Popa Alexia-Theodora
12230788
Master Computer Science

An assistive communication device based on electromyography (EMG)

Praktikum II Report

1. Abstract

This report endeavors to offer a detailed explanation of an electromyography experiment, delving into its practical application and potential impact on a child experiencing near-complete paralysis. It elucidates the selection of tools and assessments deemed pertinent by the team, observing the obstacles profiled while also outlining the envisioned pathways leading to a favorable and meaningful outcome.

2. Motivation

This initiative stands as a beacon of hope for a child bravely navigating one of life's most formidable challenges, offering solace and support within a meticulously crafted environment. The prospect of contributing to this project carries profound emotional weight, knowing that our efforts could truly make a difference in the lives of not just one, but countless individuals. It resonates deeply as a journey of compassion and empathy, tapping into the very essence of our humanity to harness technology for the betterment of human lives.

3. Summary on Praktikum I - What changes now

The triumph achieved with the Psychopy program, seamlessly exported and deployed in JavaScript, marks a significant milestone. Yet, our journey doesn't end there. We're poised to elevate our efforts by transitioning to a more compatible and robust library: Pygame. This transition signifies the establishment of a sturdy bridge between hardware and software, a pivotal enhancement identified during the refinement of Project P1.

The suggestion from our team to integrate Myo armbands into the gameplay experience emerged as a transformative insight. These armbands offer a unique avenue for processing data derived from muscle contractions, unlocking the potential for meaningful communication within the game environment. Imagining a scenario where the child can

respond to questions by intuitively guiding the movement of a ball, a gesture imbued with the power of expression and interaction has become reality.

Central to achieving this vision was the imperative task of connecting two Myo armbands to the Pygame script. This represented a focal point of our endeavors, embodying our unwavering commitment to realizing a seamless and immersive gaming experience for the child.

4. Tools and technologies

4.1 Myo Armbands

The Myo armband utilizes muscle sensing technology through electromyography (EMG) sensors, capturing and interpreting the electrical signals from muscle contractions. This allows for precise gesture control and interaction with various digital devices. Users can seamlessly control computers, smartphones, virtual reality (VR) devices, and other digital platforms using hand and arm gestures detected by the Myo armband, providing a hands-free and immersive experience. Connectivity is facilitated via Bluetooth, enabling wireless communication for easy integration with different platforms and applications.

Designed for compatibility with Windows, macOS, iOS, and Android, the Myo armband offers versatility across a range of devices and operating systems. Its accompanying software development kit (SDK) empowers developers to create custom applications, fostering innovation in gaming, healthcare, and virtual reality. Beyond gaming and entertainment, Myo armbands find utility in diverse industries such as healthcare, virtual reality, robotics, and human-computer interaction, offering a novel approach to technology interaction. With its ergonomic design for comfortable forearm wear and lightweight build, the armband is suitable for extended use.

At the core of the Myo armband are electromyography (EMG) sensors strategically positioned on its inner surface. These sensors make direct contact with the user's skin, detecting the electrical signals produced by muscle contractions. This technology forms the basis for the armband's ability to interpret various hand and arm movements. Complementing the EMG sensors are Inertial Measurement Unit (IMU) sensors, which typically include accelerometers and gyroscopes. These sensors capture data related to the orientation and movement of the user's arm, enhancing the accuracy of gesture recognition.

A vital component of the Myo armband is its microcontroller, tasked with processing the data collected by the EMG and IMU sensors. Through sophisticated algorithms, the

microcontroller interprets muscle activity and arm movements in real-time, facilitating seamless gesture recognition and interaction with digital devices. Enabling wireless communication is a Bluetooth module integrated into the Myo armband. This module establishes connections with external devices such as computers, smartphones, or VR headsets, allowing for the transmission of data and reception of commands within its operational range.

Powering the Myo armband is an internal rechargeable battery. This battery provides the necessary energy to sustain the operation of the armband's sensors, microcontroller, and Bluetooth module. The longevity of the battery life may vary depending on usage patterns and settings. The Myo armband's components are encased within an ergonomic enclosure designed for comfortable wear on the user's forearm. This enclosure boasts a sleek and lightweight design, ensuring a secure fit while minimizing discomfort during prolonged usage.

4.2 PyGame

Pygame stands out as a notable Python library tailored for game development, offering a robust set of modules that streamline the creation of 2D games. In this report, we delve into its key features, advantages, and community support, highlighting its suitability for both novice and experienced developers. Pygame is a meticulously crafted ensemble of Python modules explicitly crafted for crafting captivating video games. Its repertoire encompasses

a spectrum of tools and functionalities catering to graphics, sound, input devices, and various other facets pivotal to game development.

Renowned for its user-friendly nature, Pygame prides itself on its simplicity and minimalistic design, rendering it approachable even for novices. Leveraging the Simple DirectMedia Layer (SDL) library as its foundation, Pygame presents a straightforward interface, expediting the game development process. While its forte lies in 2D game development, Pygame exhibits unparalleled prowess in rendering intricate graphics. Boasting a plethora of functions for shape drawing, image manipulation, and sprite management, Pygame offers an extensive toolkit for crafting visually appealing games.

One of Pygame's standout attributes is its cross-platform compatibility, seamlessly traversing various operating systems such as Windows, macOS, and Linux. This cross-compatibility empowers developers to create games that transcend platform limitations, fostering a wider audience reach. Pygame thrives within a vibrant community, bolstered by comprehensive documentation. This collaborative ecosystem facilitates knowledge exchange, providing developers with ample resources, support, and exemplars to expedite their game development journey.

Embracing an event-driven programming paradigm, Pygame enables developers to orchestrate interactive gameplay by responding to diverse events such as keyboard inputs, mouse clicks, and timers. This approach enhances the dynamism and engagement levels of the gaming experience. Built upon an open-source framework, Pygame embodies the principles of accessibility and flexibility. Its transparent source code empowers developers to tailor the library to suit their bespoke requirements, fostering a culture of innovation and customization.

While Pygame furnishes a robust foundation for game development, its extensibility shines through in its compatibility with supplementary libraries and tools. This modularity allows developers to augment Pygame's functionalities, further enriching the gaming experience with bespoke features and enhancements.

In conclusion, Pygame emerges as a formidable ally for game developers, offering an array of features, cross-platform compatibility, and a supportive community. Its simplicity, extensibility, and focus on 2D graphics make it an invaluable asset for crafting immersive gaming experiences across diverse platforms.

4.3 LSL Library

LSL, or Lab Streaming Layer, serves as a pivotal communication protocol utilized for the real-time exchange of time-series data in research environments, notably within the realms of neuroscience and experimental psychology. This report delves into its functionalities,

applicability, and advantages in facilitating seamless data integration and synchronization across diverse software and hardware components. LSL emerges as a sophisticated communication protocol, facilitating the seamless transmission of time-series data in real-time among disparate software and hardware entities. Developed with the primary aim of harmonizing the integration of various tools and devices prevalent in experimental setups, particularly within neuroscience research, LSL plays a pivotal role in enabling cohesive data exchange and analysis.

Central to LSL's functionality is its capability to transmit a wide array of time-series data types, encompassing signals sourced from EEG (electroencephalography), ECG (electrocardiography), motion capture systems, eye trackers, and other pertinent data streams encountered in experimental contexts. This flexibility renders LSL indispensable for researchers seeking to amalgamate and analyze multifaceted data sources concurrently. A key hallmark of LSL is its innate compatibility across diverse operating systems, underscoring its versatility and adaptability within varied experimental setups. This cross-platform compatibility ensures seamless operation across different environments, fostering interoperability and ease of integration with an extensive spectrum of software and hardware configurations.

To harness the functionalities offered by LSL, developers can leverage an assortment of LSL libraries available in multiple programming languages, including Python, C++, and others. These libraries furnish comprehensive toolsets, empowering users to seamlessly implement LSL functionality within their applications, from stream creation to data reception and analysis. Embodying the ethos of open-source development, LSL stands as a collaborative endeavor, with its source code freely accessible to the research community. This open-access paradigm not only encourages contributions from researchers and developers but also fosters innovation and continuous enhancement of the protocol's capabilities.

LSL finds widespread adoption in experimental settings, where the synchronization and collection of data from disparate sources are imperative. By providing robust timestamping mechanisms, LSL ensures the accurate alignment of diverse data streams, facilitating coherent analysis and interpretation of research findings. A testament to its user-centric design, LSL is accompanied by comprehensive documentation and an active community dedicated to assisting users in harnessing the protocol's capabilities effectively. The official LSL website serves as a repository of invaluable resources, including documentation, tutorials, and forums, fostering knowledge exchange and collaboration among users.

In essence, LSL emerges as a dynamic and versatile protocol, facilitating the seamless exchange and synchronization of time-series data in real-time within research

environments. Its cross-platform compatibility, open-source nature, and robust documentation make it an indispensable asset for researchers in neuroscience and experimental psychology, enabling them to unravel insights from complex data streams with precision and efficiency.

4.4 PyoMyo library

The PyoMyo library is the first library used in order to make possible the connection between the armbands and the game. It prints surface electromyography (sEMG) readings at a frequency of 200Hz directly from the Myo's analog-to-digital converter (ADC) in the raw EMG mode. Each EMG reading falls within the range of -128 to 127, representing the most unprocessed data the Myo can provide. However, it typically requires additional processing to be meaningful. This file also encompasses the implementation of the Myo driver, utilizing Serial commands transmitted over Bluetooth to communicate with the Myo device.

Another important component from this library is the classifier, i.e. Classifier.py script. This module implements a real-time classifier using the k-nearest neighbors algorithm. Users can assign labels to incoming data by pressing a number key from 0 to 9, representing different classes. Pressing 'e' allows users to clear all gathered data. Upon creating two distinct classes, new data is automatically classified. Labeled data is stored as a numpy array in the data directory.

5. Myo Armband:

5.1. Myo Armband: Mono Connection

As previously outlined, establishing a connection with a single armband is facilitated seamlessly through the utilization of the Pyomyo library alone. The repository offers a wealth of exemplary resources, serving as invaluable guides for users to commence their exploration of the library's functionalities and data processing mechanisms. By leveraging these comprehensive examples, users can gain insightful insights into the workings of the library and its adeptness in handling and processing data streams. This approach not only facilitates a smoother initiation into the library's capabilities but also fosters a deeper understanding of its intricacies, empowering users to harness its full potential in their respective endeavors.

5.3. Myo Armband: Dual Connection -MioConnect Repository

The codebase is meticulously documented, ensuring clarity and ease of comprehension. It encompasses several key functionalities designed to facilitate seamless communication with Myo armbands and handle data streams effectively. The code begins by initiating a disconnect message to ensure the termination of any lingering connections from previous sessions. This proactive approach helps maintain the integrity of subsequent connections and data exchanges.

To manage the complexities of Bluetooth communication, the code implements dedicated handlers for each anticipated event. This meticulous event management ensures that the application responds appropriately to various Bluetooth signals, enhancing reliability and robustness. A crucial aspect of the code involves establishing connections with Myo armbands. Through a series of connection procedures, including device discovery and direct connection establishment, the code ensures seamless integration with the Myo devices, enabling smooth data transmission.

To maintain continuous operation and responsiveness, the code monitors critical events and awaits responses, effectively synchronizing with the Myo devices and ensuring real-time data exchange. Additionally, the code disables sleep functionality to prevent interruptions and maintain consistent data streaming. This proactive measure enhances the reliability and stability of the application, particularly during extended usage periods.

Configuring and initiating data streaming is another essential aspect of the code. By adhering to parameters specified in the configuration file, the code effectively sets up and begins streaming EMG, IMU, and Classifier data, catering to specific application requirements. The code subscribes to events related to data streams, ensuring that relevant data is captured and processed efficiently. This event-driven approach enhances the responsiveness and adaptability of the application, enabling dynamic data processing and analysis.

Central to the code's functionality is its message processing mechanism. Through the set_handlers method, the code defines how incoming messages are processed, with particular emphasis on critical functions such as handle_imu and handle_emg, where the Open Sound Control (OSC) protocol is implemented. An infinite loop ensures continuous event monitoring, allowing the application to respond promptly to incoming data and events. This iterative process facilitates real-time interaction and data processing, contributing to the overall responsiveness and effectiveness of the application.

In the event of a keyboard interrupt (Ctrl+C), the code gracefully handles termination by interrupting the loop and triggering disconnect messages. This proactive approach ensures a seamless exit process, preventing data loss or connection issues. Finally, the code incorporates a reconnection routine to address instances where a Myo device disconnects unexpectedly. By leveraging provided configuration parameters, the code initiates a reconnection process, seamlessly restoring connectivity and data exchange.

Overall, the code embodies a robust and comprehensive approach to managing Myo armband communication and data streaming, ensuring reliability, responsiveness, and adaptability in diverse application scenarios

5.4. Myo Armband: Dual Connection - Obstacles

Establishing a connection with a singular armband proves straightforward, facilitated by the utilization of the Pyomyo library in isolation. However, complications arise when endeavoring to extend this connectivity to multiple processes within the Pygame framework. Regrettably, Pygame demonstrates inadequacy in facilitating such multi-process integration, as attempts to employ multiprocessing functionality result in an error pertaining to the inability to pickle 'pygame.surface.Surface' objects. This limitation hampers the seamless extension of connectivity beyond a singular armband within the Pygame environment.

The completion of my research endeavor was notably challenging, primarily attributed to the dearth of comprehensive documentation surrounding the intricacies of working with armbands. Additionally, the absence of precise points of contact further compounded the hurdles encountered during the research process. However, the collaborative efforts of my team proved invaluable in navigating these obstacles. Through diligent exploration, we succeeded in identifying a pertinent resource: the MioConnect repository highlighted above. Noteworthy for its alignment with the Pyomyo library, this repository offers tailored adjustments that significantly enhanced the efficacy of my research efforts. Consequently, leveraging the insights and modifications encapsulated within the MioConnect repository proved instrumental in overcoming the challenges posed by the limited documentation and accessibility barriers inherent in the field of armband technology.

The execution of my research encountered significant challenges primarily attributable to the lack of comprehensive documentation available for working with armbands. This

absence of well-documented resources hindered the progress of my work, making it inherently difficult to navigate the intricacies of armband integration.

Compounding the challenge was the difficulty in locating precise points of contact for acquiring relevant information or assistance in overcoming hurdles encountered during the research process. The absence of a readily available support network or direct communication channels further complicated the resolution of issues.

Fortunately, the collaborative efforts of my team proved invaluable in mitigating these challenges. Through extensive exploration and collaborative problem-solving, we identified and leveraged a pertinent Git repository, namely the MioConnect repository. This repository, tailored specifically for armband integration, significantly enhanced the efficiency and efficacy of my work.

It is noteworthy that the MioConnect repository is intricately linked with the Pyomyo library, featuring bespoke adjustments that address specific requirements and challenges encountered in the research context. This tailored repository, with its nuanced modifications, played a pivotal role in improving the overall workflow and outcomes of my research endeavors.

Data collection and observations entailed a comprehensive exploration of the CSV data obtained from the Myo bands. Initially, our focus centered on discerning whether the streams contained raw or processed data—an endeavor that posed initial challenges. To address this, we employed the Discrete Time Fourier Transform (DTFT) technique to meticulously assess data accuracy. This analytical approach unveiled a significant revelation: the data retrieved from the Myo bands indeed comprised raw information.

This discovery heralded promising implications, as it positioned us to process and analyze the data with greater flexibility and precision. By possessing access to unaltered data streams, we could tailor our analytical methodologies to suit specific research objectives, thereby streamlining the process of identifying thresholds and extracting meaningful insights from the data. Thus, the recognition of raw data within the CSV files represented a pivotal milestone, propelling our research endeavors forward with newfound clarity and efficiency.

An additional challenge arose when encountering delays in the movement of the ball within the game environment. To address this issue, I implemented a strategy involving the utilization of two queues for data transmission. While this approach initially seemed

promising, it quickly became apparent that accessing elements within each queue posed significant challenges due to the rapid pace at which data was being sent.

Subsequently, I conducted a thorough assessment to pinpoint the source of the delays, focusing on potential issues originating from each armband. Interestingly, upon connecting to only one armband and analyzing its data stream, no issues were detected. However, integrating data from both armbands simultaneously proved to be problematic, resulting in substantial disruptions.

In pursuit of a viable solution, I devised a novel approach centered around consolidating data transmission into a single queue. Leveraging a dictionary structure, wherein the connection number served as the key and the corresponding data from each channel constituted the value, enabled seamless integration of data from both armbands into a coherent format. This streamlined approach facilitated easier access to the data during subsequent processing stages, as information regarding the origin of each data stream (i.e., left or right armband) was preserved within the dictionary. Consequently, this refined data management strategy laid the foundation for achieving the desired outcome of accurately moving the ball based on predefined thresholds, effectively mitigating delays and optimizing gameplay experience. Some of the snippets code that are worth sharing:

```
data_new = []
builder = udp_client.OscMessageBuilder("/myo/emg")
builder.add_arg(str(conn), 's')
for i in struct.unpack('<8b ', data):
    builder.add_arg(i / 127, 'f')  # Normalize
    data_new.append(i)
if conn == 0:
    dict0 = {str(conn): data_new}
    self.myo_data0.put(dict0)
if conn == 1:
    dict1 = {str(conn): data_new}
    self.myo_data0.put(dict1)


self.osc.send(builder.build())
```

```
myo_driver.receive()


while not (myo_driver.data_handler.myo_data0.empty()):
    data_both_samples = myo_driver.data_handler.myo_data0.get()

    emg1 = []
    emg2 = []
    if(data_both_samples.get("1")):
        emg2 = list(data_both_samples.get("1"))
    if(data_both_samples.get("0")):
        emg1 = list(data_both_samples.get("0"))
    plot(scr, [e / 500. for e in emg1], [e1 / 500. for e1 in emg2])
    print("left: {}, right {}".format(emg1, emg2))

    if emg1 != []:
        outlet_emg1.push_sample(emg1)
    if emg2 != []:
        outlet_emg2.push_sample(emg2)
```

```
def handle_emg(self, payload):
    """
    Handle EMG data.
    :param payload: emg data as two samples in a single pack.
    """
    myo_data0 = []
    myo_data1 = []
    if self.printEmg:
        print("EMG", payload['connection'], payload['atthandle'], payload['value'])

    # Send both samples
    self._send_single_emg(payload['connection'], payload['value'][0:8])
    self._send_single_emg(payload['connection'], payload['value'][8:16])
```

5.5. LSL usage motivation

LSL (Lab Streaming Layer) and its inherent advantages present a superior communication framework compared to previous methodologies, particularly notable in the context of Project P1. In conjunction with the MioConnect tools, LSL serves as a pivotal conduit facilitating seamless communication between game software and hardware components. Its utilization is paramount for the accurate processing of electromyography (EMG) data and subsequent transmission to Pygame. Leveraging LSL streams, the processed data enables dynamic interactions within the game environment, such as controlling the movement of a ball. Thus, the integration of LSL not only enhances the efficiency and accuracy of data processing but also empowers real-time interactions within the game software, contributing to a richer and more immersive user experience.

6. Future improvements:

The game holds immense potential, particularly when integrated with the Raspberry Pi platform. This fusion opens up a plethora of possibilities for enhanced functionality and accessibility. To streamline the user experience, initiating the game is made effortless with the inclusion of an executable icon directly on the Raspberry Pi interface.

Upon launching the game, users are presented with a choice between two distinct files: the training module and the actual gameplay environment, now redesigned to promote free communication. This reimagined format encourages a more dynamic and engaging interaction between the player and the game.

As the game commences, users are prompted to adjust various parameters such as threshold and distance, ensuring optimal customization tailored to individual preferences and gameplay requirements. What sets this adaptation apart is its dynamic nature, allowing parameters to be fine-tuned in real-time as the game progresses, ensuring a seamless and adaptive experience.

A comprehensive log file meticulously records all interactions and gameplay data, providing invaluable insights into user performance and progression. Additionally, to facilitate further analysis and research, EMG data is systematically stored in a designated repository, enabling thorough examination and interpretation.

Incorporating a simple yet effective feedback mechanism, users are presented with emotive responses in the form of smiley and sad faces, enabling intuitive feedback on goal achievements and responses to in-game questions. This emotive feedback adds a layer of depth to the gaming experience, fostering a more immersive and engaging atmosphere.

Moreover, the inclusion of amplitude-based thresholds for EMG signals enhances gameplay dynamics, allowing users to modulate their interactions based on muscle activity intensity. This nuanced approach not only promotes finer motor control but also adds a layer of challenge and immersion to the gameplay experience.