RotateJointsMethod(float angle1, float angle2, float angle3)

{

RotateToAngle(jointZ1, angle1);

RotateToAngle(jointZ2, angle2);

RotateToAngle(jointZ3, angle3);

}

private void RotateToAngle(Transform joint, float targetAngle)

{

float angle = Mathf.MoveTowardsAngle(joint.localEulerAngles.z, targetAngle, rotationSpeed \* Time.deltaTime);

joint.localEulerAngles = new Vector3(joint.localEulerAngles.x, joint.localEulerAngles.y, angle);

}

}

**Explanation of the Script:**

**Namespace and Class Declarations:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class RotateJointZ : MonoBehaviour

This part includes necessary namespaces and declares the RotateJointZ class which inherits from MonoBehaviour, allowing it to be attached to GameObjects in Unity.

**Public Variables:**

public Transform jointZ1;

public Transform jointZ2;

public Transform jointZ3;

public float targetZAngleJoint1 = 90f;

public float targetZAngleJoint2 = 180f;

public float targetZAngleJoint3 = 270f;

public float rotationSpeed = 10f;

Transform jointZ1, jointZ2, jointZ3: References to the Transform components of the three joints to be rotated.

targetZAngleJoint1, targetZAngleJoint2, targetZAngleJoint3: Desired rotation angles for each joint.

rotationSpeed: Speed at which the joints will rotate.

**Private Variables:**

private Vector3 originalAngleJointZ1;

private Vector3 originalAngleJointZ2;

private Vector3 originalAngleJointZ3;

These variables store the original rotation angles of each joint for later use.

**Start Method:**

private void Start()

{

originalAngleJointZ1 = jointZ1.localEulerAngles;

originalAngleJointZ2 = jointZ2.localEulerAngles;

originalAngleJointZ3 = jointZ3.localEulerAngles;

}

Called when the script starts. It initializes the private variables with the current rotation angles of the joints.

**Update Method:**

private void Update()

{

if (Input.GetKey(KeyCode.R))

{

RotateJointsMethod(targetZAngleJoint1, targetZAngleJoint2, targetZAngleJoint3);

}

else if (Input.GetKey(KeyCode.F))

{

RotateJointsMethod(originalAngleJointZ1.z, originalAngleJointZ2.z, originalAngleJointZ3.z);

}

}

Executed every frame. Checks if the 'R' or 'F' key is pressed.

If 'R' is pressed, it calls RotateJointsMethod with the target angles.

If 'F' is pressed, it calls RotateJointsMethod with the original angles.

The pressing of R and F is just to visualize it before importing the streamed data from the sensor, which will control the hand movements.

**RotateJointsMethod:**

private void RotateJointsMethod(float angle1, float angle2, float angle3)

{

RotateToAngle(jointZ1, angle1);

RotateToAngle(jointZ2, angle2);

RotateToAngle(jointZ3, angle3);

}

A helper method that applies rotation to each joint using the RotateToAngle method.

**RotateToAngle**

private void RotateToAngle(Transform joint, float targetAngle)

{

float angle = Mathf.MoveTowardsAngle(joint.localEulerAngles.z, targetAngle, rotationSpeed \* Time.deltaTime);

joint.localEulerAngles = new Vector3(joint.localEulerAngles.x, joint.localEulerAngles.y, angle);

}

}

This method performs the actual rotation. It smoothly transitions the current angle of a joint to the target angle. Mathf.MoveTowardsAngle is used for a smooth rotation, and Time.deltaTime ensures the movement is consistent across different frame rates.

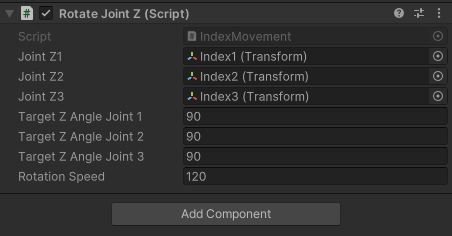


Figure Showing a sample movement, each finger has 3 joints with a rotation angle and rotation Speed

1. **Streaming data from Mindrove ArmBand to Python:**

The Mindrove Armband, is an EMG-based armband designed for detecting muscle movements. It is worn on the forearm. This device utilizes electromyography (EMG) to capture and analyze muscle activity, making it suitable for applications that require monitoring or interpreting physical movements or gestures​.

The armband captures EMG data across 8 channels, with an additional 2 channels for bias and reference, ensuring precise muscle activity measurement. All electrodes are evenly spaced to maintain consistent data quality. It operates at a sampling rate of 500Hz for accurate and timely data collection. Additionally, the armband is equipped with an integrated IMU sensor, which comprises a 3-axis gyroscope and a 3-axis accelerometer, enhancing its capability to track and analyze movement.

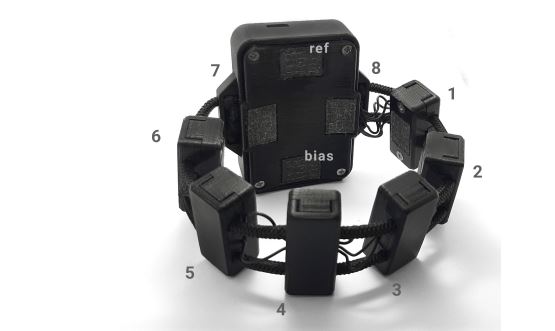
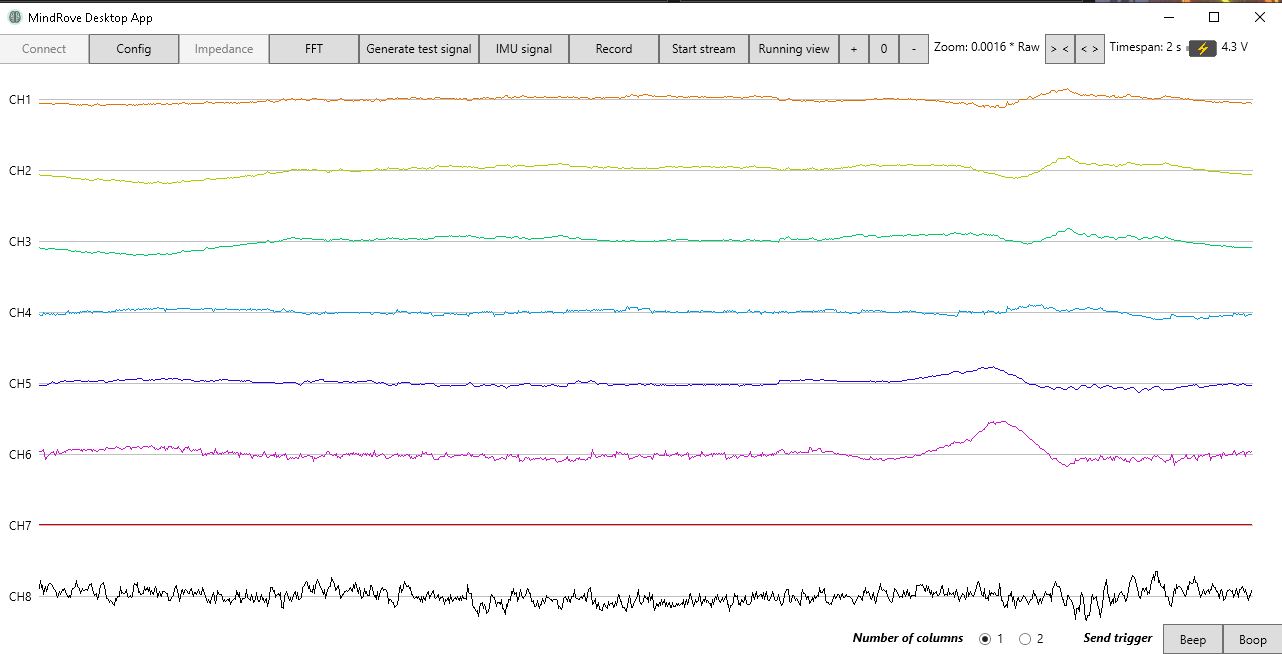


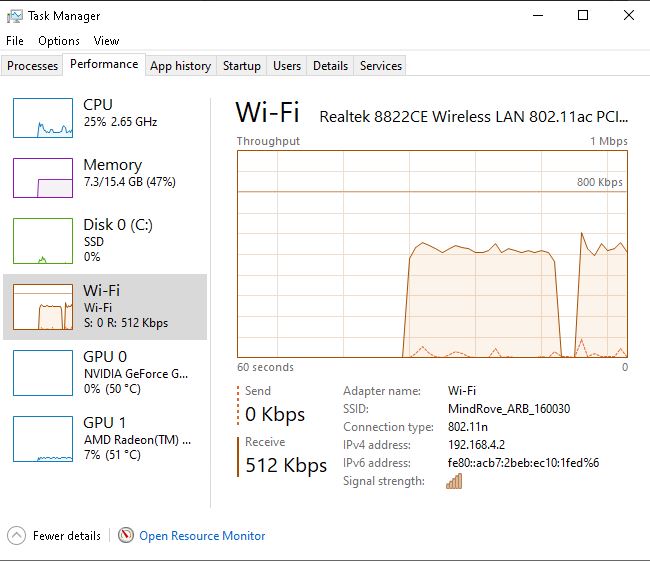
Figure showing the Mindrove armband

**2.1 Setting up the Mindrove board (Armband):**

Following the user manual, the mindrove board is connected using Wifi through a dongle, we proceed then to verify whether the board is recognized by using the MindRove Visualizer Software, then check the network process to see if there are any inconsistencies with the established connection.



Mindrove Software Showing EMG signals received.



Cut-outs in the network prevent having consistent data.

After establishing the connection, we proceed to installing the mindrove SDK in python.

This SDK provides the necessary tools and functions to interact with the MindRove ArmBand. Then we import the required libraries from the SDK, typically BoardShim, MindRoveInputParams, and BoardIds.

We can proceed to write a python code that prints the values given by the Accelerometer using a loop, this will allow to see examine the values and make them work in our application. The same applies for gyroscope data and EMG data.

from mindrove.board\_shim import BoardShim, MindRoveInputParams, BoardIds  
import time  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print\_hi('PyCharm')  
  
 # Setting up the MindRove Board  
 BoardShim.enable\_dev\_board\_logger()  
 params = MindRoveInputParams()  
 board\_id = BoardIds.MINDROVE\_WIFI\_BOARD  
 board\_shim = BoardShim(board\_id, params)  
 board\_shim.prepare\_session()  
 board\_shim.start\_stream()  
  
 # Getting accelerometer channels and sampling rate information  
 accel\_channels = BoardShim.get\_accel\_channels(board\_id)  
 sampling\_rate = BoardShim.get\_sampling\_rate(board\_id)  
  
 # Setting up the window size and number of points  
 window\_size = 2 # seconds  
 num\_points = window\_size \* sampling\_rate  
  
 # Continuous data fetching and printing  
 try:  
 while True:  
 # Fetching accelerometer data  
 data = board\_shim.get\_board\_data(num\_points)  
 accel\_data = data[accel\_channels]  
 gyro\_data = data[gyro\_channels]  
 # Unpacking the accelerometer data for each axis  
 x\_axis\_data = accel\_data[0]  
 y\_axis\_data = accel\_data[1]  
 z\_axis\_data = accel\_data[2]  
  
 # Printing the accelerometer data with labels  
 print("Accelerometer Data:")  
 print("X-axis:", x\_axis\_data)  
 print("Y-axis:", y\_axis\_data)  
 print("Z-axis:", z\_axis\_data)  
  
 # Waiting for half a second  
 time.sleep(0.5)  
 except KeyboardInterrupt:  
 print("Stream stopped by user.")  
 finally:  
 board\_shim.stop\_stream()  
 board\_shim.release\_session()

**2.2 Processing the data:**

There are three types of data:

-Acceleration Data from Accelerometer Channels

-Rotation Data from Gyroscope Channels

-EMG data from 8 EEG channels.

For the Acceleration and rotation:

After printing out the data, 2 types of approaches were taken to use the accelerometer and gyroscope data:

The first approach is to subtract the offset value from x, y and z axis then sending it to Unity.

The second approach is to use standard deviation and using those values.

A combination of two approaches was used:

x\_data\_std = np.std(accel\_data[0] - x\_ref)

y\_data\_std = np.std(accel\_data[1] - y\_ref)

z\_data\_std = np.std(accel\_data[2] - z\_ref)

The results are then sent to Unity (Next step) and different approaches are also going to be taken to move the hand position.

**EMG Data for Gesture Recognition:**

Machine learning algorithms process EMG data to distinguish between hand gestures.

Two primary states are identified:

Classification Machine learning model is one of the approaches to distinguish between three states:

State 0: Hand at Rest

State 1: Hand Closing (Clenching Fist)

State 2: Hand Opening

However opening and closing the hand give similar EMG signals, for this reason another approach could be used, which includes only two states:

State 0: Rest (No hand movement)

State 1: Hand opens or closes, depending on the initial real-world hand state.

Unity receives these states as binary values, triggering corresponding virtual hand movements.

**Accelerometer Data for Motion Control:**

Accelerometer data are applied as forces to the virtual hand model using Unity's AddForce method.

This approach simulates directional movements and enhances the naturalness of hand gestures.

**Gyroscope Data for Orientation Control:**

Gyroscope data are used to determine the orientation of the hand.

Quaternion rotations derived from gyroscope readings enable realistic rotational movements of the virtual hand in Unity.

**Sending Processed data to Unity:**

Unity will receive data from python using UDP socket, to do that, we have to:

**Set Up a UDP Socket:**

Use Python's socket library to create a UDP socket.

Send Data: Serialize and send the data (e.g., accelerometer data) to a specific IP address and port where the Unity application is listening.

import socket

# Setup UDP socket

udp\_ip = "127.0.0.1" # Replace with your Unity application's IP

udp\_port = 5005 # Replace with the port your Unity application is listening on

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

**Set Up a UDP Listener:**

Creating a script in Unity to listen for incoming UDP packets on the same IP address and port. The data is then parsed and used as needed once it is received in Unity.

try:

# Connect to the server

sock.connect((host, port))

# Prepare the data by packing it into a binary format

packed\_data = struct.pack('fff', accx\_data\_std, accy\_data\_std, accz\_data\_std)

# Send the packed data

sock.sendall(packed\_data)

# Receive response

response = sock.recv(1024).decode("utf-8")

print(response)

finally:

sock.close()

#Parsing the data

byte[] buffer = new byte[12]; // 3 floats \* 4 bytes per float

int bytesRead = stream.Read(buffer, 0, buffer.Length);

if (bytesRead == 12) {

float accx = BitConverter.ToSingle(buffer, 0);

float accy = BitConverter.ToSingle(buffer, 4);

float accz = BitConverter.ToSingle(buffer, 8);

}

The acceleration data is applied as a Force to the object, using AddForce, and the gyroscope data is applied to the rotation.

As for the EMG data, Unity will receive a value of 0 when no changes occurs (meaning that the hand did not move). When it receives a value 1, the hand opens/closes (depending on the initial state; opened or closed).

1. Conclusion:

While practical implementation is constrained, the theoretical results are promising. The machine learning model, equipped with anomaly detection, is expected to effectively differentiate hand states based on EMG data. The integration of accelerometer and gyroscope data aims to provide a seamless and responsive control over the virtual hand model. However, the challenge lies in accurately synchronizing the real and virtual hand states to avoid inverse movements.

This theoretical study underscores the potential of integrating machine learning with motion control techniques for intuitive EMG-based interfaces. The conceptual framework, despite its current limitations, lays the groundwork for future advancements in this field. As coding expertise and computational resources evolve, realizing such a system could significantly enhance user experiences in virtual environments and prosthetic control.

Bibliography:

[1] M. S. Totty and E. Wade, “Muscle Activation and Inertial Motion Data for Noninvasive Classification of Activities of Daily Living,” *IEEE Trans. Biomed. Eng.*, vol. 65, no. 5, pp. 1069–1076, May 2018, doi: 10.1109/TBME.2017.2738440.

[2] R. Meattini, S. Benatti, U. Scarcia, D. De Gregorio, L. Benini, and C. Melchiorri, “An sEMG-Based Human–Robot Interface for Robotic Hands Using Machine Learning and Synergies,” *IEEE Trans. Compon. Packag. Manuf. Technol.*, vol. 8, no. 7, pp. 1149–1158, Jul. 2018, doi: 10.1109/TCPMT.2018.2799987.

[3] G. S. Rash, “Electromyography Fundamentals”.

[4] V. Gohel and N. Mehendale, “Review on electromyography signal acquisition and processing,” *Biophys. Rev.*, vol. 12, no. 6, pp. 1361–1367, 2020.

[5] J. Tchimino, M. Markovic, J. L. Dideriksen, and S. Dosen, “The effect of calibration parameters on the control of a myoelectric hand prosthesis using EMG feedback,” *J. Neural Eng.*, vol. 18, no. 4, p. 046091, 2021.

[6] D. Xiong, D. Zhang, X. Zhao, and Y. Zhao, “Deep learning for EMG-based human-machine interaction: A review,” *IEEECAA J. Autom. Sin.*, vol. 8, no. 3, pp. 512–533, 2021.

[7] A. Phinyomark, R. N. Khushaba, and E. Scheme, “Feature extraction and selection for myoelectric control based on wearable EMG sensors,” *Sensors*, vol. 18, no. 5, p. 1615, 2018.

[8] Z. Zhang, K. Yang, J. Qian, and L. Zhang, “Real-time surface EMG pattern recognition for hand gestures based on an artificial neural network,” *Sensors*, vol. 19, no. 14, p. 3170, 2019.

[9] M. Gomez-Correa and D. Cruz-Ortiz, “Low-Cost Wearable Band Sensors of Surface Electromyography for Detecting Hand Movements,” *Sensors*, vol. 22, no. 16, p. 5931, Aug. 2022, doi: 10.3390/s22165931.

[10] S. Pancholi and A. M. Joshi, “Portable EMG Data Acquisition Module for Upper Limb Prosthesis Application,” *IEEE Sens. J.*, vol. 18, no. 8, pp. 3436–3443, Apr. 2018, doi: 10.1109/JSEN.2018.2809458.

[11] M. Tavakoli, C. Benussi, P. A. Lopes, L. B. Osorio, and A. T. de Almeida, “Robust hand gesture recognition with a double channel surface EMG wearable armband and SVM classifier,” *Biomed. Signal Process. Control*, vol. 46, pp. 121–130, 2018.

[12] “Unity (game engine),” *Wikipedia*. Nov. 25, 2023. Accessed: Dec. 06, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Unity\_(game\_engine)&oldid=1186824491

[13] “Reallusion,” *Wikipedia*. Oct. 01, 2023. Accessed: Dec. 06, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Reallusion&oldid=1178085334

[14] J. Maw, K. Y. Wong, and P. Gillespie, “Hand anatomy,” *Br. J. Hosp. Med.*, vol. 77, no. 3, pp. C34–C40, 2016.