

**DEVELOPMENT OF A WEARABLE GLOVE FOR
VIRTUAL REALITY APPLICATION**

LOW ZE XIAN

UNIVERSITI TUNKU ABDUL RAHMAN

**DEVELOPMENT OF A WEARABLE GLOVE FOR VIRTUAL
REALITY APPLICATION**

LOW ZE XIAN

**A project report submitted in partial fulfilment of the
requirements for the award of Bachelor of Mechatronics
Engineering with Honours**

**Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman**

May 2023

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature :



Name : LOW ZE XIAN

ID No. : 18UEB02784

Date : 30/4/2023

APPROVAL FOR SUBMISSION

I certify that this project report entitled “**DEVELOPMENT OF A WEARABLE GLOVE FOR VIRTUAL REALITY APPLICATION**” was prepared by **LOW ZE XIAN** has met the required standard for submission in partial fulfillment of the requirements for the award of Bachelor of Mechatronics Engineering with Honours at Universiti Tunku Abdul Rahman.

Approved by,

Signature : 

Supervisor : Chee Pei Song

Date : 17/05/2023

Signature : 

Co-Supervisor : Tan Lee Fan

Date : 19/05/2023

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2023, LOW ZE XIAN. All right reserved.

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Dr. Chee Pei Song for his invaluable advice, guidance and his enormous patience throughout the development of the research.

In addition, I would also like to express my gratitude to my loving seniors who had helped and given me advice throughout this project.

ABSTRACT

In recent years, as entered the era of 5G and the Internet-of-things (IoT), significant developments have been made in human-machine interfaces (HMIs) that enable more intuitive interactions between humans and the digital world. However, the current glove-based HMI solutions available in the market have limitations due to non-conformal sensor integration, which constrains finger movement. To address this issue, this project proposes a haptic-feedback glove-based HMI with a graphene thread sensor, ESP-NOW for wireless connection, a haptic actuator, and MPU6050 accelerometers. The graphene thread exhibits a sensitivity of $700 \Omega/^\circ$ and the graphene thread sensors show no deterioration for 100 cycles. Furthermore, in this project, the detection of multidirectional bending events in virtual space using the piezoresistive signals for various degrees of freedom on the human hand has been demonstrated. This project also perform haptic mechanical stimulation via motor vibration to realize the augmented HMI. Through the integrated demonstration of multidimensional manipulation and haptic feedback, our glove shows its potential as a promising solution for advanced human-machine interaction. It can benefit diversified areas, including entertainment, home healthcare, sports training, and the medical industry.

TABLE OF CONTENTS

DECLARATION		i
APPROVAL FOR SUBMISSION		ii
ACKNOWLEDGEMENTS		iv
ABSTRACT		v
TABLE OF CONTENTS		vi
LIST OF TABLES		ix
LIST OF FIGURES		x
LIST OF SYMBOLS / ABBREVIATIONS		xiv
LIST OF APPENDICES		xv
CHAPTER		
1	INTRODUCTION	1
1.1	General Introduction	1
1.2	Importance of the Study	1
1.3	Problem Statement	2
1.4	Aim and Objectives	2
1.5	Scope and Limitation of the Study	3
2	LITERATURE REVIEW	4
2.1	Introduction	4
2.2	Data Glove Introduction	4
2.3	Wearable Sensor	6
2.3.1	Piezo-electric Sensor	6
2.3.2	Half-Effect Sensor	6
2.3.3	Piezoresistive Sensor	7
2.3.4	Optical Sensor	8
2.3.5	Capacitive Bend Sensor	9
2.3.6	Triboelectric Sensor	10
2.4	Virtual Reality	11
2.4.1	Level of Immersion	11

	2.4.2 VR Technology	12
	2.4.3 Application of VR Combined with Glove System	12
2.5	Tools to Construct VR Environment and Model	15
	2.5.1 Blender	15
	2.5.2 Unity Editor	17
2.6	3D Tracker	17
2.7	Haptic Technology	19
	2.7.1 Introduction	19
	2.7.2 Vibration Haptic Technology	20
3	METHODOLOGY AND WORK PLAN	25
	3.1 Introduction	25
	3.2 Fabrication of Data Glove	26
	3.3 Characterisation and Calibration of Data Glove	28
	3.3.1 Data Collection from Finger Sensor	30
	3.4 MPU-6050 Module	33
	3.4.1 Data Collection from MPU6050	34
	3.5 Haptic Actuator	36
	3.5.1 Haptic Actuator Signal	37
	3.6 Setup in Arduino IDE	37
	3.6.1 Data Transmission through Wireless and Serial Communication	38
	3.7 Power Source	41
	3.8 Overall Hardware Design	42
	3.9 Build Up a Virtual Environment and Hand Model	44
	3.9.1 Blender	44
	3.9.2 Unity Editor	48
	3.10 VR Game Design	48
	3.10.1 Grabbing and Colliding Motion	48
	3.10.2 Game Scene	50
4	RESULTS AND DISCUSSION	53
	4.1 Introduction	53
	4.2 Graphene Thread Sensor Result	53
	4.3 Prototype Soldering and Combining Result	59

4.4	VR Game Environment	63
4.4.1	Main Menu Scene	63
4.4.2	Game Selection Scene	63
4.4.3	Difficulty Selection Scene	64
4.4.4	In-Game Scene	65
4.5	Serial and Wireless Communication Result	67
4.6	Summary	75
5	CONCLUSIONS AND RECOMMENDATIONS	76
5.1	Conclusions	76
5.2	Recommendations for future work	76
	REFERENCES	78
	APPENDICES	83

LIST OF TABLES

Table 2.1: Types of Data Glove with Respective Characteristics (Caeiro-Rodríguez, <i>et al.</i> , 2021)	4
Table 2.2: Summary of Features of Blender (Blender, n.d.)	16
Table 3.1: MPU6050 Output Parameters	35
Table 3.2: Hand Model	45
Table 3.3: Armature	46
Table 4.1: Equation to Calculate Angle Data	59

LIST OF FIGURES

Figure 2.1: Types of Data Glove (Caeiro-Rodríguez, <i>et al.</i> , 2021)	5
Figure 2.2: Piezo-electric Sensor (Variomh Eurosenosr, 2019)	6
Figure 2.3: Hall Effect Sensor (Electronics Tutorials, n.d.)	7
Figure 2.4: Structure of Piezoresistive Sensor (Jonathan, <i>et al.</i> , 2016)	8
Figure 2.5: Optical Sensor (David, 2020)	9
Figure 2.6: Structure of Capacitive Bend Sensor (Alapati <i>et al.</i> , 2017)	10
Figure 2.7: Working Principle of Triboelectric Sensor (Kim <i>et al.</i> , 2020)	11
Figure 2.8: Goniometer (IndiaMart, n.d.)	14
Figure 2.9: Blender Logo (Blender, n.d.)	15
Figure 2.10: Unity Logo (Unity, 2023)	17
Figure 2.11: Illustration of Force Feedback Haptic (Christian <i>et al.</i> , 2019)	19
Figure 2.12: Illustration of Air Vortex Ring (Kim, 2013)	20
Figure 2.13: Illustration of Ultrasonic Haptic Technology (Lisa, 2017)	20
Figure 2.14: Structure of ERM (Jason <i>et al.</i> , 2018)	21
Figure 2.15: Exploded View of LRA (Haptics, 2013)	22
Figure 2.16: Design of Piezo Haptic Actuator (TDK, n.d.)	23
Figure 2.17: Illustration of Vibration Motion of Piezo Haptic Actuator (TDK, n.d.)	24
Figure 3.1: Flowchart of Work Plan	26
Figure 3.2: Graphene Thread Sensor	26
Figure 3.3: Illustration of Graphene Thread Processing Steps	27
Figure 3.4: Graphene Conductive Ink	27
Figure 3.5: Ultrasonic Bath	28
Figure 3.6: Vacuum Oven	28

Figure 3.7: Resultant Graphene Thread	28
Figure 3.8: Hardware Structure of DOIT ESP32 DEV KIT V1	29
Figure 3.9: Connection of Voltage Divider Circuit to ESP32	30
Figure 3.10: Voltage Divider	31
Figure 3.11: Analog Reading Vs Bending Angle	32
Figure 3.12: Flowchart of Data Collection of Graphene Thread Sensor	32
Figure 3.13: MPU-6050 Module	33
Figure 3.14: Connection of MPU6050 to ESP32	33
Figure 3.15: MPU6050 Process Flow	35
Figure 3.16: Haptic Actuator	36
Figure 3.17: Circuit Connection of DC Motor Vibration to ESP32	36
Figure 3.18: Flowchart of Haptic Actuator Process	37
Figure 3.19: Arduino Logo	37
Figure 3.20: Data Transmission Process	39
Figure 3.21: MAC Address Displayed in Serial Monitor	40
Figure 3.22: Process Flow of Wireless Communication	41
Figure 3.23: Process Flow Involving in Second ESP32	41
Figure 3.24: LiPo Battery	42
Figure 3.25: Overall Hardware Connection for First ESP32	42
Figure 3.26: Overall Process Flow of First ESP32	43
Figure 3.27: Overall Process Flow of Second ESP32	44
Figure 3.28: Blender Logo	44
Figure 3.29: Grabbing Motion of Hand Model	47
Figure 3.30: Capsule Collider on Game Object	49
Figure 3.31: Capsule Collider on Hand Model	49

Figure 3.32: Balloon Design	51
Figure 3.33: Overall Game Logic	52
Figure 4.1: Graphene Thread Sensor on Glove	53
Figure 4.2: Resistance Change of Graphene Thread Sensor on Index	54
Figure 4.3: Resistance Change of Graphene Thread Sensor On Middle	54
Figure 4.4: Resistance Change of Graphene Thread Sensor On Thumb	55
Figure 4.5: Resistance Change of Graphene Thread Sensor on Pinky	55
Figure 4.6: Resistance Change of Graphene Thread Sensor on Ring	56
Figure 4.7: Normalized Data of Sensors	57
Figure 4.8: Resistance Change under Bending and Releasing of Finger	58
Figure 4.9: Top View of Combined Prototype	59
Figure 4.10: Bottom View of Combined Prototype	60
Figure 4.11: MPU6050 Soldered on Board	60
Figure 4.12: Top View of Prototype with Cover	61
Figure 4.13: Bottom View of Cover	61
Figure 4.14: Cover with LiPo Battery	62
Figure 4.15: Prototype on User Hand	62
Figure 4.16: Main Menu Scene	63
Figure 4.17: Game Selection Scene	63
Figure 4.18: Difficulty Selection Scene	64
Figure 4.19: In-Game Preparation Time	65
Figure 4.20: In-Game Scene	65
Figure 4.21: Boom Balloon Effect	66
Figure 4.22: Normal Balloon Effect	66
Figure 4.23: Ending Menu	67

Figure 4.24(a): Finger Motion Control via Communications	68
Figure 4.24(b): Finger Motion Control via Communications	69
Figure 4.24(c): Finger Motion Control via Communications	69
Figure 4.24(d): Finger Motion Control via Communications	70
Figure 4.25(a): Hand Model Control via Communications	70
Figure 4.25(b): Hand Model Control via Communications	71
Figure 4.25(c): Hand Model Control via Communications	71
Figure 4.25(d): Hand Model Control via Communications	72
Figure 4.26: In-Game Condition 1	73
Figure 4.27: In-Game Condition 2	74

LIST OF SYMBOLS / ABBREVIATIONS

c_p	specific heat capacity, J/(kg·K)
h	height, m
K_d	discharge coefficient
M	mass flow rate, kg/s
P	pressure, kPa
P_b	back pressure, kPa
R	mass flow rate ratio
T	temperature, K
v	specific volume, m ³
α	homogeneous void fraction
η	pressure ratio
ρ	density, kg/m ³
ω	compressible flow parameter
ID	inner diameter, m
MAP	maximum allowable pressure, kPa
MAWP	maximum allowable working pressure, kPa
OD	outer diameter, m
RV	relief valve

LIST OF APPENDICES

Appendix A: Overall Arduino Code of First ESP32	83
Appendix B: Overall Arduino Code of Second ESP32	90
Appendix C: Code to Control Finger Bending and Releasing motion, Quaternion and Acceleration of Hand Model and Haptic Actuator Signal	92
Appendix D: Exit Button Code	97
Appendix E: Play Button Code/ Scene Switcher	98
Appendix F: Code to Ensure Hand Model Not Destroyed	99
Appendix G: Back Button Code	100
Appendix H: Difficulty Level Selection Code	101
Appendix I: Code to Spawn Balloon	102
Appendix J: Code to Define the Balloon Moving Speed	103
Appendix K: Code to Restrict Movement of Hand Model in Frame	104
Appendix L: Timer Code	105
Appendix M: Score Manager Code	107
Appendix N: Code for In Game Preparation Count Down	109
Appendix O: Destroy Normal Balloon Code	110
Appendix P: Destroy Boom Balloon Code	111
Appendix Q: Ending Menu Code	112
Appendix R: ESP32 Holder	113
Appendix S: MPU6050 Holder	114

CHAPTER 1

INTRODUCTION

1.1 General Introduction

In the fourth industrial revolution, the Internet of Things (IoT) emerged as a major exploration, development, and implementation trend. IoT is an advanced system comprising elements such as computing devices, digital machines, mechanical devices, objects, and people, each with unique identities (UIDs). With IoT, the elements and components in a system can send data across a network without human involvement (Alexander, n.d.). Due to IoT, numerous wearable sensors have been designed and developed, which can be used in various industrial areas such as gaming, fitness, entertainment, security, defense, and healthcare. These wearable sensors can also be incorporated into human-machine interfaces (HMI) to monitor and collect essential data for future processes.

For hand rehabilitation purposes, a data glove, also known as a cyberglove or wired glove, has been designed and developed. The leading players in the data glove market include Nansense, ProGlove, HaptX, Manus, AiQ Synertial, IBM, Gest, Virtual Motion Labs, and others. Meanwhile, the leading commercialized data gloves in the market include 5DT, CyberGlove III, X-IST, DG5 VHand 2.0, NuGlove, GoGlove, Manus VR, Dexmo VR, and others (Manual, 2021).

Due to the state-of-the-art technology and lack of universal standardization, data gloves have a slightly high price ranging from a few hundred to a few thousand dollars. Therefore, in this project, a self-made data glove will be fabricated and combined with elements such as VR environments and haptic actuators through microprocessors and wireless communication to provide users with a better experience in hand rehabilitation.

1.2 Importance of the Study

The importance of this project is to create a wearable glove for virtual reality applications that can benefit patients who require rehabilitation, such as stroke patients. Traditionally, stroke patients must always travel to the hospital for

rehabilitation, and their family members may need to sacrifice more time to accompany and transport them. Additionally, the high demand for physiotherapists can result in high costs for hiring them to assist in the rehabilitation process.

To overcome this problem, the development of a wearable glove for VR applications enables patients to undergo the rehabilitation process at home. This can help them save costs, save their family members' time, and simultaneously improve the patient's psychological and emotional state.

1.3 Problem Statement

According to the New Straits Times (2022), in less than 20 years, there will be three senior citizens aged 65 and above for every 20 Malaysians, and this number is projected to keep increasing. By 2040, the number of senior citizens is expected to triple from today's two million to over six million. The aging population comes with the prevalent health issues of the elderly, such as hypertension, high cholesterol, diabetes, and others, which can lead to an increased risk of stroke. After the stroke, patients usually experience a functional barrier in the cerebral cortex, which means they require ongoing rehabilitation training to avoid physical function degeneration (Hwang *et al.*, 2012). This process can take one or more years to show results, resulting in high demand for clinicians and therapists. Moreover, equipment such as the FEI 13-1168 Kinetec Maestra CPM and MediTouch BalanceTutor, used during the rehabilitation process, is expensive and has a large scale. This can pose a problem for small hospitals that may not have enough space for such equipment. Therefore, researchers are constantly seeking ways to balance the effectiveness, simplicity, and convenience of the rehabilitation process.

1.4 Aim and Objectives

The main objectives of this project are to design and develop a wearable glove integrated with haptic feedback and a VR environment that will enable users to engage in games while undergoing rehabilitation. Specifically, the project aims to:

- i. To design and develop a wearable glove with haptic actuators.
- ii. To integrate the glove to provide feedback to users.

- iii. To apply the developed glove in a virtual environment.

1.5 Scope and Limitation of the Study

This project will review various methods for creating a wearable glove and VR environment, including analyzing and comparing different types of wearable sensors and haptic actuators. The necessary hardware and software, such as the DOIT ESP32 DEV KIT V1, MPU-6050 Module, Arduino IDE, Blender, and Unity Editor, will also be well-studied to ensure the success of the project.

However, one limitation of the proposed system is that the user cannot feel the structure and shape of virtual objects in the virtual environment. Additionally, the VR experience will be non-immersive, providing less visual and sound immersion for the user.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In this literature review, existing data glove types, wearable sensors, virtual reality, tools in constructing VR environments and hand models, data glove accessories, and haptic technology will be reviewed.

2.2 Data Glove Introduction

A well-characterized and systemized data glove facilitates tactile sensing and fine-motion control in robotics and virtual environments. Data gloves are considered one of several electromechanical devices used in haptic applications (TechTarget, 2016). They are comfortable to wear, environmentally friendly, and do not restrict the user's hand movements. The initial concept for a hand-based device was proposed more than four decades ago in 1978 (Sabatini, *et al.*, 2008). Since then, researchers have explored different methods and approaches to utilizing data gloves in various industries. Data gloves can be categorized into several types, which are summarised in Table 2.1 and Figure 2.1, along with their respective characteristics.

Table 2.1: Types of Data Glove with Respective Characteristics (Caeiro-Rodríguez, *et al.*, 2021)

Glove Types	Characteristics
Exoskeleton	Some strings or rigid links are attached to the finger, which provide kinaesthetic feedback.
Fabric	Fabric covers the hand and fingers in its entirety. Several sensors and actuators are embedded throughout the fabric to enable the desired capabilities.
Strips of fabric, plastic or other materials.	This glove doesn't cover the whole fingers and hand but just the sensor

	and actuator's location. This type of glove may accommodate a variety of hand and finger shapes and sizes.
Open fingertips	This glove doesn't cover the fingertips to use in the application such that the sense of touch is important.



Figure 2.1: Types of Data Glove (Caeiro-Rodríguez, *et al.*, 2021)

The sensors used in the data glove can be further classified into several classes, such as accelerometer type, flex-sensor type, hall-effect sensor type, magnetic sensor type, and stretch-sensor type (Hasan, *et al.*, 2019)

2.3 Wearable Sensor

2.3.1 Piezo-electric Sensor

The piezoelectric sensor, shown in Figure 2.2, is suitable for developing data gloves due to its electromechanical properties. Piezoelectric ceramics such as PZT ceramics and single-crystal materials like quartz are the two main sensing materials used for piezoelectric sensors. The working principle of piezoelectric sensors is based on the piezoelectric effect. When a force is applied to a piezoelectric material, it generates an electric charge proportional to the applied force across the material. This effect was first discovered by Pierre Curie and Jacques Curie in 1880, while the converse effect was mathematically proven by Lippman in 1881 and confirmed by the Curie brothers in 1882 (Manbachi & Cobbold, 2011). Typically, a charge amplifier is combined with the piezoelectric sensor to convert the charge signal into voltage. Piezoelectric sensors have several advantages, such as not requiring an external power supply to output a signal, having a wide frequency bandwidth, a high signal-to-noise ratio, and a simple structure. However, they also exhibit some limitations. For example, when pressure and temperature increase, sensitivity may be reduced due to twin formation, and they may crack if overstressed (Basel, 2021).



Figure 2.2: Piezo-electric Sensor (Variotm Eurosenosr, 2019)

2.3.2 Half-Effect Sensor

The hall-effect sensor is a magnetic sensor, and the material usually used to fabricate the sensor are semi-conductors such as Indium Antimonide (InSb), Gallium Arsenide (GaAs), and Indium arsenide (InAs). The sensor's operating concept is based on the hall effect, which Edwin Hall discovered in the 1870s. Figure 2.3 shows when a hall element experiences a magnetic field, the magnetic flux produced will cause the particles of the element, such as charge carriers, electrons, and holes, to deflect to either side of the semiconductor slab.

Then, a certain amount of Hall Voltage, V_H will be output when the magnetic flux density surrounding the sensor reaches a threshold value. The amount of this Hall voltage is directly proportional to the strength of the magnetic field passed through the semiconductor materials. Nevertheless, the Hall Voltage produced is normally very small such that only a few microvolts. Thus, the Hall-effect sensor is normally manufactured with a built-in DC amplifier to improve the sensor's performance, such as sensitivity, hysteresis, and output voltage.

The advantages of a Hall-effect sensor are that it has a robust structure, can be miniaturized for surface mount applications, offers fast response with no contact bounce, is extremely durable, and its performance is not significantly affected by dust or dirt, unlike optical sensors. Additionally, it also has some drawbacks, such as requiring a magnet for operation, limited output voltage, and vulnerability to magnetic fields.

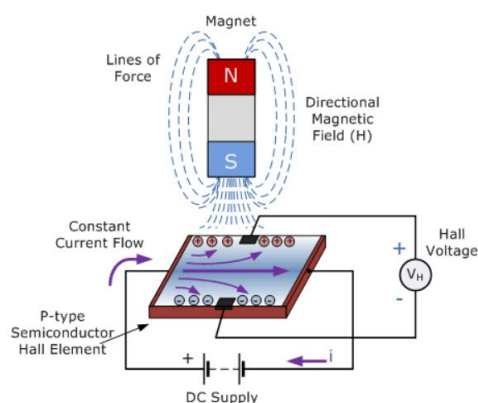


Figure 2.3: Hall Effect Sensor (Electronics Tutorials, n.d.)

2.3.3 Piezoresistive Sensor

The popular materials for piezoresistive sensors include graphene, carbonized melamine, polypyrrole, silicon, and germanium. The working principle of a piezoresistive sensor is based on the piezoresistive effect, which was discovered by Lord Kelvin in 1856 (Sir William Thomson, Belfast, 1824–1907). The development of piezoresistive sensors progressed rapidly since the discovery of the effect in Silicon (Si) and Germanium (Ge) in 1954. The piezoresistive effect causes a change in electrical resistance when an external force is applied to a

semiconductor. This alteration solely impacts the electrical resistivity of the material. Unlike the piezoelectric effect, this phenomenon cannot be exploited to create a voltage across a device. Figure 2.4 shows the structure of piezoresistive sensor. When an applied force alters the material's band structure, it makes it easier for electrons to be stimulated into the conduction band. As a result, the density of current carriers varies, as does the material's resistance.

The advantages of the piezoresistive sensor include stable electrical and mechanical characteristics, high-resolution measurement, strong resistance to shock, vibration, and dynamic pressure change, and lower cost compared to other sensors. However, the sensor's output is reliant on temperature change. The resistance decreases, and power consumption increases as its size is reduced.

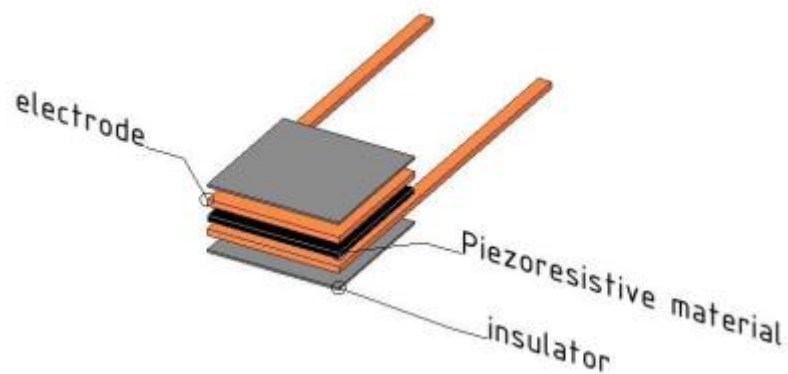


Figure 2.4: Structure of Piezoresistive Sensor (Jonathan, *et al.*, 2016)

2.3.4 Optical Sensor

An optical sensor, shown in Figure 2.5, consists of a solid-state light source, a photodetector, and an optical fiber. The working principle of the optical fiber is to measure the strength of the light source. The light source, such as an LED, is detected by the photodetector and converted into an electrical signal. It can be used as a wearable sensor in a data glove because hand motions such as bending and twisting can cause some paths of light transmission in the optical fiber to be blocked, reducing the power generated. To ensure wearability, the fibers have been made of flexible and stretchable elastomers such as a polyurethane rubber core with silicone composite cladding (Zhao *et al.*, 2016) or a polydimethylsiloxane core with gold nanoparticles and a cured

polydimethylsiloxane with a lower refractive index as cladding (Guo *et al.*, 2019).

The advantages of an optical sensor are that the materials used in optical sensors are chemically inert, allowing them to be applied in highly reactive environments. Additionally, they are less sensitive to electromagnetic waves and have high linearity of performance. However, optical sensors also have some disadvantages, such as slower response times and sensitivity to ambient light.

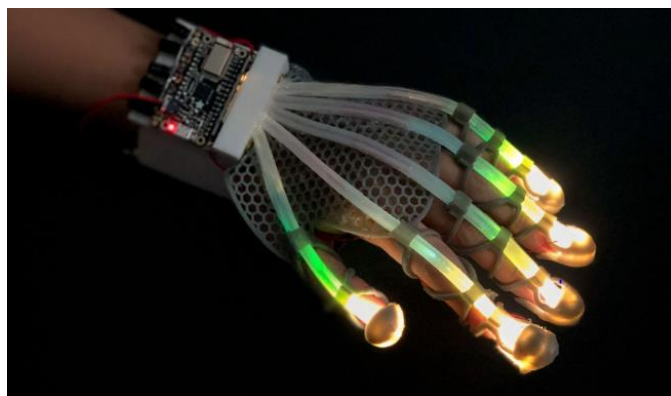


Figure 2.5: Optical Sensor (David, 2020)

2.3.5 Capacitive Bend Sensor

A capacitive sensor comprises two layers of comb-shaped conductive polymer separated by a dielectric, as shown in Figure 2.6. The degree of bending of the sensor causes a change in the overlapped electrode surface, resulting in a change in capacitance. Increasing the deformability of the dielectric layer is crucial to enhancing the sensor's sensitivity. Elastomeric materials with high compressibility and low Young's modulus can build up the dielectric layer, substantially increasing the sensor's sensitivity while eliminating the hysteresis effect caused by viscoelastic behavior.

Capacitive bend sensors have advantages such as high sensitivity, less sensitivity to the surrounding temperature, less sensitivity to drift, and lower energy consumption. According to Anderson (2019), capacitive sensors exhibit linearity of change in capacitance when subjected to strain, but only up to a certain strain point (Amjadi *et al.*, 2016). Capacitive sensors consume less

power as no real power is consumed (DeHennis *et al.*, 2016). However, they are sensitive to temperature due to the thermo-mechanical strain caused by the difference in thermal expansion between silicon and Pyrex (Blasquez *et al.*, 2000; Beddiaf *et al.*, 2016).

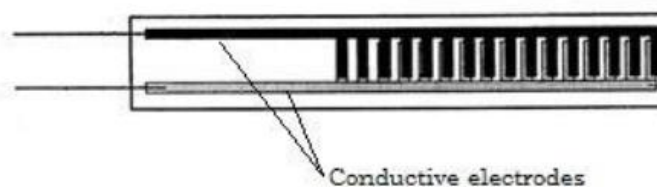


Figure 2.6: Structure of Capacitive Bend Sensor (Alapati *et al.*, 2017)

2.3.6 Triboelectric Sensor

The triboelectric nanogenerator has many advantages, such as being self-powered, inexpensive, high output range, and stable and lightweight. These advantages make it a preferred choice of researchers for adapting it to industry applications (Chen *et al.*, 2020). The TENG was first built for energy harvesting in 2012 due to its ability to convert mechanical force to electricity.

The working principle of the triboelectric sensor is based on the triboelectric effect (shown in Figure 2.7). The dielectric surfaces, such as EcoFlex - human skin (An *et al.*, 2020) or PET - Kapton (Wang *et al.*, 2018), come into contact and separate to produce a static electrical charge. The amount of charge produced is dependent on the relative displacement of the materials in the triboelectric series, where materials located farther apart from each other result in more charge being transferred between materials (Fan *et al.*, 2012).

The triboelectric sensor has several advantages, such as not requiring an external power source due to its self-powering mechanism. Additionally, it is lightweight, flexible, and has a simple construction. However, the triboelectric sensor also exhibits some drawbacks, such as being sensitive to temperature due to the change in mechanical properties of materials at various temperatures, which affects the effective friction between the materials (Wen *et al.*, 2014). It is also sensitive to humidity, with more electrical charge produced in lower humidity (Nguyen & Yang, 2013).

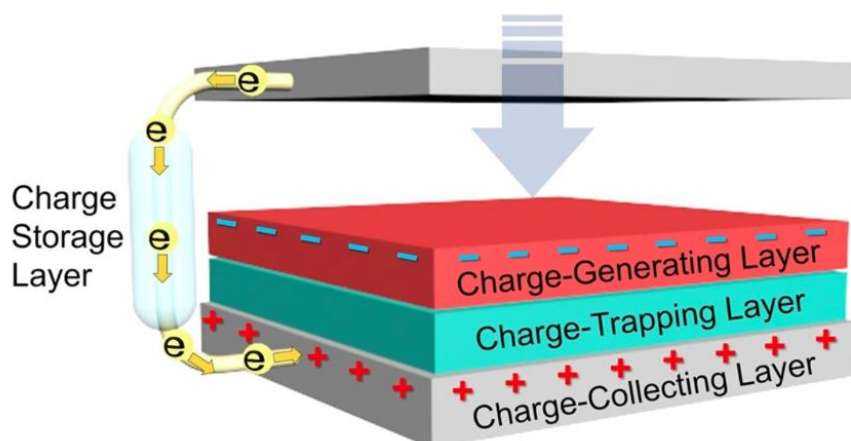


Figure 2.7: Working Principle of Triboelectric Sensor (Kim *et al.*, 2020)

2.4 Virtual Reality

Virtual Reality (VR) is an advanced system that comprises computer modeling and simulation to create a virtual environment that can interact with humans. Various devices, such as VR goggles, headsets, gloves, and body suits, receive input from users and simulate human senses to make VR more immersive. VR has gained the attention of investors and the general public over the past five years, especially since Mark Zuckerberg's \$2 billion acquisition of Oculus in 2014 (Castelvecchi, 2016). Major players in VR include HTC Vive, Oculus Rift, and PlayStation VR (PSVR). Initially, VR was used in computer graphics, but technological advancements have implemented it in many other industries and fields (Choi *et al.*, 2015). For example, Moser evaluated the usage of VR, emphasizing its significance in clinical treatment and research (Minderer *et al.*, 2016). Although the definition of VR may vary from person to person and from different perspectives, they all share three characteristics: immersion, the impression of being present in a world, and interactivity with that environment (Sundar *et al.*, 2010).

2.4.1 Level of Immersion

VR immersion is defined as the degree of involvement of a user in the virtual environment. A highly immersive VR system provides a lot of stimuli to simulate the user's senses such as sound, smell, touch, and others. There are three levels of immersion systems: non-immersive, semi-immersive, and

immersive. A non-immersive system is the simplest and least expensive system with the fewest stimuli. Normally, a non-immersive system uses a desktop to reproduce the image. For semi-immersive, the system starts to use more advanced gadgets such as stereographic glasses to display some stereo images of a virtual environment, such as Fish Tank VR. For immersive systems, it involves more devices to provide more stimuli to human senses, for example, VR goggles stimulate the user's vision, VR earphones provide sound stimulation, and haptic actuators stimulate the user's touch sense. A high-level immersion system can create the illusion of technological non-mediation and a sensation of 'being-in' or being present in the virtual environment for the user.

2.4.2 VR Technology

The devices used in VR can be separated into two parts: input and output (Burdea *et al.*, 2003). Input devices, such as trackers, capture motion input from the user and display the corresponding motion in the virtual environment. For example, bend-sensing gloves capture the finger motion of users and display it in the virtual environment. VR motion platforms capture the walking motion of the user for video games.

Output devices allow the user to sense everything in the virtual environment through sight, sound, taste, touch, and smell. There is a vast selection of output devices, ranging from the least immersive, such as a computer monitor, to the most immersive, such as VR goggles, helmets, HMD, CAVE systems, VR earphones, haptic actuators, and others.

2.4.3 Application of VR Combined with Glove System

Nowadays, many VR applications exist in different industries, such as military training, gaming, education, architectural design, simulation of surgeons, and others (Freeman *et al.*, 2017). There are numerous ways that virtual reality technology can be used as a stimulus, taking the place of actual stimuli and accurately duplicating experiences that are hard to achieve in the real world. For this reason, virtual reality (VR) is commonly used in research on innovative psychological training or therapeutic techniques, such as for phobia-related difficulties (Botella *et al.*, 2017). It can also improve the current rehabilitation methods (Borrego *et al.*, 2016) by creating games that ease the tasks. The

application of VR combined with glove system can be further classified into two groups: classical application and recent application.

2.4.3.1 Classical Application

Design and manufacturing is one of the classic applications of VR. With VR, manufacturers and designers can visualize their products in a virtual environment before the construction or manufacturing process, thus eliminating costly mock-ups. Users can view 3-D graphical models built in the virtual environment and perform design testing, improving, and changing processes without altering the real-world product beforehand. For example, Daimler-Benz developed a VR system for product testing and design, allowing users to select different options and models for furnishing the interior of Mercedes using the Data Glove. Another example is Boeing, an aircraft manufacturer, which carries out design and maintenance tasks for the military aircraft Joint Strike Fighter with Cyberglove by 'walking' around the virtual aircraft.

Art and entertainment are also classic applications of VR. The entertainment industry has combined with VR for a long time. Gloves have been utilized in video games, computer-animated characters (Muses, 2002), and movie production (Minoh, 2007). A glove system enables applications that require control of many degrees of freedom, such as robotic and musical performance (Singh, 2002). An example of musical entertainment using VR is Sound Sculpting, prototyped by Mulder and Fels. There are also games that promote exercise, such as controlling a 3-D virtual basketball while wearing two CyberGloves. Hand motions can be used to change the position and orientation of the virtual basketball, which links to changes in acoustic parameters.

2.4.3.2 Recent Application

Medicine and healthcare are the most popular recent applications of data gloves in combination with VR. They have been used for motor rehabilitation, motor performance analysis, ergonomics, medical education, and training. In the area of motor rehabilitation, researchers are exploring the possibility of using data gloves as a tool for hand-functional assessment. Hand functional assessment requires a wide range of data, such as hand strength and, most importantly, the range of motion of hand joints, to make further decisions. Formerly, mechanical

or electronic goniometers measured these parameters manually, as shown in Figure 2.8.

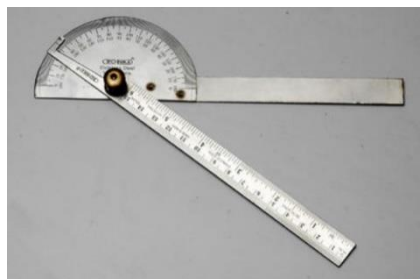


Figure 2.8: Goniometer (IndiaMart, n.d.)

This method has some disadvantages. Even when conducted by a skilled therapist, the procedure is time-consuming and restricted in its precision (Hellebrandt *et al.*, 1949). Using the data glove to undergo hand assessment can ultimately resolve these problems. The glove system can automatically and simultaneously measure the range of motion of each joint and finger with specific commands and tasks, thus reducing the time consumed and increasing accuracy. Some groups have used sensorized gloves to build a more complex quantitative system for hand assessment and rehabilitation (Thalmann *et al.*, 2007). For example, Greenleaf Medical System has created the Wrist System for quantitative evaluation of upper extremity function. The system measured wrist flexion, extension, radial, and ulnar motion using a data glove with a custom-fitted sensor (Greenleaf, 1996). In addition to the Greenleaf Medical System, Burdea and colleagues also developed a workstation consisting of two subsystems: a diagnosis subsystem and a therapy subsystem for hand motion-disabled patients. The diagnosis subsystem records the patient's hand motion with a glove system and transmits it to the computer to assess the patient's hand capability and plan therapy accordingly. In the therapy subsystem, Burdea has implemented Rutgers Master I haptic glove, which can provide force feedback to the user (Burdea *et al.*, 1992).

In the field of ergonomics, data gloves are utilized to enhance the design of goods, tasks, and settings in order to adhere to ergonomic principles. For instance, Protomic of the Netherlands utilized CyberGlove to gather data to fine-tune the design of their DataStealth ergonomic keyboard, which aims to

increase user comfort and decrease the danger of repetitive strain injury. Using CyberGlove, they recorded the motion of the user's fingers while they typed on the DataStealth and other conventional keyboards. The data collected was then used to evaluate a biomechanical model from the perspective of the impact of stresses and strains on the soft tissues of the hand to analyze which design minimized the users' effort the most.

In the area of medical education and training, VR and data gloves have been used to simulate rare cases or surgeries for medical training. This brings along some advantages, such as cost-reducing and also making mistakes on a virtual organ instead of the real human organ. Delp and Rosen, for instance, built a tendon transplant simulator that allows inexperienced surgeons to explore and become familiar with the structure of the lower limb skeleton through VR goggles and data gloves. Surgeons could analyze the effect of changing the insertion locations of tendons to plan a procedure to repair individuals with gait disorders, and the gait could be predicted with the simulator (Delp *et al.*, 1992).

2.5 Tools to Construct VR Environment and Model

2.5.1 Blender



Figure 2.9: Blender Logo (Blender, n.d.)

Blender (shown in Figure 2.9) is an open-source 3D computer graphics toolkit developed by the Dutch non-profit Blender Foundation. Blender is generally used to make animated movies, cinematography, artwork, 3D-printed models, demo reel, interactive 3D applications, virtual reality and video games. Blender contains several functions with varied functions. Table 2.2 summarizes all the features with their corresponding functions.

Table 2.2: Summary of Features of Blender (Blender, n.d.)

Feature	Function
Modelling	Advanced polygon modelling system accessible via edit mode, with supported features like extrusion, beveling, and subdividing. It supports several geometric primitives, including polygon mesh, metaballs, icospheres, text, and others.
Modifiers	Support rendering and exporting function, which can apply non-destructive effects.
Sculpting	Simplify models for exporting purposes, for example, exporting models as game assets.
Simulation	Simulate effects, including fire, rainfall, dirt, fabric, fluid, fur, and rigid bodies.
Animation	Allow users to manipulate elements such as inverse kinematics, armatures, hooks, curve- and lattice-based deformations, shape keys, non-linear animation, constraints, and vertex weighting.
Plugins and scripts	Allow integration with several external render engines via plugins and add-ons.
Import and Export	Allow users to export and import file in different formats such as Alembic, 3D Studio (3DS), FBX, DXF, SVG, STL, UDIM, USD, VRML, WebM, X3D, and OBJ

2.5.2 Unity Editor



Figure 2.10: Unity Logo (Unity, 2023)

Unity Editor (Figure 2.10) is a game engine that controls the motion of certain models by using scripts with languages such as C#, C, C++, Rust, Iron Python, and Lua. Scripting is an essential element in all games used to create visual effects, control motion, or even apply a self-trained AI system for characters in the virtual environment. Unity Editor provides graphic features such as lighting, cameras, shaders, particle systems, visual effects, mesh components, texture components, rendering components, and others. Unity Editor obeys physics and can be separated into 3D physics and 2D physics. The Unity Editor provides audio settings, including clips, sources, listeners, importing, and sound settings. The Unity Editor also comes equipped with a navigation system. The navigation system allows the creation of intelligently moving characters in the game world. The navigation system reasons about the surroundings using navigation meshes, which are generated automatically from Scene geometry. Dynamic barriers provide rerouting of characters in runtime, while off-mesh linkages enable the user to construct specialized actions, such as opening doors or falling off a cliff. In creating a game, models should be imported as assets and then utilized as GameObjects in scripts. Assets can be imported to the project by moving the model file into the "Assets" in the Finder. Then, users can simply drag the asset file from the Project View window into the Hierarchy or Scene View. Whenever an asset file is updated, it will automatically reflect in the game.

2.6 3D Tracker

A 3-D tracker is a device that tracks the motion of a data glove in 3 axes of translation and 3-axis of rotation in real-time and transforms it into the virtual reality (VR) environment. With the advancements in technology, various types

of trackers have been developed with different key performance parameters such as accuracy, jitter, drift, and latency. Initially, 3-D trackers were mainly mechanical, but these trackers were visually obtrusive and expensive. Therefore, non-contact trackers are now widely used to replace mechanical trackers. These non-contact trackers feature kinematic structures composed of sensorized linkages that are less burdensome and do not restrict the user's mobility. Some examples of non-contact trackers include magnetic-based, ultrasonic-based, optical-based, and inertial-based trackers.

A magnetic non-contact tracker employs a magnetic field generated by a stationary transmitter to identify the location of a moving receiver device. Magnetic trackers are cheap and have acceptable accuracy, and most importantly, they do not require a direct line of sight between the transmitter and receiver as long as the receiver is under the effect of the magnetic field. However, magnetic trackers are sensitive to ferromagnetic materials and magnetic fields and should always be kept away from metallic objects as they might affect their accuracy.

An ultrasonic non-contact tracker employs an ultrasonic signal generated by a stationary transmitter to pinpoint the location of a moving receiver. Unlike magnetic trackers, it is not affected by metallic interference, but it is susceptible to echoes from hard surfaces and requires a direct line of sight between the transmitter and receiver with no obstacles between them; otherwise, the signal might be lost. Additionally, it has a slightly slower updating rate than magnetic trackers.

An optical non-contact tracker employs optical sensing to identify the location or orientation of an object in real-time. Optical trackers, like ultrasonic trackers, require a direct line of sight between the transmitter and receiver and are not affected by metallic interference. However, optical trackers have a higher update rate and larger work envelopes than ultrasonic trackers. Nevertheless, optical trackers are sensitive to the reflected light, which might decrease their accuracy.

An inertial tracker is a type of tracker that is self-equipped with sensors that measure the rate of change of an object's orientation and translation velocity. It has advantages, including sourceless operation with theoretically unlimited range, no line-of-sight limits, and extremely low sensor distortion.

However, the output of inertial trackers must be integrated to derive an object's orientation and position.

2.7 Haptic Technology

2.7.1 Introduction

Haptic technology is defined as kinaesthetic communication or 3D touch that can generate force, vibration, or motion to the user through the sense of touch as part of an interface. Nowadays, many applications have already embedded haptic actuators to enhance the user's experience. For example, smartphones provide different strengths of vibration depending on the caller, vibration feedback when playing video games through a joystick, and tangible feedback when using a touchscreen. There are several types of haptic implementations, such as force feedback, air vortex ring, ultrasound, and vibration.

Force feedback haptic technology uses devices such as motors to simulate the sensation of holding objects in a virtual environment for the user. Figure 2.11 illustrates the implementation of force feedback haptic technology.

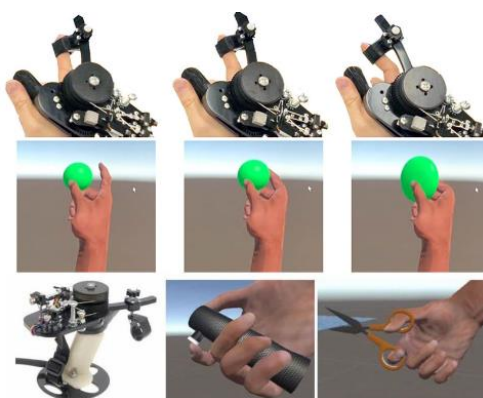


Figure 2.11: Illustration of Force Feedback Haptic (Christian *et al.*, 2019)

The Direct-drive wheels introduced in 2013 implemented the force feedback by stimulating force to turn the steering wheel when cornering a real vehicle by servomotor.

Air vortex rings are circular air pockets made up of concentrated gusts of air. The air created has proven can create stimuli and sense to humans such that it can blow out a candle. Both Microsoft Research (AirWave)(Gupta *et al.*, 2013) and Disney Research (AIREAL)(Sodhi *et al.*, 2013) have implemented

air vortex rings in their system to create non-contact haptic feedback for their user to better users' experience (Shtarbanov *et al.*, 2018). Figure 2.12 illustrates the implementation of air vortex rings.



Figure 2.12: Illustration of Air Vortex Ring (Kirn, 2013)

The ultrasound haptic technology enables users to have a focused pressure sense with any actual contact. The location of pressure feeling generated can be adjusted to our desired location or point by manipulating the strength and phase of an ultrasound transducer. Figure 2.13 illustrates the implementation of ultrasound haptic technology.



Figure 2.13: Illustration of Ultrasonic Haptic Technology (Lisa, 2017)

The ultrasound haptic also enables the user to feel the virtual 3D object (Benjamin *et al.*, 2014) through the sensation of vibration (Heather *et al.*, 2018) and also can use to gather information on surface texture.

2.7.2 Vibration Haptic Technology

Vibration haptic technology is the most popular used haptic in electronic applications. It offers vibration feedback for a better user experience. There are several vibration haptic actuators such as Eccentric Rotating Mass actuators

(ERM), Linear Resonant Actuators (LRA) and Piezo Haptic actuators. Each of them has a different structural design and operation but similarly, each provides vibration feedback to the user. Since the vibration haptic actuator is suitable for our project, the three vibration haptic actuators will be introduced and explained in the following section.

2.7.2.1 Eccentric Rotating Mass Actuator (ERM)

Eccentric Rotating Mass (ERM) actuators are miniature DC motors that spin an eccentric, unbalanced mass to produce the necessary vibration. The function of the ERM is realized by equipping a DC motor with a non-symmetric mass attached to the shaft. Therefore, when the motor rotates, a resulting net centrifugal force is generated due to the presence of the non-symmetric mass. When it rotates at a constantly high speed, the motor will be constantly displaced and moved by these asymmetric forces. When the process is repeated again and again, the vibration is created. The frequency of the vibration can be controlled by the DC voltage. This is because the rotational speed is directly proportional to the DC voltage, and the frequency. Figure 2.14 shows the structure of the ERM.

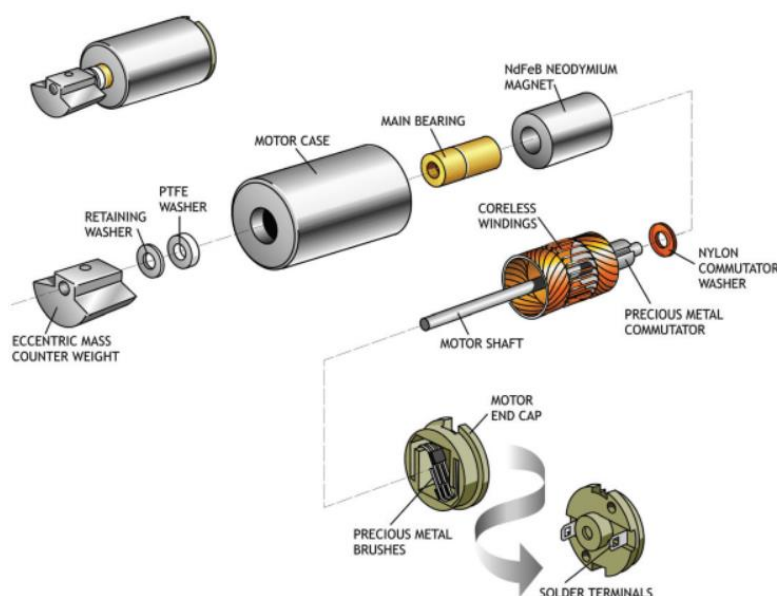


Figure 2.14: Structure of ERM (Jason *et al.*, 2018)

ERM's advantages include inexpensive cost, widespread availability, established technology, a wide variety of specifications, and easy usage and design. While it has several disadvantages, such as vibration signal amplitude being dependent on driving frequency and requiring much power, it has a delayed reaction when beginning and stopping.

2.7.2.2 Linear Resonant Actuator (LRA)

A Linear Resonant Actuator is an AC-driven actuator that functions based on the theory of a spring-mass system. The moving mass of the actuator moves back and forth under the action of a voice coil, producing a desired strength of vibration that is similar to that of a speaker. Figure 2.15 shows an exploded view of an LRA.

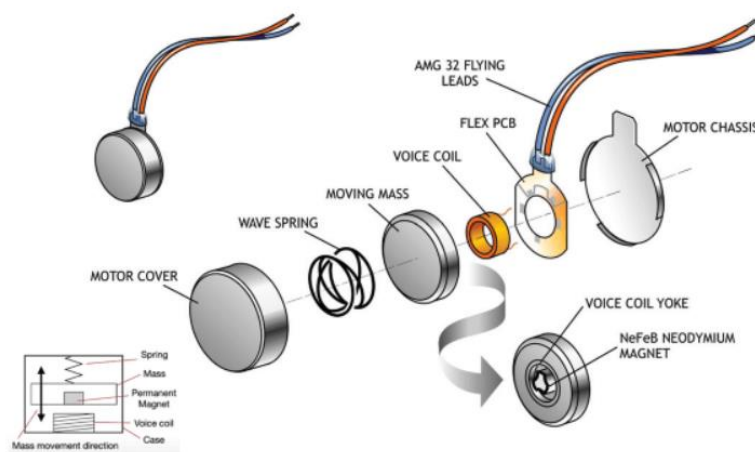


Figure 2.15: Exploded View of LRA (Haptics, 2013)

From the exploded view of the LRA, there are components such as a wave spring, moving mass, magnet, voice coil, and others. The function of the LRA, which creates the sensation of vibration, mainly depends on the four components mentioned above. The combination of the moving mass and the magnet acts as a part, followed by the wave spring and then the voice coil in sequence from outer to inner. As the AC drives the voice coil, it provides the motive force to the cone by the reaction between the magnetic field and the current passing through it, or the so-called electromagnetic force. Then, the moving mass and wave spring will be moved linearly under the effect of the electromagnetic force produced by the voice coil, creating a back-and-forth

displacement and, therefore, a vibration motion. The process is repeated, and the vibration reaches its maximum when the frequency of the AC supplied matches the resonant frequency of the spring.

LRA has advantages such as easy amplitude modulation, less power consumption compared to ERM, shorter response time, smaller size, more acceleration than the same size of ERM, reliability, and high Q. However, it also exhibits some drawbacks, such as vibrating only in one axis, different resonant frequencies between each LRA, and a narrow operating bandwidth, which causes lower vibration amplitude outside the resonant frequency.

2.7.2.3 Piezo Haptic Actuator

Piezo Haptic actuators work based on the piezoelectric effect, which generates a certain amount of electric charge in response to applied mechanical stress. The operation is reversible. Piezo Haptic actuators are usually made up of ceramics, crystals, bone, and other materials. Figure 2.16 shows some designs of piezo haptic actuators.

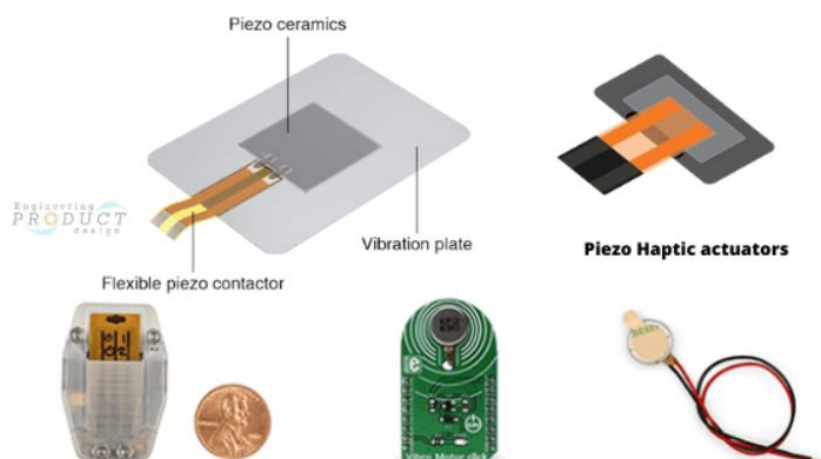


Figure 2.16: Design of Piezo Haptic Actuator (TDK, n.d.)

When an AC source is supplied to the actuator, the material expands and contracts, causing the element to stretch upwards and downwards, as shown in Figure 2.17, producing a vibrational motion.

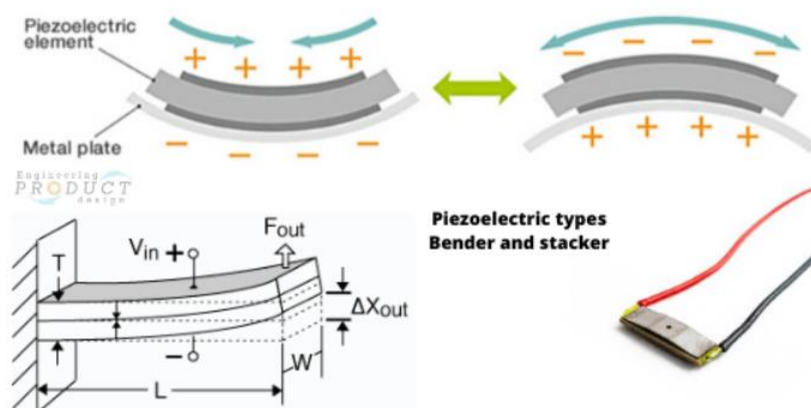


Figure 2.17: Illustration of Vibration Motion of Piezo Haptic Actuator (TDK, n.d.)

Piezo Haptic Actuators are more accurate than ERMs or LRAs because they are capable of vibrating at a wide range of frequencies, and the amplitude can be adjusted independently. The applications of piezo haptic actuators include sounders in audio systems, linear drives, autofocus systems, laser tuning, and others. The advantages of the piezo haptic actuator include being non-magnetic, not requiring holding power, having a quick response time, consuming low power, being smaller in size, and having high precision, among others. However, it also exhibits disadvantages, such as being non-linear and sensitive to electrical overdrive.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

The project begins with the fabrication of the graphene thread sensor, followed by the connection of the sensor with the microprocessor. The hardware components can be divided into three parts: two-way communication and electronic gadgets such as 2 MPU-6050 modules, 2 DOIT ESP32 DevKit V1, 5 DC motor vibrations, 5 graphene thread sensors, and a mini LiPo battery. The first part involves obtaining input from the graphene thread sensors and MPU-6050 modules attached to the glove. The graphene thread sensors track the user's finger bending motion, while the MPU-6050 modules track the 6 degrees of freedom (DOF) movement of the entire hand. Additionally, there will be some signal output from ESP32 in the first part to control the vibration of the haptic actuator.

The second part includes data processing using ESP32, and the third part involves outputting the processed data to the hand model and game in the virtual environment of Unity Editor. The process starts with the graphene sensors well-attached to the surface of the glove. Then, the terminal of the sensors on each finger and MPU-6050 modules is connected to the ESP32. The characterization process begins with serial communication to analyze the output response of each sensor under different bending angles to create a mathematical model equation to represent it. The equation is then inserted into the Arduino Script to convert the analog reading to an angle value. All the data, including finger and hand motion data, is then sent to the Unity game through wireless and serial communication to control the model motion to play games in the virtual environment.

In reverse, after some set conditions, a signal will be sent to ESP32 to control the vibration of the haptic actuator. To make the whole circuit neater, some soldering work is also done to combine every gadget on a PCB board. After that, packaging will be done to cover and equip the whole PCB board on the user's hand. Figure 3.1 shows the work plan of the project.

An interactive and interesting game is designed using Unity Editor and Blender software. Blender is used to export the hand model to Unity Editor. Then, C# script will be coded in the Unity Editor for game design. The user can play the game while doing the rehabilitation process.

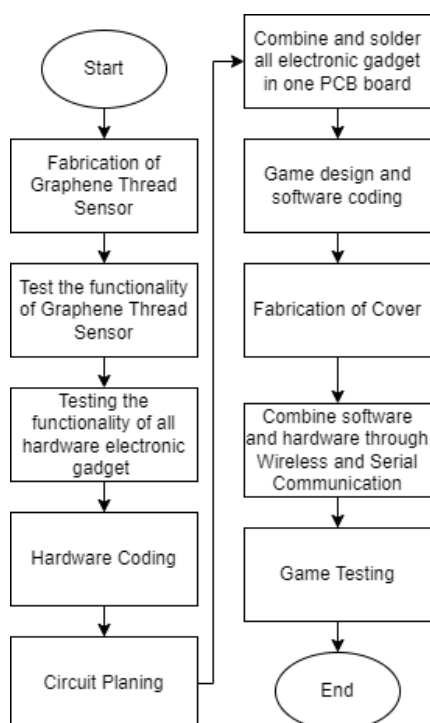


Figure 3.1: Flowchart of Work Plan

3.2 Fabrication of Data Glove

This project uses a graphene thread sensor as the main component that senses finger motion. Figure 3.2 shows the graphene thread sensor.



Figure 3.2: Graphene Thread Sensor

The graphene thread sensor was created using various chemical processes on a regular polyester thread. Graphene conductive ink was used to

coat the polyester threads and consists of graphene nanoplatelets, PEDOT: PSS, and DMSO. Graphene nanoplatelets serve as the active sensing materials in ink and provide conductivity to the threads by forming a conductive coating through Van der Waals forces between the GnPs and polyester threads. DMSO is the primary solvent used to disperse GnPs, as it has high solvation power to dissolve GnPs due to its aprotic nature. DMSO contains hydrophilic and hydrophobic parts, and the hydrophobic parts form weak Van der Waals forces with GnPs to disperse GnPs in an aqueous medium. To further enhance the ink's conductivity, PEDOT: PSS was added as a co-solvent, as it is a naturally conductive polymer. PEDOT: PSS is highly compatible with DMSO and can effectively wash away the non-conductive PSS shell of PEDOT: PSS, increasing the ratio of conductive PEDOT in the ink. This improves the overall conductivity of the solution. Figure 3.3 illustrates the steps in processing the graphene thread from a regular polyester thread.

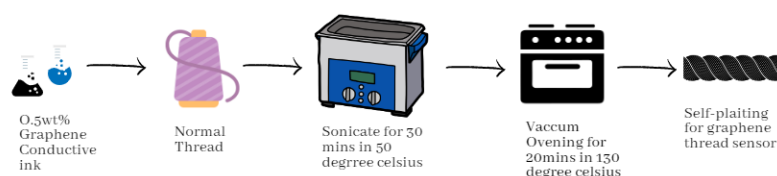


Figure 3.3: Illustration of Graphene Thread Processing Steps

First, a normal polyester thread is immersed into 0.5 wt% graphene conductive ink, as shown in Figure 3.4.

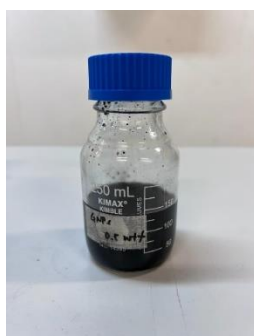


Figure 3.4: Graphene Conductive Ink

Next, the entire bottle was placed in an ultrasonic bath set at 50 degrees Celsius and sonicated for 30 minutes to agitate the particles in the solvent,

resulting in a better coating. The thread was then removed from the bottle and placed into the vacuum oven at 130 degrees Celsius for 20 minutes to completely dry the graphene thread. The ultrasonic bath can be seen in Figure 3.5, the vacuum oven in Figure 3.6, and the resultant graphene thread immediately after being removed from the vacuum oven in Figure 3.7.



Figure 3.5: Ultrasonic Bath



Figure 3.6: Vacuum Oven



Figure 3.7: Resultant Graphene Thread

The graphene thread is now ready to undergo processes such as plaiting and covering with copper tape to increase its strength and sensitive.

3.3 Characterisation and Calibration of Data Glove

The process of Characterisation and Calibration involves finding the output response pattern by using Arduino IDE and serial communication to map the

output data with their corresponding bending angle by creating a mathematical model equation. In this project, the ESP32 is the main hardware component used to process the data input from the graphene thread sensor. Figure 3.8 shows the hardware structure of the DOIT ESP32 DEV KIT V1.

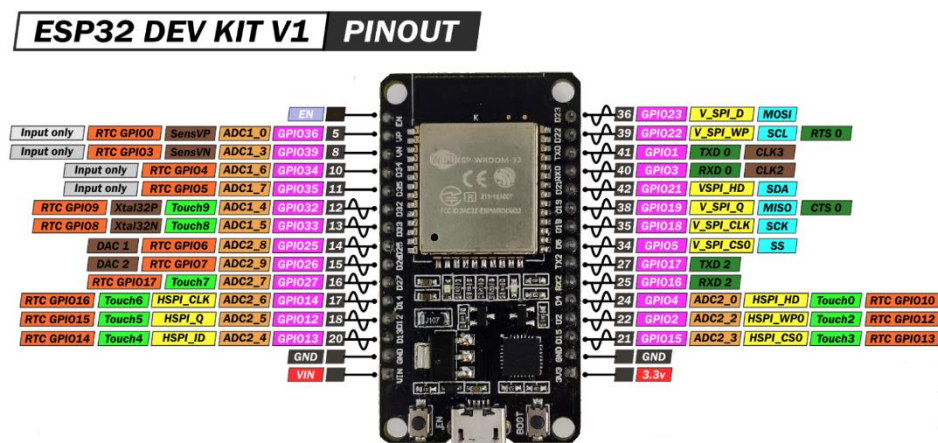


Figure 3.8: Hardware Structure of DOIT ESP32 DEV KIT V1

There are three ways to activate the ESP32: by connecting a power supply to the Micro-USB port, through the Vin pin, or through the 3.3V pin. Due to its internal LDO voltage regulator, the ESP32 outputs a constant 3.3V, regardless of the input supply voltage. However, it is recommended to supply the ESP32 with a voltage in the range of 3.3V to 5V to balance its lifespan and activation. The ESP32 has 15 ADC pins that enable it to read analog input. The ADC input channels have 12-bit resolution, which means analog readings range from 0 to 4095. A reading of 0 corresponds to 0V, while 4095 corresponds to 3.3V. Additionally, there are two 8-bit DAC pins that enable the ESP32 to convert digital signals into analog voltage signals for output. All pins can be set to input or output, except GPIO36, GPIO39, GPIO34, and GPIO35, which can only be used as input pins due to a lack of internal pull-up or pull-down resistors. Nine touch pins enable the ESP32 to sense variations in electrical charges, such as those produced by human skin. SCL and SDA pins enable the I2C protocol, which allows the ESP32 to communicate with multiple slave devices in half duplex. The SPI pins enable communication with only one slave device at a time and in a full duplex. RTC pins are used to wake up the ESP32 in deep sleep mode, saving power consumption.

For this project, the required pins include ADC pins for input analog readings and output voltage, SCL and SDA pins for the I2C protocol, the 3.3V pin for activating the ESP32, and GND pins. To calibrate and characterize the self-plaiting and graphene thread sensors, 5 ADC input pins will be needed, and their connections are shown in Figure 3.9.

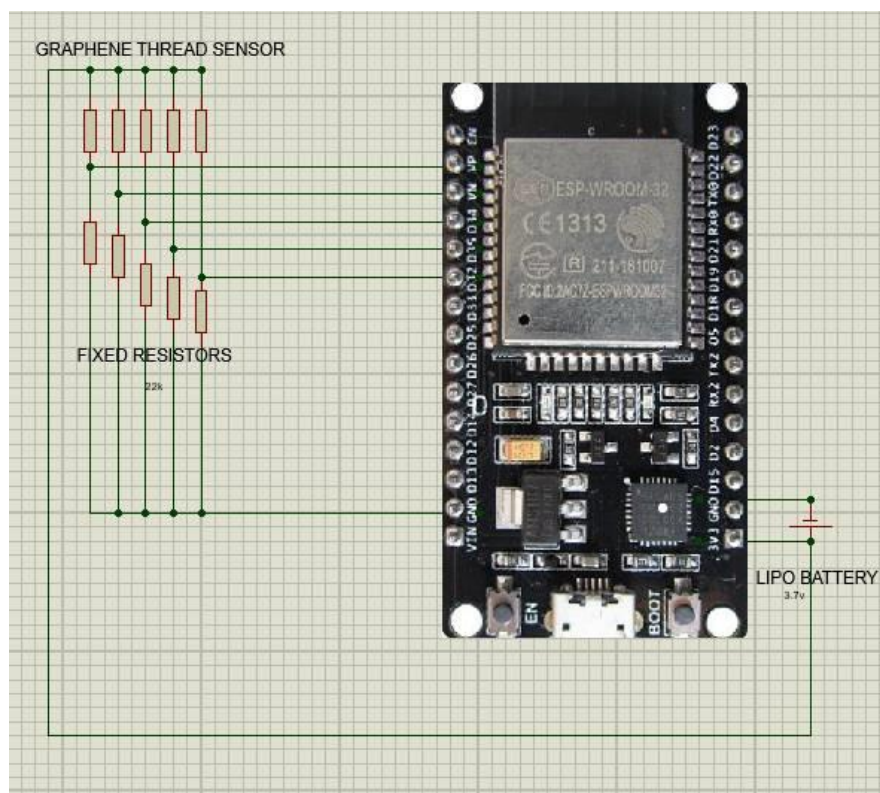


Figure 3.9: Connection of Voltage Divider Circuit to ESP32

3.3.1 Data Collection from Finger Sensor

To enhance the graphene thread sensor's connectivity, strength, and sensitivity, two copper tapes with jumper wires soldered onto them have been used and attached to both ends of the graphene thread sensor. One end of the jumper wire will connect to a power source, while the other will connect to a 22k ohm resistor and ground, as shown in Figure 3.9. These can be connected to either end because there is no polarity difference between the ends of the graphene thread sensor, forming a voltage divider circuit. Then, connect the voltage output port of the voltage divider circuit to GPIO 32, GPIO 34, GPIO 35, GPIO 36, and GPIO 39 pins for the 5 thread sensors, respectively. The GPIO 32 to 39 pins act as input pins to read the analog value input from the voltage divider.

The reason for selecting a 22k ohm resistor as the fixed resistor in the voltage divider circuit is that the output resistance of fabricated graphene thread sensors in the optimum condition is between 30k to 100k ohm when stretched and released, which is close to 22k ohm.

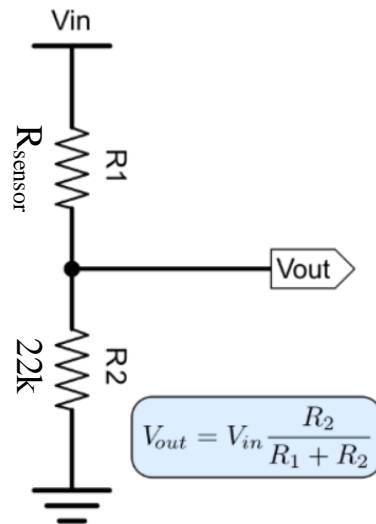


Figure 3.10: Voltage Divider

As shown in Figure 3.10, two undesired conditions will not occur: when the fixed resistance (R_2) is too high, changes in R_{sensor} will not cause obvious changes in V_{out} , and when the fixed resistance (R_2) is too low, the V_{out} will be close to 0. As finger bending causes the stretching of the graphene thread sensor, it changes its resistance and, consequently, in the V_{out} . The input analog reading of the ESP32, the finger bending angle, can be mapped from the input analog reading. The analog reading corresponding to different bending angles of the finger can be called out and displayed on the serial monitor, as shown in Figure 3.11.

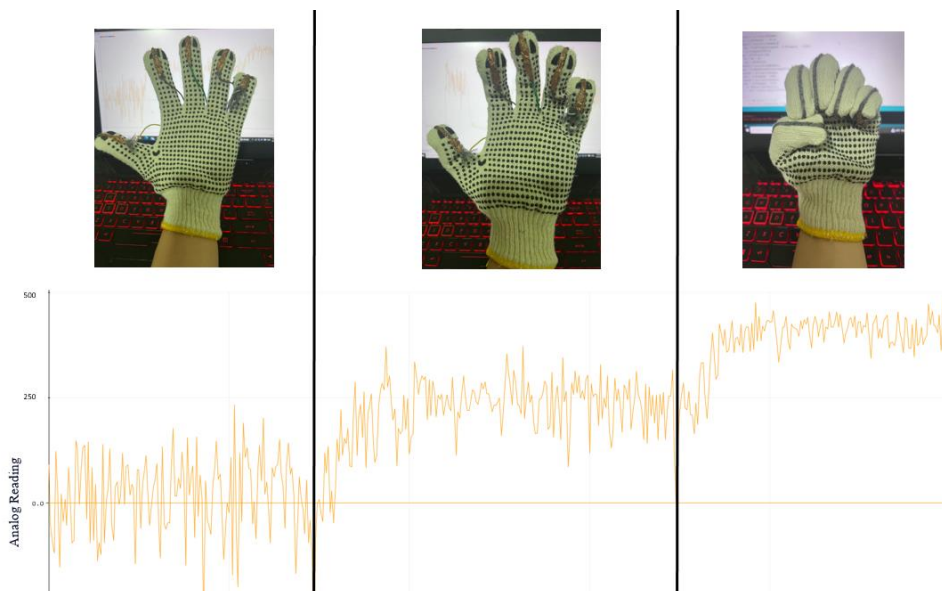


Figure 3.11: Analog Reading Vs Bending Angle

After that, the bending angle can be mapped from the analog value by forming a quadratic equation, $\theta = Ax^2 + Bx + C$, where x represents the input analog reading and θ represent the angle data. Then, the angle data can be sent out through wireless connection algorithms to the game environment. Figure 3.12 shows the flowchart for the data collection from the graphene thread sensor.

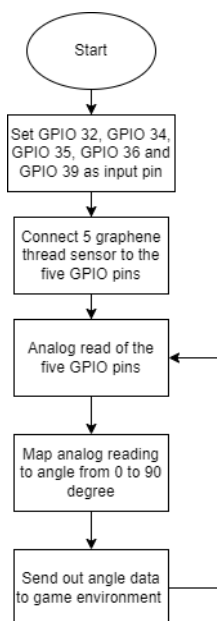


Figure 3.12: Flowchart of Data Collection of Graphene Thread Sensor

3.4 MPU-6050 Module

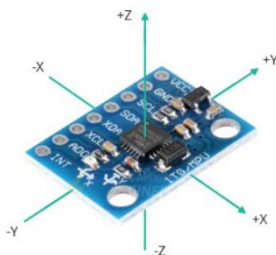


Figure 3.13: MPU-6050 Module

The MPU-6050 Module (Figure 3.13) is used to detect the orientation of the real-world hand and control the motion of the hand model in the virtual environment based on the motion of the real-world hand, with the aid of the ESP32. The MPU-6050 Module is a 6-axis motion tracking device that includes a 3-axis gyroscope and a 3-axis accelerometer. The gyroscope tracks the rotational gravity, while the accelerometer tracks the gravitational creation. The MPU-6050 Module also has a built-in integrated DMP, which provides high motion-related computational power. The module uses the I2C protocol for bi-directional communication between the module and external devices via the SDA and SCL ports. For this project, two MPU-6050 Modules will be used, and their connections are shown in Figure 3.14.

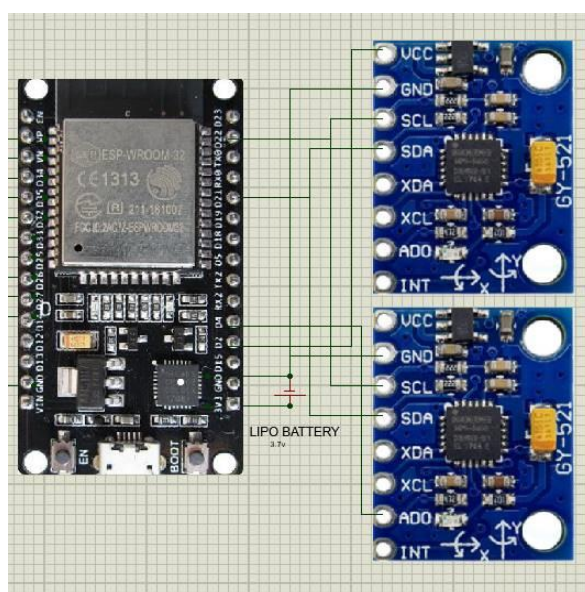


Figure 3.14: Connection of MPU6050 to ESP32

3.4.1 Data Collection from MPU6050

Both MPU6050 modules act as slaves to the ESP32 and have duplex communication with it. Each of them serves a different purpose and function in the project. The first MPU6050 is used to detect and collect quaternion rotation data of the user's hand, while the second MPU6050 is used to detect and collect acceleration in the vertical and horizontal directions of the user's hand.

As shown in Figure 3.14, the SCL pin of the MPU6050 is connected to the SCL pin of the ESP32, which is GPIO 22, while the SDA pin of the MPU6050 is connected to the SDA pin of the ESP32, which is GPIO 21. To enable communication between the two MPU6050s and the ESP32 simultaneously, the I2C protocol is used. To enable the I2C protocol, the GPIO 2 and GPIO 4 pins are set as output pins to provide power to the MPU6050s. The V_{cc} pin of the first MPU6050 is connected to GPIO 2 and powered to high, while the AD0 pin of the second MPU6050 is connected to GPIO 4 and powered to high. Both GND pins of the MPU6050s are connected to the GND pin of the ESP32. After finishing the connection of the MPU6050s to the ESP32, coding for them is taking place. First, the I2C address for both of them is set; one is 0x68, the default address for the I2C protocol, while the other is 0x69 for the MPU6050 with the AD0 pin high. Then, the function class is defined to tell the MPU6050 to output values corresponding to desired parameters. For this project, function classes are set to <OUTPUT_READABLE_QUATERNION> to get quaternion orientation and <OUTPUT_READABLE_WORLDACCEL> to get horizontal and vertical acceleration. The offset is then set and calibrated by setting the offset value to the MPU6050s to eliminate the zero error. After that, both MPU6050 are initialized by checking the functionality of the DMP (Digital Motion Processor) and getting the initial DMP FIFO Packet Size for later comparison and calculation. The process is repeated by checking whether the DMP is ready. If so, the current DMP FIFO Packet Size is obtained to compare with the previous one to output the parameter values corresponding to the current motion. Then, the data is sent out through the wireless algorithm of the ESP32. Figure 3.15 shows the overall MPU6050 process flow.

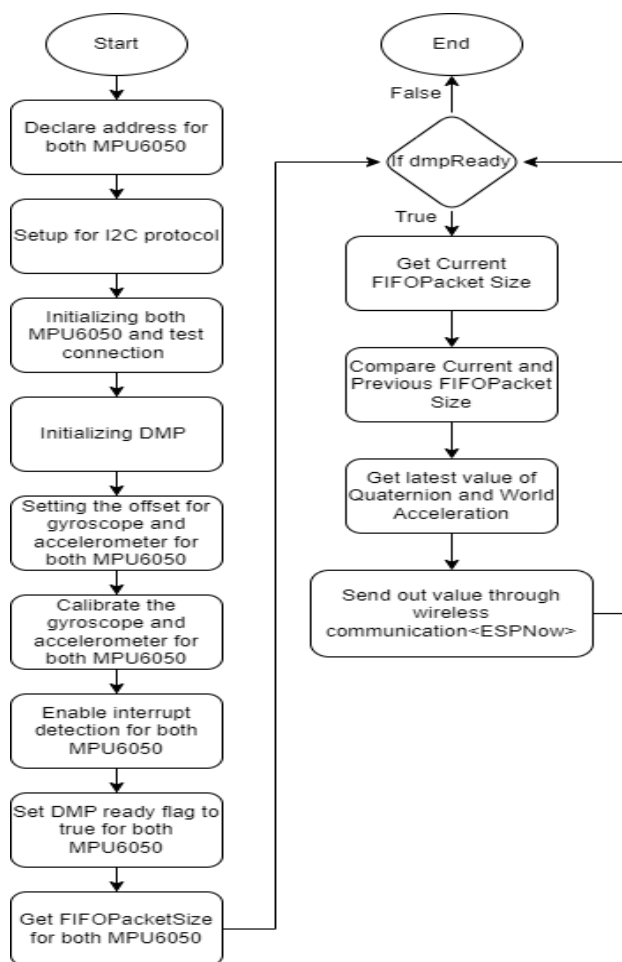


Figure 3.15: MPU6050 Process Flow

In this project, there are total of 7 parameters value will be output by two of the MPU6050 as shown in Table 3.1.

Table 3.1: MPU6050 Output Parameters

MPU6050	Output Parameter	Represent
First	q.w	Quaternion Scaler
	q.x	Quaternion vector in X
	q.y	Quaternion vector in Y
	q.z	Quaternion vector in Z
Second	aaWorld.x	World acceleration in X
	aaWorld.y	World acceleration in Y
	aaWorld.z	World acceleration in Z

3.5 Haptic Actuator



Figure 3.16: Haptic Actuator

A haptic actuator (Figure 3.16) was added to the glove to enhance the user's experience and interaction with the game. An ERM (Eccentric Rotating Mass) type of DC motor vibration haptic actuator has been selected because it is small and lightweight and functions with DC voltage input, which is easier to control by the ESP32 since it outputs DC voltage. The ERM haptic actuator operates at a rated DC voltage of 3V and creates vibration when supplied with DC voltage. When certain criteria are met, a signal can be sent to the ESP32 to output DC voltage and activate the ERM haptic actuator. Figure 3.17 shows how the DC motor vibration is connected to the ESP32.

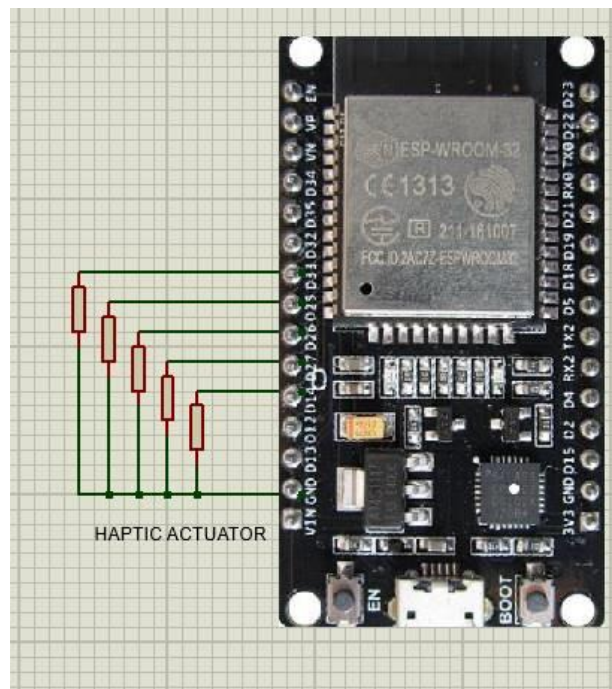


Figure 3.17: Circuit Connection of DC Motor Vibration to ESP32

3.5.1 Haptic Actuator Signal

In this project, five haptic actuators have been used for each finger. In the game environment, when a certain condition is met, a signal will be sent from the game environment to the ESP32, which will then control the vibration of the DC motor vibrations. To achieve this, GPIO 14, GPIO 25, GPIO 26, GPIO 27, and GPIO 33 have been set as output pins to supply voltage to the five DC motor vibrations. For example, when the player grabs the boom balloon in the game environment, a signal 'a' will be sent from the game environment to the ESP32. When the ESP32 detects the signal 'a', it will control the five GPIO pins that are initially low by setting them high for 0.5 seconds and then turning them low again. Figure 3.18 illustrates this process.

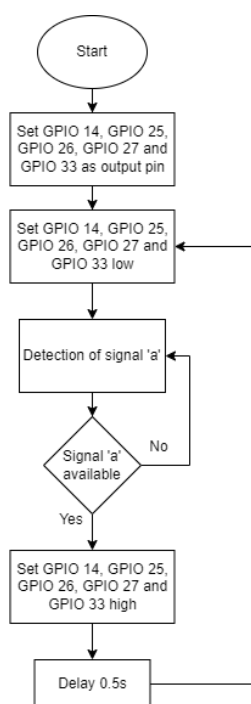


Figure 3.18: Flowchart of Haptic Actuator Process

3.6 Setup in Arduino IDE



Figure 3.19: Arduino Logo

The Arduino IDE (Figure 3.19) is a software tool based on the C++ language, which allows users to write and upload code to the Arduino hardware chip through a USB cable. Certain essential steps must be followed to ensure proper functioning. For example, the board setting in the Arduino IDE must match the Arduino hardware used. This setting can be configured through Tools >> Board >> Board Manager. In Board Manager, various board types are available, and the user must select the appropriate one based on the hardware type. However, there may be some problems in this step, such as when the downloaded version of Arduino IDE is too new or old to have a board type that matches the Arduino hardware type. In such cases, the user should add an online source through File >> Preferences >> Additional Boards Manager URLs. The source URLs can be copied and inserted to solve this problem. Additionally, the corresponding library must be included in the code if a particular function needs to be realized. For example, if the 'WiFi' function needs to be included in the application, the 'WiFi' library must be included by following the step (Sketch >> Include Library >> Manage Libraries) to find the particular library in the Library Manager and install it. If the library is not available in the Library Manager, the user should download its zip file online and add it to the Arduino IDE through (Sketch >> Include Library >> Add .ZIP Library). Once the codes are written in the Arduino IDE, they can be uploaded to the chip through a USB cable and do not need to be uploaded again unless some part of the code needs to be changed. The Arduino IDE also provides some features, such as the Serial Plotter, which helps to plot the outcome of serial communication. In this project, the Doit ESP32 DevKit V1 board needs to be installed, and an extra library, such as 'elapsedMillis', must be added from an online source.

3.6.1 Data Transmission through Wireless and Serial Communication

In this project, two types of communication were used to transmit data which are wireless and serial communication. Figure 3.20 shows the data transmission process.

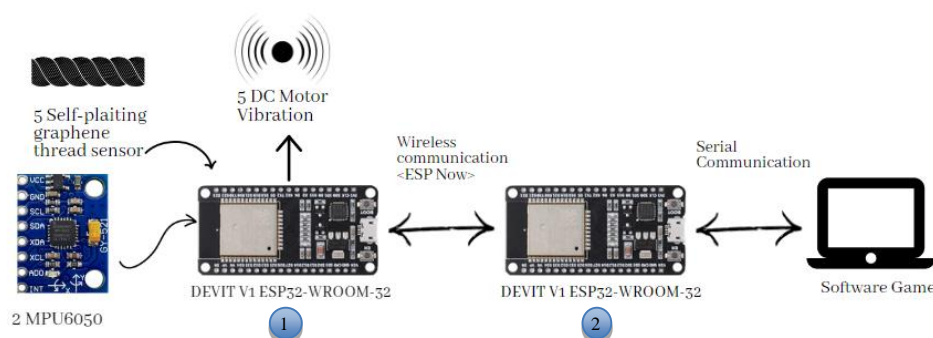
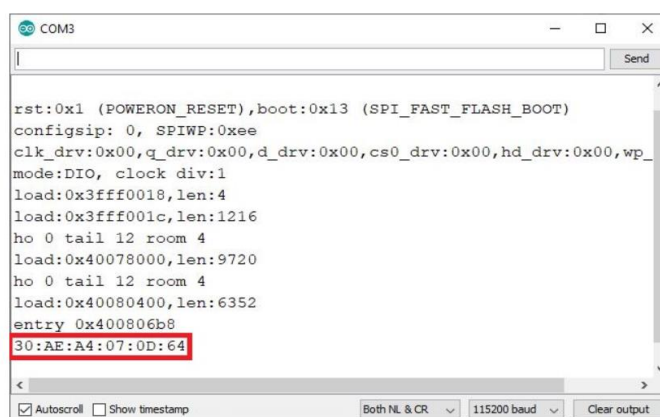


Figure 3.20: Data Transmission Process

In the first DOIT ESP32 DEV KIT V1, all electronic gadgets and sensors will be connected and coded to obtain the desired data for use in the game environment. The data will then be transmitted wirelessly to the second DOIT ESP32 DEV KIT V1. The second DOIT ESP32 DEV KIT V1 will be connected to a computer through a USB port and will receive data wirelessly from the first DOIT ESP32 DEV KIT V1, which will be transmitted to the computer through serial communication. Conversely, signals from the game environment can be transmitted to the second DOIT ESP32 DEV KIT V1 through serial communication and then to the first DOIT ESP32 DEV KIT V1 through wireless communication, activating the haptic actuators to provide haptic feedback to the user. For this reason, a wireless data glove that can interact with the game environment and provide haptic feedback can be fabricated.

The wireless communication algorithm used in this project is called ESP-NOW, which is one of the reasons why DOIT ESP32 DEV KIT V1 has been selected as the main microprocessor used in this project. ESP-NOW is a fast wireless communication protocol that allows users to exchange small messages of up to 250 bytes between ESP32 boards without using Wifi. ESP-NOW can be set up as one-way or two-way communication, depending on the setup and application. In this project, two-way communication is set up to transmit signals and data between electronic gadgets and the game environment. To set up two-way communication of ESP-NOW, some libraries, such as `esp_now.h` and `WiFi.h`, need to be included in the Arduino script. `WiFi.h` is not used to transmit data through Wifi but is used to get the unique MAC address of each DOIT ESP32 DEV KIT V1 and set them as a standalone wifi station to send data through their internal antenna. Each ESP32 board has its own unique

MAC address. Figure 3.21 shows an example of a MAC address that has been called out and displayed in the serial monitor of the Arduino IDE.



```
COM3
Send

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:9720
ho 0 tail 12 room 4
load:0x40080400,len:6352
entry 0x400806b8
30:AE:A4:07:0D:64
```

Figure 3.21: MAC Address Displayed in Serial Monitor

Once the MAC addresses of each ESP32 board have been obtained, they are set as the data receiver for each other. The MAC address of the second ESP32 is set as the data receiver address in the first ESP32 Arduino script, and vice versa. Then, a structure that contains all the variables needed to be sent and received between the two ESP32s is created. A callback function can be optionally created to notify the user whether the message has been sent successfully. All the data from the electronic gadgets and sensors are then assigned to variables defined in the message structure. The message structure is then sent and received between the two ESP32s using the `OnDataSent` and `OnDataRecv` commands through the internal antenna of ESP32. Figure 3.22 shows the process flow of wireless communication.

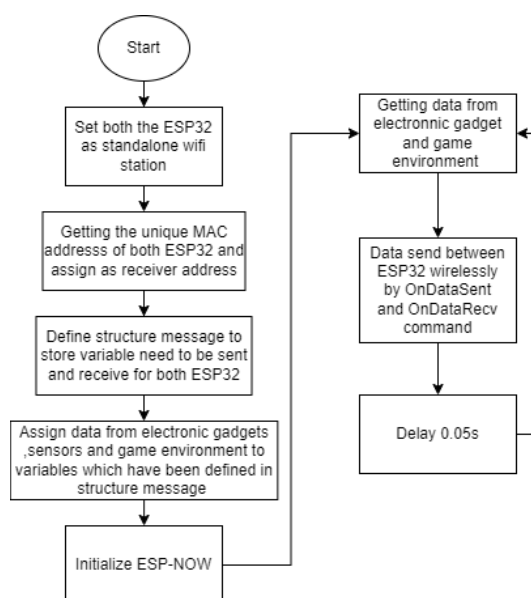


Figure 3.22: Process Flow of Wireless Communication

After the second ESP32 receives all the data from the first ESP32 through wireless communication, the data will be sent to the computer and game environment via serial communication. Conversely, when there is data from the game environment, data will be sent from the game environment to the second ESP32 via serial communication and wireless communication to the first ESP32. Figure 3.23 shows the process involved in the second ESP32.

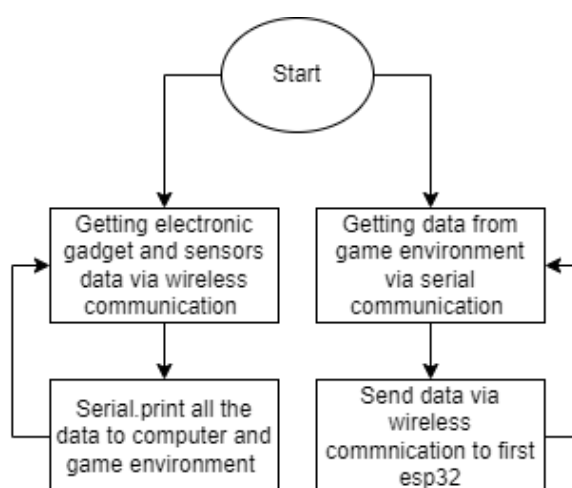


Figure 3.23: Process Flow Involving in Second ESP32

3.7 Power Source

In this project, a wireless connection is applied, thus causing the ESP32 cannot be directly powered up by the computer through a USB port. For that, a mini in

size, light in weight, and portable battery, as shown in Figure 3.24 has been introduced.



Figure 3.24: LiPo Battery

The battery capacity selected for this project is 600 mAh, which means it can supply a current of 600 mA for 1 hour at the nominal voltage of 3.7 V. This is because the power consumption of the ESP32 in ESP-NOW mode is about 20 mA, the MPU6050 will draw about 10 mA each, and the haptic actuator will draw about 60 mA in the 3V voltage supply each. Therefore, the total current required is about 340 mA. The 600 mA current supply of the LiPo battery is more than enough and can continuously power the whole prototype for about 1.5 hours before recharging is needed. The positive terminal of the LiPo battery will be connected to the 3.3V pin of the first ESP32, as well as one end of the Graphene Thread Sensor, and the negative terminal will be connected to the GND pin of the first ESP32.

3.8 Overall Hardware Design

Figure 3.25 shows the complete circuit for the first ESP32 project.

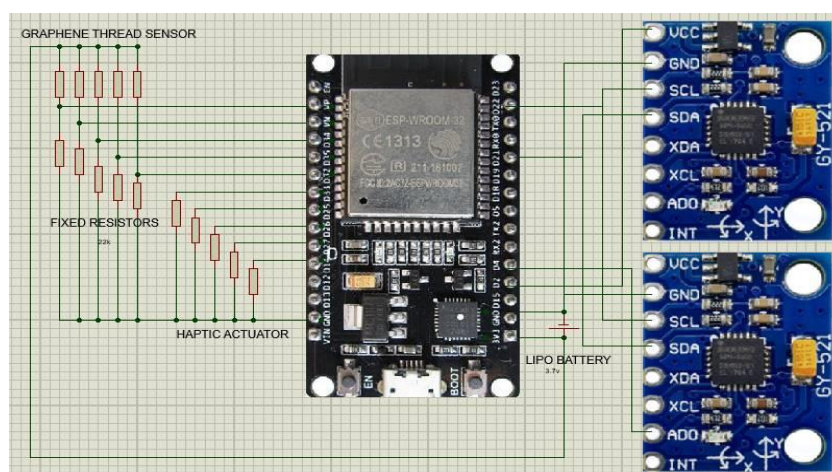


Figure 3.25: Overall Hardware Connection for First ESP32

Figure 3.26 shows the overall process flow of the first ESP32, and Figure 3.27 shows the overall process flow of the second ESP32

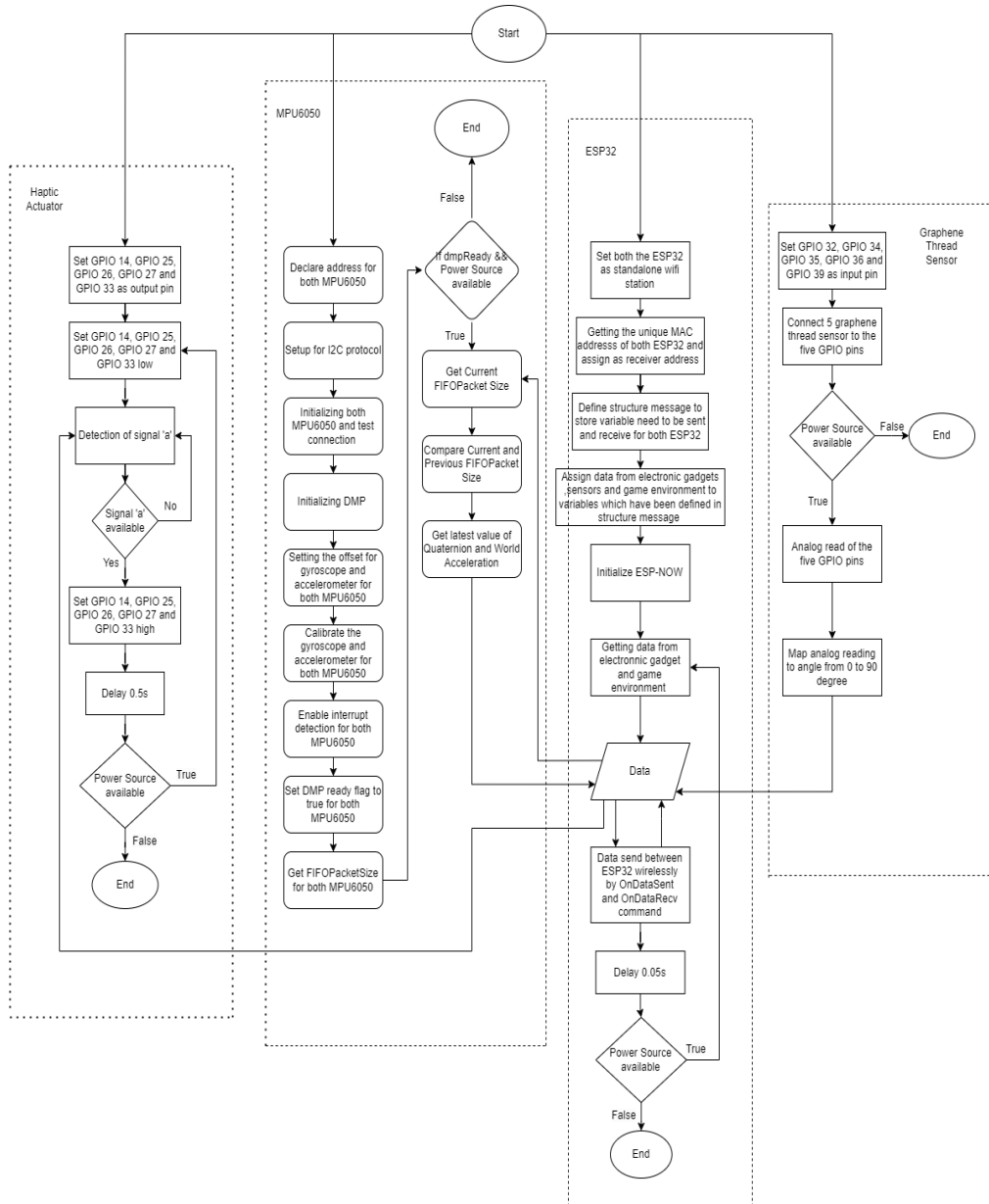


Figure 3.26: Overall Process Flow of First ESP32

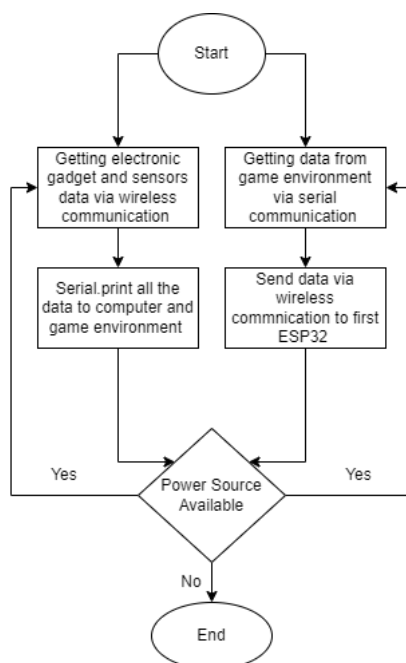


Figure 3.27: Overall Process Flow of Second ESP32

3.9 Build Up a Virtual Environment and Hand Model

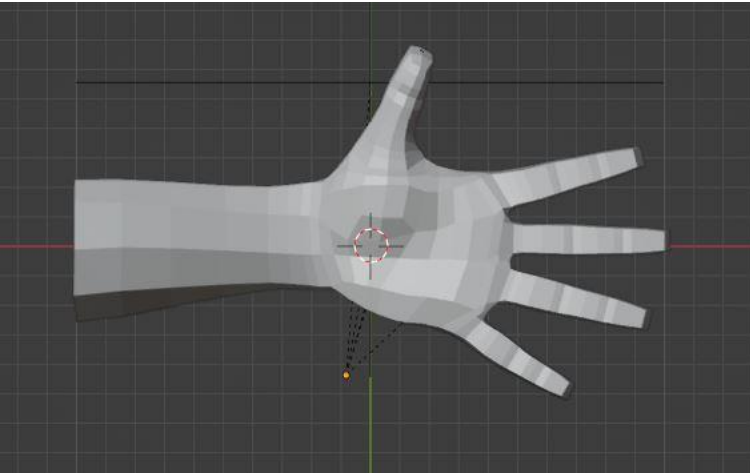
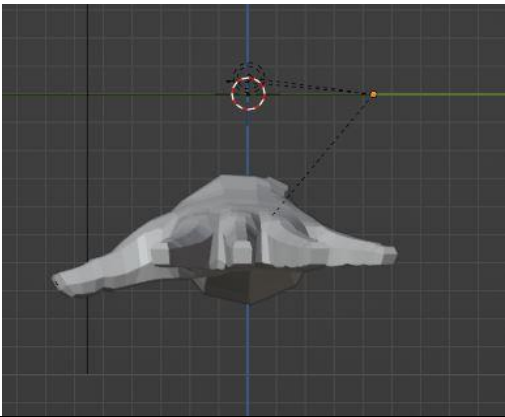
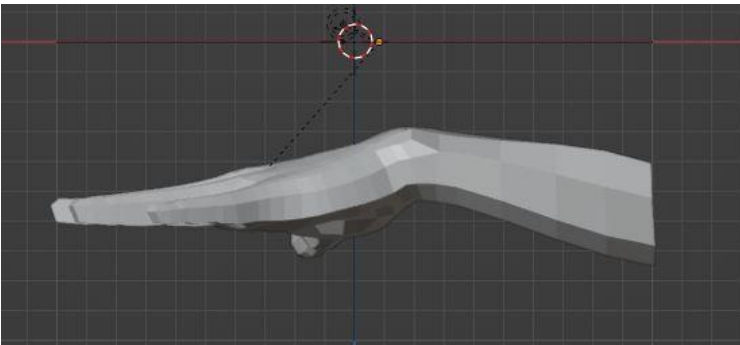
3.9.1 Blender



Figure 3.28: Blender Logo

To build up the virtual hand model, the software used are such as Blender(Figure 3.28) and Unity Editor. The Blender is mainly used to get the hand model. The Blender will be cooperated with Sketchfab to import online open-source hand models to the Blender. The Sketchfab can be added-on to the Blender with the step (Edit >> Preferences >> Add-ons). Then will need to register a Sketchfab account. After that, log in to Sketchfab account to select a suitable hand model and import it into Blender. In Blender, the armature will be added to the hand model following the pattern of model's finger and then set the relationship between the armature and hand model as parent-children. Therefore, the hand model will move simultaneously when the armature is moving. Table 3.2 shows the structure and design of the selected hand model, and Table 3.3 shows the design of the armature attached from a different perspective.

Table 3.2: Hand Model

Perspective	Hand Model
Bottom View	 A 3D perspective view of a hand model from the bottom. The hand is rendered in a light gray, low-poly style. A red dashed circle highlights the wrist area, with a red crosshair centered on it. A green dashed line extends from the center of the wrist towards the palm. A blue dashed line extends from the center of the wrist towards the fingers. A yellow dot is located at the end of the blue dashed line. The background is a dark gray grid.
Front View	 A 3D perspective view of a hand model from the front. The hand is rendered in a light gray, low-poly style. A red dashed circle highlights the wrist area, with a red crosshair centered on it. A green dashed line extends from the center of the wrist towards the palm. A blue dashed line extends from the center of the wrist towards the fingers. A yellow dot is located at the end of the blue dashed line. The background is a dark gray grid.
Side View	 A 3D perspective view of a hand model from the side. The hand is rendered in a light gray, low-poly style. A red dashed circle highlights the wrist area, with a red crosshair centered on it. A green dashed line extends from the center of the wrist towards the palm. A blue dashed line extends from the center of the wrist towards the fingers. A yellow dot is located at the end of the blue dashed line. The background is a dark gray grid.

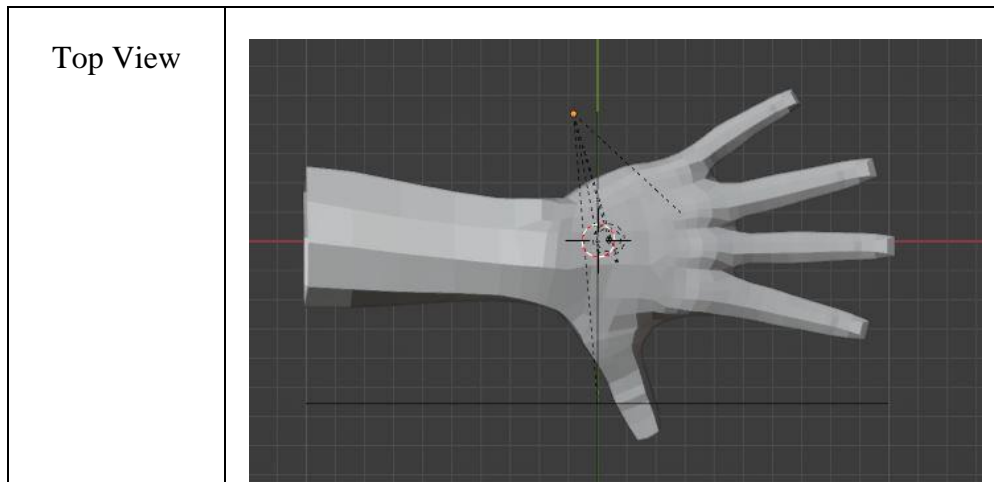
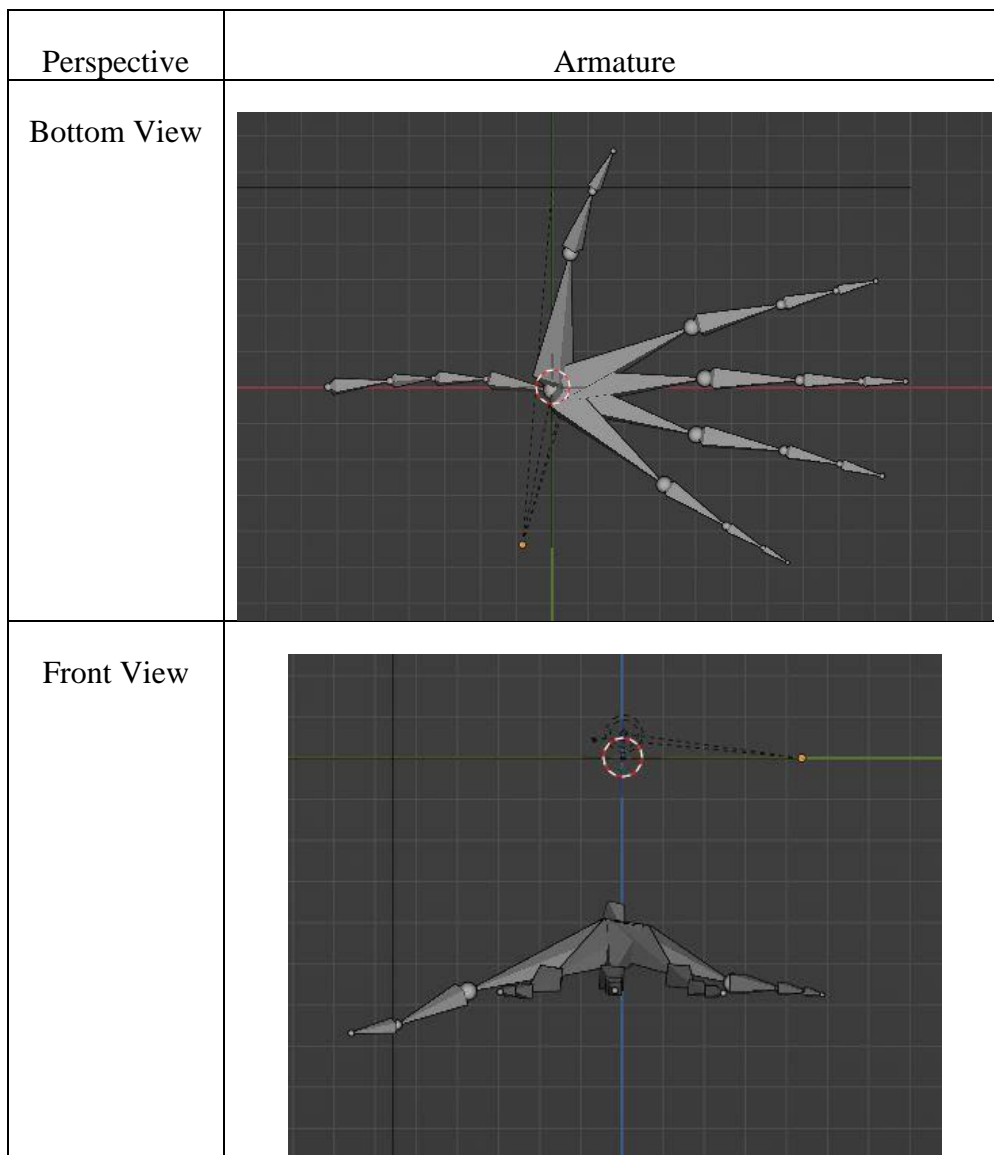


Table 3.3: Armature



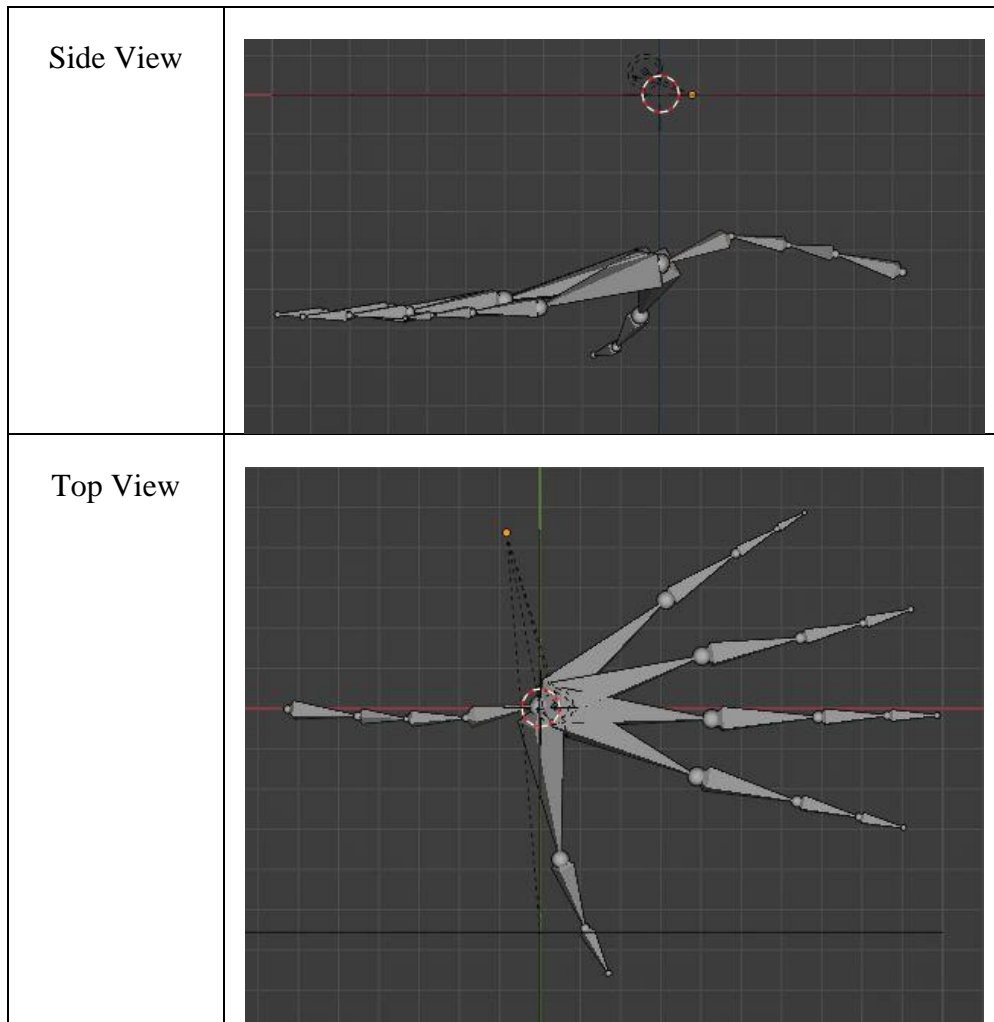


Figure 3.29 shows the hand model's grabbing motion by manually controlling the armature after they have been successfully set as the parent-children relationship.

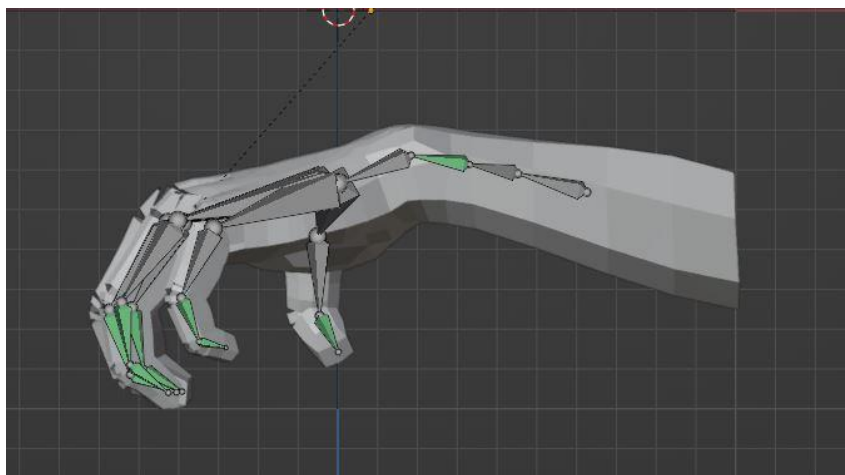


Figure 3.29: Grabbing Motion of Hand Model

3.9.2 Unity Editor

After the hand model has been created and attached to an armature, the file was exported in FBX format to the Unity Editor. Unity Editor is a game engine that enables users to create their own games using custom design models and controls their motion using C# scripts. The Unity Editor will treat the exported hand model from Blender and the C# script as assets. In this step, a mathematical model equation to represent the motion of the hand model's fingers and MPU-6050 module can be inserted into the C# script to control the motion of the hand model in the virtual world based on the real-world motion with the aid of ESP32 through wireless and serial communication. In Unity Editor, some games will be created for the patient to play during the rehabilitation process. Some haptic feedback will be sent back to the patient in a set condition as vibration from the haptic actuators.

3.10 VR Game Design

Four scenes were designed in the VR game environment, including the main menu, game selection, difficulty selection, and in-game scene. The data from the second ESP32 will be sent to the game environment via serial communication and will be bound to their respective parts in the game environment. For example, the data from the finger sensor will be bound to each finger of the hand model, while the data from the MPU6050 was bound to the hand model's body. Algorithms were also implemented to detect the grabbing and colliding motion of the hand model with game objects to make decisions in the game. Additionally, background music, sound and visual effects, a timer, a score manager, and other features will be introduced to make the game more engaging and enjoyable.

3.10.1 Grabbing and Colliding Motion

Hand model finger orientation fully depends on angle data senses from the graphene thread sensor. If the angle data is 90, the hand model will be presented as grabbing motion, and if the data decrease, the hand model finger bending angle will also decrease until 0. The grabbing and colliding motion is the important motion which need to be detected in game environment as a lot of decision making is mainly depends on them. To detect the grabbing motion, the

algorithms can be set to detect whether the angle data is 90 or near to 90 as the hand model motion is fully depending on angle data. For colliding motion, some function in the Unity Editor has been used to detect it such as `OnTriggerEnter()`, `OnTriggerStay()` and `OnTriggerExit()`. They are used to detect the hand model first collide with a certain game object, stay colliding with a certain game object and move away from colliding a certain game object respectively. To enable the three functions above, 3D collider need to be added to game object and hand model such that when the collider is collided then can be captured by code algorithm. There are many type of collider in Unity Editor, but the capsule collider has been selected as capsule collider can let user freely adjust the size and coverage of the collider. Figure 3.30 and Figure 3.31 show the example of capsule collider which attaches on a game object and hand model.

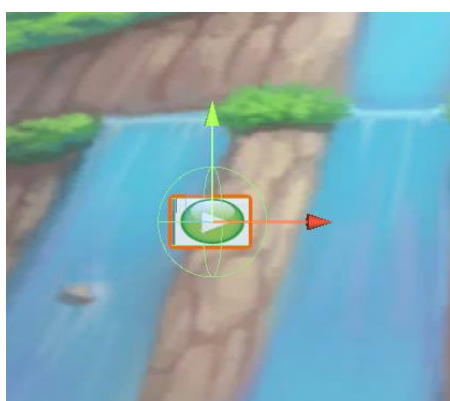


Figure 3.30: Capsule Collider on Game Object

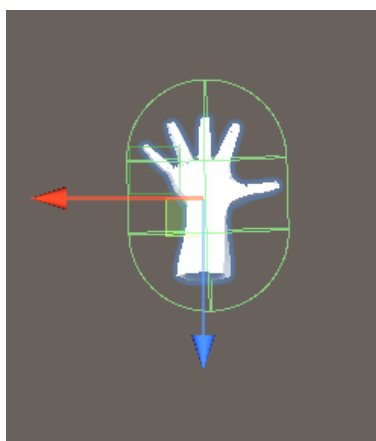


Figure 3.31: Capsule Collider on Hand Model

To make a decision in the game, the user must simultaneously move the hand model to collide with the desired game object and perform the grabbing motion. Two Boolean variables, `bool2`, and `value`, are assigned to detect grabbing motion and collision, respectively. The decision can only be made when `bool2` and `value` are both true. For example, if a user wants to play a particular game in the game station, they can move the hand model to collide with the game object or button corresponding to that game and perform the grabbing motion to make the selection. If the game algorithms do not detect the hand model performing the grabbing motion or the hand model does not collide with a particular game object, no decision will be made.

3.10.2 Game Scene

The main menu scene had two buttons: the play button and the exit button, which the user could select. To select the game station, the user can grab the play button, and to exit, the user can grab the exit button. If the user selects the play button, the scene manager will change the scene from the main menu to the game selection scene. In the game selection scene, the user can select their desired game by grabbing the corresponding button. Additionally, the user can choose to go back to the main menu scene by grabbing the back button. Once the user selects their desired game, three difficulty buttons will appear in the difficulty selection scene, which is easy, medium, and hard. Similarly, if the user wants to change the game, they can grab the back button, and the scene manager will take them back to the game selection scene. If the difficulty level is selected, the scene manager will change the scene to the in-game scene. For now, only one game, called Balloon Pop, has been designed. In Balloon Pop, balloons of different colors keep moving upwards at different speeds, depending on the difficulty level selected. For instance, the balloons move slowly on the easy level and quickly on the hard level. Figure 3.32 shows the design of the balloons in the game.



Figure 3.32: Balloon Design

In Balloon Pop, the user was given 60 seconds to play the game, and the highest score in history will be displayed for reference. During these 60 seconds, the user can freely move around and grab the normal balloons, and one point will be added to the score regardless of the balloon's color. If the user grabs the boom balloon, a punishment of 5 seconds will be deducted from the timer, and a signal 'a' will be sent out through serial and wireless communication to activate the haptic actuator.

To prevent cheating, algorithms have been designed to detect whether the user continuously holds the grabbing gesture without releasing it. A Boolean variable (bool1) has been assigned to detect whether the angle data is less than 20, indicating that the user has performed the hand opening motion.

After the time is up, the score will be compared with the highest score in history. If the score exceeds the highest score, it will be recorded as the new highest score. The user will then be given three options: press (R) to restart the game, press (G) to go back to the game selection scene to choose another game, or press (Esc) to exit the game station entirely. Figure 3.33 shows the overall game logic.

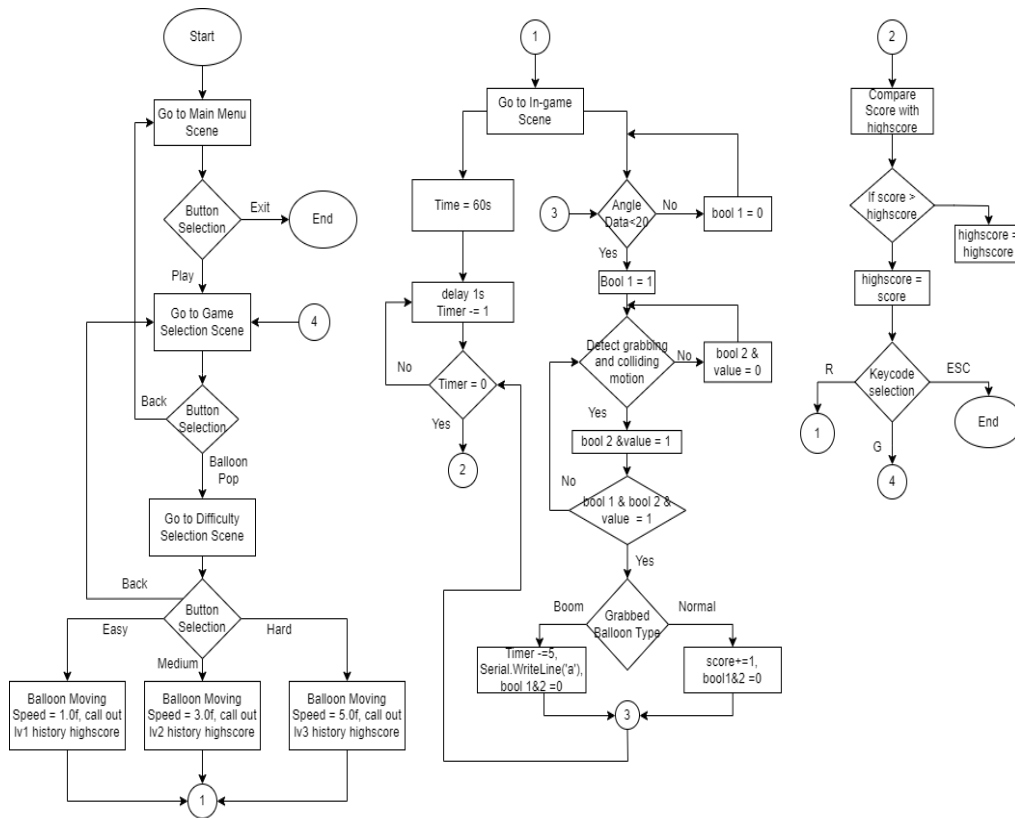


Figure 3.33: Overall Game Logic

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

In the following section, the fabricated graphene thread sensors result, the combined prototype on the PCB board, the VR Game design, and the effectiveness of serial and wireless communication will be evaluated and discussed.

4.2 Graphene Thread Sensor Result

Five graphene thread sensors have been well-fabricated and glued on the glove, as shown in Figure 4.1.



Figure 4.1: Graphene Thread Sensor on Glove

Figure 4.2 to Figure 4.6 show the graph of resistance change of each graphene thread sensor when the bending angle changes from 0 ° to 90 °.

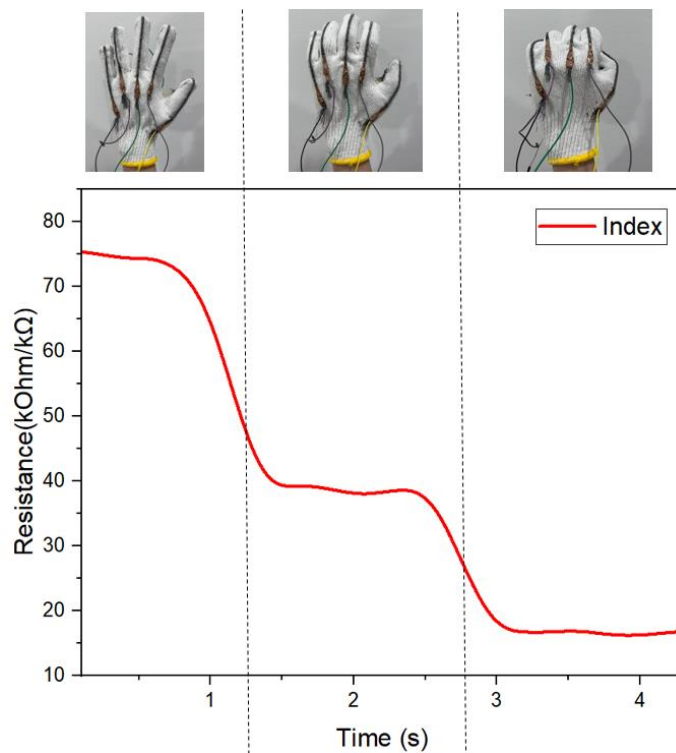


Figure 4.2: Resistance Change of Graphene Thread Sensor on Index

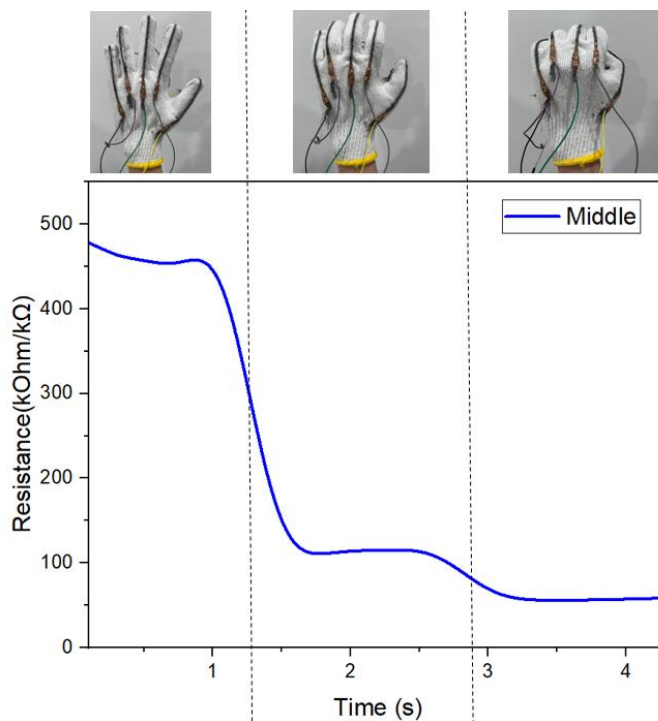


Figure 4.3: Resistance Change of Graphene Thread Sensor On Middle

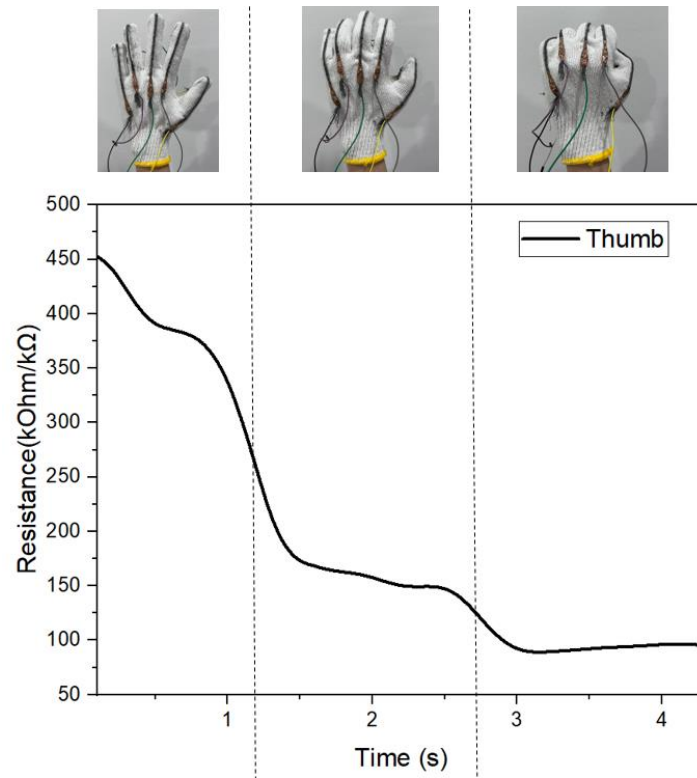


Figure 4.4: Resistance Change of Graphene Thread Sensor On Thumb

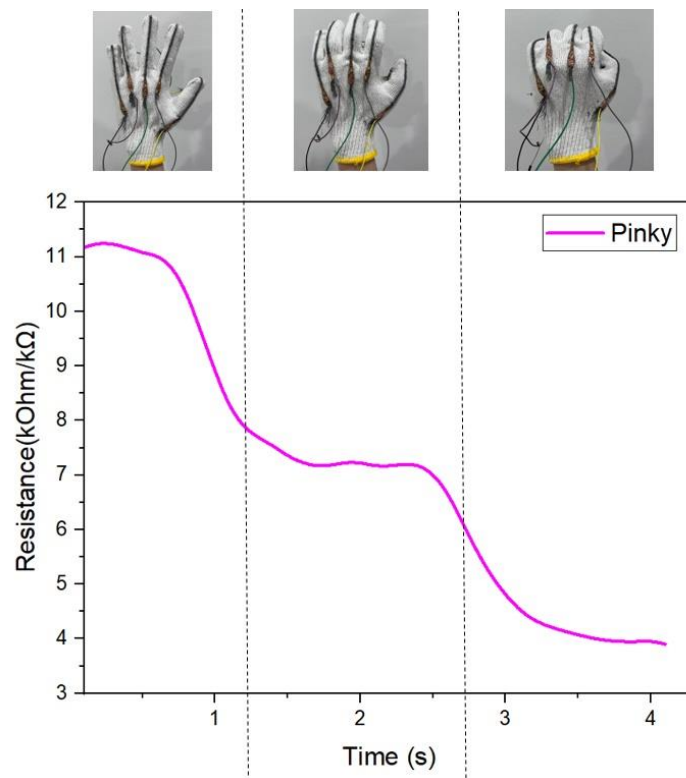


Figure 4.5: Resistance Change of Graphene Thread Sensor on Pinky

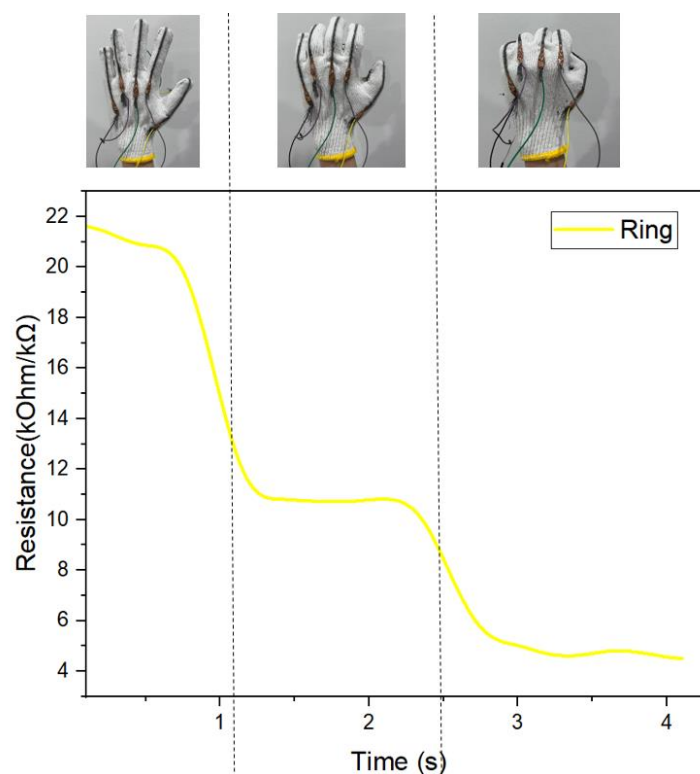


Figure 4.6: Resistance Change of Graphene Thread Sensor on Ring

Figure 4.2 to Figure 4.6 show that although the resistance range of the graphene thread sensors is different, they follow the same pattern where the resistance decreases as the bending angle increases. This is because the thread sensors are plaited from several processed single threads. When the sensor is stretched, the stretching force applied to it reduces the gaps between every single thread, causing the connectivity to increase and the resistance to decrease.

The different resistance ranges among the sensors may be due to human involvement in the sensor fabrication process. In each process, there may be some contamination or unavoidable scraping down of some graphene particles, which are coated on the thread during the plaiting and reclaiming process, further causing the intensity of graphene particles on each sensor to be slightly different. This condition might differ the connectivity and sensitivity of each sensor. However, this will not affect the sensing result of finger bending angle as it follows the pattern of decreasing resistance with increasing bending angle.

To further compare the five sensors, a normalized data graph has been plotted and is shown in Figure 4.7.

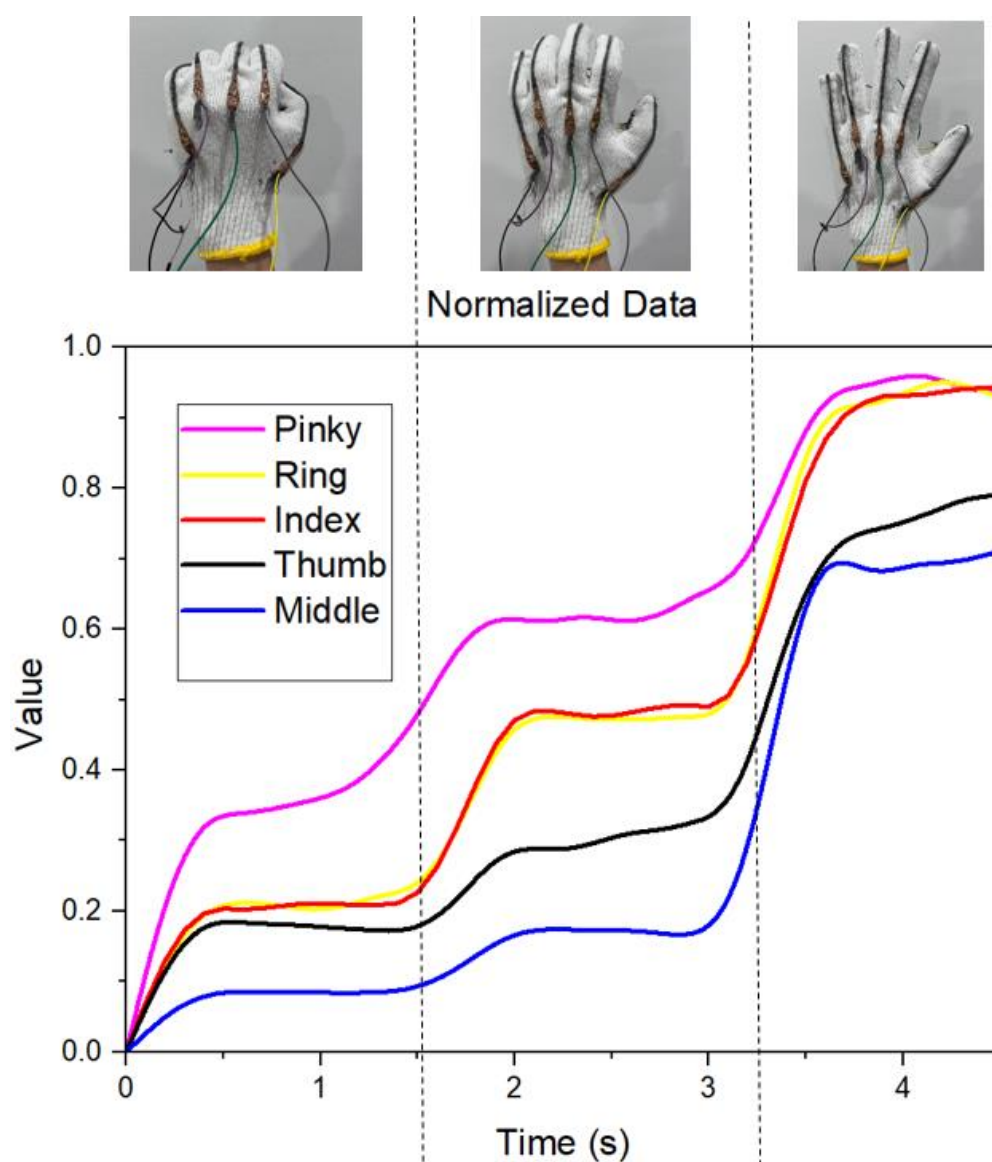


Figure 4.7: Normalized Data of Sensors

The normalized data is calculated by dividing the resistance data by their respective highest data value in each sensor, with bending angles ranging from 90° to 0° degrees, to put them in the same data range for better comparison. From the normalized data graph above, the ring and index finger sensors have steeper slopes than others. Thus, we can conclude that they are more sensitive to force applied on them, have less contamination, and have less human error during fabrication processes. Figure 4.8 shows the resistance change of the

graphene thread sensor under the bending and releasing motion of the finger for 100 cycles.

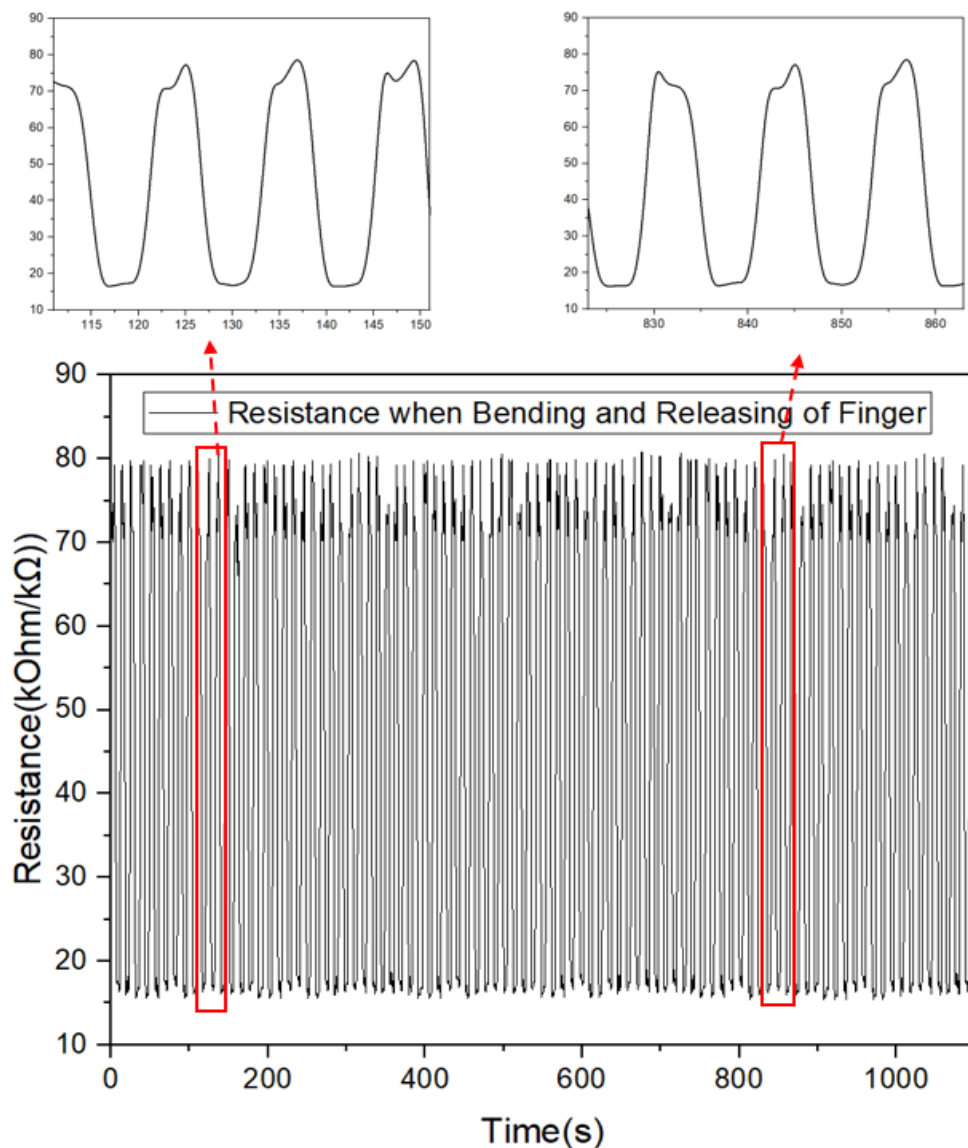


Figure 4.8: Resistance Change under Bending and Releasing of Finger

The equation in Table 4.1 maps the analog reading of the input pin of the ESP32 to the corresponding finger-bending angle data. This data is used to control the motion of the hand model in the game environment. Figure 4.7 illustrates the relationship between the analog reading and finger bending angle data for all five fingers. The figure shows that the angle data increases as the analog reading decreases. It is important to note that the equation may vary slightly for each sensor due to the differences in resistance range and sensitivity of each sensor.

Therefore, calibration may be required to ensure accurate finger bending angle data.

Table 4.1: Equation to Calculate Angle Data

Finger	Equation
Thumb	$\theta = 0.00015x^2 - 0.555x + 495$
Index	$\theta = 0.00060x^2 - 0.510x + 900$
Middle	$\theta = 0.00500x^2 - 0.070x - 123.75$
Ring	$\theta = 0.00075x^2 - 1.275x + 540$
Pinky	$\theta = 0.00080x^2 - 1.144x + 424.29$

4.3 Prototype Soldering and Combining Result

After the characterization and calibration of graphene thread sensor, all the components have been soldered on a PCB board using a solder gun. Figure 4.9 and Figure 4.10 show the result of a combined prototype on one PCB board.

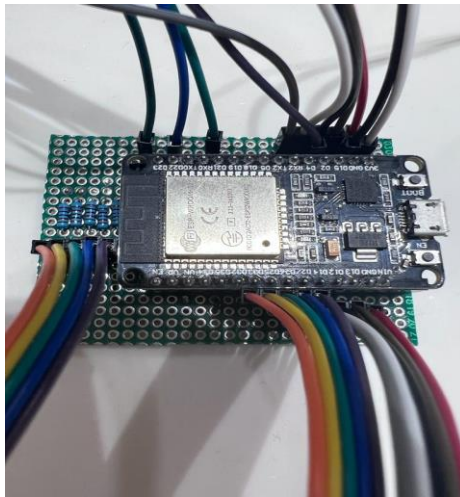


Figure 4.9: Top View of Combined Prototype

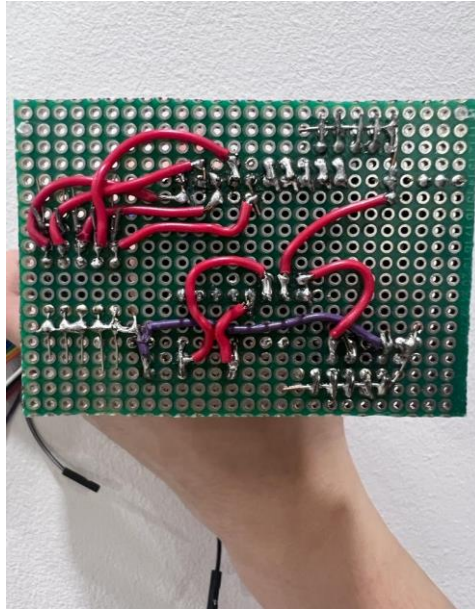


Figure 4.10: Bottom View of Combined Prototype

The board has a total of 26 jumper wires that were soldered onto it. Ten of these wires are used to connect the two end terminals of the five graphene thread sensors. Another ten wires are used to connect the positive and negative terminals of the five haptic actuators. Two wires are used to connect the positive and negative terminals of the LiPo battery, and four are used to connect the AD0 pin, GND pin, SCL pin, and SDA pin of one of the MPU6050 sensors. Additionally, another MPU6050 sensor is directly soldered onto the board underneath the ESP32 to reduce the space occupied by the components, as shown in Figure 4.11.

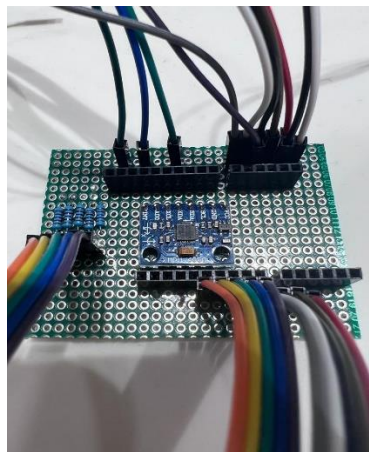


Figure 4.11: MPU6050 Soldered on Board

Five 22k ohm resistors are also soldered onto the board to act as fixed resistors in the voltage divider circuit. In addition, two covers were designed and 3D printed. One is used to cover the combined prototype on the PCB board, while the other is used to cover the other MPU6050 to prevent the user from getting injured by physical pins of parts when equipping it on the hand. Figure 4.12 and Figure 4.13 show the results.

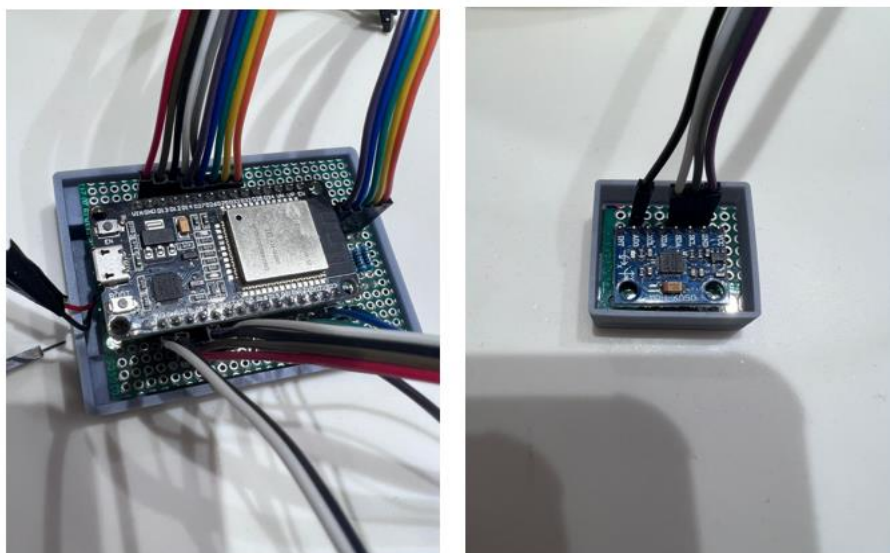


Figure 4.12: Top View of Prototype with Cover

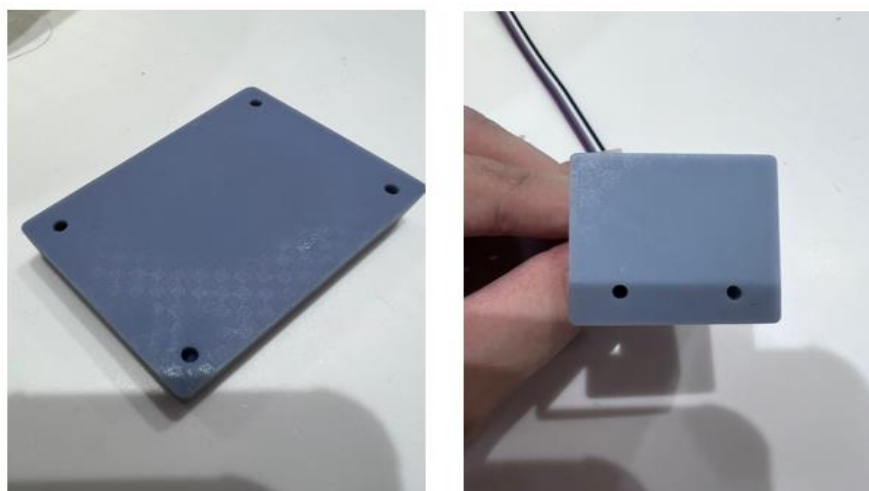


Figure 4.13: Bottom View of Cover

The cover of ESP32 is purposely made thicker to allocate LiPo battery together, as shown in Figure 4.14. Figure 4.15 shows the whole prototype in the user's hand.

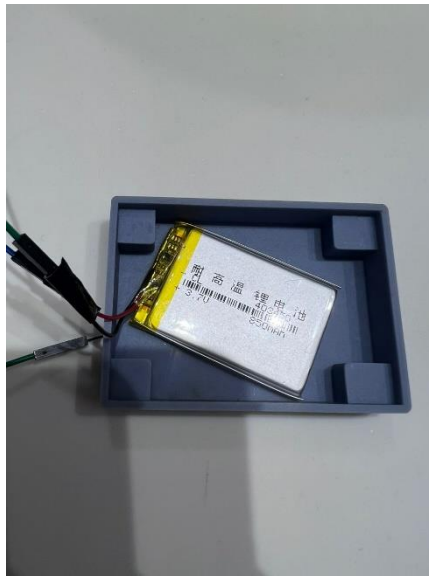


Figure 4.14: Cover with LiPo Battery

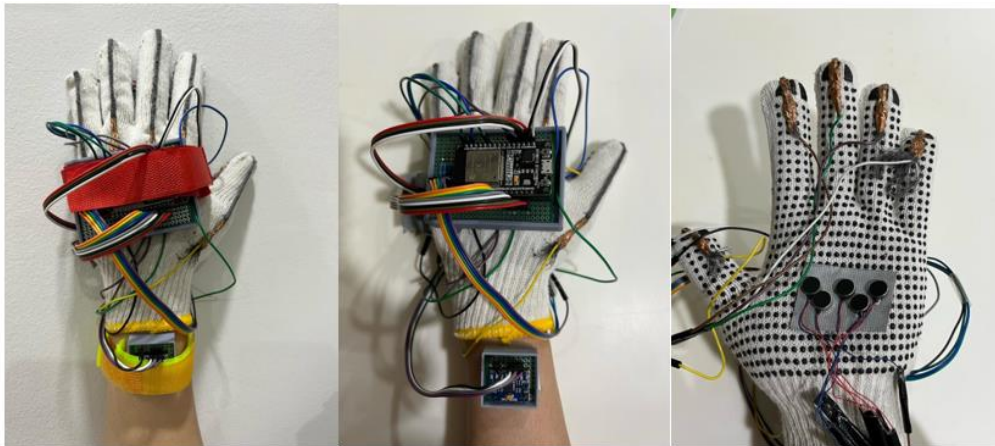


Figure 4.15: Prototype on User Hand

4.4 VR Game Environment

4.4.1 Main Menu Scene



Figure 4.16: Main Menu Scene

Figure 4.16 shows the design of the main menu scene in the game station. There are two buttons, Play, and Exit, to be selected by the user. If the Exit is grabbed, the game will be exit; if the Play is grabbed, the scene will be changed to Game Selection Scene.

4.4.2 Game Selection Scene



Figure 4.17: Game Selection Scene

Figure 4.17 depicts the design of the Game Selection Scene in the game station. Currently, only one game, Balloon Pop, is available, but more games can be added. The user can select between the Balloon Pop button and the Back

button. The scene will change to the Main Menu Scene if the Back button is selected. On the other hand, if the Balloon Pop button is selected, the scene will change to the Difficulty Selection Scene.

4.4.3 Difficulty Selection Scene



Figure 4.18: Difficulty Selection Scene

Figure 4.18 shows the Difficulty Selection Scene. There are 4 button to be selected by the user: Easy button, Medium button, Hard button, and Back button. If the Easy button is grabbed, the moving speed of the balloon in the Balloon Pop game will be the slowest, medium speed for the Medium button, and fastest for the Hard button. If the user grabs the Back button, the scene will change back to the game selection scene to select other desired games.

4.4.4 In-Game Scene



Figure 4.19: In-Game Preparation Time

Before the game starts, the user is provided with 3 seconds of preparation time, as shown in Figure 4.19.

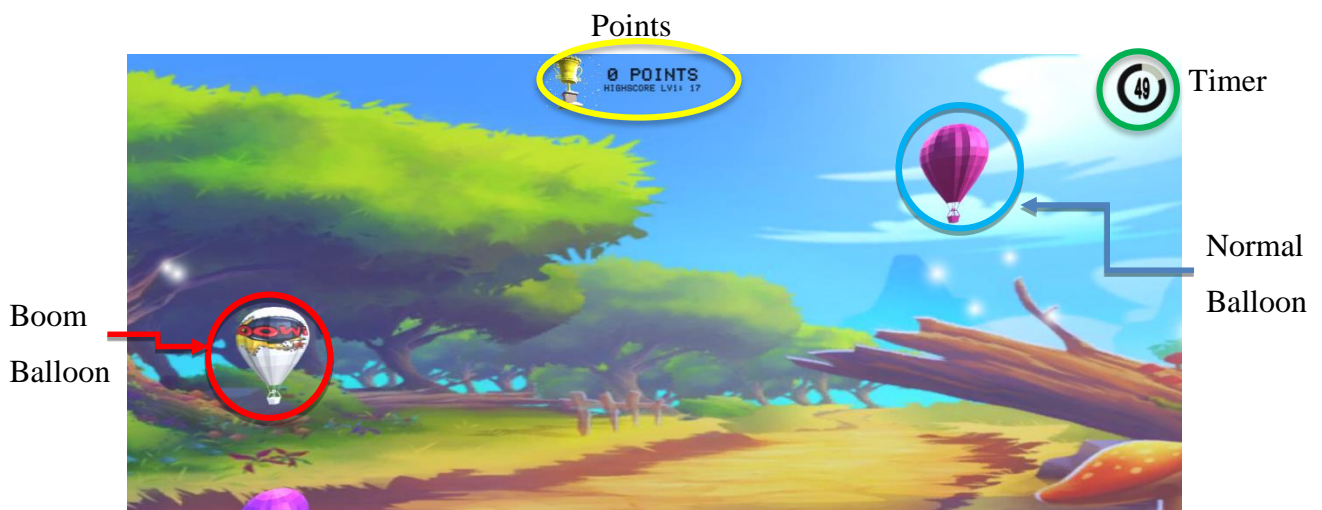


Figure 4.20: In-Game Scene

Figure 4.20 depicts the In-Game Scene that appears when the game starts. The user can grab two types of balloons - Normal Balloons and Boom Balloons. One point is added to their score when the user grabs a Normal Balloon. However, if the user grabs a Boom Balloon, they will be punished with a deduction of 5 seconds. The game also features a timer, a history of high scores,

and sound and visual effects. At the end of the game, the user's score is compared to the history of high scores. If the user's score is higher, it will overwrite the previous high score and remain in the record until the next high score is broken. The sound and visual effects make the game more interesting and engaging, notifying the user whenever a balloon is grabbed and popped. The accompanying figures (Figure 4.21 and Figure 4.22) showcase these effects.



Figure 4.21: Boom Balloon Effect

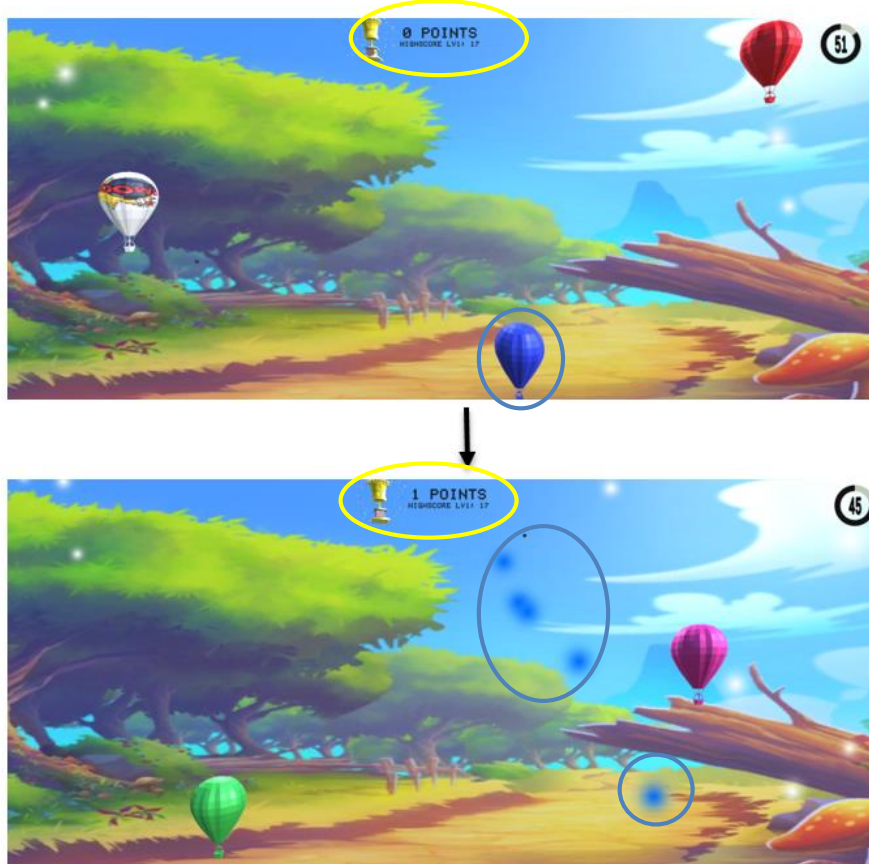


Figure 4.22: Normal Balloon Effect

As shown in Figure 4.21, when the Boom Balloon is grabbed, the timer will deduct 5 seconds, as seen in the figure where the timer goes from 60 to 55 seconds left. The visual effect will be accompanied by a short ‘boom’ sound effect and a signal ‘a’ will be sent out to activate the haptic actuator to notify the user that the Boom Balloon has been popped. On the other hand, as seen in Figure 4.22, when a Normal Balloon is grabbed, one point will be added to the point record regardless of the balloon color. The particle visual effect with a color corresponding to the grabbed balloon color and a short ‘pop’ sound effect will be played to notify the user that the Normal Balloon has been popped. An ending menu will pop out when the time is up, as shown in Figure 4.23.

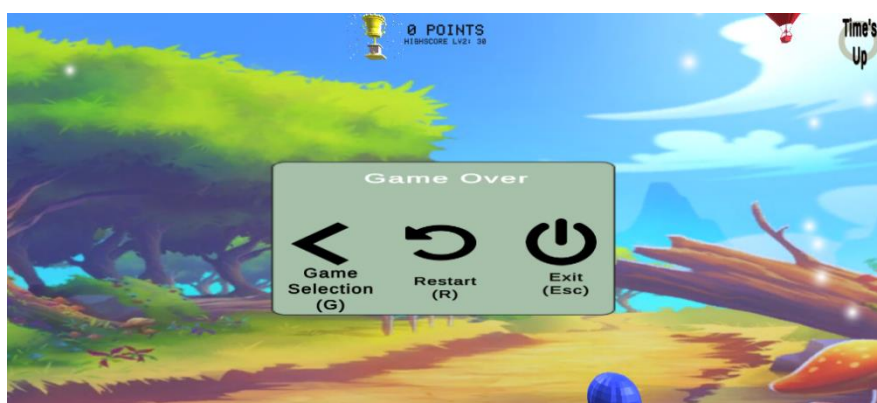


Figure 4.23: Ending Menu

After the game is over and the ending menu is displayed (as shown in Figure 4.23), the user has several options. They can go back to the Game Selection Scene to play another game (if available) by pressing G, restart the game by pressing R, or exit the game by pressing ESC.

4.5 Serial and Wireless Communication Result

In this project, there are 12 data values: 5 for data values from the graphene thread sensor to control finger motion, 4 data values from the first MPU6050 to control the quaternion orientation of the hand model, and 3 data values from the second MPU6050 to control the acceleration of the X, Y, and Z axis. These data values need to be sent out forward, from the first ESP32 to the second ESP32 through wireless communication and eventually to the computer and game via

serial communication. In reverse way, there is 1 data value that needs to be sent out as a signal to control the vibration of the haptic actuator via serial communication from the computer and game to the second ESP32. Figure 4.24 (a)- (d) show the result of the two communications.



Figure 4.24(a): Finger Motion Control via Communications



Figure 4.25(b): Finger Motion Control via Communications



Figure 4.26(c): Finger Motion Control via Communications



Figure 4.27(d): Finger Motion Control via Communications



Figure 4.28(a): Hand Model Control via Communications

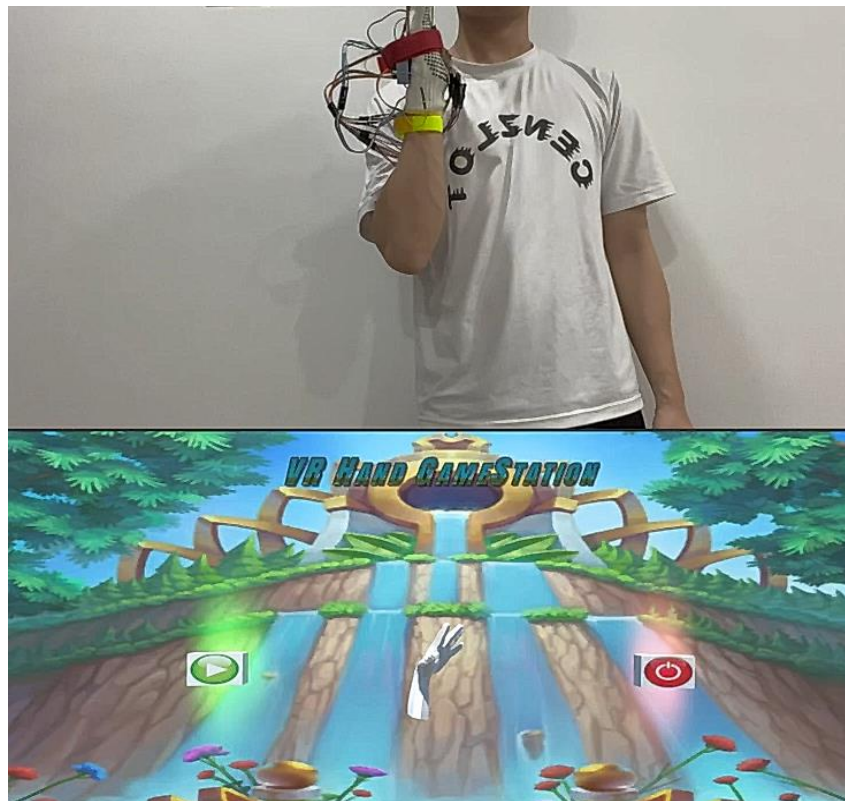


Figure 4.29(b): Hand Model Control via Communications



Figure 4.30(c): Hand Model Control via Communications



Figure 4.31(d): Hand Model Control via Communications

As shown in Figure 4.28(a) - (d), the user was able to control the hand model in the VR game environment through serial and wireless communication. By bending and releasing his fingers one by one, the user was able to control the bending and releasing of the fingers in the hand model in the game. Moreover, the user could control the hand model's motion by rotating and moving his hand wherever he wanted. Figure 4.32 and Figure 4.33 show the in-game condition.

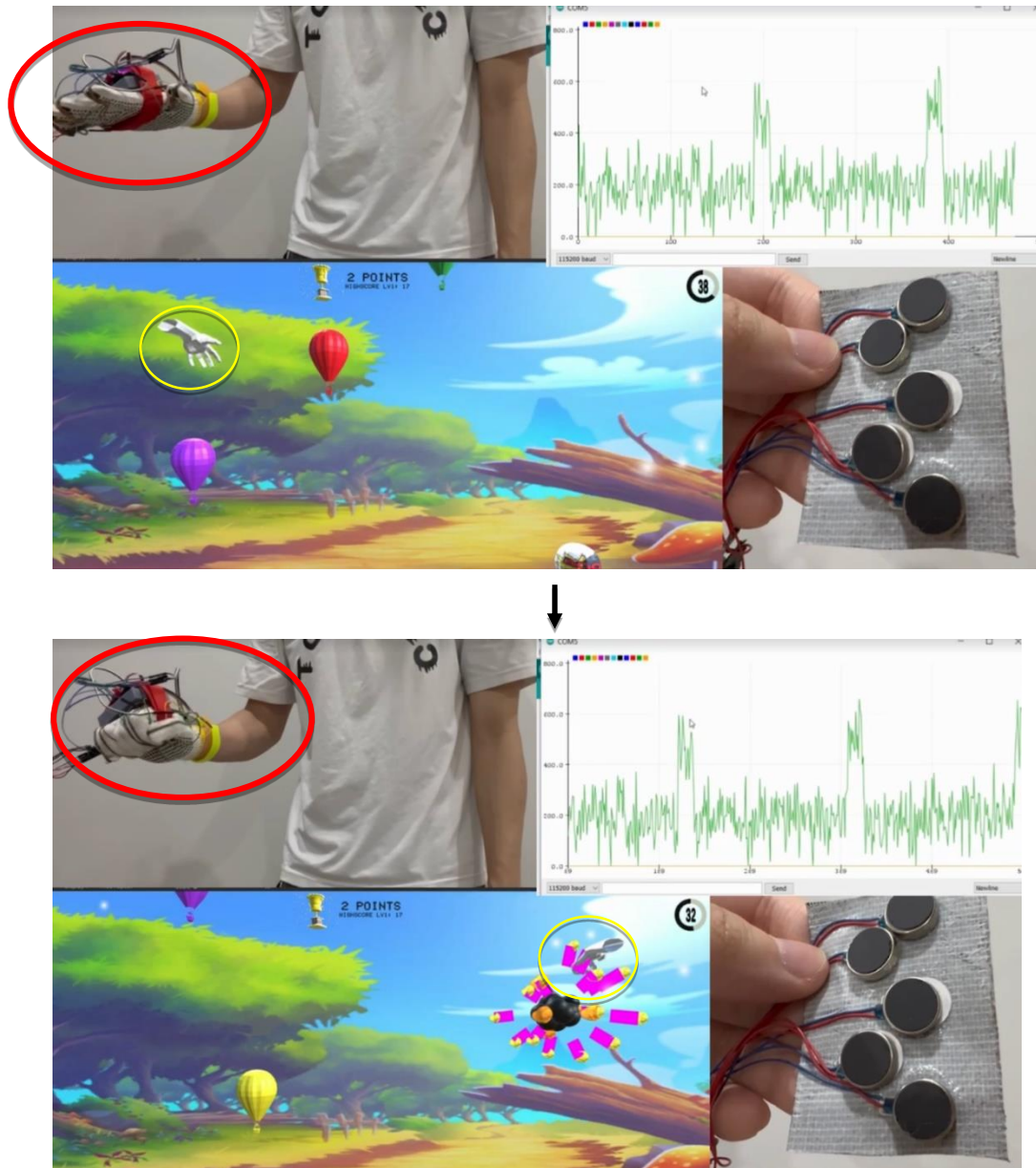


Figure 4.32: In-Game Condition 1

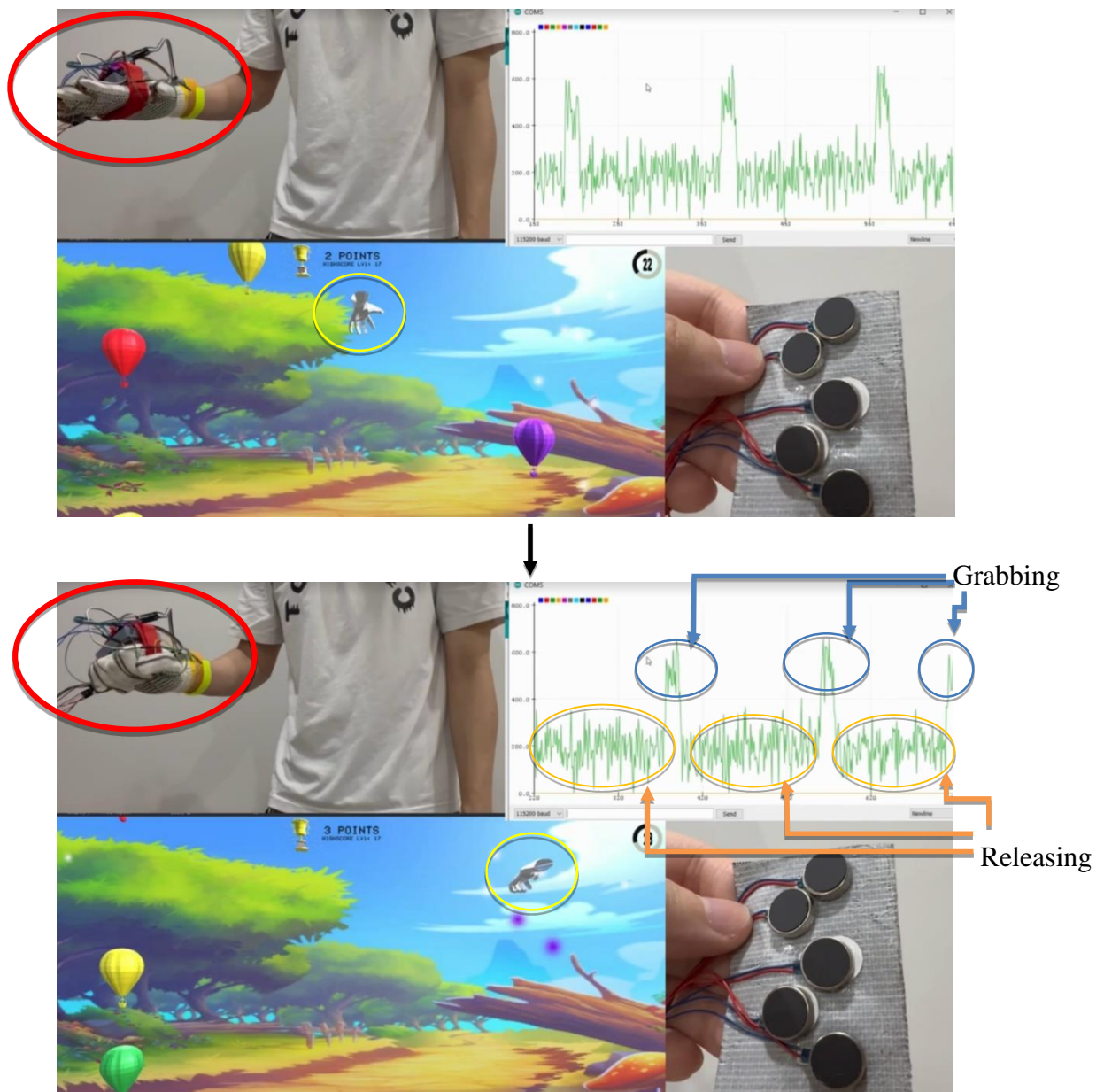


Figure 4.33: In-Game Condition 2

Figure 4.32 and Figure 4.33 show that the user was able to control the hand model in the game environment to move around and perform grabbing and releasing motions as desired. The graph beside it shows the analog reading changes caused by the bending and releasing of the graphene sensor every time the user performs the grabbing and releasing motion. The haptic actuators also successfully received signals sent from the game environment every time the Boom Balloon was grabbed and provided haptic feedback to the user.

4.6 Summary

The fabricated graphene thread sensor is capable of sensing the user's finger bending and releasing motion, and the user successfully controls the quaternion and acceleration motion via the MPU6050. All hardware components have been combined and soldered onto one PCB, following the desired hardware algorithm design. The VR game, with numerous fascinating and fancy features, has been designed to make the rehabilitation process more interesting. The serial and wireless communication algorithm successfully and smoothly transmits all data values throughout the project bi-directionally.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

Throughout this project, a graphene thread sensor has been produced and fabricated from normal polyester thread by using graphene conductive ink. Electronic gadgets such as MPU6050, DOIT ESP32 DEV KIT V1, LiPo Battery, and DC motor vibration have been studied thoroughly. Coding algorithms, such as wireless and serial communication, have been integrated into the glove for data transmission. Game design coding was used to design a fancy, interesting, and bug-free game. Applications and tools, such as Blender, Unity Editor, Arduino, SolidWorks, solder gun, ultrasonic machine, vacuum oven, Cura, and others, have been utilized to complete the project.

5.2 Recommendations for future work

The power source selected to power up the ESP32 for wireless connection is a LiPo battery, as it is rechargeable, compact and lightweight. However, it has some drawbacks, such as its voltage range fluctuating from 4.2V when fully charged to 3.0V when fully discharged, despite its nominal voltage being 3.7V. This fluctuation can affect the angle data mapped from the analog reading of the output voltage of the voltage divider rule, which is controlled by the graphene thread sensor powered by the LiPo battery. To address this issue, it is recommended to build and install an external voltage regulator of 3.3V after the LiPo battery to ensure the stability of the voltage output from the LiPo battery.

Secondly, it has been observed that the graphene thread sensor absorbs moisture from the surrounding air, causing changes in its resistance range. This is undesirable, particularly when the mapped angle data highly depends on the resistance of the graphene thread sensor. To solve this issue, it is recommended to add insulation at the outer layer of the sensors or explore a suitable chemical to add to the graphene conductive ink to prevent the graphene from absorbing moisture from the surrounding air. Furthermore, automating the production

process of the graphene thread sensor is recommended to ensure the stability and consistency of the sensors.

Thirdly, to make the entire game station more complete and engaging, adding more games in the future is recommended. The game should be designed to be easy to use, allowing the patient or user to get started quickly.

REFERENCES

A. Mulder, S. Fels and K. Mase, (1997). *Empty-handed gesture analysis in MAX/FTS*. Proc. AIMI Int. Workshop Kansei---Technol. Emotion, pp. 87-90 ed.

Andersen, S.M. and Thorpe, J.S. (2009). An IF–THEN theory of personality: Significant others and the relational self. *Journal of Research in Personality*, 43(2), pp.163–170. doi:10.1016/j.jrp.2008.12.040.

Aw, K., Budd, J. and Wilshaw-Sparkes, T. (2022). Data Glove Using Soft and Stretchable Piezoresistive Sensors. *Micromachines*, [online] 13(3), p.372. doi:10.3390/mi13030372.

Bailenson, J.N., Yee, N., Merget, D. and Schroeder, R. (2006). The Effect of Behavioral Realism and Form Realism of Real-Time Avatar Faces on Verbal Disclosure, Nonverbal Disclosure, Emotion Recognition, and Copresence in Dyadic Interaction. *Presence: Teleoperators and Virtual Environments*, 15(4), pp.359–372. doi:10.1162/pres.15.4.359.

Biocca F., Harms C., Gregg J. (2001). ‘The networked minds measure of social presence: pilot test of the factor structure and concurrent validity,’ in 4th Annual International Workshop on Presence, Philadelphia, PA, 1–9. (n.d.).

Borrego A., Latorre J., Llorens R., Alcañiz M., Noé E. (2016). *Feasibility of a walking virtual reality system for rehabilitation: objective and subjective parameters*. *J. Neuroeng. Rehabil.* 13:68. 10.1186/s12984-016-0174-171.

Botella, C., Fernández-Álvarez, J., Guillén, V., García-Palacios, A. and Baños, R. (2017). Recent Progress in Virtual Reality Exposure Therapy for Phobias: A Systematic Review. *Current Psychiatry Reports*, 19(7). doi:10.1007/s11920-017-0788-4.

Brown, A. and Green, T. (2016). Virtual Reality: Low-Cost Tools and Resources for the Classroom. *TechTrends*, 60(5), pp.517–519. doi:10.1007/s11528-016-0102-z.

Burdea G. C and Coiffet P. (2003). *Virtual Reality Technology Vol. 1 Hoboken, NJ: John Wiley & Sons.*

Burdea, G., Patounakis, G., Popescu, V. and Weiss, R.E. (1999). Virtual reality-based training for the diagnosis of prostate cancer. *IEEE Transactions on Biomedical Engineering*, 46(10), pp.1253–1260. doi:10.1109/10.790503.

Caeiro-Rodríguez, M., Otero-González, I., Mikic-Fonte, F.A. and Llamas-Nistal, M. (2021a). A Systematic Review of Commercial Smart Gloves: Current Status and Applications. *Sensors*, [online] 21(8), p.2667. doi:10.3390/s21082667.

Caeiro-Rodríguez, M., Otero-González, I., Mikic-Fonte, F.A. and Llamas-Nistal, M. (2021b). A Systematic Review of Commercial Smart Gloves: Current Status and Applications. *Sensors*, 21(8), p.2667. doi:10.3390/s21082667.

Castelvecchi, D. (2016). Low-cost headsets boost virtual reality's lab appeal. *Nature*, 533(7602), pp.153–154. doi:10.1038/533153a.

Culbertson, H., Schorr, S.B. and Okamura, A.M. (2018). Haptics: The Present and Future of Artificial Touch Sensation. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1), pp.385–409. doi:10.1146/annurev-control-060117-105043.

DELP, S.L. and ZAJAC, F.E. (1992). Force- and Moment-Generating Capacity of Lower-Extremity Muscles Before and After Tendon Lengthening. *Clinical Orthopaedics and Related Research*, &NA;(284), p.247??259. doi:10.1097/00003086-199211000-00035.

Dipietro, L., Sabatini, A.M. and Dario, P. (2008). A Survey of Glove-Based Systems and Their Applications. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, [online] 38(4), pp.461–482. doi:10.1109/TSMCC.2008.923862.

Gillis, A. (2022). *What is IoT (Internet of Things) and How Does it Work? - Definition from TechTarget.com*. [online] IoT Agenda. Available at: <https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT>.

Gonçalves L.M.G. and Musse, S.R. (). *Proceedings, XV Brazilian Symposium on Computer Graphics and Image Processing : 7-10 October 2002, Fortaleza-CE, Brazil*. [Piscataway, N.J.]: IEEE.

Greenleaf, W.J. (1996). Developing the tools for practical VR applications [Medicine]. *IEEE Engineering in Medicine and Biology Magazine*, [online] 15(2), pp.23–30. doi:10.1109/51.486714.

Guo, J. *et al.*, 2019. Stretchable and Highly Sensitive Optical Strain Sensors for Human-Activity Monitoring and Healthcare. *ACS applied materials & interfaces*, 21(9), p. 33589–33598. (n.d.).

Gupta, S., Morris, D., Patel, S.N. and Tan, D. (2013). AirWave. *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*. doi:10.1145/2493432.2493463.

Heeter, C. (2000). Interactivity in the Context of Designed Experiences. *Journal of Interactive Advertising*, 1(1), pp.3–14. doi:10.1080/15252019.2000.10722040.

Llorens R., Noé E., Ferri J., Alcañiz M. (2014). *Virtual reality-based telerehabilitation program for balance recovery. A pilot study in hemiparetic individuals with acquired brain injury*. *Brain Inj.* 28:169.

Lombard, M. and Ditton, T. (2006). At the Heart of It All: The Concept of Presence. *Journal of Computer-Mediated Communication*, [online] 3(2). doi:10.1111/j.1083-6101.1997.tb00072.x.

Long, B., Seah, S.A., Carter, T. and Subramanian, S. (2014). Rendering volumetric haptic shapes in mid-air using ultrasound. *ACM Transactions on Graphics*, 33(6), pp.1–10. doi:10.1145/2661229.2661257.

Loomis, J.M., Blascovich, J.J. and Beall, A.C. (1999). Immersive virtual environment technology as a basic research tool in psychology. *Behavior Research Methods, Instruments, & Computers*, 31(4), pp.557–564. doi:10.3758/bf03200735.

M. Minoh, H. Obara, T. Funtatomi, M. Toyura and K. Kakusho (2007). *Direct manipulation of 3D virtual objects by actors for recording live video content*. Proc. Conf. Inf. Res. Dev. Knowl. Soc. Infrastructure, pp. 11-18 ed.

Minderer, M., Harvey, C.D., Donato, F. and Moser, E.I. (2016a). Virtual reality explored. *Nature*, 533(7603), pp.324–325. doi:10.1038/nature17899.

Minderer, M., Harvey, C.D., Donato, F. and Moser, E.I. (2016b). Virtual reality explored. *Nature*, 533(7603), pp.324–325. doi:10.1038/nature17899.

P. Coiffet and G. C. Burdea (2003). *Virtual Reality Technology*, New York: Wiley.

Rashid, A. and Hasan, O. (2019). Wearable technologies for hand joints monitoring for rehabilitation: A survey. *Microelectronics Journal*, 88, pp.173–183. doi:10.1016/j.mejo.2018.01.014.

Rathore, A. (2019). *What Is A Piezoelectric Sensor?* [online] Electronics For You. Available at: <https://www.electronicsforu.com/technology-trends/learn-electronics/piezoelectric-sensor-basics> [Accessed 8 Sep. 2022].

Sensor, F.P. (2020). *Piezoresistive Pressure Sensors vs. Capacitive Pressure Sensors - Advantages & Disadvantages*. [online] PressureSensorFinder.com. Available at: <https://www.pressuresensorfinder.com/piezoresistive-pressure-sensors-vs-capacitive-pressure-sensors-advantages-disadvantages/>.

Shtarbanov, A. and Bove Jr., V.M. (2018). Free-Space Haptic Feedback for 3D Displays via Air-Vortex Rings. *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*. doi:10.1145/3170427.3188622.

Singh, G. (2002). Surrendering control. *IEEE Computer Graphics and Applications*, 22(4), pp.4–5. doi:10.1109/mcg.2002.1016690.

Slater, M. (2009). Place illusion and plausibility can lead to realistic behaviour in immersive virtual environments. *Philosophical Transactions of the Royal Society B: Biological Sciences*, [online] 364(1535), pp.3549–3557. doi:10.1098/rstb.2009.0138.

Sodhi, R., Poupyrev, I., Glisson, M. and Israr, A. (2013). AIREAL. *ACM Transactions on Graphics*, 32(4), pp.1–10. doi:10.1145/2461912.2462007.

Sundar, S.S., Xu, Q. and Bellur, S. (2010). Designing interactivity in media interfaces. *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*. [online] doi:10.1145/1753326.1753666.

Time. (n.d.). *Facebook Buying Oculus Virtual-Reality Company for \$2 Billion*. [online] Available at: <http://time.com/37842/facebook-oculus-rift> [Accessed 16 Jan. 2022].

Ware, C., Arthur, K. and Booth, K.S. (1993). Fish tank virtual reality. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '93*. doi:10.1145/169059.169066.

Yeh, S.-C., Hwang, W.-Y., Huang, T.-C., Liu, W.-K., Chen, Y.-T. and Hung, Y.-P. (2012a). *A Study for the Application of Body Sensing in Assisted Rehabilitation Training*. [online] IEEE Xplore. doi:10.1109/IS3C.2012.240.

Yeh, S.-C., Hwang, W.-Y., Huang, T.-C., Liu, W.-K., Chen, Y.-T. and Hung, Y.-P. (2012b). *A Study for the Application of Body Sensing in Assisted Rehabilitation Training*. [online] IEEE Xplore. doi:10.1109/IS3C.2012.

APPENDICES

Appendix A: Overall Arduino Code of First ESP32

```

#include <elapsedMillis.h>
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif
#include "esp_now.h"
#include "WiFi.h"

MPU6050 mpu;
MPU6050 mpul(0x69); // <-- use for AD0 high

#define OUTPUT_READABLE_QUATERNION
#define OUTPUT_READABLE_WORLDACCEL
#define INTERRUPT_PIN 2 // use pin 2 on Arduino Uno & most boards
#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;

// MPU control/status vars
bool dmpReady = false; // set true if DMP init was successful
uint8_t mpuIntStatus; // holds actual interrupt status byte from MPU
uint8_t devStatus; // return status after each device operation (0 = success, != 0 = error)
uint16_t packetSize; // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount; // count of all bytes currently in FIFO

// orientation/motion vars
Quaternion q; // [w, x, y, z] quaternion container
VectorInt16 aa; // [x, y, z] accel sensor measurements
VectorInt16 aaReal; // [x, y, z] gravity-free accel sensor measurements
VectorInt16 aaWorld; // [x, y, z] world-frame accel sensor measurements
VectorFloat gravity; // [x, y, z] gravity vector
float euler[3]; // [psi, theta, phi] Euler angle container
float ypr[3]; // [yaw, pitch, roll] yaw/pitch/roll container and gravity vector

// MPU1 control/status vars
bool dmpReady1 = false; // set true if DMP init was successful
uint8_t mpulIntStatus; // holds actual interrupt status byte from MPU
uint8_t devStatus1; // return status after each device operation (0 = success, != 0 = error)
uint16_t packetSize1; // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount1; // count of all bytes currently in FIFO
uint8_t fifoBuffer1[64]; // FIFO storage buffer

// MPU1 orientation/motion vars
Quaternion q1; // [w, x, y, z] quaternion container
VectorInt16 aal; // [x, y, z] accel sensor measurements
VectorInt16 aaReal1; // [x, y, z] gravity-free accel sensor measurements
VectorInt16 aaWorld1; // [x, y, z] world-frame accel sensor measurements
VectorFloat gravity1; // [x, y, z] gravity vector
float euler1[3]; // [psi, theta, phi] Euler angle container

```

```

float yprl[3];          // [yaw, pitch, roll] yaw/pitch/roll container and gravity vector

const int Thumb = 32;
const int Index = 35;
const int Middle = 34;
const int Ring = 39;
const int Pinky = 36;
const int vibration1 = 33;
const int vibration2 = 25;
const int vibration3 = 26;
const int vibration4 = 27;
const int vibration5 = 14;
const int MPU1 = 2;
const int MPU2 = 4;|
float angle1;
float ag1;
float angle2;
float ag2;
float angle3;
float ag3;
float angle4;
float ag4;
float angle5;
float ag5;

uint8_t broadcastAddress[] = {0xE0, 0x5A, 0x1B, 0x77, 0x18, 0x98};
typedef struct struct_message{
    char v;
    float t;
    float m;
    float i;
    float r;
    float p;
    float wl;
    float xl;
    float yl;
    float zl;
    float aax2;
    float aay2;
    float aaz2;

}struct_message;

struct_message My_data;

void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len){
    memcpy(&My_data, incomingData, sizeof(My_data));
    Serial.print(My_data.v);
}

esp_now_peer_info_t peerInfo;

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status){
    // Serial.print("\r\nLast Packet Send Status:\t");
    // Serial.print(status == ESP_NOW_SEND_SUCCESS? "Delivery Success\n" : "Delivery Fail\n");
}

int IMU_CHECK_INTERVAL_MSEC = 10;
elapsedMillis sinceLastIMUcheck;

```

```

// =====
// ===                INTERRUPT DETECTION ROUTINE                ===
// =====

volatile bool mpuInterrupt = false;    // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}

volatile bool mpulInterrupt = false;    // indicates whether MPU interrupt pin has gone high
void dmpDataReady1() {
    mpulInterrupt = true;
}

// =====
// ===                INITIAL SETUP                ===
// =====

void setup() {

    pinMode(MPU1, OUTPUT);
    pinMode(MPU2, OUTPUT);
    digitalWrite(MPU1, HIGH);
    digitalWrite(MPU2, HIGH);
    pinMode(vibration1, OUTPUT);
    pinMode(vibration2, OUTPUT);
    pinMode(vibration3, OUTPUT);
    pinMode(vibration4, OUTPUT);
    pinMode(vibration5, OUTPUT);
    digitalWrite(vibration1, LOW);
    digitalWrite(vibration2, LOW);
    digitalWrite(vibration3, LOW);
    digitalWrite(vibration4, LOW);
    digitalWrite(vibration5, LOW);

    sinceLastIMUCheck = 0;
    Serial.begin(115200);
    WiFi.mode(WIFI_STA);
    if(esp_now_init() != ESP_OK){
        Serial.print("Error initializing ESP-NOW");
        return;
    }
    esp_now_register_send_cb(OnDataSent);
    esp_now_register_rcv_cb(OnDataRecv);
    memcpy(peerInfo.peer_addr, broadcastAddress, 6);
    peerInfo.channel = 0 ;
    peerInfo.encrypt = false;

    if (esp_now_add_peer(&peerInfo) != ESP_OK){
        Serial.print("Failed to add peer");
        return;
    }

}

```

```

// join I2C bus (I2Cdev library doesn't do this automatically)
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    Wire.begin();
    Wire.setClock(400000); |
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
    Fastwire::setup(400, true);
#endif

// initialize serial communication
// (115200 chosen because it is required for Teapot Demo output, but it's
// really up to you depending on your project)
Serial.begin(115200);
while (!Serial); // wait for Leonardo enumeration, others continue immediately

// initialize device
Serial.println(F("Initializing I2C devices..."));
mpu.initialize();
mpul.initialize();
//    pinMode(INTERRUPT_PIN, INPUT);

// verify connection
Serial.println(F("Testing device connections..."));
Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));
Serial.println(mpul.testConnection() ? F("MPU6050.1 connection successful") : F("MPU6050.1 connection failed"));

// load and configure the DMP
Serial.println(F("Initializing DMP..."));
devStatus = mpu.dmpInitialize();
Serial.println(F("Initializing DMP1..."));
devStatus1 = mpul.dmpInitialize();

// supply gyro offsets, scaled for min sensitivity
mpu.setXGyroOffset(220);
mpu.setYGyroOffset(76);
mpu.setZGyroOffset(-85);
mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

mpul.setXGyroOffset(220);
mpul.setYGyroOffset(76);
mpul.setZGyroOffset(-85);
mpul.setZAccelOffset(1788);

// make sure it worked (returns 0 if so)
if (devStatus == 0) {
    // Calibration Time: generate offsets and calibrate our MPU6050
    mpu.CalibrateAccel(6);
    mpu.CalibrateGyro(6);
    mpu.PrintActiveOffsets();
    // turn on the DMP, now that it's ready
    Serial.println(F("Enabling DMP..."));
    mpu.setDMPEntered(true);

    // enable Arduino interrupt detection
    Serial.print(F("Enabling interrupt detection (Arduino external interrupt "));
    Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));
    Serial.println(F(")..."));
    attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
    mpuIntStatus = mpu.getIntStatus();

    // set our DMP Ready flag so the main loop() function knows it's okay to use it
    Serial.println(F("DMP ready! Waiting for first interrupt..."));
    dmpReady = true;

```

```

    // get expected DMP packet size for later comparison
    packetSize = mpu.dmpGetFIFOpacketSize();
} else {
    // ERROR!
    // 1 = initial memory load failed
    // 2 = DMP configuration updates failed
    // (if it's going to break, usually the code will be 1)
    Serial.print(F("DMP Initialization failed (code "));
    Serial.print(devStatus);
    Serial.println(F(""));
}

if (devStatus1 == 0) {
    // Calibration Time: generate offsets and calibrate our MPU6050
    mpul.CalibrateAccel(6);
    mpul.CalibrateGyro(6);
    mpul.PrintActiveOffsets();
    // turn on the DMP, now that it's ready
    Serial.println(F("Enabling DMP1..."));
    mpul.setDMPEntered(true);

    // enable Arduino interrupt detection
    Serial.print(F("Enabling interrupt detection (Arduino external interrupt "));
    Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));
    Serial.println(F(")..."));
    attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady1, RISING);
    mpulIntStatus = mpul.getIntStatus();

    // set our DMP Ready flag so the main loop() function knows it's okay to use it
    Serial.println(F("DMP1 ready! Waiting for first interrupt..."));
    dmpReady1 = true;

    // get expected DMP packet size for later comparison
    packetSize1 = mpul.dmpGetFIFOpacketSize();
} else {
    // ERROR!
    // 1 = initial memory load failed
    // 2 = DMP configuration updates failed
    // (if it's going to break, usually the code will be 1)
    Serial.print(F("DMP1 Initialization failed (code "));
    Serial.print(devStatus1);
    Serial.println(F(""));
}

```



```

// =====
// ===                                MAIN PROGRAM LOOP                                ===
// =====

void loop() {

  angle1 = analogRead(Thumb);
  ag1 = 0.00015*angle1*angle1 - 0.555*angle1 + 495;
  angle2 = analogRead(Index);
  ag2 = 0.0006*angle2*angle2 - 0.51*angle2 + 90;
  angle3 = analogRead(Middle);
  ag3 = 0.00053571428571429*angle3*angle3 - 0.069642857142857*angle3 - 123.75;
  angle4 = analogRead(Ring);
  ag4 = 0.00075*angle4*angle4 - 1.275*angle4 + 540;
  angle5 = analogRead(Pinky);
  ag5 = 0.0008*angle5*angle5 - 1.144*angle5 - 424.29;

  if (My_data.v == 'a'){
    digitalWrite(vibration1,HIGH);
    digitalWrite(vibration2,HIGH);
    digitalWrite(vibration3,HIGH);
    digitalWrite(vibration4,HIGH);
    digitalWrite(vibration5,HIGH);
    delay(500);

    digitalWrite(vibration1,LOW);
    digitalWrite(vibration2,LOW);
    digitalWrite(vibration3,LOW);
    digitalWrite(vibration4,LOW);
    digitalWrite(vibration5,LOW);
  }
  // if programming failed, don't try to do anything
  if (!dmpReady) return;
  // read a packet from FIFO
  if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer)) { // Get the Latest packet
    #ifdef OUTPUT_READABLE_QUATERNION
      // display quaternion values in easy matrix form: w x y z
      mpu.dmpGetQuaternion(&q, fifoBuffer);

    #endif

    #ifdef OUTPUT_READABLE_WORLDACCEL
      // display initial world-frame acceleration, adjusted to remove gravity
      // and rotated based on known orientation from quaternion
      mpu.dmpGetQuaternion(&q, fifoBuffer);
      mpu.dmpGetAccel(&aa, fifoBuffer);
      mpu.dmpGetGravity(&gravity, &q);

      mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
      mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);

    #endif

    // blink LED to indicate activity
    blinkState = !blinkState;
    digitalWrite(LED_PIN, blinkState);
  }

  if (!dmpReady1) return;
  // read a packet from FIFO
  if (mpu1.dmpGetCurrentFIFOPacket(fifoBuffer1)) { // Get the Latest packet
    #ifdef OUTPUT_READABLE_QUATERNION
      // display quaternion values in easy matrix form: w x y z
      mpu1.dmpGetQuaternion(&q1, fifoBuffer1);

    #endif

```

```
#ifndef OUTPUT_READABLE_WORLDACCEL
    // display initial world-frame acceleration, adjusted to remove gravity
    // and rotated based on known orientation from quaternion
    mpul.dmpGetQuaternion(&q1, fifoBuffer1);
    mpul.dmpGetAccel(&aal, fifoBuffer1);
    mpul.dmpGetGravity(&gravity1, &q1);
    mpul.dmpGetLinearAccel(&aaReal1, &aal, &gravity1);
    mpul.dmpGetLinearAccelInWorld(&aaWorld1, &aaReal1, &q1);

#endif

    My_data.t = ag1;
    My_data.m = ag2;
    My_data.i = ag3;
    My_data.r = ag4;
    My_data.p = ag5;
    My_data.w1 = q.w ;
    My_data.x1 = q.x;
    My_data.y1 = q.y;
    My_data.z1 = q.z;
    My_data.aax2 = aaWorld1.x/5;
    My_data.aay2 = aaWorld1.y;
    My_data.aaz2 = aaWorld1.z/5;

    esp_err_t result = esp_now_send(broadcastAddress, (uint8_t*) &My_data, sizeof(My_data));

    // blink LED to indicate activity
    blinkState = !blinkState;
    digitalWrite(LED_PIN, blinkState);

    delay(50);
}
}
```

Appendix B: Overall Arduino Code of Second ESP32

```

#include "esp_now.h"
#include "WiFi.h"

uint8_t broadcastAddress[] = {0xE0, 0x5A, 0x1B, 0x77, 0x24, 0xF4};
typedef struct struct_message{
    char v;
    float t;
    float m;
    float i;
    float r;
    float p;
    float wl;
    float xl;
    float yl;
    float zl;
    float aax2;
    float aay2;
    float aaz2;
}struct_message;

struct_message My_data;

void onDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len){
    memcpy(&My_data, incomingData, sizeof(My_data));
    Serial.print(My_data.t);
    Serial.print(",");
    Serial.print(My_data.m);
    Serial.print(",");
    Serial.print(My_data.i);
    Serial.print(",");
    Serial.print(My_data.r);
    Serial.print(",");
    Serial.print(My_data.p);
    Serial.print(",");
    Serial.print(My_data.wl);
    Serial.print(",");
    Serial.print(My_data.xl);
    Serial.print(",");
    Serial.print(My_data.yl);
    Serial.print(",");
    Serial.print(My_data.zl);
    Serial.print(",");
    Serial.print(My_data.aax2);
    Serial.print(",");
    Serial.print(My_data.aay2);

    Serial.println(My_data.aaz2);
}

esp_now_peer_info_t peerInfo;

void onDataSent(const uint8_t *mac_addr, esp_now_send_status_t status){
    // Serial.print("\r\nLast Packet Send Status:\t");
    // Serial.print(status == ESP_NOW_SEND_SUCCESS? "Delivery Success\n" : "Delivery Fail\n");
}

void setup() {
    Serial.begin(115200);
    WiFi.mode(WIFI_STA);

    if(esp_now_init() != ESP_OK){
        Serial.print("Error initializing ESP-NOW");
        return;
    }
}

```

```
esp_now_register_send_cb(OnDataSent);
esp_now_register_recv_cb(OnDataRecv);

memcpy(peerInfo.peer_addr, broadcastAddress, 6);
peerInfo.channel = 0 ;
peerInfo.encrypt = false;

if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.print("Failed to add peer");
    return;
}

}

void loop() {

//
// strcpy(My_data.a, "ESP_NOW ON!");
// if (Serial.available()){
//   My_data.v = Serial.read();
//   esp_err_t result = esp_now_send(broadcastAddress, (uint8_t*) &My_data, sizeof(My_data));
// }
//
// if(result == ESP_OK){
//   Serial.println("Sending Confirmed");
// }
// else{
//   Serial.println("Sending error");
// }
// delay(50);
}
```

Appendix C: Code to Control Finger Bending and Releasing motion, Quaternion and Acceleration of Hand Model and Haptic Actuator Signal

```

using System.IO.Ports;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Linq;

public class FingerMotion : MonoBehaviour
{
    SerialPort stream = new SerialPort("COM5", 115200);
    public static FingerMotion Boom;
    public string strReceived;
    public string[] strData = new string[12];
    public string[] strData_received = new string[12];
    public float qw, qx, qy, qz, aax, aay, aaz;
    public float Speed = 10f;
    public float speedFactor = 15.0f;
    public static bool bool1;
    public Vector3 rotationOffset;
    public GameObject Sphere;
    public GameObject Rectangle;
    public Vector3 LastFramePos;
    public Transform Thumb_Root;
    public Transform Index_Root;
    public Transform Middle_Root;
    public Transform Ring_Root;
    public Transform Pinky_Root;

    public Transform Thumb_1;
    public Transform Thumb_2;

    public Transform Index_1;
    public Transform Index_2;
    public Transform Index_3;

    public Transform Middle_1;
    public Transform Middle_2;
    public Transform Middle_3;

    public Transform Ring_1;
    public Transform Ring_2;
    public Transform Ring_3;

    public Transform Pinky_1;
    public Transform Pinky_2;
    public Transform Pinky_3;

    public float F1, F2, F3, F4, F5;

    public GameObject Thumb_2_end;
    public GameObject Index_3_end;
    public GameObject Middle_3_end;

```

```

public GameObject Ring_3_end;
public GameObject Pinky_3_end;

// Start is called before the first frame update
void Awake()
{
    Boom = this;
}
void Start()
{
    //Start Streaming
    stream.Open();

    //Record current position of hand
    LastFramePos = transform.position;
}

// Update is called once per frame
void Update()
{
    strReceived = stream.ReadLine();

    strData = strReceived.Split(',');

    F1 = converter(strData[1]);
    F2 = converter(strData[1]);
    F3 = converter(strData[1]);
    F4 = converter(strData[1]);
    F5 = converter(strData[1]);

    if(F1 < 20 && F2 < 20 && F3 < 20 && F4 < 20 && F5 < 20)
    {
        bool1 = true;
    }

    if(F1 > 70 && F2 > 70 && F3 >70 && F4 > 70 && F5 > 70
&& bool1)
    {
        Exit.bool2 = true;
        SceneSwitcher.bool2 = true;
        B2.bool2 = true;
        SwitchSceneB1.bool2 = true;
        Level1.bool2 = true;
        Level2.bool2 = true;
        Level3.bool2 = true;
        Difficulty.bool2 = true;
        DestroyBalloon.bool2 = true;
        DestroyBoom.bool2 = true;
    }
    else
    {

```

```

        Exit.bool2 = false;
        SceneSwitcher.bool2 = false;
        B2.bool2 = false;
        SwitchSceneB1.bool2 = false;
        Level1.bool2 = false;
        Level2.bool2 = false;
        Level3.bool2 = false;
        Difficulty.bool2 = false;
        DestroyBalloon.bool2 = false;
        DestroyBoom.bool2 = false;
    }

    //update transform variable of all segment of fingers
    //localEulerAngles = store rotation of game object to parent
    coordinate system in the form of Vector 3 structure

    Thumb_1.localEulerAngles = new Vector3(-F1,
Thumb_1.localEulerAngles.y, Thumb_1.localEulerAngles.z);
    Thumb_2.localEulerAngles = new Vector3(-F1,
Thumb_2.localEulerAngles.y, Thumb_2.localEulerAngles.z);

    Index_1.localEulerAngles = new
Vector3(Index_1.localEulerAngles.x, Index_1.localEulerAngles.y,
F2);
    Index_2.localEulerAngles = new
Vector3(Index_2.localEulerAngles.x, Index_2.localEulerAngles.y,
F2);
    Index_3.localEulerAngles = new
Vector3(Index_3.localEulerAngles.x, Index_3.localEulerAngles.y,
F2);

    Middle_1.localEulerAngles = new
Vector3(Middle_1.localEulerAngles.x,
Middle_1.localEulerAngles.y, F3);
    Middle_2.localEulerAngles = new
Vector3(Middle_2.localEulerAngles.x,
Middle_2.localEulerAngles.y, F3);
    Middle_3.localEulerAngles = new
Vector3(Middle_3.localEulerAngles.x,
Middle_3.localEulerAngles.y, F3);

    Ring_1.localEulerAngles = new
Vector3(Ring_1.localEulerAngles.x, Ring_1.localEulerAngles.y,
F4);
    Ring_2.localEulerAngles = new
Vector3(Ring_2.localEulerAngles.x, Ring_2.localEulerAngles.y,
F4);
    Ring_3.localEulerAngles = new
Vector3(Ring_3.localEulerAngles.x, Ring_3.localEulerAngles.y,
F4);

    Pinky_1.localEulerAngles = new
Vector3(Pinky_1.localEulerAngles.x, Pinky_1.localEulerAngles.y,
F5);

```

```

        Pinky_2.localEulerAngles = new
Vector3(Pinky_2.localEulerAngles.x, Pinky_2.localEulerAngles.y,
F5);
        Pinky_3.localEulerAngles = new
Vector3(Pinky_3.localEulerAngles.x, Pinky_3.localEulerAngles.y,
F5);

        LastFramePos = transform.position;

        if (strData[5] != "" && strData[6] != "" &&
strData[7] != "" && strData[8] != "" && strData[9] != "" &&
strData[10] != "" && strData[11] != "")
        {
            strData_received[5] = strData[5];
            strData_received[6] = strData[6];
            strData_received[7] = strData[7];
            strData_received[8] = strData[8];
            strData_received[9] = strData[9];
            strData_received[10] = strData[10];
            strData_received[11] = strData[11];

            qw = float.Parse(strData_received[5]);
            qx = float.Parse(strData_received[6]);
            qy = float.Parse(strData_received[7]);
            qz = float.Parse(strData_received[8]);
            aax = float.Parse(strData_received[9]);
            aay = float.Parse(strData_received[10]);
            aaz = float.Parse(strData_received[11]);

            Rectangle.transform.localRotation = Quaternion.Lerp
(Rectangle.transform.localRotation, new Quaternion(qy, -qz, -
qx, qw), Time.deltaTime* speedFactor);
            Sphere.transform.localPosition += new Vector3
(aay/20, -aaz/15, 0) * Time.deltaTime;
        }
        Rectangle.transform.parent.transform.eulerAngles =
rotationOffset;
    }

    float converter (string angle)
    {
        float converter_angle;

        converter_angle = float.Parse(angle);
        if (converter_angle < 0)
        {
            converter_angle = 0;
        }
        else if(converter_angle > 90)
        {
            converter_angle = 90;
        }
    }

```



```
        return converter_angle;
    }

    public void HapAc()
    {
        stream.WriteLine("a");
        print("a");
    }
}
```

Appendix D: Exit Button Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Exit : MonoBehaviour
{
    bool value;
    public GameObject Camera;
    public static bool bool2;

    void OnTriggerEnter(Collider other)
    {
        value = true;
        Debug.Log(value);
    }

    void OnTriggerStay(Collider other)
    {
        value = true;
        Debug.Log(value);
    }

    void OnTriggerExit(Collider other)
    {
        value = false;
        Debug.Log(value);
    }

    void Update()
    {
        if(bool2 && value == true)
        {
            Camera.SetActive(false);
            value = false;
            bool2 = false;
        }
    }
}
```

Appendix E: Play Button Code/ Scene Switcher

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class SceneSwitcher : MonoBehaviour
{
    bool value;
    public static bool bool2;

    void OnTriggerEnter(Collider other)
    {
        value = true;
        Debug.Log(value);
    }

    void OnTriggerStay(Collider other)
    {
        value = true;
        Debug.Log(value);
    }

    void OnTriggerExit(Collider other)
    {
        value = false;
        Debug.Log(value);
    }

    void Update()
    {
        if(bool2 && value == true)
        {

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex
+ 1);
            value = false;
            bool2 = false;
        }
    }
}

```

Appendix F: Code to Ensure Hand Model Not Destroyed

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DontDestroy : MonoBehaviour
{
    public static DontDestroy Instance;
    void Start()
    {
        if(Instance != null)
        {
            Destroy(this.gameObject);
            return;
        }

        Instance = this;
        GameObject.DontDestroyOnLoad(this.gameObject);
    }
}
```

Appendix G: Back Button Code

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class SwitchSceneB1 : MonoBehaviour
{
    // Start is called before the first frame update
    bool value;
    public static bool bool2;
    // Start is called before the first frame update
    void OnTriggerEnter(Collider other)
    {
        value = true;
        Debug.Log(value);
    }

    void OnTriggerStay(Collider other)
    {
        value = true;
        Debug.Log(value);
    }

    void OnTriggerExit(Collider other)
    {
        value = false;
        Debug.Log(value);
    }

    void Update()
    {
        if(bool2 && value == true)
        {
            SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex
- 1);
            bool2 = false;
        }
    }
}

```

Appendix H: Difficulty Level Selection Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Selection : MonoBehaviour
{
    public static bool selection1;
    public static bool selection2;
    public static bool selection3;
    public static float MoveBalloonSpeed;
    public static float SpawnInterval;
    // Start is called before the first frame update

    // Update is called once per frame
    void Update()
    {
        if(selection1)
        {
            MoveBalloonSpeed = 2.0f;
            SpawnInterval = 3.0f;
            ScoreManager.HgSc1 = true;
            ScoreManager.HgSc2 = false;
            ScoreManager.HgSc3 = false;
        }
        if(selection2)
        {
            MoveBalloonSpeed = 5.0f;
            SpawnInterval = 1.5f;
            ScoreManager.HgSc1 = false;
            ScoreManager.HgSc2 = true;
            ScoreManager.HgSc3 = false;
        }
        if(selection3)
        {
            MoveBalloonSpeed = 9.0f;
            SpawnInterval = 1.0f;
            ScoreManager.HgSc1 = false;
            ScoreManager.HgSc2 = false;
            ScoreManager.HgSc3 = true;
        }
    }
}
```

Appendix I: Code to Spawn Balloon

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BalloonSpawner : MonoBehaviour
{
    public Transform[] SpawnPoints;
    public GameObject[] BalloonPrefabs;

    public float Interval;

    void Start()
    {
        Interval = Selection.SpawnInterval;
        InvokeRepeating("Spawn", 0.0f, Interval);
    }

    void Spawn()
    {
        int randBalloon = Random.Range(0,
BalloonPrefabs.Length);
        int randSpawnPoint = Random.Range(0,
SpawnPoints.Length);

        Instantiate(BalloonPrefabs[randBalloon],
SpawnPoints[randSpawnPoint].position, transform.rotation);
    }
}
```

Appendix J: Code to Define the Balloon Moving Speed

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MoveBalloon : MonoBehaviour
{
    public static float speed;

    // Update is called once per frame
    void Start()
    {
        speed = Selection.MoveBalloonSpeed;
    }
    void Update()
    {
        GetComponent<Rigidbody>().velocity = new Vector3 (0,
speed, 0);
    }
}
```


Appendix K: Code to Restrict Movement of Hand Model in Frame

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Restrict : MonoBehaviour
{
    // Start is called before the first frame update
    void Update()
    {
        // transform.Translate(Input.GetAxis("Horizontal") *
        Time.deltaTime *15f,0f,0f);
        // transform.Translate(0f,Input.GetAxis("Vertical") *
        Time.deltaTime *15f,0f);

        if (transform.position.x > 32)
        {
            transform.position = new Vector3(32,
transform.position.y, transform.position.z);
        }

        if (transform.position.x < 7)
        {
            transform.position = new Vector3(7,
transform.position.y, transform.position.z);
        }

        if (transform.position.y > -46)
        {
            transform.position = new
Vector3(transform.position.x, -46, transform.position.z);
        }

        if (transform.position.y < -55)
        {
            transform.position = new
Vector3(transform.position.x, -55, transform.position.z);
        }
        if(transform.position.z != -24)
        {
            transform.position = new
Vector3(transform.position.x, transform.position.y, -24);
        }
    }
}

```

Appendix L: Timer Code

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class Timer : MonoBehaviour
{
    public static Timer timer;
    [SerializeField] private Image Fill;
    [SerializeField] private TextMeshProUGUI Text;

    public int Duration;
    public int remainingDuration;
    public AudioClip TimesUp;
    public GameObject Ending;

    public void Awake()
    {
        timer = this;
    }

    // Start is called before the first frame update
    private void Start()
    {
        Being(Duration);
    }

    // Update is called once per frame
    public void Being(int Second)
    {
        remainingDuration = Second;
        StartCoroutine(UpdateTimer());
    }

    public IEnumerator UpdateTimer()
    {
        yield return new WaitForSeconds(3f);
        while(remainingDuration >= 0)
        {
            Text.text = $"{remainingDuration % 60:00}";
            Fill.fillAmount = Mathf.InverseLerp(0, Duration,
remainingDuration);
            remainingDuration--;
            yield return new WaitForSeconds(1f);
        }
        OnEnd();
    }

    public void OnEnd()
    {

```

```
        Text.text = "Time's Up";
        AudioSource.PlayClipAtPoint(TimesUp, new
Vector3(0,0,0));

GameObject.Find("Background").GetComponent<AudioSource>().enabl
ed = false;
        Ending.SetActive(true);
        Time.timeScale = 0f;
    }

    public void SubtractTime()
    {
        remainingDuration -= 4;
    }
}
}
```

Appendix M: Score Manager Code

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class ScoreManager : MonoBehaviour
{
    public static ScoreManager instance;
    public TextMeshProUGUI ScoreText;
    public TextMeshProUGUI HighScoreText;
    public static bool HgSc1;
    public static bool HgSc2;
    public static bool HgSc3;

    public int Score1 = 0;
    public int Score2 = 0;
    public int Score3 = 0;
    public int HighScore1 = 0;
    public int HighScore2 = 0;
    public int HighScore3 = 0;

    public void Awake()
    {
        instance = this;
    }

    // Start is called before the first frame update
    void Start()
    {
        if(HgSc1)
        {
            HighScore1 = PlayerPrefs.GetInt("HighScoreLv1", 0);
            ScoreText.text = Score1.ToString() + " POINTS";
            HighScoreText.text = "HIGHSCORE LV1: " +
HighScore1.ToString();
        }

        if(HgSc2)
        {
            HighScore2 = PlayerPrefs.GetInt("HighScoreLv2", 0);
            ScoreText.text = Score2.ToString() + " POINTS";
            HighScoreText.text = "HIGHSCORE LV2: " +
HighScore2.ToString();
        }

        if(HgSc3)
        {

```

```

        HighScore3 = PlayerPrefs.GetInt("HighScoreLv3", 0);
        ScoreText.text = Score3.ToString() + " POINTS";
        HighScoreText.text = "HIGHSCORE LV3: " +
HighScore3.ToString();

    }
}

// Update is called once per frame
public void AddPoint()
{
    if(HgSc1)
    {
        Score1 += 1;
        ScoreText.text = Score1.ToString() + " POINTS";
        if (HighScore1<Score1)
        {
            PlayerPrefs.SetInt("HighScoreLv1", Score1);
        }
    }

    if(HgSc2)
    {
        Score2 += 1;
        ScoreText.text = Score2.ToString() + " POINTS";
        if (HighScore2<Score2)
        {
            PlayerPrefs.SetInt("HighScoreLv2", Score2);
        }
    }

    if(HgSc3)
    {
        Score3 += 1;
        ScoreText.text = Score3.ToString() + " POINTS";
        if (HighScore3<Score3)
        {
            PlayerPrefs.SetInt("HighScoreLv3", Score3);
        }
    }
}
}

```

Appendix N: Code for In-Game Preparation Count Down

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class Countdown : MonoBehaviour
{
    public GameObject countdown;
    public TextMeshProUGUI CountdownManager;
    public AudioClip CountdownSound;

    // Start is called before the first frame update
    private void Start()
    {
        StartCoroutine(CountDownCoroutine());
    }

    // Update is called once per frame
    IEnumerator CountDownCoroutine()
    {
        yield return new WaitForSeconds(0.5f);
        CountdownManager.text = "3";
        AudioSource.PlayClipAtPoint(CountDownSound, new
Vector3(0, 0, 0));
        countdown.SetActive(true);

        yield return new WaitForSeconds(1.0f);
        // countdown.SetActive(false);
        CountdownManager.text = "2";
        AudioSource.PlayClipAtPoint(CountDownSound, new
Vector3(0, 0, 0));
        // AudioSource.PlaySound(CountDownSound);
        countdown.SetActive(true);

        yield return new WaitForSeconds(1.0f);
        // countdown.SetActive(false);
        CountdownManager.text = "1";
        AudioSource.PlayClipAtPoint(CountDownSound, new
Vector3(0, 0, 0));
        countdown.SetActive(true);

        yield return new WaitForSeconds(1.0f);
        countdown.SetActive(false);

        GameObject.Find("Background").GetComponent().enabled = true;

        GameObject.Find("Spawner").GetComponent<BalloonSpawner>().enabled = true;
    }
}

```

Appendix O: Destroy Normal Balloon Code

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DestroyBalloon : MonoBehaviour
{
    public AudioClip clip;
    public GameObject Effect;
    bool value = false;
    public static bool bool2;

    // Start is called before the first frame update
    void OnTriggerEnter(Collider other)
    {
        value = true;
        Debug.Log(value);
    }

    void OnTriggerStay(Collider other)
    {
        value = true;
        Debug.Log(value);
    }

    void OnTriggerExit(Collider other)
    {
        value = false;
        Debug.Log(value);
    }

    void Update()
    {
        if(bool2 && value == true)
        {
            Instantiate(Effect, new
            Vector3(transform.position.x-5 ,transform.position.y -3,
            transform.position.z +4), Quaternion.identity);
            Destroy(gameObject);
            PlaySound();
            ScoreManager.instance.AddPoint();
            bool2 = false;
        }
    }

    void PlaySound()
    {
        AudioSource.PlayClipAtPoint(clip, transform.position);
    }
}

```

Appendix P: Destroy Boom Balloon Code

```

using System.IO.Ports;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Linq;

public class DestroyBoom : MonoBehaviour
{
    public AudioClip BoomSound;
    public GameObject BoomEffect;
    bool value = false;
    public static bool bool2;

    void OnTriggerEnter(Collider other)
    {
        value = true;
        Debug.Log(value);
    }

    void OnTriggerStay(Collider other)
    {
        value = true;
        Debug.Log(value);
    }

    void OnTriggerExit(Collider other)
    {
        value = false;
        Debug.Log(value);
    }

    // Update is called once per frame
    void Update()
    {
        if(bool2 && value == true)
        {
            Instantiate(BoomEffect, new
            Vector3(transform.position.x-5 ,transform.position.y -3,
            transform.position.z +4), Quaternion.identity);
            Destroy(gameObject);
            PlaySound();
            Timer.timer.SubstractTime();
            FingerMotion.Boom.HapAc();
            bool2 = false;
            Cube.bool1 = false;
        }
    }

    void PlaySound()
    {
        AudioSource.PlayClipAtPoint(BoomSound, new Vector3(19,-
51,-14));
    }
}

```


Appendix Q: Ending Menu Code

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using UnityEngine.SceneManagement;

public class Ending : MonoBehaviour
{
    public TextMeshProUGUI Greeting;
    public TextMeshProUGUI Quit;
    public TextMeshProUGUI GameSelection;
    public TextMeshProUGUI Restart;
    public GameObject Camera3;
    // Start is called before the first frame update

    // Update is called once per frame
    void Update()
    {
        Greeting.text = "Game Over";
        Quit.text = "Exit (Esc)";
        GameSelection.text = "Game Selection (G)";
        Restart.text = "Restart (R)";

        if(Input.GetKeyDown("r"))
        {
            Time.timeScale = 1f;

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex
);
        }

        if(Input.GetKeyDown("g"))
        {
            Time.timeScale = 1f;

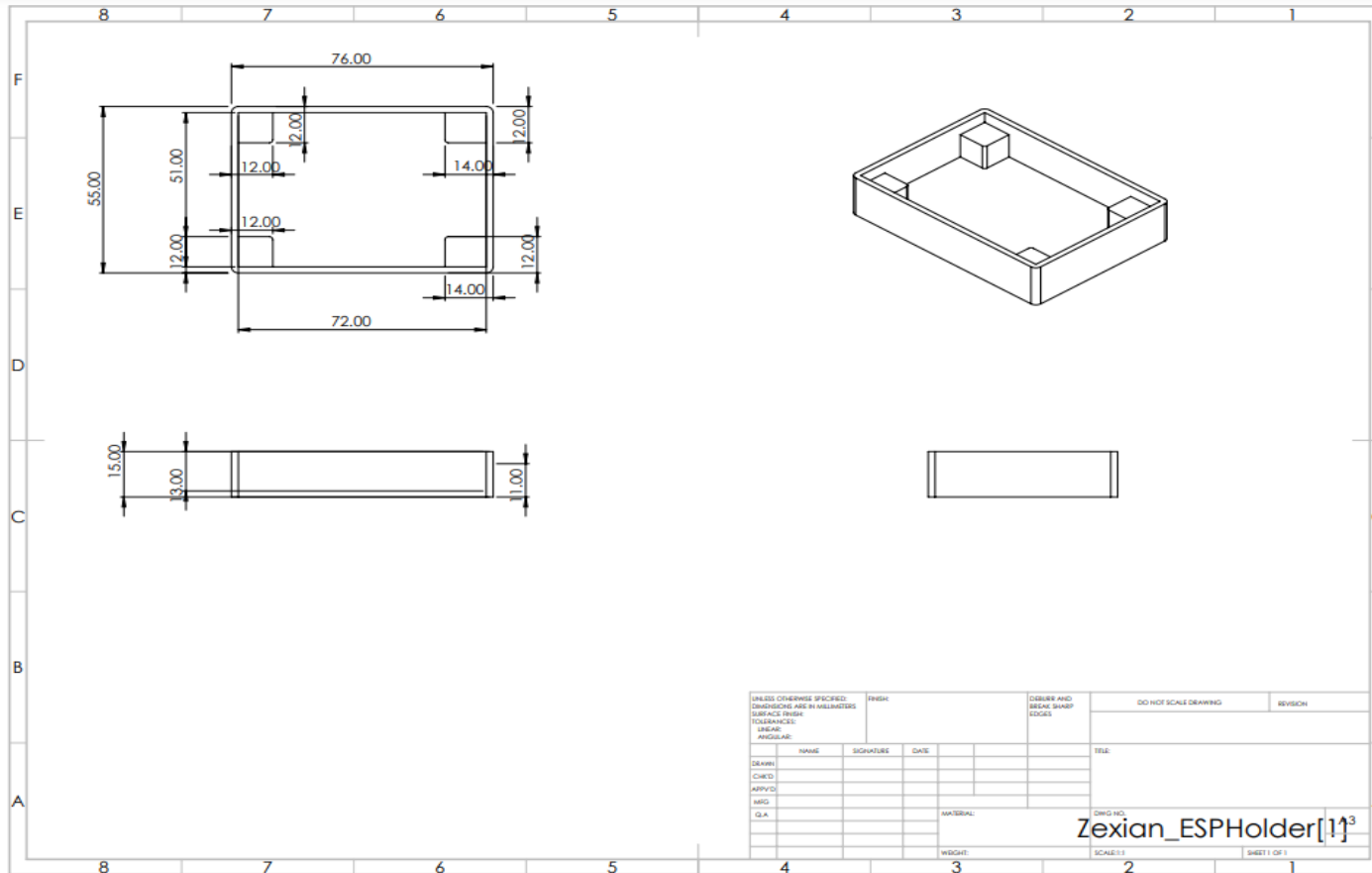
SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex
- 1);

            Selection.selection1 = false;
            Selection.selection2 = false;
            Selection.selection3 = false;
        }

        if(Input.GetKeyDown(KeyCode.Escape))
        {
            Camera3.SetActive(false);
        }
    }
}

```

Appendix R: ESP32 Holder



Appendix S: MPU6050 Holder

