Bachelor's thesis

Information and Communications Technology

2020

Minh Hoang Nguyen

INTERNET-OF-THINGS APPLICATIONS WITH HAND MOTION FOR REMOTE CONTROL

- A case study on Home automation and Robotic arm

TURKU AMK

TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2020 | 40

Minh Hoang Nguyen

INTERNET-OF-THINGS APPLICATIONS WITH HAND MOTION FOR REMOTE CONTROL

- A case study on Home automation and Robotic arm

This thesis aimed to introduce two possible implementations with Myo armband, which contains surface electromyography (sEMG) electrodes and a nine-axis Inertial Measurement Unit (IMU). sEMG is the reading of electrical signals generated in forearm muscles contractions, while IMU measures the angles of movement, thus when used together, they can predict the gesture. The proposed projects, namely, Home Assistant control and MeArm robot control, aimed to provide reusable and simple applications for future research about Myo gesture-recognition in Assistive technology. With regards to Myo's feature extraction, pre-built Python libraries were used for handling the Bluetooth communication with Myo band and performing handler on the received raw data including sEMG, poses, and IMU readings. By utilizing the MQTT (Message Queue Telemetry Transport) protocol, the communication between Myo and external microcontrollers, including an ESP-12E (also known as Node MCU) and an Arduino UNO, can be established.

The purpose of the Home Assistant project was to develop a sandbox application to control smart devices. In the Home Assistant implementation, a desk lamp was calibrated and controlled by Node MCU. After setting up the MQTT broker server on Intel gateway, it was possible to subscribe and publish messages to control the remote ESP. Hence, the result was promising, which has proved that this application could be scaled up and control other smart devices such as Phillips Hue light bulb, smart air-conditioner, or smart TV.

With a similar technique, the implementation with the MeArm robot was successful in demonstrating a simple gesture controlled robotic application. The combination of pose and arm movements were used to compose the set of command protocol with seven gestures. However, there were drawbacks to the current design. Firstly, connectivity with Myo Bluetooth dongle can sometimes be unstable. Secondly, Myo built-in gesture-recognition does not always make correct predictions. Lastly, only two readings from Myo's IMU (inertial measurement unit) were used in the implementation. Nevertheless, performing data processing and pattern recognition for different pose categorization would be possible for further development in the field of machine learning.

KEYWORDS:

EMG, IMU, gesture recognition, Myo armband, BLE, Home Assistant, MQTT

CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ATT	Attribute Protocol
BLE	Bluetooth Low Energy
EMG	Electromyography
GAP	Generic Access Profile
GATT	Generic Attribute and Profile
HA	Home Assistant
HCI	Human-Computer Interface
IMU	Inertial Measurement Unit
L2CAP	Logical Link Control and Adaptation Protocol
LED	Light-Emitting Diode
MQTT	Message Queue Telemetry Transport
M2M	Machine-to-Machine
OS	Operating System
PWM	Pulse Width Modulation
ROS	Robot Operating System
STX	S-Band Transmitter
SRX	S-Band Receiver
TCP/IP	Transmission Control Protocol/Internet Protocol
UUID	Universally Unique Identifier
VM	Virtual Machine
Wi-Fi	Wireless Fidelity
sEMG	surface Electromyograph

1 INTRODUCTION

During the past decade, the number of studies in home automation technology, as well as in healthcare technology, has seen a drastic increase. Most of these studies research technologies support in-door appliances such as lighting control systems, automated HVAC (heater, ventilator, and air-conditioner). Regarding assistive technology, researchers focus on the use of the robotic control system that utilizes the nervous-based sensors, such as capturing muscular activities of our arms with electromyography (EMG) to recognize the hand's gesture.[1] Among those technologies, Myo armband is one of the most renowned implementations that can be used to perform such task.

The Thalmic Labs-designed Myo armband device that can read surface Electromyography (sEMG) signals from the wearer’s arm allows control over other Bluetooth enabled devices using the user’s hand gesture. It features the advantage of not relying on any external sensors (such as motion-capture radars); on the other hand, EMG and Inertial Measurement Unit (IMU) sensors are embedded in the armband itself to recognize the user’s hands gesture and to act accordingly. [1]

Several voice-assistant models were developed, which support physically disabled people and elderly persons with controlling their home appliances and monitoring the condition in-door [2]–[4]. However, these models required voice assistant integration in their design; thus, they need to be controlled with voice access. In case that the user has speech difficulties or hearing-impaired, these implementations cannot sufficiently operate. Therefore, it is worth researching and developing solutions in human-activity-recognition in technologies that do not require voice-over-controls.

In this thesis work, experiments were performed with an ESP8266EX (more specifically ESP-12E model) microcontroller, a compact system-on-chip (SoC) that integrates with a 32-bit Tensilica processor, standard peripherals interfaces, antenna switches, filters, RF balun, low noise receive amplifier, and power management modules. Its processor features extra-low power consumption while the Wi-Fi stack is integrated and consumes about 80% of the processing power.[5] At the same time, the Home Assistant application toolset was installed to the gateway device to monitor and control the lighting with Myo armband via ESP-12E. However, the configuration of the aforementioned project was only aimed to achieve the minimum working product and serve as a template for future development. In the second project, an Arduino-based-robotic-arm controlled by pose recognition feature of Myo was developed in order to demonstrate a possible novel use case for industrial application. The main objective of those implementations was to propose a wearable and customizable assistive solution for the vocally disabled, or elderly or special groups. Arduino UNO was used in this thesis project in order to cope with the ESP-12E disadvantage while handling Wi-Fi-related works, which will be mentioned in the Discussion part.

In the following chapters, the theoretical background about sEMG and technical architecture of this thesis work will be presented in more detail.

2 THEORY AND METHODS

This chapter presents the related studies about human-machine interaction (HCI) that will be presented and compares the gesture-recognition methodologies. This section also describes a general picture of this thesis work's implementations focusing on utilizing Myo armband features.

2.1 Gesture recognition

Human-computer interaction based on gesture recognition has various application scenarios, for example, playing games without a keyboard or mouse, controlling robots in remote and hazardous environments. Gesture recognition has become an essential area for development in Human-computer interaction (HCI). Specifically, this technology has heavily relied on visual-based methods (body movements, facial expression recognition), audio-based methods (auditory emotion recognition, speaker recognition, and speech recognition), or device-based (wearable sensors). However, these types of systems suffer from many drawbacks such as light sensitivity, varying distance, hand motion modeling complexity, and hand's position. [6] On the other hand, hand-gesture recognition systems based on sensor node detection, depend only on muscle contractions [7], which is more reliable and efficient. Thus, gesture-based control is considered to be a preferable human interaction classification method.

2.2 EMG and Myo armband

Electromyography (EMG) records the electrical activity in our arm from contracting muscles. Changes in amplitude of these signals can be used to detect motion, while advanced processing methods analyze underlying patterns, which can be used to detect hand gestures.

The Myo armband, developed by Thalmic Labs [8], includes eight cubbies, each of which has a SEMG electrode, and the central one has a nine-axis inertial measurement unit (IMU) to detect the arm's movements [9]. The IMU comprises a three-axis gyroscope, a three-axis accelerometer, and a three-axis magnetometer. In order to synchronize with the band [10], it is recommended that we wear the band on the thickest part of our arm

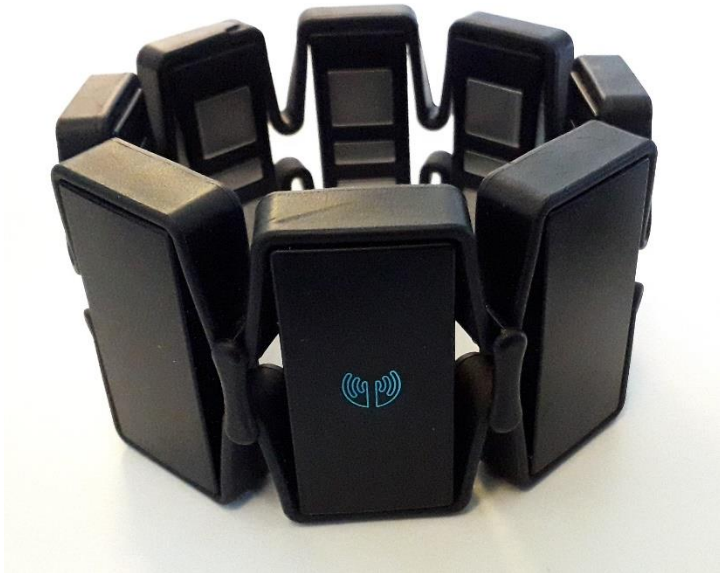


Figure 1. Myo armband

(practically just below the elbow) then perform the "sync" gesture [11]. Figure1 showcases the Myo armband, with two sizing-clips already attached. For simplicity, Myo will be used to refer to the armband.

Initially, the armband needs to be set up via Myo Connect using a supported operating system, which helps the Myo armband, to initiate communication with the Bluetooth Dongle. Additionally, Myo runs with an internal lithium-ion rechargeable battery. If the horizontal LED near the Myo's Logo emits orange-color light, it means we need to charge the armband through its micro USB port, until it changes to green color (fully charged).

2.3 Computing hardware and software used

Python was the selected programming language in the current development environment because of its popularity and versatility among other platforms. Although Android, Java, C#, and NodeJS have well-supported libraries and ready-made projects, they are not as preferable in terms of compatibility with respect to the Ubuntu environment.

According to the technical properties of the Myo Bluetooth adapter, also known as Bluegiga, its transfer rate is 9600 bit/sec (bps). The armband's collected data is transmitted via Bluetooth LE (Low Energy) to the computer via Myo’s Bluegiga, which is well-compatible with USB 3.0 port of Aaeon Intel-based gateway computer [12]. Regarding the Operating System of the gateway, Ubuntu Xenial was pre-installed and intended for testing with robotic platforms, such as ROS (Robotic Operating System). In this thesis work, however, data collection and processing were carried out with pre-built Python libraries. Thanks to the work of Dzhu [13], author of the Myo Bluetooth library written in Python, and the work of Sam Pfeiffer et al. [14], who have introduced ros_myo, an implementation with Myo's gesture recognition was developed with ROS. Those developers managed to send teleoperation commands to Turtle robot, an inexpensive robot that was meant for ROS learners to simulate basic movements with the developed programming language[15]. Together, those previous works have provided a strong foundation for any starter project with Myo's communication and feature extraction in the Python environment.

With respect to the stability of the Bluetooth connection, Dzhu's Python library, called "myo-raw", can only last up to 15 minutes of continuous operation. Fernando's work, however, improved the design by reimplementing the armband manager user interface of Myo Connect in the latest version of PyoConnect, which is a Linux alternative of Myo Connect on Windows and MacOS.[16] As a result, connection with the band can be re-established each time it was lost (indicated by the horizontal LED near to Myo's logo), which somewhat improved the design for user-friendliness.

External hardware units that were chosen for the experiments in the two projects are Arduino-based microcontrollers, namely, ESP-12E, an energy-efficient microcontroller of ESP8266EX module family [17] and Arduino UNO, a well-known open-sourced development board within the Arduino family that is inexpensive and useful in various scenarios [18]. In particular, ESP-12E, also known as Node MCU v1.0, includes a Wi-Fi 2.4Ghz module, which supports WPA/WPA2 and integrates with TCP/IP protocol stack. This feature is utilized to perform M2M communication within the same local network via MQTT (Message Queue Telemetry Transport) protocol. Hence, ESP-12E plays a key role in the designed system, which communicates with the Myo armband via the gateway. The secondary component that was included in the setup of the robotic arm project, Arduino UNO, on the other hand, supports ESP Node MCU to control the robot's servos.

2.4 Bluetooth protocol and features extraction with Myo

Thalmic Labs developed Myo utilizes Bluetooth Low Energy (BLE) communication protocol to send data packets to the computer. Thus, a thorough investigation with insights is needed in order to understand how this technology is implemented in Myo.

According to Townsend et al. [19], a BLE device is comprised of 3 main parts, namely, host, controller, and application, where each part is divided into several layers with various functionalities. The host part stays on the upper layer while the controller part is on the lower layer of the Bluetooth protocol stack. Application interfaces with Bluetooth protocol stack to exhibit a particular use case.[20]

The controller includes Host Controller Interface on the controller side, Link layer, and Physical layer.[19] In the Physical layer, the analog signals are modulated and demodulated to transform them into binary data. The radio of BLE uses the 2.4 GHz ISM (Industrial, Scientific, and Medical) band to communicate and divides this band into 40 channels from 2.4000GHz to 2.4835GHz, where the last three channels (37, 38, and 39) are used as advertising channels to set up connections and send broadcast data [19]. The Link layer directly interfaces with the Physical layer and is implemented as a combination of custom hardware and software. This layer manages the radio state, which defines how the hosting devices connect to each other. Typically, smartphone devices tend to act as masters, while smaller, simpler, and memory-constrained devices such as sensors node adopt the slave role. According to the definition of Bluetooth protocol [19], Link layer includes the following roles:

- Advertiser: a device that sends advertising packets.
- Scanner: a device that scans for advertising packets.
- Master: a device that initiates a connection and manages it later.
- Slave: a device that accepts connection requests and follows the master's timing.

These roles can be logically grouped into two pairs: advertiser and scanner (when not in an active connection); master and slave (when in a connection). [19]

BLE has only one packet format and two types of packets (advertising and data packets). Advertising packets serve two purposes: broadcasting data for applications that do not need overhead of a full connection establishment, discovering slaves, and connecting to them. Such packets are broadcast blindly over the air by the advertiser without knowing the presence of any scanning device. These packets are sent at a fixed rate defined by the advertising interval. The shorter the interval, the higher probability that those packets are received by a scanner. However, the higher number of packets transmitted the higher power consumption by the BLE module. [19]

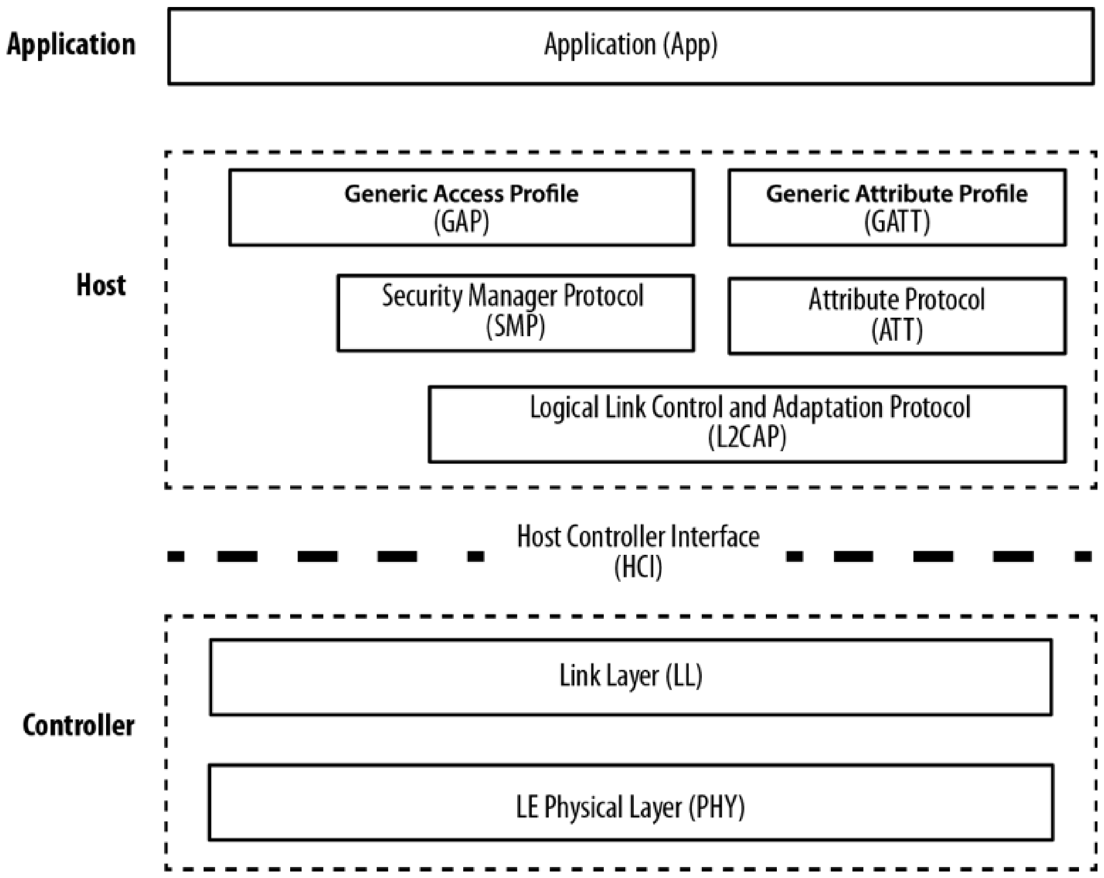

Application
Application (App)
Host
Generic Access Profile (GAP)
Generic Attribute Profile (GATT)
Security Manager Protocol (SMP)
Attribute Protocol (ATT)
Logical Link Control and Adaptation Protocol (L2CAP)
Host Controller Interface (HCI)
Controller
Link Layer (LL)
LE Physical Layer (PHY)

Figure 2. Bluetooth protocol stack

The Host Controller Interface layer is defined by Bluetooth specification as a set of commands and events that allows interactions between host and the controller, along with a data packet format and a set of rules for flow control and other procedure. Since this layer is constrained by the real-time requirements, and contact with the PHY layer, it is reasonable to draw a line at this level, as shown in Figure 2. Thus, it separates between the host and the controller. Recently, semiconductor technology has become inexpensive enough to allow a single chip to incorporate the complete host, controller, and application in a single package (a System-on-Chip). Therefore, it is common in many embedded device applications to implement the sensor using a single chip that runs all three layers concurrently on a low-power CPU. [19]

Regarding the Host part of the BLE protocol, it includes the following layers: GAP (Generic Access Profile), GATT (Generic Attribute Profile), Attribute Protocol (ATT),

Logical Link Control and Adaptation Protocol (L2CAP), Security Manager (SM), and Host Controller Interface on the host side. [19] The L2CAP layer provides two functionalities.

On the transmission path, it serves as the protocol multiplexer that receives protocols from the upper layers and encapsulates them into the standard BLE packet format. Specifically, it performs fragmentation and recombination with the sending packets from the upper layers, breaking them into 27-byte chunks, which is the maximum payload size of BLE packets on the transmitting side. On the reception path, it receives multiple fragmented packets and recombines them into a single large packet, which is then sent to the appropriate entity in the upper layers of the host. For BLE, the L2CAP layer responsible for routing two main protocols: ATT protocol and Security Manager Protocol (SMP).[19]

The ATT protocol is a simple client/server stateless protocol based on attributes presented by a device. In terms of master-slave sequencing, the protocol is stringent. If a request is still pending, no further requests can be sent until the response is received and processed. This applies to both directions independently in case where two peers are acting both as a client and server. Each server contains a 16-bit-attribute handle, a universally unique identifier (UUID), a set of permissions, and a value. UUID specifies the type and nature of the data contained in the value. When a client device wants to read or write attribute values from or to a server, it issues a read or writes a request to the server with the handle. The server will respond with the attribute and an acknowledgment. In the case of a read operation, it is up to the client to parse the value and understand the data type based on the UUID of the attribute. On the other hand, during a write operation, the client is expected to provide data that is consistent with the attribute type, and the server is free to reject the operation if that is not the case.[19]

Regarding the Security Manager protocol, it serves as a series of security algorithms designed to provide the Bluetooth protocol stack with the ability to generate and exchange security keys. This enables the peers to communicate securely over an encrypted link, to trust the identity of the remote device, and finally, to hide the public Bluetooth Address if required to avoid malicious peers tracking a particular device. Specifically, SMP describes two roles in the initialization procedures, namely, initiator, which corresponds to the Link Layer master (thus, the GAP central) and responder, which corresponds to the Link Layer slave (thus, the GAP peripheral). Although it is always up to the initiator to triggers the beginning of the procedure, the responder can asynchronously request the start of any step of the procedure. Hence, there are no guarantees for the responder that the initiator will eventually heed the request. Therefore, security requests can logically be issued only by the slave or peripheral end of the connection.[19]

SMP includes pairing, bonding, and encryption re-establishment. In the pairing procedure, a temporary common security encryption key is generated to be able to switch to a secure, encrypted link. This temporary key is not stored and is therefore not reusable in subsequent connections. In the bonding procedure, after a sequence of pairing, permanent security keys are generated and, which is to be stored in non-volatile memory and, therefore, creating a permanent bond that allows the devices to quickly set up a secure link in subsequent connections without having to perform bonding again. After a bonding procedure is complete, If encryption keys have been stored, encryption re-establishment procedure defines how to use those keys in subsequent connections to re-establish a secure, encrypted connection without having to go through the pairing (or bonding) procedure again.[19]

The Generic Access Profile (GAP) defines how devices interact with each other at the lower level, outside of the actual protocol stack. GAP can be considered to define the BLE topmost control layer, given that it specifies how devices perform control procedures such as device discovery, connection, security establishment, and others to ensure interoperability and to allow data exchange to take place between devices from different vendors. GAP establishes different sets of rules and concepts to regulate and standardize the low-level operation of devices. Specifically, GAP list four roles that a device can adopt to join the BLE network, namely, broadcaster, observer, central, and peripheral. If it takes the broadcaster role, it will periodically send the advertising packets with the data. Theoretically, this broadcaster could be used with transmitter-only radios, but in practice, the device is usually capable of both transmitting and receiving. If the device has taken the observer role, it is optimized for receive-only applications that want to collect data from the broadcasting peer. Regarding the Myo’s implementation, the observer, which is the computer connected with Myo Bluetooth adapter, listens to the EMG, IMU and recognizes gesture data from the broadcasting device, the armband. The central device is capable of establishing connections to multiple peers, which is always the initiator of connection and allows devices onto the network. Thus, this role corresponds to the Link Layer master and, thereafter, requires resourceful computing capability, such as a gateway device. Lastly, the peripheral role corresponds to the Link Layer slave. This role uses advertising packets to allow the central device to detect it and, subsequently, to establish a connection with it. BLE optimized device with few resources to undertake peripheral implementation, at least in terms of processing power and memory. [19]

The Generic Attribute Profile (GATT) initializes how all profile and client information are communicated over a BLE network. As opposed to GAP, which characterizes low-level cooperations with devices, GATT just handles with real information and formats. As similar to other conventions or profiles in the Bluetooth, GATT characterizes two jobs that communicating entities can adopt, in particular, client and server. The client device sends update requests to the server and waits for responses from it. In practice, client knows nothing ahead of time about the server, and subsequently, it must inquire about the presence and the nature of those properties by performing service discovery. Subsequent to finishing the service discovery, it can then begin perusing and writing attributes found in the server, as well as server-initiated updates. Server, on the other hand, gets update demands from the client and sends responses back, which may incorporate server-initiated update if it was designed to do so. GATT server is liable for putting away and making the user information accessible to the client, as written in attributes. Each BLE device must incorporate with at least a basic GATT server that can respond to client demands, regardless of whether it would just return error messages.[19]

As previously described, GATT uses the Attribute (ATT) Protocol, where each attribute contains information about itself and about the actual data, including handle, type, permission, and value field. Handle field is a unique 16-bit identifier for each attribute on a GATT's server. This part of the attribute makes it addressable and is guaranteed to stay unchanged across bonded devices, even across connections. Attribute's type represents UUID of the device, which can take up two, four, or 16 bytes in data capacity. It determines the kind of data present in the value of the attribute and the mechanisms that are available to be discovered based exclusively on attribute type. Permission field is metadata that specifies which ATT operations can be executed on each particular attribute and with which security requirements. ATT defines the following permissions: Access permission, which determines whether the client can read or write (or both) an attribute value; Encryption, which determines whether a certain level of encryption is required for the client to access the attribute; and Authorization, which determines whether user permission is required to access this attribute. The value field holds the actual data content. Depending on the type of attribute, it may hold additional information about the attributes themselves or actual, useful, user-defined application data.[19]

Concerning the project work, the armband acts as peripheral and plays the broadcaster role, which periodically sends the advertising packets. The Myo dongle uses the Link Layer observer role to scan for advertising packets, which contains a single 128bit UUID for the control service (D5060001-A904-DEB9-4748-2C7F4A124842). Once connected, the armband can be treated as a GATT server. At the same time, the gateway device

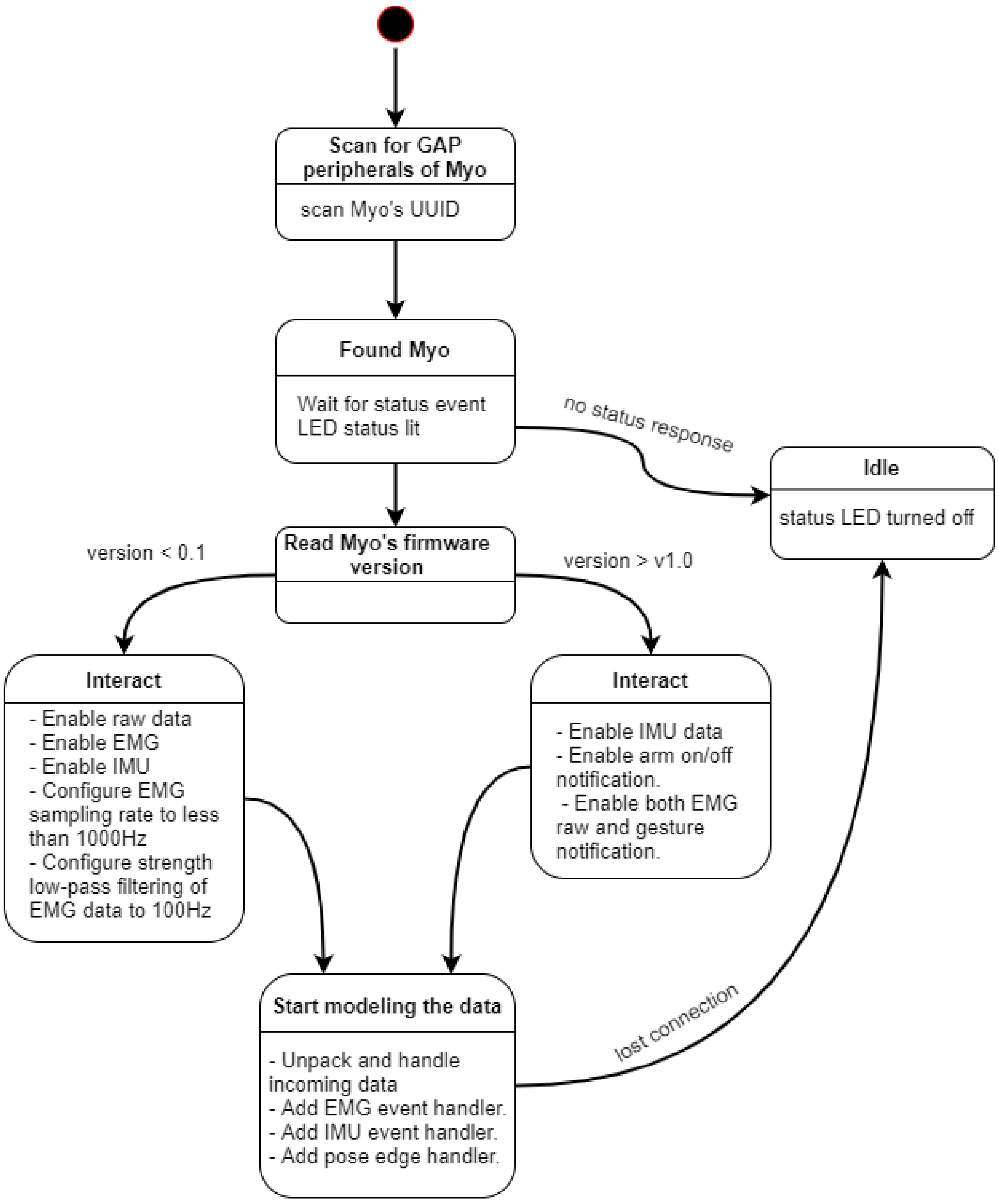


Figure 3. State diagram of Bluetooth communication flow with myo_raw Python library.

becomes a client, which reads and writes the characteristic values and subscribes to the changes of the user's arm movements and/or poses.[21]

In Dzhu's implementation [13], which utilizes the Python's serial library, the MyoRaw class initializes communication with Myo's band and sends command according to the corresponding protocol of the released header file [22]. As shown in Figure 2, it first scans the matching UUID packets that are received with Myo Dongle then waits for a response message from the armband, which triggers the horizontal LED on the band to lit. After successfully connected, the MyoRaw Python object then starts to check the firmware version info that is exposed within the sequence package exchanged between the gateway device and the band. This plays an essential part in the communication process [21] because it clarifies the underlying attributes, and Myo's command service that can be used, which may vary between different versions. At the time of writing this thesis, the firmware version was 1.5.1931.2. Following the firmware identification, the Python object sends the enabling command in order to retrieve EMG, IMU, and categorized pose data.

The later part of the Python library includes creating callback functions and store in a list (i.e., mutable), which are called when an EMG, inertial movement, or pose recognition data is received from Myo. Depending on which data is needed, we can append the handler to the corresponding list. For example, in the ros_myo application [14], the author retrieved the accelerometer, gyroscope, and magnetometer readings and rescaled them for demonstrating the orientation using the Euler formula. After that, these data are sent to ROS topic of IMU readings, where the ROS's 3D visualization tool is subscribed to display the real-time rotation of the Myo band with respect to the pitch, roll, and yaw axes. ROS topics are named channels in which nodes can publish or subscribe to talk with each other without knowing whom they are communicating with.[23] Furthermore, Python's ASCII graph library is used to display EMG signal strength from 8 channels by converting them on a 2048-bar-scale. The application in this thesis, however, only used the pose recognition data in combination with Myo's accelerometer reading to recognize the user's action and his arm's elevation.

2.5 Communication with other microcontrollers using MQTT

The communication tool in both implementations of this thesis is Message Queue Telemetry Transport (MQTT), a powerful yet lightweight “machine-to-machine” (M2M) communication protocol. It can be used in a wide range of IoT infrastructures, such as environmental sensors which communicate via satellite, dial-up connection with healthcare providers, home automation, and small beginner engineering project. Its popularity is due to the minimalistic design principle, which makes it an ideal protocol for any resource-constrained devices operating in low-bandwidth, high latency networks. There exist several MQTT-broker-service providers available, namely, HiveMQ, Adafruit, ActiveMQ, Rabbit MQ, Fleshpi, etc. [24] In this thesis's projects, Mosquitto was selected as a message broker. The reason was that it is a free Open Source, and compatible with various development platforms, thus easily available for all types of devices [25], and, more importantly, supports Home Assistant OS as a third party service.

Regarding the robot arm project, Mosquitto broker server, in this scenario, was installed directly into Ubuntu host gateway and was used for sending a command from Myo to ESP8266EX. After receiving the MQTT message, ESP-12E will then convert the message's payload into simple numeric values, which correspond with certain control commands, and send them to Arduino UNO [26] via the virtual serial bus. Even though there are already many projects where microcontrollers can perform as an MQTT client while controlling other hardware components, ESP-12E microcontroller is restrained from performing hardware driving tasks (such as servo control) while reading the Wi-Fi data packets. [27] Since MQTT is lightweight and optimized for low bandwidth constraints, it serves as an ideal approach for performing communication with the peripherals. After installing Mosquitto in the Intel gateway, it was possible to initialize the MQTT messaging channel with the participating clients, the MyoRaw Python object and the ESP8266EX, when they are connected to the same Wi-Fi network. The Python library for initializing communication with the gateway's server is an open-source Eclipse project written by Roger Light [28]. It provides client implementation with the MQTT server and MQTT-SN (MQTT for sensor network). Hence, the data generated from the sEMG and IMU handler of PyoConnect can be written into packages' payloads and published to exchange messages with the Node MCU. In the next chapter, a simple MQTT-based implementation is designed for controlling a desk lamp with ESP Home-Assistant's integration along with Mosquitto broker service.

3 HOME ASSISTANT CONTROL WITH NODE MCU V1.0

The original idea of using Myo in HCI was to create a non-invasive and interactive way to manage control over the devices that we would otherwise need to use accessory to control. In the following project, the aim was to simulate a simple gesture-control-smart-home network, with ESP8266EX as a sensor node, and communicate with the gateway within the same network.

3.1 Home Assistant installation

In order to install Home Assistant in an Ubuntu machine, it is recommended to choose the Supervised version, which is technically a Home Assistant (HA) Operating System (OS) running inside of a Docker container [29]. The advantage of Docker container, in comparison to Virtual Machines (VM), is the lightweight in terms of resource usage and the ability to isolate the application, Home Assistant, from the OS level, Ubuntu. [30] Moreover, since the purpose of this project is not to analyze the HA itself, it is unnecessary to have full access to all functionalities if we were to use VM. With that being said, it should be noted that there might be some custom configurations of the host OS that can potentially influence the Home-Assistant-Docker container. Therefore, ensuring system control over the HA supervisor container was essential during the development process. [29]

3.2 ESPHome

ESPHome is a simplified version of Home Assistant integration to monitor and control ESP node remotely. Instead of programming with C or Arduino-related language, we can compile codes using the ESP Addon of Home Assistant Web UI with the configuration file (.yaml) [31]. Expressly, ESPHome core component (i.e., Wi-Fi, GPIO, switches) are set up with the local network SSID and password, which was declared in the configuration file. After connecting the node MCU to the gateway through the serial port, the program can be flashed into the board to integrate with the Home Assistant. Moreover, once the board is connected to the Wi-Fi network, the Over-the-air update will enable publishing the codes and checking logs of the ESP-12E board.

The node MCU controls the switch with peripheral output port D2 as an MQTT client component for reading messages from gateway's server. Initially, a test with tactile-button as GPIO switch component was performed, which also shows that ESPHome pre-built library designed while countermeasure against debouncing effect was taken into account [32]. On setting up MQTT (Mosquitto) service on the gateway with ESP client credentials, automation can be configured to react upon reception of the subscribed topic message. Figure 2. exhibits an overview of MQTT message flow in the Home Assistant project between Node MCU, Myo armband, and Intel gateway.

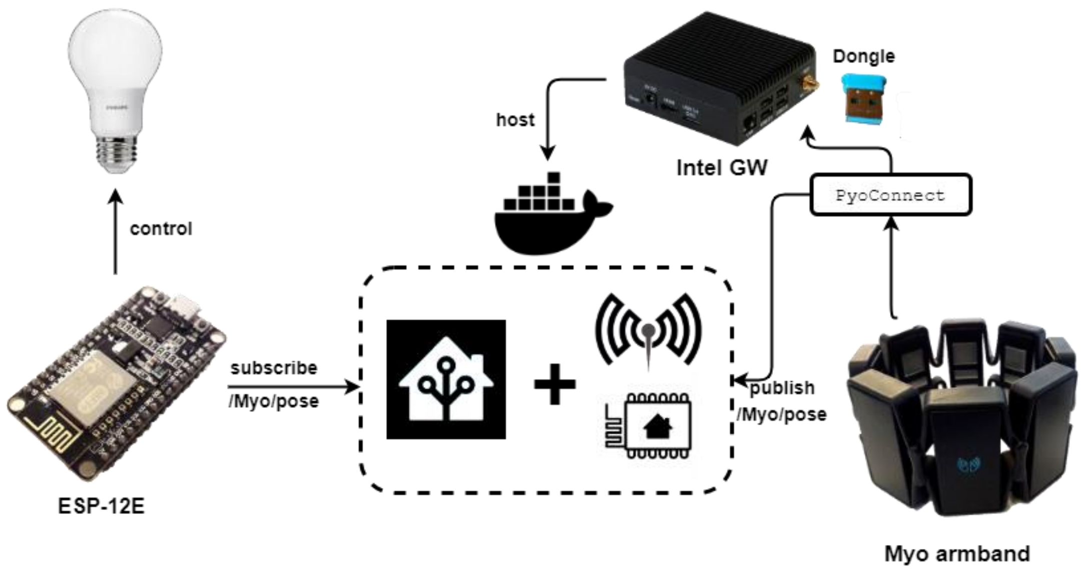


Figure 4. Data flow of MQTT communication channel between Myo module and ESP-12E through Home Assistant running in gateway’s Docker container.

3.3 MQTT messaging channel with Myo band

In the previously mentioned MQTT connection with Python, the user credential for initializing connection with the MQTT server (Home Assistant) is configured with QoS1 (quality of service level 1). Thus, the broker process the MyoRaw's handler message publish request as the user makes a matching pose (tapping middle finger and thumb twice, called "DOUBLE_TAP") and waits until it receives the message sent confirmation response, called PUBACK packet, from the receiver [33]. The receiver, ESP node, reads the package and responds with the PUBACK packet to the server for claiming that it accepts the ownership of the message. From then, it will treat all incoming packets with the same Packet Identifier as a new publication, regardless of the duplication flag [34].

The YAML scripts in Program 1 initialize the connection with gateway's Mosquitto broker server on ESPHome Add-on User Interface. Regarding the resulting output, as shown in Table 1, it can be seen that the moment when DOUBLE_TAP message (recognized as “double-tap” from PyoConnect library) was sent when from PyoConnect's handler, ESP node will send toggle command to LED.

Program 1. MQTT server configuration for espnode user.

```
# Connection with HomeAssistant MQTT broker
mqtt:
    broker: 192.168.xxx.xxx
    username: espnode
    password: EspHome!
    keepalive: 10s
    reboot_timeout: 0s
    on_message:
      topic: "/Myo/pose"
      payload: "DOUBLE_TAP"
      qos: 1
      then:
        - switch.toggle: led_d2
```

Table 1. Log output of ESPHome User Interface.

```
## ESPHome logs
INFO Reading configuration /config/esphome/mcu1.0.yaml...
INFO Starting log output from esp12e/debug
INFO Connected to MQTT broker!
[21:09:55][D][switch:045]: 'LED': Sending state OFF
[21:13:16][D][switch:029]: 'LED' Toggling ON.
[21:13:16][D][switch:045]: 'LED': Sending state ON
[21:13:18][D][switch:029]: 'LED' Toggling OFF.
[21:13:18][D][switch:045]: 'LED': Sending state OFF
## Python's output from PyoConnect
device name: MyoBand
unlock
Myo pose activated!
MQTT client initiated
Connected with result code: 0
[21:13:14]:received: rest
[21:13:16]:received: doubleTap
[21:13:16]:received: rest
[21:13:18]:received: doubleTap
[21:13:19]:received: rest
```

For the sake of demonstration, a desk lamp was calibrated to be controlled with ESP-12E via relay module. Initially, the lamp’s power cord was attached with a single-pole switch, which consists of a regular ON and regular OFF positions. After detaching the two stripped cords from the switch, a relay module, JQC-3FF, was used to replace the power switch. As shown in Figure 3, the relay was connected with power cords at Normal On (NO) and Common (C) sockets, which means raised (HIGH) signal from Digital pin 2 of ESP will close the circuit, thus, turn on the light and vice versa. [35]

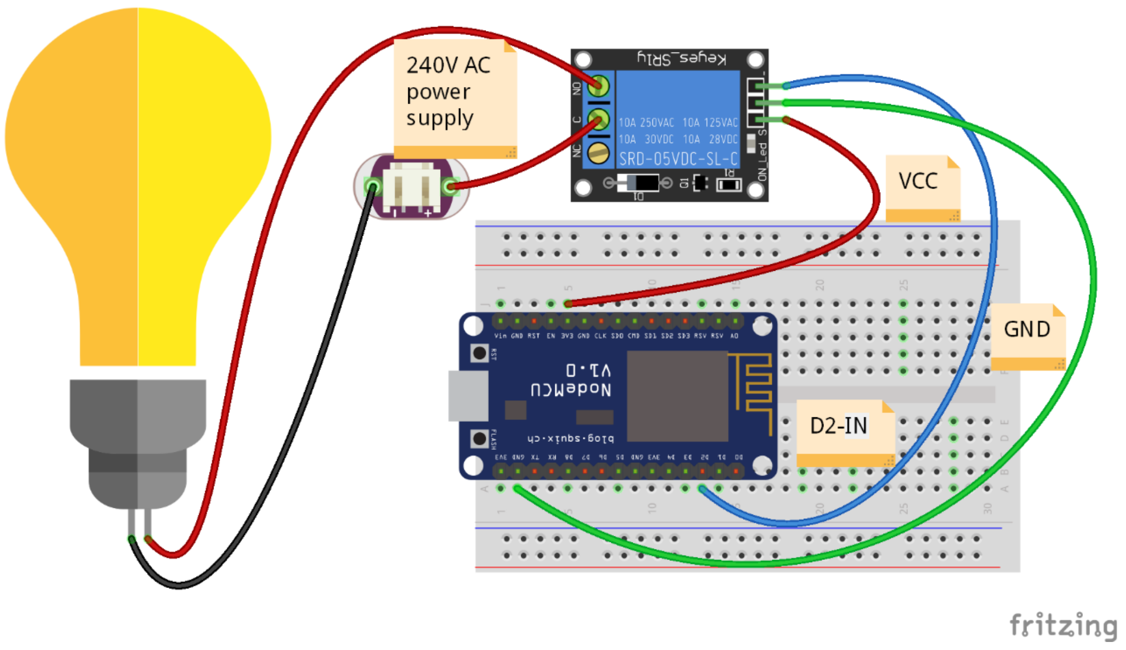


Figure 5. Lightbulb controlled by Digital signal from Node MCU via a JQC-3FF relay module.

3.4 Future development

The initial proposal of this project was initially intended to implement Myo band pose-recognition for controlling many different types of smart home appliances. During the planning phase of this project, a full set of home automation with Phillips smart devices, such as lightbulbs, air-conditioners, vacuum cleaners, were intended to be included in the smart system control. However, there was a slight change in the plan because of a shortage of available smart devices at the lab's premises. As a result, this project's goal has shifted towards a more scalable and straightforward approach, which only controls a desk lamp.

The response rate can be further tested even though each client of the network seems to have been optimized. Specifically, by lowering the active-standby ratio (keep-a-live time reduced from 15s to 10s), we can improve the overall stability of the connection. Additionally, MQTT reset time (reboot_timeout) can be initialized to zero, which reboots right after connection with the server is lost. With respect to the sending method, using QoS 1, which is known as the two-way-hand-shake approach, guarantees the delivery of messages [33]. However, it should be noted that the current design has not considered multiple nodes MCU listening to the same topic, or to multiple topics at the same time. In other words, a scalability test is needed to evaluate the performance of this in future development fully.

Another issue, which was essential yet neglected, is the stability of Myo armband connectivity with the USB Dongle. It would regularly disconnect with Myo after a few minutes of standby from the user. Even though PyoConnect user interface provides connection status, a more robust solution is needed to prolong the connection. According to [21], one possible way is to enable myohw_sleep_mode_never_sleep in the sleep mode setting. It should be taken into concern that this would also consume more battery capacity than the normal mode.

4 ROBOT CONTROL WITH NODE MCU V1.0 AND ARDUINO UNO

In the following design, the recognized gestures from Myo band, in combination with accelerometer readings, are collected for pose classification. The goal is to achieve ease-of-use control while maximizing the response rate and accuracy of the pose-recognition mechanism.

4.1 Previous related work with robotic control design

Regarding rehabilitation in robotic applications, there were several projects where the raw sEMG data is pre-processed in order to recognize custom gestures. For example, Hussein et al. [6] employed overlap technique and classifier algorithm, such as Support Vector Machine (SVM), k-Nearest Neighbor (KNN), Linear Discriminant Analysis (LDA), to identify gestures from recorded sEMG signals. Their proposed system achieved up to 95.26% accuracy. In another work [7], the KNN algorithm was used to classify ten different gestures; however, they later removed some, which were unused to enhance the accuracy of the system. Even though designing a gesture-recognition application would provide a more flexible tool for testing, in this thesis work, the main focus is to discover possible usage of Myo armband and enhance the reaction rate of connected devices (servo motors of the robot in this case).

4.2 Command protocol between Myo and controllers

Myo's python node, similar to the previously mentioned project, evaluates its recognized gesture in combination with IMU data and translates the corresponding message then sends them through the MQTT broker of the gateway. Arduino, on the other hand, communicates with ESP through the UART channel (i.e., Serial communication bus). It should be noted that even though pin 0 and 1 are hardware UART ports, they are reserved for Serial Monitor communication with Arduino IDE [36]. Therefore, it is recommended to use software UART ports (pin 2-SRX and pin 3-STX) instead. Thus, pin D2 and D1 (RX and TX respectively) of ESP-12E is to connect with pin 3-STX and pin 2-SRX of Arduino. Additionally, the baud rate of Arduino's serial bus is 115200 bps in order to communicate with the ESP-12E board.

Program 2. Arduino code snippet shows the configuration of SUART on pin 3-SRX and pin 4-STX, and Interrupt handler on pin 2

```
## Arduino UNO ##
#include <Servo.h>
#include <SoftwareSerial.h>
SoftwareSerial SUART( 3, 4); //SRX  = DPin-3; STX = DPin-4
const byte interrupt_pin = 2;
...
void restISR(){
  rest = true;
  Serial.println("Arm stopped!");
}
void setup(){
    SUART.begin(115200);
    Serial.begin(9600);
    pinMode(interrupt_pin, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(interrupt_pin), restISR, RISING);
    ...
}

void loop(){
if (SUART.available() != 0){
     /** print the received message  **/
    command = SUART.read();
    Serial.print("Command received: ");
    Serial.println((char)command);
    /** Handling the command messages **/
    ...
}
```

Program 3. Arduino code snippet demonstrates SoftwareSerial is configured at port D2 and D1 of ESP-12E.

```
## ESP-12E ##
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <SoftwareSerial.h>
SoftwareSerial SUART(4, 5); //SRX=Dpin-D2; STX-DPin-D1
const byte interrupt_pin 0; //DPin-D3

void setup()
{
    Serial.begin(9600);
    SUART.begin(115200);
    pinMode(interrupt_pin, OUTPUT)
    ...
    # Wifi & MQTT broker connection configuration
    ...
}
```

The Arduino code of ESP-12E snippets, as shown in Program. 3, reproduce the message from MQTT server's/Myo/pose topic and convert it into a simple command (from zero to six). Arduino UNO, on the other hand, handles those messages from software serial and controls the rotation of servos gear shaft with Pulse-width-modulation (PWM) signals. Pulse-Width-Modulation is a series of repeating pulses with different duty cycles, which represent the proportion HIGH state (5V) throughout one period of the signal. [37]

The state diagram, as demonstrated in Figure 5, is comprised of a list of decision-making blocks for message translation before the commands are sent to Arduino UNO. It starts by checking for the existence of a new MQTT message from /Myo/pose topic. If the espnode (the designated name for using ESP-12E in MQTT server) client detects a message, it will then check whether the message’s payload matches with those in Table 2, to send as commands to Arduino UNO through software serial bus. If the message is unrecognizable, an error message will be printed to Serial Monitor (which runs at 9600 baud-rate), which can be visible from Arduino IDE’s Serial monitor user interface. Once the command message is sent via the Software Serial bus, the whole process is repeated.

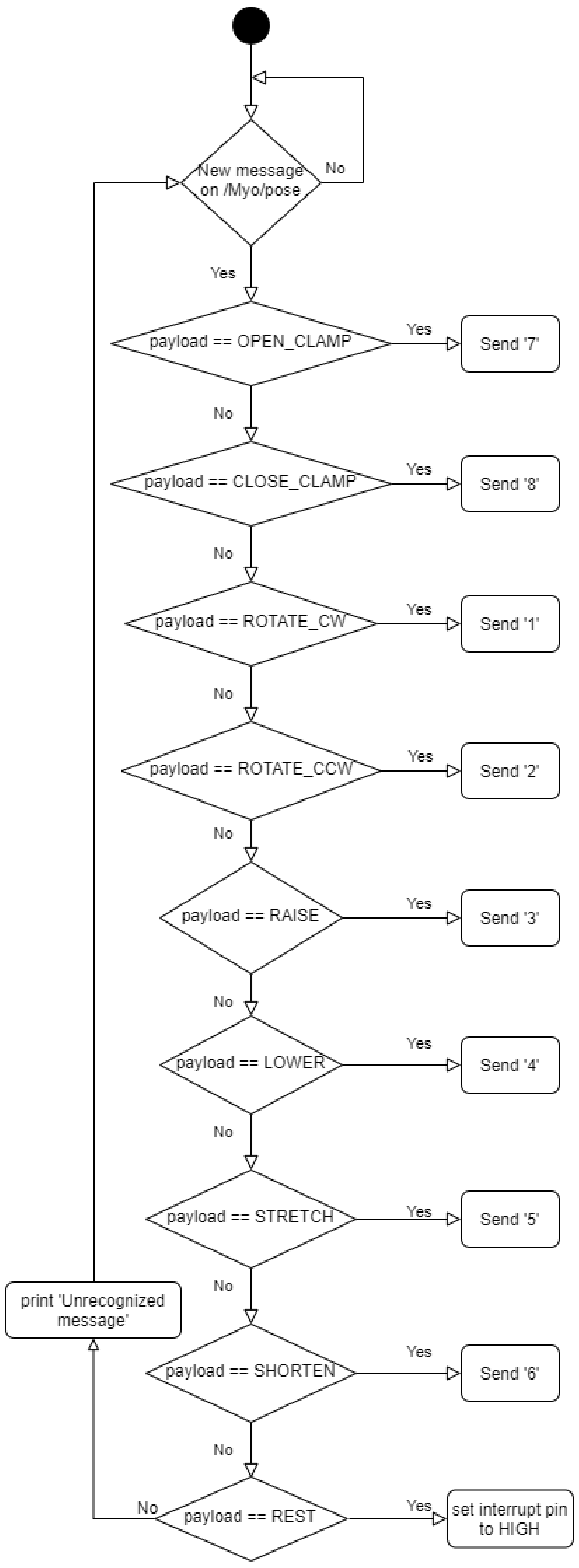
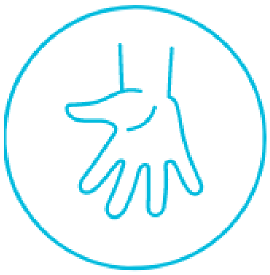
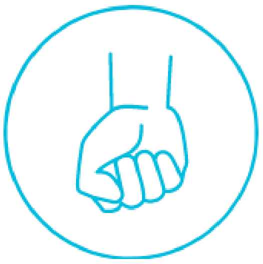
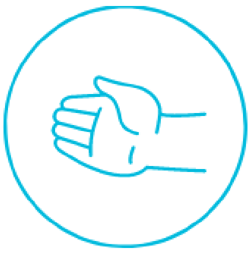
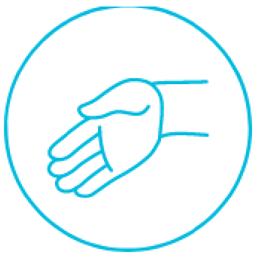
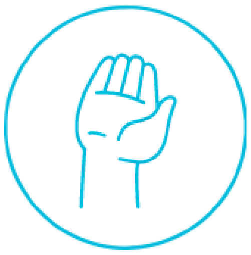


Figure 6. MQTT message handler logic of ESP-12E

Table 2. Command protocol of Arduino Uno with respect to /Myo/pose MQTT message. Pose figures that were retrieved from [38].

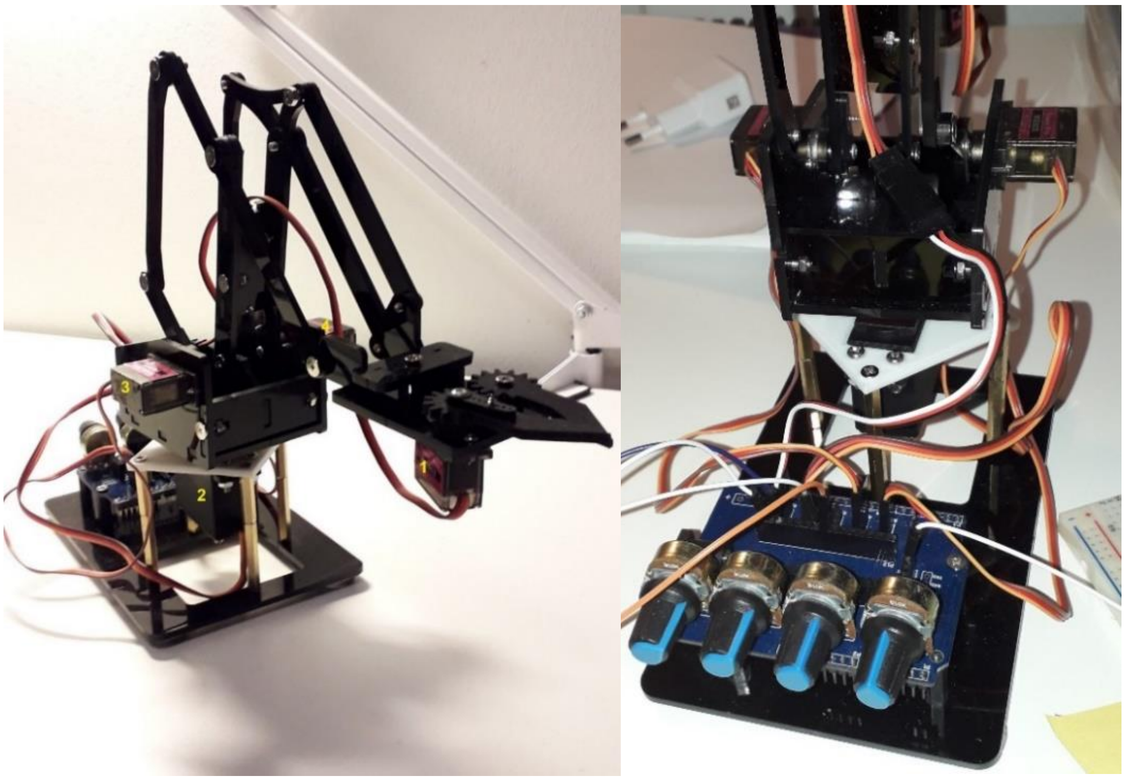
Pose	Pose description	/Myo/pose message	Command	Description
	Fingers spread while the forearm is lowered	OPEN_CLAMP	7	Open clamping hand
	Fingers closed (fist) while forearm is lowered	CLOSE_CLAMP	8	Close clamping hand
	Waving out while the forearm is at adduction	ROTATE_CW	1	Clockwise rotation of the base servo
	Waving out while the forearm is at adduction	ROTATE_CCW	2	Counter-clockwise rotation of the base servo
	Waving out while the forearm is raised	RAISE	3	Raise arm (Clockwise rotation of shoulder servo)

	Waving in while the forearm is lowered	LOWER	4	Lower arm (Counter-clockwise rotation of shoulder servo)
	Fingers spread while the forearm is at adduction	STRETCH	5	Stretch arm reach (Counter-clockwise rotation of elbow servo)
	Finger closed while the forearm is adduction	SHORTEN	6	Shorten arm reach (Clockwise rotation of elbow servo)

In order to achieve multiple robotic arm movement controls, existing features of Myo's band that were extracted in this implementation are combined. In detail, the accelerometer sensor readings from IMU and Myo’s recognized gesture are utilized together to compose the set of commands protocol. However, for the sake of simplicity, the forearm angle with the chest is evaluated by x and z vector’s value pair from the IMU’s accelerometer, which are scaled by gravitational acceleration (g). Specifically, by comparing the sign and value pair (x-z), it would be possible to determine the relative position of the wearer’s forearm. For instance, if their product is negative or, in other words, opposite signs, and the measured acceleration’s value on vector x is less than -0.5g, then the forearm is predicted to the body. Nevertheless, to ensure the Myo armband to read correctly, users need to wear the armband so that the Myo’s logo is facing up, with the status LED points toward the hand.

Regarding the Arduino UNO’s program, the Arduino servo library’s functions were adopted to monitor and control the arm's servo motors. In detail, a continuous-update loop is implemented, which targets at their virtual extreme values (minimum and maximum angles) depending on their intended range of movement. Those values are calculated based on the initial experiments with the arm. Namely, handling clamp’s servo, located at the tip of the arm, has a range of zero to 70 (open and close hands); bottom motor ranges from zero to 180, where 90 is the default position; 80-160 with left servo; and 15-85 with servo right. On receiving the REST signal (idling pose categorized by Myo), ESP-12E sends interrupt-signal to Arduino UNO, as shown in Program 2, restISR() interrupt handler function, which will stop any currently active commands.

4.3 Robot Arm



(a) (b)

Figure 7. (a) Robot arm with four servos: 1 - handling clamp; 2 - swivel base; 3 - elbow; 4 – shoulder; (b) DC voltage regulator platform.

The modeled robot arm is an ArduinoNano-compatible version of MeArm with a rotary-encoder-controlled platform instead of a joystick-controlled board of the original version. MeArm is a kick-starter project launched since March 2014. The project’s owner aims to bring a mini and inexpensive version of industrial robot arm design and programming to any level of education [39]. Since this is an Open Source project, there have been several enthusiasts who have written their own similar robotic arm projects as well as tutorials on Arduino Project hub [40], Instructables [41], or Hack-a-day [42]. Therefore, with a different version of the MeArm robot being used, it was reasonably simple to calibrate the servos and assemble the robot from individual parts.

The robot’s platform is built on a DC motor driver with a 5V voltage regulator. Thus, each servo’s female jumper wires can be attached to the male headers on the board. In this project’s implementation, slots 1, 2, 4, and 5 are used with servo 1 (clamp), 2 (base), 4(shoulder), and 3(elbow), respectively. Through reverse engineering, as shown in Figure 8, we can detect which female pins pair with those headers, by sending the PWM signal to each of those selected slots. In Figure 9, the simplified schematic describes how the components are connected.

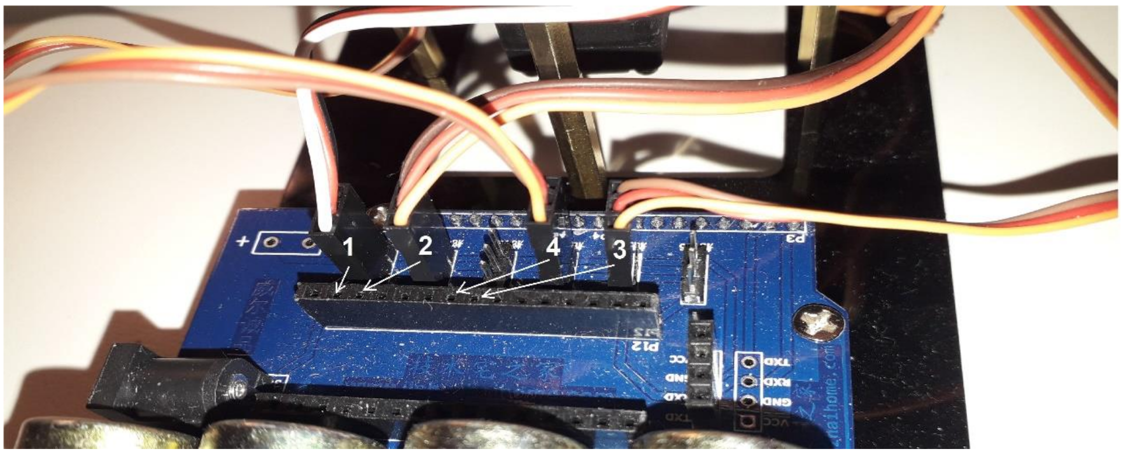
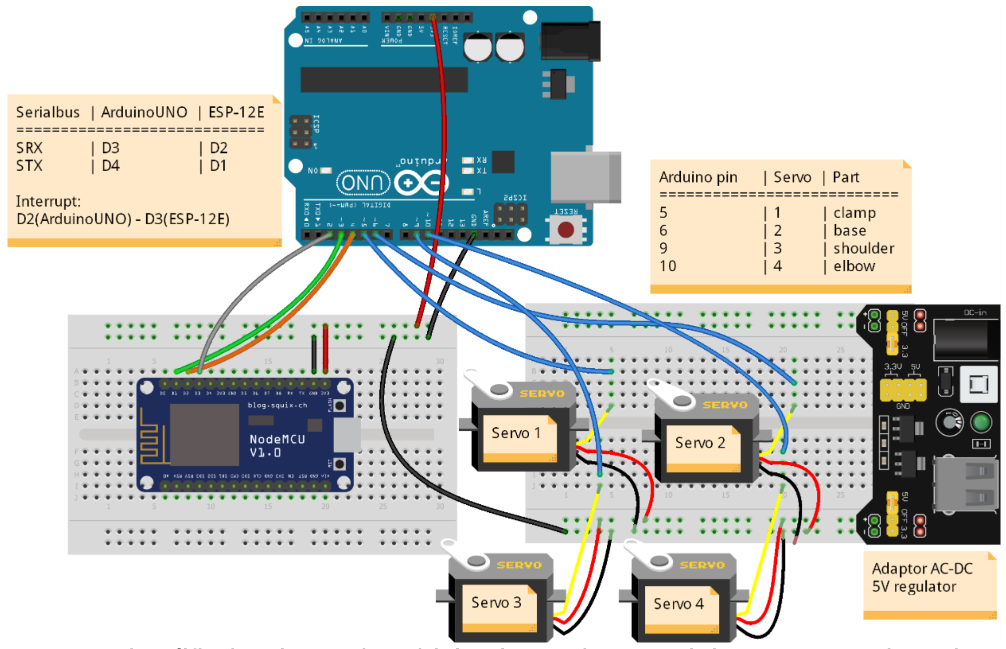


Figure 8. Jumper wires, which are numbered with respect to servos, are connected to the male pin headers of DC voltage regulator.



```
Serialbus  | ArduinoUNO  | ESP-12E
============================
SRX        | D3          | D2
STX        | D4          | D1

Interrupt:
D2(ArduinoUNO) - D3(ESP-12E)
```

```
Arduino pin     | Servo  | Part
============================
5               | 1      | clamp
6               | 2      | base
9               | 3      | shoulder
10              | 4      | elbow
```

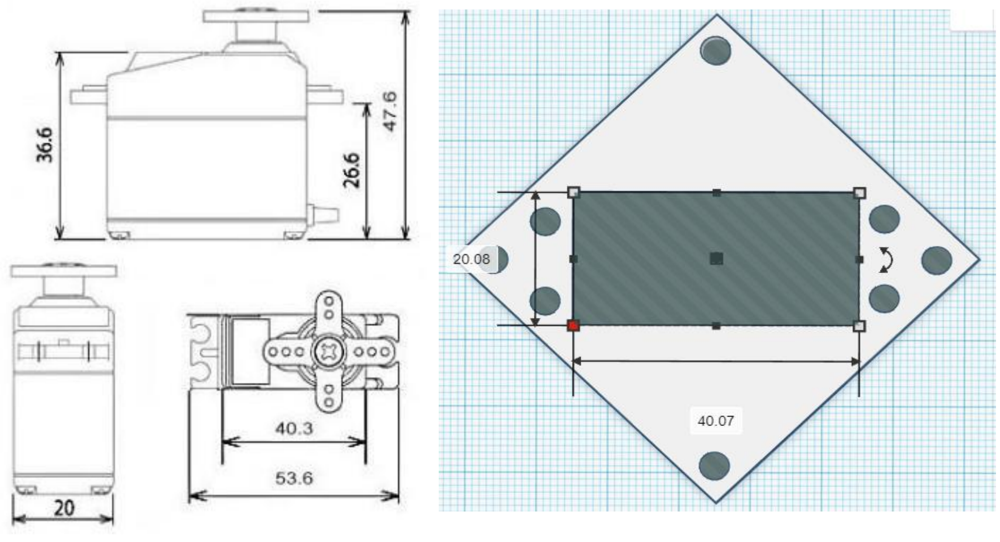
Figure 9. Simplified schematic which shows how Arduino UNO and NodeMCU and MeArm's servo motors are linked.

4.3.1 Servo motors

The robot arm includes four servos, three of which are MG90S, while the other (at the bottom) is MG996R. In the original form, all four servos are MG90S, and they were designed to be controlled with a potentiometer platform (at the back) where an Arduino nano can be attached. In order to improve the stability of the robot leg, the bottom servo motor was replaced with an MG996R model with a stall torque of 9.6kg/cm compared to 1.8kg/cm of MG90S. Since the PWM signal is required to control servo, Arduino UNO's pin 5, 6, 9, and 10 were used as they support PWM outputs [26]. However, since it draws a current of around 500-900 mA, [43] while Arduino UNO limit output current from 5V DC pin to 200 mA,[44] the AC-DC voltage regulator is needed to regulate currents to drive the servos.

As mentioned earlier, each servo has a specific DoF (degree-of-freedom) depending on which part on the robot's arm that it controls. For example, elbow's servo can make a turn between its 80-to-160-degree range, while the shoulder's servo has a broader range of movement, 90-to-180. DoFs' value ranges are obtained by manually reading the servo value while moving each servo during the calibration process because the robot arm was not well documented.

4.3.2 Replaced arm's base



(a) (b)

Figure 10. (a) - MG996R's dimension details (24); (b) Corresponding 3D printed base (drawn with TinkerCad online software)

Since base servo is replaced, the bearing base needed to be replaced with 3D printing. The model was drawn using TinkerCad 3D online application. As shown in Figure 10, we designed the base model so that it fits the top of MG996R servo width and length (20mm and 54 mm, respectively). In addition, there were eight screw holes for the robot’s legs, which were also replaced to fit with the height of MG996R, attachment. The 3D printing tool that was used here is one of the lab’s equipment, called Anycubic i3 printer. In this model, since the circular-shaped (screw holes) corners located near to rectangular-shaped corners, it was analyzed with Ultimaker Cura software to include supports (raft) during the printing process to preserve the shape of the desired model.

4.4 Compromise with ESP8266 limitation

While it is possible to control the servos with only ESP-12E board, it would accidentally crash and reboot automatically when the running program is in blocking state and waits for a message from Myo armband. The reason behind this is that ESP8266 has a watchdog timer (WDT), which calls the timer reset whenever the program is detected to runs inside an infinite loop. In the previous version of Arduino code in ESP-12E, the program keeps polling for any message from MQTT Myo/pose’s topic channel. Thus, WDT eventually reset Node MCU when it detects that the program runs into a blocking state [45].

With the addition of an Arduino UNO, acts as the motor driver, it is possible to perform MQTT message polling while sending an interrupt to control the servo. In the new Arduino implementation, a hardware interrupt is attached to pin 2 of Arduino UNO to stop the servo that is currently moving. Hence, it serves as a coherent and effective approach for controlling the output signal from the Arduino since software interrupt can instantly respond to the REST message from the gateway’s server.

4.5 Result

After some mock tests with MQTT messaging, the MeArm robot can accurately control its servo and move according to commands sent from Node MCU. After receiving the ROTATE_CW message from Mosquitto server, Arduino UNO will read the corresponding command, with value one, from ESP and, thus, send clockwise rotation signal to the base servo. To simplify the procedure of this implementation, the developer himself has tested the system by wearing Myo’s armband on the right arm and making poses according to the protocol in Table 2. As expected, the gesture messages from Myo’s armband were correctly translated to control MeArm’s servos. However, the PyoConnect gesture recognition handler cannot read correctly read the actual arm’s gestures. For example, it would mistakenly recognize a waving out gesture while the user is spreading fingers. As a result, it can be estimated that around 10% of this pose was misread. In addition, since each person may have different EMG readings, it requires a little practice with making gestures while wearing Myo armband before we can manage to control the robot arm fluently.

5 DISCUSSION

There were some changes during the planning phase of this thesis work. The initial intention this Myo's armband project was to analyze the EMG signal and perform gesture recognition using pattern algorithms. It can be seen that there was a wide range of possible research that has already been carried out regarding EMG's signal processing. However, the required time and effort to renovate ideas for those researches were overlooked. Therefore, it was not until Myo's gesture recognition feature is extracted from Myo band that the decision to alter the plan with a more feasible one was made, which aims to build a simple robotic application using existing features from Myo armband.

It should be noted that the gyroscope and magnetometer's feature Myo's IMU are ignored. It can be seen that we can utilize those features in calculating pitch, roll, and yaw for replicating the rotation of armband to rotate the robot arm movement. For example, we can use the Euler formula to compute pitch, roll, and yaw [46]:

$$pitch = 180 * \frac{\operatorname{atan}\left(\frac{accel_x}{\sqrt{\left(accel_y^2 + accel_z^2\right)}}\right)}{\pi}$$

$$roll = 180 * \frac{\operatorname{atan}\left(\frac{accel_y}{\sqrt{(accel_x^2 + accel_z^2)}}\right)}{\pi}$$

$$yaw = 180 * \frac{\operatorname{atan}\left(\frac{accel_z}{\sqrt{(accel_x^2 + accel_z^2)}}\right)}{\pi}$$

Equation 1. Euler formula of calculating pitch, roll, and yaw from the accelerometer

With the orientation readings, it would be possible to have a more comprehensive way to implement command instructions to the robot arm. Since it can mimic the forearm movement, it is easier to be controlled by users. Nevertheless, this approach would fundamentally change the logic of Pyo's pose edge handler functions. These values can interpret the rotational angles of each servo motor. For instance, pitch, roll, and yaw of Myo armband can determine MeArm's shoulder, bearing base, and ankle motor's angular value, respectively. In that case, stop interrupt signal would not need to be included in the command protocol as the targeted angles were already be specified. Additionally, that would require the MQTT channel to be able to process a much higher influx of messages from Myo's side, since the IMU signals are sent at 50Hz [22].

Regarding the outcome, this thesis work has succeeded in illustrating two possible ways to apply pose-recognition and IMU's features of Myo's armband. Even though there exist possible improvements that can still be done in those projects, they have partly met initial goals that were set at the beginning. Tests are not performed by real patients with speech difficulties, but only by a healthy individual. However, it has shown successes in terms of the performance of the feedback rate. HomeAssistant project, in particular, desk lamp light turned on instantly after the user makes a thumb-to-middle-finger pose. Though this implementation is small in terms of scaling, it brings a fundamental step for any further development with similar ideas. Secondly, the MeArm robotic project has also displayed a promising result. By utilizing the Serial communication bus of Arduino and Node MCU, the outcome was improved as compared to relying only on the MQTT message handler of ESP Node MCU to control the servos. In addition, it has enhanced the stability of the robot by replacing the base servo motor and, thus, reduced the shattering of servos movements.

Shortcomings exist in both implementations. Firstly, the gesture recognition feature of Myo armband cannot read the users' poses correctly. Each person may have different sEMG signal patterns when making the same set of poses. This has influenced the accuracy of the system and, thus, would confuse the users when they unintentionally send a different command. It could be improved by applying training software, which utilizes machine learning algorithms such as k-nearest-neighbor (KNN) or Support Vector Machine (SVM), to perform pose-pattern categorization instead of relying on reusing the set recognizable gestures. The second drawback of the design is that readings from the inertial movement unit of Myo armband were not applied. If the command to control the robot was the movement of users' forearms, it would have been more immersive from the perspective of the wearer. Thirdly, even though the developed Python library, PyoConnect, has been proven as a functioning Linux alternative to MyoConnect, there exist interruptions during the connection period due to interferences with surrounding Wi-Fi signals. Therefore, a simple work-around solution for maintaining synchronization with the armband is to let the gateway automatically recall for the Myo armband Bluetooth signal whenever it disconnects.

6 CONCLUSION

This thesis work was planned to illustrate a possible implementation with a gesture-based HCI, which was initially intended to support people with speech defects, thus cannot use their voice to control their smart home devices. In the first project, the Home Assistant system has succeeded in providing an assistive automation solution where these persons can instead use hand gestures in combination with arm movements and switch on and off the lights. Moreover, with further integrations with other smart devices, it would be possible even to control air-conditioners, ventilators, or television with ease. In the project with the robotic arm, even though many enthusiasts and developers have already implemented their model with MeArm, the developed gesture-controlled version would be a useful and important milestone for students or researchers who are interested. Overall, this project proved to be a simple yet reusable implementation for students who would like to reference in their robotic project.

Concerning the outcome, this thesis work has demonstrated working schemes of Myo armband in two use cases, namely home automation, and robotic application. In spite of the limited scale, the HomeAssistant project can be easily extended to other smart home devices or installed with other UI integrations that provide more visuals into the HA’s dashboard. Since the project aims to prioritize minimalistic in terms of design, it successfully illustrates a simple implementation of Myo’s armband using MQTT. Regarding the robotic arm project, users were able to send commands from Myo armband to the ESP Node MCU and Arduino UNO to control the MeArm robot. For instance, while the user is performing a pose, the robot would continue the corresponding movement until the user stops by sending the “REST” message to ESP Node MCU, thus, interrupt Arduino UNO from sending PWM control signals to the running servo motor. Even though the Myo’s gesture recognition feature cannot always predict correctly, its accuracy at 90% was enough to meet our expectation. In addition, the connectivity with Myo’s Bluetooth dongle through the PyoConnect library seems unstable. Hence, some self-maintenance effort is needed from the gateway. Nevertheless, this outcome of this project can serve as a working proof-of-concept for a gesture-based IoT system.

REFERENCES

[1] F. Gaetani, P. Primiceri, G. Antonio Zappatore, and P. Visconti, “Hardware design and software development of a motion control and driving system for transradial prosthesis based on a wireless myoelectric armband,” *IET Sci. Meas. Technol.*, vol. 13, no. 3, pp. 354–362, 2019, doi: 10.1049/iet-smt.2018.5108.

[2] P. Mtshali and F. Khubisa, “A Smart Home Appliance Control System for Physically Disabled People,” in *2019 Conference on Information Communications Technology and Society (ICTAS)*, Mar. 2019, pp. 1–5, doi: 10.1109/ICTAS.2019.8703637.

[3] A. Khusnutdinov, D. Usachev, M. Mazzara, A. Khan, and I. Panchenko, “Open Source Platform Digital Personal Assistant,” in *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, May 2018, pp. 45–50, doi: 10.1109/WAINA.2018.00062.

[4] H. V. Bhatnagar, P. Kumar, S. Rawat, and T. Choudhury, “Implementation model of Wi-Fi based Smart Home System,” in *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, Jun. 2018, pp. 23–28, doi: 10.1109/ICACCE.2018.8441703.

[5] “ESP8266 Overview | Espressif Systems.” https://www.espressif.com/products/hardware/esp8266ex/overview/ (accessed Apr. 12, 2020).

[6] H. F. Hassan, S. J. Abou-Loukh, and I. K. Ibraheem, “Teleoperated robotic arm movement using electromyography signal with wearable Myo armband,” *J. King Saud Univ. - Eng. Sci.*, May 2019, doi: 10.1016/j.jksues.2019.05.001.

[7] S. Bisi, L. De Luca, B. Shrestha, Z. Yang, and V. Gandhi, “Development of an EMG-Controlled Mobile Robot,” *Robotics*, vol. 7, no. 3, p. 36, Sep. 2018, doi: 10.3390/robotics7030036.

[8] S. Rawat, S. Vats, and P. Kumar, “Evaluating and exploring the MYO ARMBAND,” in *2016 International Conference System Modeling Advancement in Research Trends (SMART)*, Nov. 2016, pp. 115–120, doi: 10.1109/SYSMART.2016.7894501.

[9] B. Stern, “Myo Armband Teardown.” Adafruit Learning System, Aug. 22, 2018, Accessed: Apr. 09, 2020. [Online]. Available: https://cdn-learn.adafruit.com/downloads/pdf/myo-armband-teardown.pdf.

[10] “How to wear the Myo armband,” *Welcome to Myo Support*. http://support.getmyo.com/hc/en-us/articles/201169525-How-to-wear-the-Myo-armband (accessed Mar. 25, 2020).

[11] “How to perform the sync gesture – Welcome to Myo Support.” https://support.getmyo.com/hc/en-us/articles/200755509-How-to-perform-the-sync-gesture (accessed Mar. 25, 2020).

[12] “Intel® Based Gateway System LoRa Certified Gateway and Network Server-AIOT-ILRA01,” *AAEON.COM*. https://www.aaeon.com/en/p/intel-lora-gateway-system-server (accessed Apr. 08, 2020).

[13] D. Zhu, *dzhu/myo-raw*. 2019.

[14] *uts-magic-lab/ros_myo*. UTS Magic Lab, 2019.

[15] “About - TurtleBot.” https://www.turtlebot.com/about/ (accessed May 09, 2020).

[16] “PyoConnect - MyoConnect for Linux.” http://www.fernandocosentino.net/pyoconnect/ (accessed Mar. 25, 2020).

[17] “esp8266-module-family [ESP8266 Support WIKI].” https://www.esp8266.com/wiki/doku.php?id=esp8266-module-family (accessed Apr. 10, 2020).

[18] "Arduino Uno Pinout, Examples, Programming and Applications," *Microcontrollers Lab*, Oct. 06, 2018. https://microcontrollerslab.com/introduction-arduino-uno/ (accessed Apr. 11, 2020).
[19] K. Townsend, C. Cufí, Akiba, and R. Davidson, *Getting Started with Bluetooth Low Energy*, 1st ed. 1005 Gravenstein Highway North, Sebastopol, CA 95472.: O'Reilly Media, Inc., 2014.
[20] "Bluetooth Low Energy (BLE) Introduction - Part 1," *EmbeTronicX*, Jul. 26, 2017. https://embetronicx.com/tutorials/tech_devices/bluetooth-low-energy-ble-introduction-part-1/ (accessed May 10, 2020).
[21] "Myo Bluetooth Protocol Released," *The Lab*, Mar. 26, 2015. https://developerblog.myo.com/myo-bluetooth-spec-released/ (accessed Mar. 24, 2020).
[22] *thalmiclabs/myo-bluetooth*. Thalmic Labs, 2020.
[23] "Topics - ROS Wiki." http://wiki.ros.org/Topics (accessed Apr. 08, 2020).
[24] "mqtt/mqtt.github.io," *GitHub*. https://github.com/mqtt/mqtt.github.io (accessed Apr. 08, 2020).
[25] "Eclipse Mosquitto," *Eclipse Mosquitto*, Jan. 08, 2018. https://mosquitto.org/ (accessed Apr. 08, 2020).
[26] "Arduino Reference." https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/ (accessed Apr. 04, 2020).
[27] V. Roland, *vroland/MyoIMUGestureController*. 2019.
[28] "paho-mqtt · PyPI." https://pypi.org/project/paho-mqtt/#id3 (accessed Mar. 29, 2020).
[29] H. Assistant, "Installing Home Assistant," *Home Assistant*. https://www.home-assistant.io/hassio/installation/ (accessed Apr. 08, 2020).
[30] "What is Docker: Docker vs VirtualBox, Home Server with Docker," *SmartHomeBeginner*, Dec. 03, 2016. https://www.smarthomebeginner.com/what-is-docker-docker-vs-virtualbox/ (accessed Apr. 08, 2020).
[31] "Getting Started with ESPHome through Hass.io," *ESPHome*. https://esphome.io/guides/getting_started_hassio.html (accessed Mar. 30, 2020).
[32] "GPIO Binary Sensor," *ESPHome*. https://esphome.io/components/binary_sensor/gpio.html (accessed Mar. 30, 2020).
[33] "Paho MQTT C Client Library: Quality of service." https://www.eclipse.org/paho/files/mqttdoc/MQTTClient/html/qos.html (accessed Mar. 31, 2020).
[34] "Internet of Things MQTT Quality of Service Levels - DZone IoT," *dzone.com*. https://dzone.com/articles/internet-things-mqtt-quality (accessed Mar. 31, 2020).
[35] C. B. | Arduino | 67, "How to Set Up a 5V Relay on the Arduino," *Circuit Basics*, Nov. 28, 2015. https://www.circuitbasics.com/setting-up-a-5v-relay-on-the-arduino/ (accessed Apr. 16, 2020).
[36] "Serial Communication between Arduino and NodeMCU." https://forum.arduino.cc/index.php?topic=605324.0 (accessed Apr. 02, 2020).
[37] "What is a Pulse Width Modulation (PWM) Signal and What is it Used For? - National Instruments." https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019OkFSAU (accessed Apr. 08, 2020).
[38] "What gestures does the Myo armband recognize?," *Welcome to Myo Support*. http://support.getmyo.com/hc/en-us/articles/202647853 (accessed Apr. 06, 2020).
[39] "MeArm - Pocket Sized Industrial Robotics for Everybody," *Kickstarter*. https://www.kickstarter.com/projects/phenoptix/mearm-pocket-sized-industrial-robotics-for-everybo (accessed Apr. 06, 2020).

[40] “MeArm Robot Arm - Your Robot - V1.0,” *Arduino Project Hub*. https://create.arduino.cc/projecthub/benbobgray/mearm-robot-arm-your-robot-v1-0-326702 (accessed Apr. 06, 2020).
[41] phenoptixFollow, “MeArm - Build a Small Hackable Robot Arm V0.3,” *Instructables*. https://www.instructables.com/id/MeArm-Build-a-Small-Hackable-Robot-Arm/ (accessed Apr. 06, 2020).
[42] “MeArm - Your Robot.” https://hackaday.io/project/181-mearm-your-robot (accessed Apr. 06, 2020).
[43] “MG996R Datasheet(PDF) - List of Unclassifed Manufacturers.” Accessed: Apr. 04, 2020. [Online]. Available: https://www.alldatasheet.com/datasheet-pdf/pdf/1131873/ETC2/MG996R.html.
[44] “Arduino Playground - ArduinoPinCurrentLimitations.” https://playground.arduino.cc/Main/ArduinoPinCurrentLimitations/ (accessed Apr. 16, 2020).
[45] “Soft WDT reset - NodeMCU.” https://forum.arduino.cc/index.php?topic=442570.0 (accessed Apr. 05, 2020).
[46] “mechanical engineering - Calculating pitch, yaw, and roll from mag, acc, and gyro data,” *Engineering Stack Exchange*. https://engineering.stackexchange.com/questions/3348/calculating-pitch-yaw-and-roll-from-mag-acc-and-gyro-data (accessed Apr. 06, 2020).