CSB1

FYP Final Report

Minority Report: Control remote devices with air gestures in total privacy

by

CHAN Cheuk Wai and LO Shih-Heng

CSB1

Advised by

Prof. Brahim BENSAOU (彭卻邦)

Submitted in partial fulfillment of the requirements for COMP 4971

in the

Department of Computer Science

The Hong Kong University of Science and Technology

2022-2023

Date of submission: April 20, 2023

Abstract

This project aims to develop a smart glove that can be used to control electronic devices wirelessly. We have designed a physical smart glove embedded with sensors and wireless communication technologies. The data collected from the smart glove will be transmitted to a server that is capable of processing the sensor data and classify these time-series data into pre-defined gestures with machine learning. These gestures can then be sent to application and perform the task we intended with the gesture in an intuitive way. The project includes several applications to demonstrate the capabilities of the system, including control of PowerPoint and LED control using gestures received. Overall, the smart glove offers a user-friendly and intuitive way to control electronic devices wirelessly, with potential applications in accessibility and home automation.

1 Introduction

1.1 Overview

The invention of electronic devices has brought convenience to the lives of people from all walks of life. However, controlling these devices can be challenging, particularly for individuals with little knowledge of the interfaces or the devices. A smart glove offers a solution to this problem, allowing users to control devices using natural hand gestures. By embedding sensors and wireless communication technology into the glove, users can interact with their environment in a seamless and intuitive way, without the need for complex interfaces. This approach has the potential to improve the accessibility and usability of smart homes, allowing users to control devices such as lights, fans, and entertainment systems with ease. Smart gloves can be customized to fit the specific needs of individual users, such as adjusting gesture sensitivity or mapping specific gestures to control different devices. These are the features we aimed to build in this project.

1.2 Objectives

The goal of this project is to design and implement a smart glove that can control devices in various ways, including controlling devices and assisting presentation. We hope to achieve this in a way that the user's natural gesture language can be reflected on the devices in real-time and make the gesture commands more humanized and natural.

To achieve our goal, we will mainly focus on the following objectives:

- Build a smart glove that is capable of collecting motion data and sending them to a server.
- Implement a server to receive motion data and perform classification of motion data into gestures
- Implement multiple application devices to receive commands and perform the commands according to the meaning

1.3 Literature Survey

Gesture Recognition has been used widely in different areas, prevailing in Human Machine Interface (HMI) and AR/VR gaming areas. Commercial devices include Wii controller, Leap Motion, Kinect, and CyberGlove. Two of the common ways they use to capture human gesture motion are vision-based and sensor-based. Vision-based systems use cameras to detect and recognize gestures, however, camera angle, limited range of space in which the user can perform, and lighting conditions are the limitations. On the other hand, sensor-based motion tracking uses sensors such as flex sensors to measure the extent of bending or flexing based on varied resistance. They tend to produce nonlinearity and cannot precisely present the motion of each finger. To understand and have a reference on how gesture motion can be captured, we have studied several related works, and they are discussed below.

1.3.1 CSB1 2021-2022: Home Automation with Hand Gestures

The FYP group that worked on this project [1] previously decided to utilize five flex sensors attached to the fingers and a nine-axis motion tracking device MPU-9250 for collecting data from the glove.

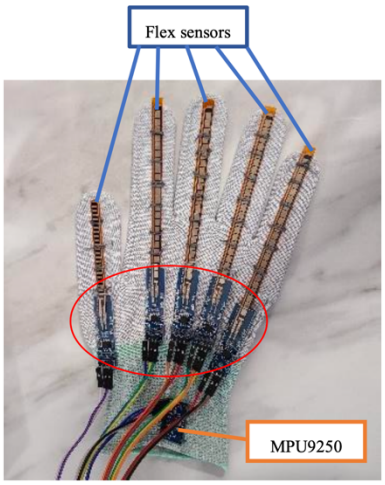


Figure 1 CSB1 2022-23 glove

The sensors on the glove will collect motion data and send it to the Arduino. These data include the bentness for each of the finger and the direction provided from the compass on MPU-9250. Afterwards, the Arduino will then send those data to a Raspberry Pi's COM port and save the data in a CSV file. It is then pipelined into a decision tree that they have previously trained. Those data will then be classified into six different gestures that they have defined.

In our opinion, there are some critical problems with this previous design. To begin with, the nine-axis motion tracking sensor is only used as a compass which can only determine the direction of the glove in north, south, east, and west. This limits the variety of some common gestures to be used. For instance, raising our thumb upwards cannot be differentiated from pointing our thumb downwards.

1.3.2 IMU Sensor-Based Hand Gesture Recognition for Human-Machine Interfaces

M. Kim et al. tried to train a 3D digit data set [2] where they drew the number in the air and used a single IMU (MPU6050) to record the acceleration with a button used to indicate the start and the end. They proposed an HGR algorithm that uses a Restricted Coulomb Energy (RCE) neural network with the distance measurement method used in RCE neural networks removed and that of Dynamic Time Warping (DTW) is employed to achieve high-performance gesture recognition.

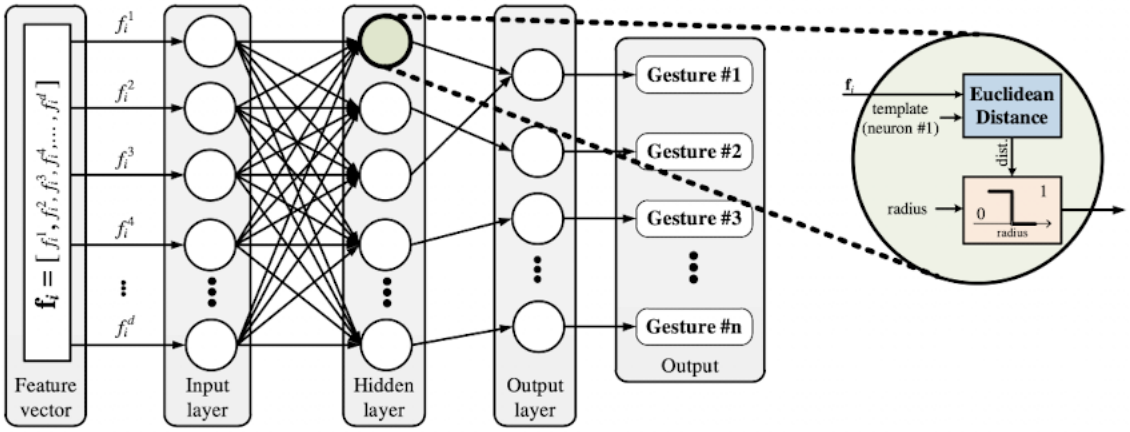


Figure 2 Structure of an RCE neural network

An RCE neural network consists of three layers: an input layer, a hidden layer, and an output layer. The input layer contains feature data and is connected in parallel to each neuron in the hidden layer. Each neuron has a center point and a radius, which forms an activation region similar to the shape of a sphere. In the feature space, the activation region creates a decision boundary to distinguish the data. As the data enter the input layer, each neuron calculates the distance between the input data and its center point. The distance is compared with the radius of each neuron to determine if that neuron is activated, and the results are passed to the output layer. The output layer uses the received information to output the label of the neuron that best matches the input data.

DTW is a technique for aligning two data sequences by warping them in a nonlinear fashion to match each other and then finding the distance between the data. This method is widely used in areas that deal with data that changes over time because it results in better performance than employing Euclidean distance on time-dependent data.

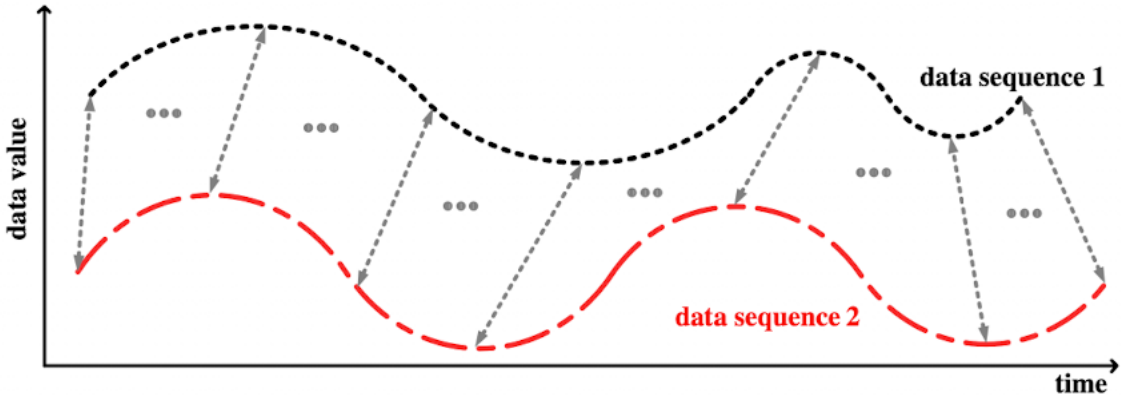


Figure 3 Dynamic Time Warping

Their model works great with handwriting-like input data and provides us with a great example of a simple and lightweight model. However, in order to make a glove that recognizes natural gesture language rather than handwriting, a single IMU is not capable of providing a complete view of gestures. Instead, multiple IMUs are needed to describe the temporal relationship between each finger and palm.

1.3.3 Real Time gesture Pattern Classification with IMU Data

A. Fu et al. have tried to use a single IMU to perform gesture classification[3] . They take the 3 dimensions of data for the gyroscope and another 3 from the accelerometer into account. Then transform them into quaternions and construct a 10-dimensional motion data per timestamp. This piece of data is then fed into an LSTM classifier since they would like to retain the temporal relationship of data among timestamps. For each of the gestures, they resize all of the gestures into the same length such that the LSTM is capable of reading the input. Their model is basically 2 LSTM layers followed by a RELU activation as well as a softmax activation. As for training and evaluation of the model, they have decided to use cross-entropy as loss and Adam optimizer.

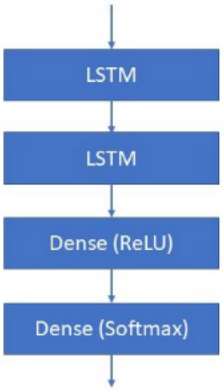


Figure 4 M. Kim et al LSTM model

Their model seemed to have a decent accuracy at above 0.9 for training and 0.7 for validation. This shows the model is overfitting a lot and we might have to consider that as well. In the paper, the research has mentioned their tiny dataset and the dataset was only collected by the two researchers and could be the cause of overfitting. Moreover, at the start and the end of the time frame is indicated by pressing an enter key while collecting the IMU data, this is not as practical for a glove to be used in real life. They have mentioned that a possible solution to this problem is to detect the start and end of gestures by creating a distinguishable local gradient.

2 Methodology

2.1 System Overview

Our gesture recognition system consists of three main stages: the glove, the ROS server, and the end appliances. The system architecture is shown as following:

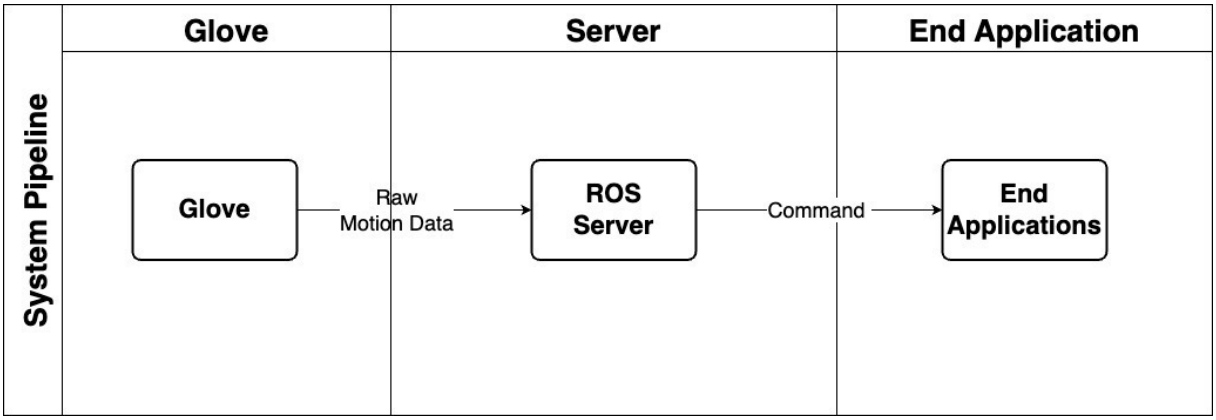


Figure 5 System architecture

The glove collects motion data and orientation information from the user's hand and sends it to the ROS server through Wi-Fi connection, details will be discussed in 2.2 Glove.

The ROS server then receives the data from the glove and applies filtering and transformation to the raw motion data. The preprocessed data is then fed into our trained gesture recognition model for real-time inferencing. The decision module analyzes the detected gesture and combines it with the orientation information from the glove to determine which command should be sent to which end devices.

The end application receives the command from the ROS server and reacts accordingly. The application could be a PowerPoint, light bulb, or etc.

Before the methodology section on the server, we will first discuss on our off-pipeline processes, including how the dataset was collected, augmented, and preprocessed. We will also discuss

how our model was trained and some setup system configuration tool, please refer to 2.3 Off-Pipeline Processes.

2.2 Glove

2.2.1 Physical Design of the Glove

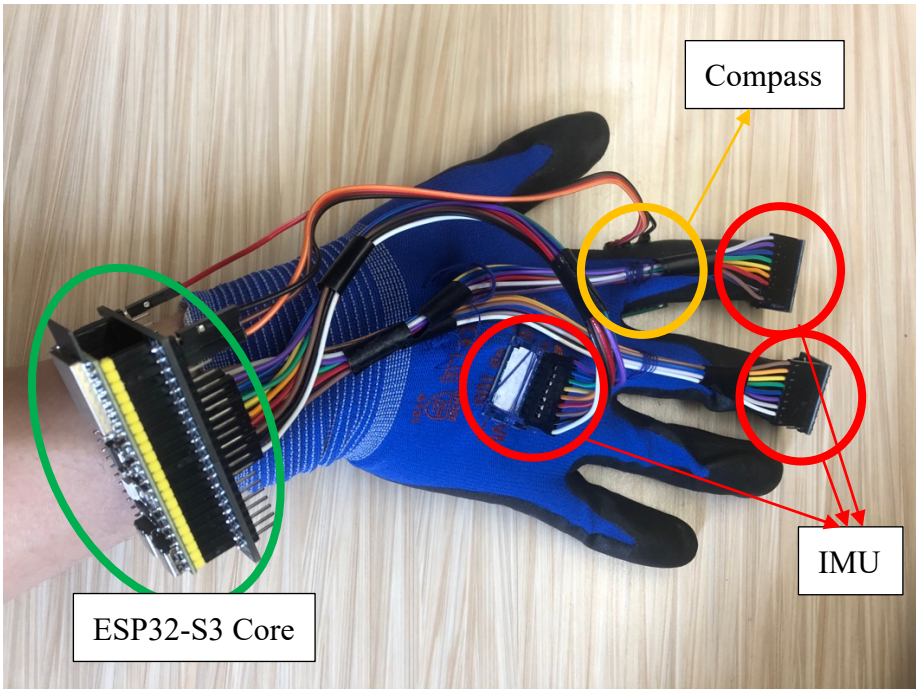


Figure 6 Physical view of our glove

Our implementation features a normal glove equipped with three IMUs located on the index finger, middle finger, and back of the palm. A microcontroller unit (MCU), an ESP32-S3, is positioned on the back of the hand, and is equipped with both WI-FI and Bluetooth connectivity, allowing communication with the server. The three IMUs (MPU9250) are connected to the ESP32-S3 through four data lines and two power lines, following the Serial Peripheral Interface (SPI) protocol which enables access to multiple slave devices of the same type on the data bus. Moreover, a compass (GY26) is connected to ESP32-S3 through two data lines and two power lines, following the Universal Asynchronous Receiver/Transmitter (UART) protocol. To power the ESP32-S3, we use a power bank.

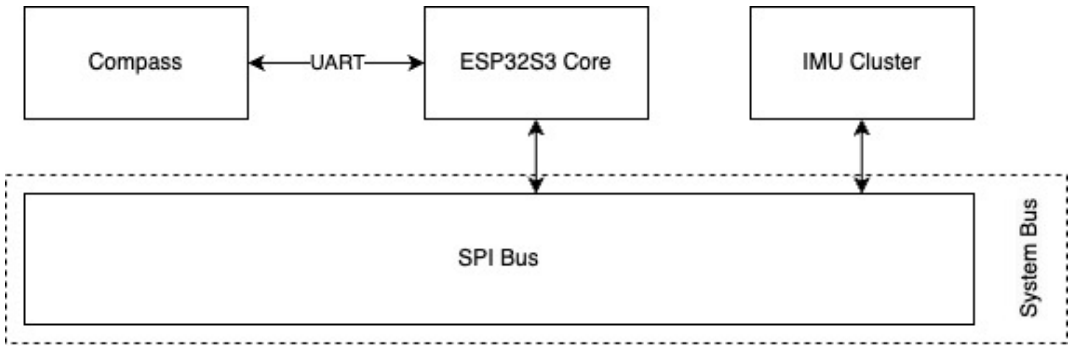


Figure 7 Hardware structure of the glove

2.2.2 Process Management (RTOS)

The code for the glove is written under the ESPIDF framework, which is the official development framework for the ESP32 microcontroller. The framework is based on RTOS (Real-Time Operating System), which embraces the use of multi-threading processing, reducing the complexity of interactions between the different components of the glove.

The three main threads in the code are the IMU thread, the Compass thread and the WI-FI thread. These three threads are running independently on the 2 cores of ESP32 to avoid interference among them. The IMU and the compass thread is designed to prioritize data collection and ensure that the IMUs are sampled quickly and accurately, even when other processes are running simultaneously. The WI-FI thread is pinned to a different core than the IMU and compass thread, allowing it to operate independently and without interfering with the data collection process. Below is the assignment of threads to the cores:

Core ID	Threads
Core 0	IMU Thread
	Compass Thread
Core 1	Wi-Fi Thread

Table 1 Core-Thread Assignment

2.2.3 Data Collection

During the data collection phase, we continuously sampled readings from the IMUs on the fingers and the compass and saved the data in a data queue every 10 milliseconds. The ESP32-S3 microcontroller communicated with the IMU using SPI, and with the compass using UART.

2.2.4 Data Transmitting

The raw data obtained from the IMUs is transmitted to the PC in a JSON format string for further processing. Wi-Fi is chosen as the transmission medium as we want the user to be able to control devices in a larger range comparing to Bluetooth. The transmission is done by sending HTTP requests via Wi-Fi with the body containing the IMU data as a JSON string.

Name	Meaning	Note
Time	Time passed since boot	
Translational acceleration	Instantaneous acceleration of IMU	3-axis (X, Y, Z) for each IMU
Angular velocity	Angular velocity of IMU	3-axis (Roll, Pitch, Yaw) for each IMU
Magnetic magnitude	Magnetic magnitude detected by the IMU	3-axis (X, Y, Z) for each IMU
Compass Orientation	Orientation of Earth's magnetic field	

Table 2 Structure of JSON sent from ESP32

```
{
    "t_sec": 1.000,
    "imu0": {
        "ax": 0.00,
        "ay": 0.00,
        "az": 0.00,
        "gx": 0.00,
        "gy": 0.00,
        "gz": 0.00,
        "mx": 0.00,
        "my": 0.00,
        "mz": 0.00,
        "t": 0.00
    },
    "imu1": {
        "ax": 0.00,
        "ay": 0.00,
        "az": 0.00,
        "gx": 0.00,
        "gy": 0.00,
        "gz": 0.00,
        "mx": 0.00,
        "my": 0.00,
        "mz": 0.00,
        "t": 0.00
    },
    "imu2": {
        "ax": 0.00,
        "ay": 0.00,
        "az": 0.00,
        "gx": 0.00,
        "gy": 0.00,
        "gz": 0.00,
        "mx": 0.00,
        "my": 0.00,
        "mz": 0.00,
        "t": 0.00
    },
    "orientation": 100.0
}
```


Figure 8 Sample of JSON sent from glove to server

2.3 Off-Pipeline Processes

2.3.1 Data Collecting for Machine Learning

Collecting a high-quality dataset is important for training a good machine learning model. However, the process of it can be really challenging, especially when there are no existing datasets that use the same hardware setup for the glove like us. In our case, we had to collect our own dataset of hand gestures. To achieve this, we followed the following steps, including:

1. Wearing the glove and connecting it to the ROS server.
2. Performing different hand gestures, including stay static, sliding up, down, left, right, releasing, and grasping. While performing the gestures, we used the ROS bag tool to record the desired ROS topics, such as /Imu0, /Imu1, /Imu2, /tf, and /tf_static, for approximately 5 seconds for each gesture.
3. Cropping the data to a length of 50 sample size and labeled it according to the gesture being performed.
4. Saving the data in a local machine.

We ensured the diversity of our dataset by collecting samples from multiple individuals with varying hand sizes and shapes. We also collected samples of the same gestures performed at different speeds, starting and ending points, and orientations. It was essential to ensure that the gestures were performed naturally as this increased the likelihood of successful recognition.

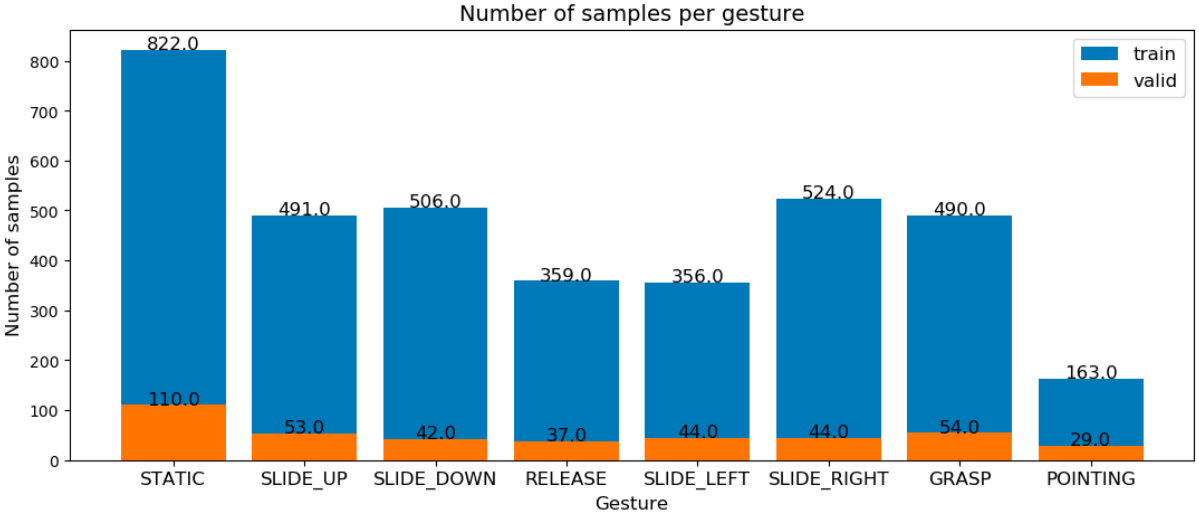


Figure 9 Dataset sample distribution

For gesture definition, please refer to 6 Appendix A: Gesture Definition.

2.3.2 Data Analysis

The dataset we collected is consisted of 45-axes data, including linear acceleration, angular velocity, orientation, and transformation information between IMUs. Each sample in the dataset has a timestep of 50 samples.

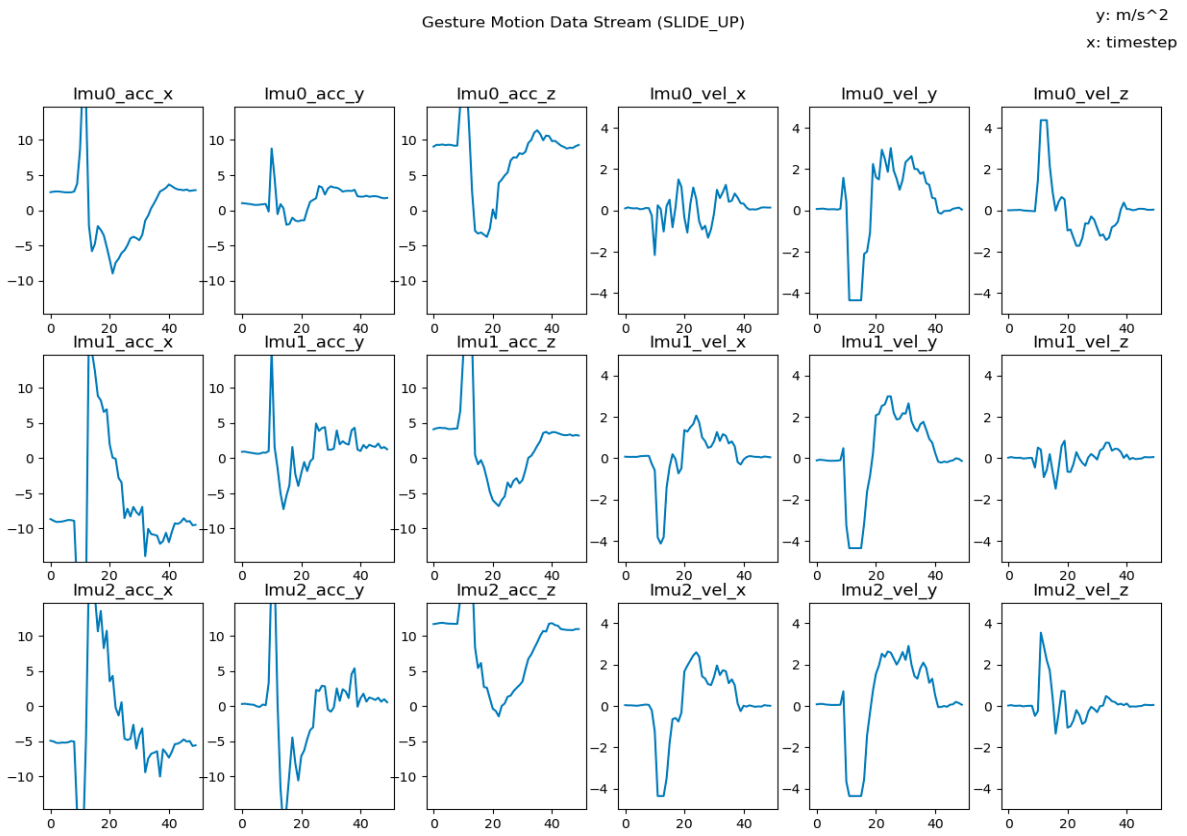


Figure 10 Sample data for "Sliding up"

2.3.3 Data Labeling

The labeler tool is an essential component of the gesture recognition system, as we collect the training data ourselves. The module subscribes to the ROS bag currently playing and utilizes a ROS time synchronizer in message filter package, to ensure that the messages are properly synchronized. Once the data has been synchronized, it is then appended to a python dictionary and saved to a csv file after the bag has finished playing, providing the user with a clean and organized dataset to use for training. Additionally, the labeler module allows the user to put a label on each data, which is then saved to another csv file for training label.

2.3.4 Data Augmentation

The data augmentation process involves applying different transformations applied to random parts of the original time series data to create new samples. We applied two types of data augmentation techniques: jittering and time warping.

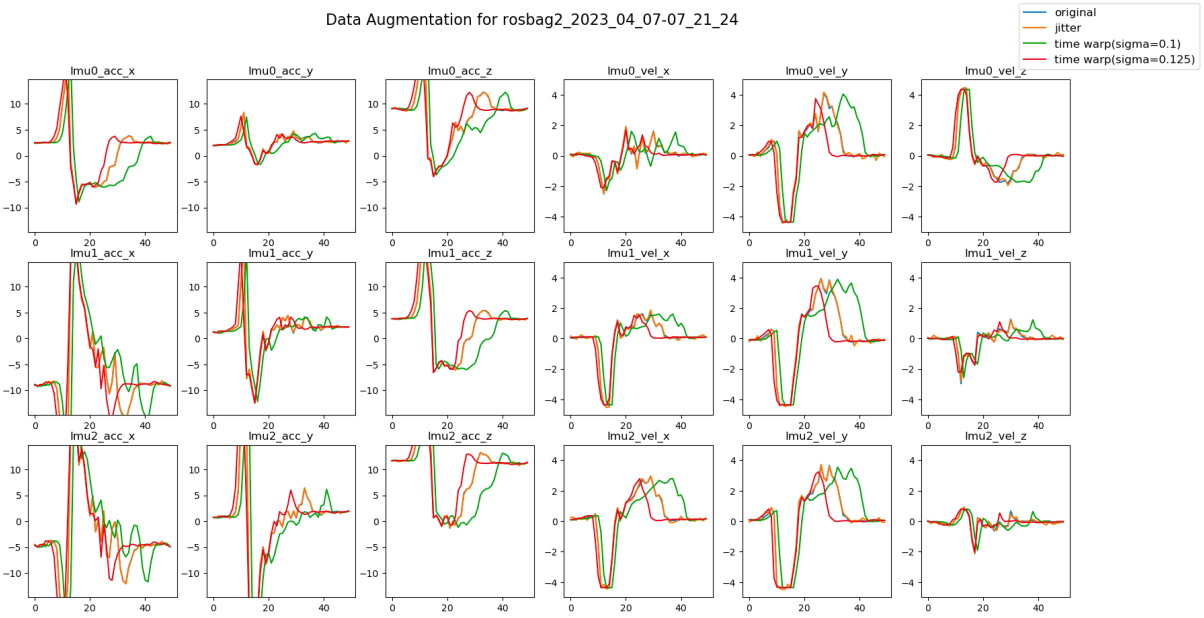


Figure 11 Data augmentation plot

The jittering method adds normally distributed random noise to the time series data, with a standard deviation of 0.1. This can help the model learning to be more robust to small variations in the data and more toleration towards Gaussian noises.

The time warping method applies a random time warp to the time series data, using a cubic spline interpolation. The spline is generated in random shapes along the time axis and applying a scaling equal to 0.1 and 0.125 factor to each knot point. The resulting spline is then used to interpolate the time series data at new time steps. This can help the model to recognize gestures with different time profiles, or with small temporal distortions. Practically, we hope the model can classify gestures of the same meaning but at different speeds.

The code iterates over all the samples in the dataset, and for each sample, applies the three data augmentation methods to generate three categories of augmented data: jitter, time warp 1, time warp 2, with time warp 1 and time warp 2 being time warps of different magnitudes. The resulting augmented data are saved as new CSV files in a new directory, expanding our dataset four times of the original one.

2.3.5 Training Gesture Recognition Model

2.3.5.1 Before Training

Before training, we load each data sample corresponding with its ID and read the raw data and label. We then drop less important features in the data frame (refers to the discussion on Feature Importance). Then we pad the data to a fixed length using a buffer size of 50 (defined as DATA_BUF_LEN, refers to discussion on Data Length Changes). The function also reads the corresponding label from a separate CSV file.

Out data was randomly shuffled and divided into training and validation set using the "train_test_split" function from the package scikit-learn.

2.3.5.2 Model Architecture

For the implementation of our gesture recognition model, we used Keras, a high-level neural networks API running on top of TensorFlow. The model architecture we designed consists of two halves. The first half is serves as feature extractor, it stakes an LSTM layer with 64 units, a dropout layer with a rate of 0.5, a layer normalization layer, and a flatten layer. The second half is serves as our neural network, this consists of a dense layer with 64 units, another dropout layer with a rate of 0.5, another layer normalization layer, and a final dense layer with Softmax activation that outputs the predicted probabilities of the different classes. Regularization is applied to the kernel and bias with a rate of 1e-2 to prevent overfitting.

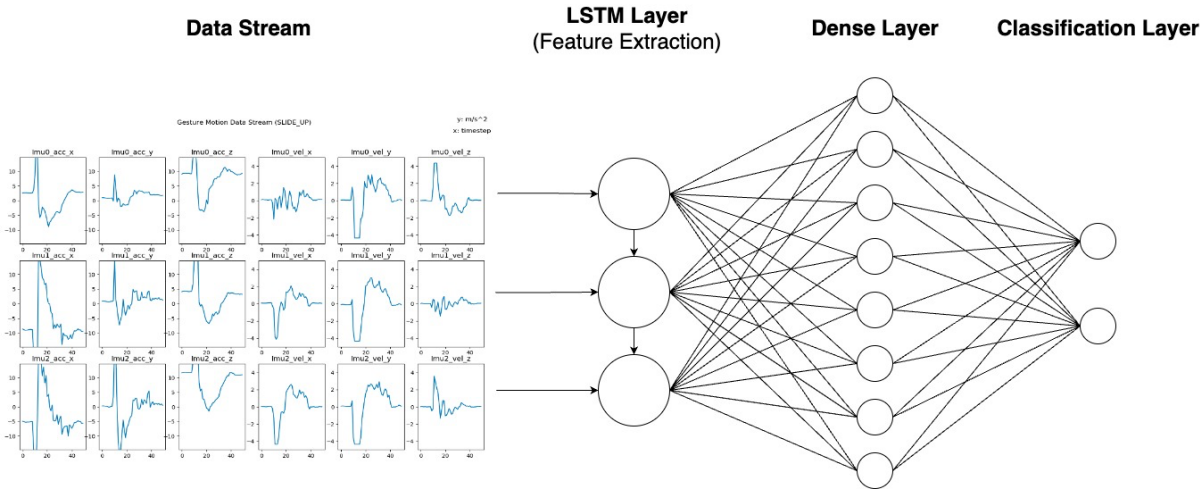


Figure 12 Model Architecture

The compiled model was trained with categorical cross-entropy as the loss function and Adam optimizer with a learning rate of 0.0005. We also monitor the accuracy metric during training. Furthermore, an early stopping callback with a patience of 3 that will halt the training if there is no improvement in loss over 3 consecutive epochs and an accuracy threshold callback has also been implemented. It stops the training when both training and validation accuracy reach 0.99 in order to save training time.

2.3.6 Device Configuration Interface

A device configuration interface is designed and implemented to allow the user in configuring the real-world device position with respect to the glove. There are buttons for the users to add or remove devices from the current configuration. Users can also click on the devices to edit the settings of the devices. These settings include the name, device type, starting angle and ending angle to be detected as well as the IP address of the device. After users finished editing the configurations, the editing window will be closed automatically. Users can choose to export the current configuration or not and if it is exported, these configurations could be loaded to a

JSON file which will be loaded by the commander module later. The interface will also load the saved configuration from the JSON when using the interface next time.

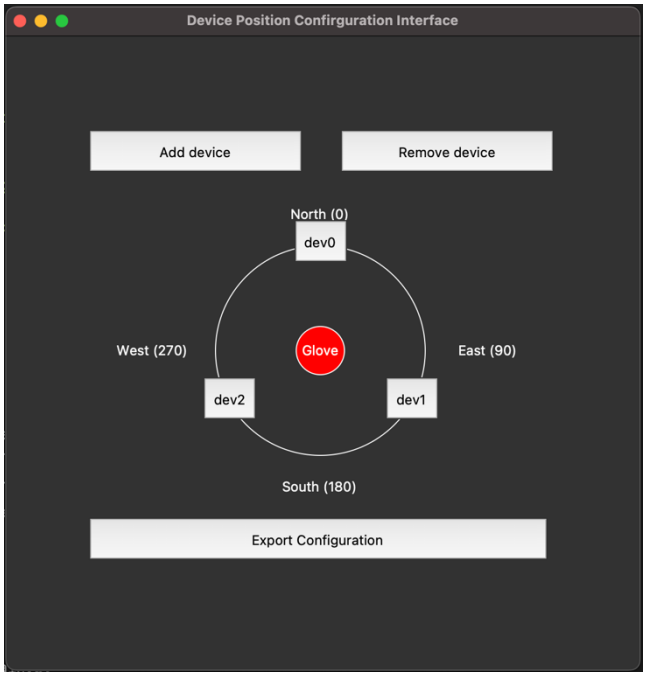


Figure 13 Main Page of the device configuration interface

```
{
    "LED_1": {
        "upper_bound": "60.0",
        "lower_bound": "300.0",
        "device_type": "LED",
        "ip_address": "192.168.4.2"
    },
    "LED_2": {
        "upper_bound": "180.0",
        "lower_bound": "60.0",
        "device_type": "LED",
        "ip_address": "192.168.4.3"
    },
    "PC_1": {
        "upper_bound": "300.0",
        "lower_bound": "180.0",
        "device_type": "LED",
        "ip_address": "192.168.4.4"
    }
}
```

Figure 14 Sample JSON file for device configuration

2.3.7 Visualization

The RViz of ROS is an important tool in the visualization of gestures in this project. The RViz display provides a clear and intuitive representation of the IMUs and the TF between each of them, making it easier for the user to label the training data and validate the gestures. This visualization capability is particularly useful when the user is labeling the training data, as they can see the orientation of the IMUs in real-time and make sure that the label they are applying is accurate. The RViz display also provides valuable information about the performance of the IMUs, allowing the user to identify any issues or inaccuracies in the data, and make necessary adjustments to improve the accuracy of the gesture recognition system.

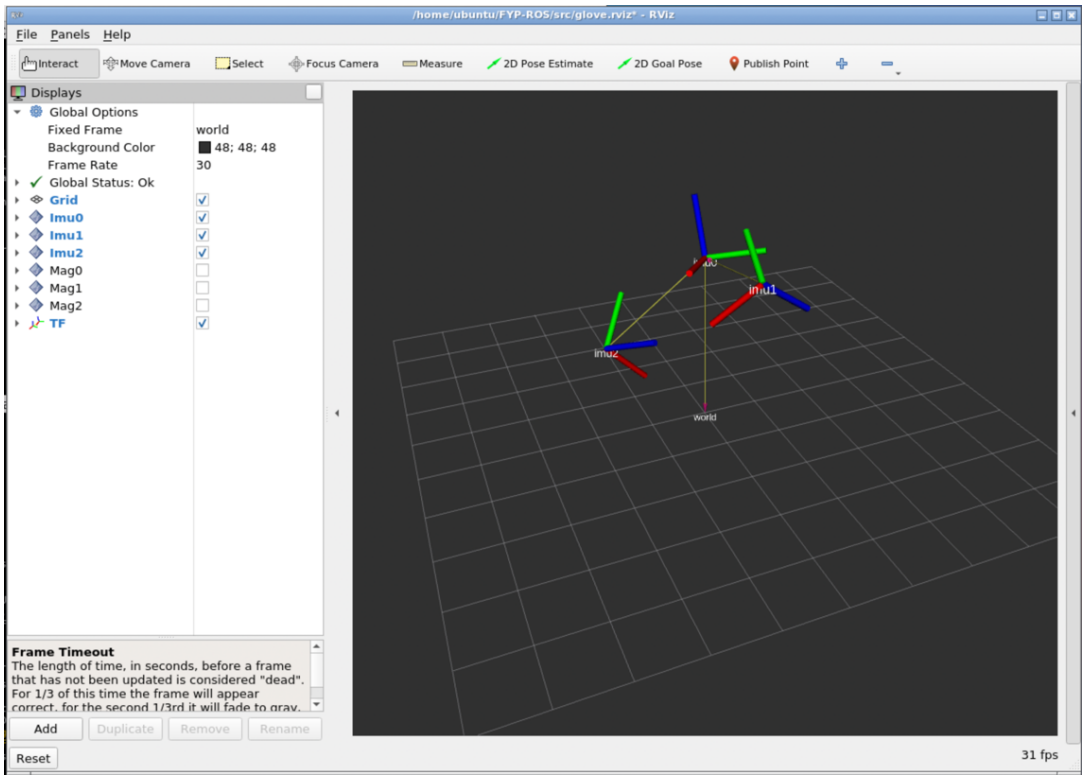


Figure 13 Rviz (ROS visualization tool)

2.4 Server

2.4.1 Overview

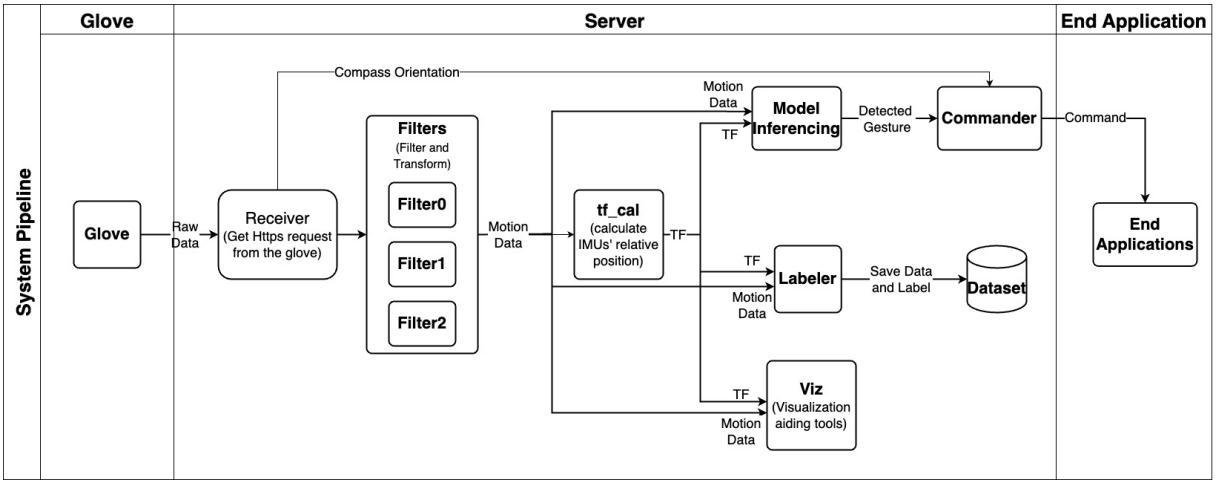


Figure 14 Server Architecture

The raw data received on the server side is in a JSON string and processed using the ROS2 framework. Firstly, a receiver node will establish a Flask server for receiving the JSON string, then, the data is transformed into local data with reference to the palm and used by different backend nodes for tasks such as filtering and transform calculation. Then, the model predicts gesture class based on the preprocessed input data. Finally, the commander node will get the output from the gesture recognition model and send the commands dedicated to the devices based on the orientation received from the receiver module.

2.4.2 Background

In this project, ROS2 Foxy, a version that is built on Ubuntu 20.04, is used for the backbone of the server. In this report, all references to ROS in this report pertain to the ROS2 Foxy version. Below are some terminologies of it to be explained:

- "Node" is an executable file in ROS that implements a specific functionality. Nodes communicate with each other using topics. A node can publish or subscribe to multiple topics.

- "Topic" in ROS is a way for nodes to communicate with each other. Think of it like a virtual mailbox where a node can send messages (known as publishing) and another node can receive these messages (known as subscribing). Nodes communicate through these topics and exchange information or data, allowing them to work together to accomplish a task.
- "Publish" and "Subscribe" refer to the mechanism for sending and receiving data over a topic. A node that sends data is called a "publisher" and a node that receives data is called a "subscriber".
- "ROS Bag" is a file format in ROS for storing and sharing data recorded from the ROS system. A bag file can contain multiple topics and can be used to play back recorded data, analyze data, or develop and test code against pre-recorded data.

2.4.3 Data Preprocessing

2.4.3.1 Receiver

With all the basics setup, we started on implementing the modules. The receiver module was the first to be worked on. To receive the http requests from ESP32, we have decided to use the Flask framework for our receiver. Being one of the most popular python web frameworks, Flask is a convenient web interface for backend development. After receiving the POST request from the ESP32, the JSON payload will be unwrapped into the reading from the 3-axis of readings from accelerometer, gyroscope, and magnetic reading for each of the IMUs and compass orientation. These data are then published to the dedicated topic ready for further processing. (For the JSON received, refer to Figure 8 Sample of JSON sent from glove to server)

2.4.3.2 Dynamic Transform Calculator

Not only the individual IMU measurement can be used as the input data for the model, but the relation between each IMU on the glove, such as their linear and angular displacement serves

as an indicator for different gestures. This relation between IMUs is called transform (TF) in ROS. The Dynamic Transform Calculator (DTF) calculates the TF whenever new IMU data arrives and publishes the TF to the topic for later gesture recognition.

To calculate the TF, the initial position of the glove is assumed to be statically placed on a flat surface, allowing the calculation of the initial TF of the glove. The linear acceleration data collected by each IMU is used to calculate the TF through a double integral of linear acceleration to displacement based on Equation 1, Equation 2 and Equation 3.

$$v_t = v_{t-1} + a * dt$$

Equation 1

$$S_t = S_{t-1} + v * dt$$

Equation 2

$$\theta_t = \theta_{t-1} + \omega * dt$$

Equation 3

However, it was found that the accumulated error from the static measurement of the IMUs caused the estimated displacement to drift over time, which increases the error due to the double integral relation from acceleration to displacement.

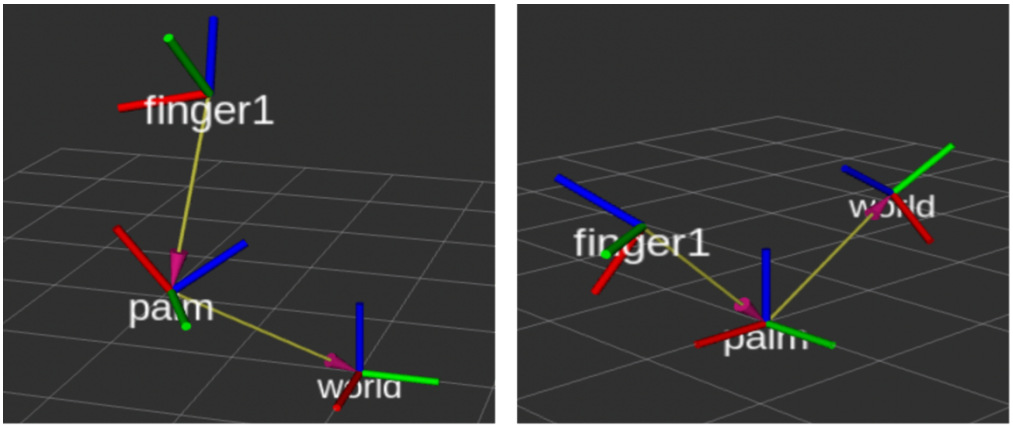


Figure 15 TF from different perspectives

The orientation of the IMUs is used to calculate the transform (TF) instead of their linear displacement. The orientations of the IMUs are relative to the world frame, which is a global absolute coordinate system. To calculate the TF between the IMUs, the orientation difference between the finger and palm IMUs is used, under the assumption that the total length between them is fixed. See Equation 4, Equation 5 and Equation 6.

$$translation_x = fingerlength * cos(pitch) * cos(yaw)$$

Equation 4

$$translation_y = fingerlength * cos(pitch) * sin(yaw)$$

Equation 5

$$translation_z = fingerlength * sin(pitch)$$

Equation 6

In our ROS implementation, when all IMU data of the same timestamp is received and synchronized by the ROS Message Filter at the DTF node, the TF calculating callback functions are invoked. The TF is calculated by transforming the TF from quaternion to roll-pitch-yaw representation, subtracting the orientation of the finger's IMU from the palm's, and then calculating the linear TF. Finally, the calculated TF is broadcasted for later use.

2.4.4 Gesture Recognition Model Inferencing

During inferencing, our gesture recognition model utilizes two techniques to optimize its performance.

First, to ensure that the data from different IMUs and transformation (TF) is correctly synchronized, we perform a synchronization step before feeding the data into the buffer. These techniques allow our model to make accurate predictions in real-time and ensure that the model operates efficiently.

Second, the model has a data buffer that continuously receives and processes new data generated by the glove. This buffer operates on a First In First Out (FIFO) basis, older data is automatically discarded as new data is added to the buffer. This ensures that the model is always working with the most current data.

2.5 End Application

To differentiate between which device the glove is trying to send command to, we have embedded a compass on the glove that can determine which directions the glove is facing at. For each iteration of recognition, we will combine the gesture recognition result with the direction obtained from the compass to send the command through HTTP requests to the target device. The following part is going to illustrate the entire process of how this is performed.

2.5.1 Commander Module

A JSON file with the device configuration will be loaded by the commander module upon launching. These saved configurations will then be used for determining what command and device that the commander is going to send. Whenever a new angle is published to the /Orientation topic, it will trigger the commander module and calls a callback function for deciding what the glove is currently pointing at. After the device that the glove points at is decided, it will be saved until the next angle is received and the decision will be made again.

The command will eventually be sent once a gesture is published to the /Gestures topic. The commander module will search the device's IP address that was loaded from the JSON configuration file and proceed to send the command with HTTP request to the device side in the following format:

Figure 16 Sample body of HTTP request sent

2.5.2 PowerPoint Control

On the PC where a PowerPoint is being presented, a Flask HTTP server will be hosted with an endpoint that allows commands to be sent as the body of the request. After receiving the commands, the body of the requests received from the server will be decoded. The decoded command will then be mapped to the functions that is implemented on the server. The functions are implemented with PyAutoGUI that can input keyboard or mouse inputs for the user. In our case of the functions implemented, we have mapped the gestures to the following meaning for the PowerPoint control interface.

Gestures	Meaning	Keyboard Input
Slide Up, Slide Right	Next Slide	"Up"
Slide Down. Slide Left	Previous Slide	"Down"
Grasp	Blacken Out the PowerPoint	"B"
Release	Un-Blacken Out the PowerPoint (Will only trigger if the PowerPoint is currently blackened out)	"B"

Figure 17 PowerPoint Gestures Mapping

2.5.3 IOT device Control

The IOT devices is currently controlled by another ESP32. It is essentially implemented as a HTTP server in the way PowerPoint control is implemented. Gestures will, similarly, be received from the ROS server commander module and decoded into the gestures. These gestures will also be mapped into functions to control the IOT device. In our case, we have used a LED light as demonstration for the IOT device and the functions will send GPIO output according to the gesture input. The following is how the LED functions is mapped with the gestures:

Gestures	**Meaning**	**GPIO Level**
Grasp	Turn off the LED	High
Release	Turn on the LED	Low

Figure 18 IOT Gestures Mapping

3 Discussion

3.1 Glove

3.1.1 Choosing Between Protocols

Several common communication protocols that are widely used in embedded systems are evaluated, including I2C, SPI, and UART/USART, to determine the best option for communication between the ESP32 and the IMUs. Initially, we considered using I2C due to its simplicity in wiring of requiring only two data wires (SDA for data and SCL for clock).

However, the I2C distinguishes devices by their unique ID, which depends on the model number. This means if multiple devices have the same model number, the master device would not be able to distinguish between the devices. Similarly, UART was not practical for our scenario as it is designed for point-to-point communication and cannot communicate with multiple IMUs at the same time on the same connection. Thus, we selected SPI as the most suitable protocol for our application. One of its advantages is its ability to support up to 256 slave devices without requiring each device to be unique, as the control of the bus is determined by the Chip Select (CS) wire. The device that is being pulled up on the CS line has control over communication with the master. Additionally, it provides a higher sampling rate. The data transmission frequency of SPI can reach up to tens of MHz, which allows for high-speed data transmission. On the other hand, I2C operates with a lower data transmission frequency compared to SPI, with typical speeds ranging from 100 Kbps to 400 Kbps.

	UART/USART	**I2C**	**SPI**
Transmission rate	~100Kbps	100~400Kbps	Several Mbps
Duplexity	Full-duplex	Half-duplex	Full-duplex
asynchronous	Asynchronous	Synchronous	Synchronous
Connection	Point to Point	Bus	Bus

Table 3 Protocols comparison

3.1.2 Choosing Between IMU Sensors

We have implemented and tried various IMU sensors that can capture the movement of the hand along the development. Three sensors were implemented and tested: MPU6050, MPU6500, and MPU9250. The model number of each sensor indicates its level of use, with MPU9250 being the most advanced.

To begin with, we tried the MPU6050, which is a 6-axis motion tracking device with gyroscope and accelerometer and only supports the I2C protocol. It features a configurable gyro precision that can measure up to 2000 degrees per second and a configurable accelerometer precision that measures up to 16g meters per second squared. It also has a 16-bit analog to digital converter and output streaming capabilities. We wrote a driver for it using C under the ESPIDF framework and verified it could collect acceleration and angular velocity data. However, we later realized that it would not be a suitable choice for our project because of limitation of the I2C protocol mentioned earlier in 3.1.1 Choosing Between Protocols.

Next, we evaluated the MPU6500, which has similar specifications to the MPU6050 but supports the SPI protocol. This allowed us to control multiple devices with the same model number. However, we soon realized the need for a global reference to locate different devices in an area after visualizing in our visualization tool. Considering the possibility of controlling multiple appliances in the future, the glove must have the means of distinguishing different devices based on their spatial location. As a result, we continued our search for a suitable sensor.

Magnetic sensor is often a source for global referencing, that is why we tried the MPU9250, which is essentially an MPU6500 with an additional magnetic sensor embedded. We used the MPU6500 from September 2022 to November 2022, but we eventually switched to the MPU9250 starting in December 2022 after realizing the importance of having a magnetic sensor in the IMUs.

	MPU6050	MPU6500	MPU9250
Supported Protocol	I2C	I2C/SPI	I2C/SPI
Measurement	Accel/Gyro	Accel/Gyro	Accel/Gyro/Magnetic

Table 4 IMU model comparison

Initially, our plan was to have a glove equipped with six IMUs, one located on each of the five fingers and one on the palm, in order to capture the complete hand movement. However, the ESPIDF framework only supports three slave devices on a single SPI bus. As a result, we considered using two ESP32S3, each controlling three IMUs, to collect the data. But after re-evaluating our pre-defined gestures, we realized that most of the gestures did not require the movement of the thumb, ring finger, and little finger, so installing IMUs on these fingers was not necessary.

3.1.3 Data Transmitting

Initially, we have decided to go with using Bluetooth to transfer data from ESP32 to the server. The reason for doing so is that we believe Bluetooth, being a point-to-point communication protocol, is the least prone to external interference. However, after discussing with our supervisor, we realized the limited range of Bluetooth would be a problem to our application. In the scenario of our users walking around a large place controlling appliances, Bluetooth's limit of range could restrict the users' activity area. Thus, we have decided to go with using Wi-Fi after careful considerations. This is because our user can essentially be using our device anywhere as long as there is an access point nearby the user.

3.1.3.1 Connection to WPA-Personal networks

As the communication protocol is decided, we have started with implementing the communication on our code. With the help of the ESPIDF library, the Wi-Fi functionalities of ESP32 can be easily accessible by including "esp_wifi.h". ESP32 is configured into using STA mode for the Wi-Fi module, which is another name for client mode. Provided with the Wi-Fi

SSID and password, ESP32 can now be connected to Wi-Fi networks with WPA-Personal adapted.

3.1.3.2 Connection to WPA-Enterprise networks

While ESP32 can now connect to WPA-Personal networks, we have found that our school's Wi-Fi *eduroam* is secured under WPA-Enterprise. While we can use our own mobile hotspots as an access point for our server and Wi-Fi, it would be more convenient if we could use directly instead. Therefore, we have found the *eduroam* CA certificate on the Internet and compile our program together with the CA certificate such that we can connect to *eduroam* easily.

3.1.3.3 HTTP Requests

As we have selected Wi-Fi to transmit data from ESP32 to the server, a communication protocol would have to be chosen to establish communication between the two. We decided to test with three different ways of implementing the transmission, namely HTTP requests, low level TCP socket, and UDP socket. We first tried using HTTP requests and found that even though it's a higher-level protocol and might be consisted of more payloads, we can reach 40Hz while transmitting data, which is satisfactory enough. We further tested TCP and UDP sockets, both are capable of performing 100Hz of data transmission frequency. However, there were times when TCP and UDP sockets between ESP32 and PC disconnects, and we would have to manually reestablish the connection. To reduce development time cost, we decided to go with HTTP requests as it's a session-less protocol and we would not have to deal with sudden spike of disconnection.

3.1.4 Synchronize with Time Server

As mentioned earlier, our IMU only provides linear acceleration, angular velocity and magnetic field readings, which require a complementary filter to estimate the orientation of the IMU. The

complementary filter requires the time between two IMU samples to work accurately. Therefore, it was necessary to find a way for the glove to keep track of time.

We considered two options for time synchronization: synchronizing with a time server (NTP) or synchronizing with a local server by getting the time from the computer. Initially, we chose for NTP synchronization as it is a standard and accurate measurement. However, we found out that our glove took too much time to connect to the NTP server.

As an alternative, we decided to sync the glove time with a local server. Fortunately, we observed that the complementary filter still worked accurately with the local server's time. This synchronization approach not only provided a faster connection but also reduced the overall complexity of the system.

3.2 Data Collecting

3.2.1 Dataset Collection

During dataset collection, we intentionally collected more samples for the "Static" gesture as we wanted our model favors “Static” more. This was because in our end appliance, "Static" means not performing any action, which is a safer option compared to other gestures such as "Sliding up/down/left/right" which are mapped to certain functions. Therefore, we deliberately created a data imbalance to favor the "Static" gesture, with the number of samples being 833, while the other gestures had a range of 358 to 514 samples (please refer to Figure 9 Dataset sample distribution). By doing so, we aimed to prevent mis-actions and ensure the safety of the end user. Nonetheless, we still made sure that the overall dataset was well balanced to prevent any bias towards any of the gestures.

3.2.2 Data Collection Improvements

Throughout the testing of our system by using the glove, we have found that the glove performs flawlessly when we are the users. However, once the user is not the two of us, the main training data contributor, it could not reach the same kind of performance as when we are the user. We then realized that even though our model might not have overfitted, our training data might be restricted to our own definition of the gestures as well as our hands. To overcome this issue, we have come up with two solutions to tackle the problem, namely extending the variation of our data and having other people for contributing data.

3.2.2.1 Extension of data variation

Understanding our training data might have a narrow variation, we have decided to collect training data with a wider range of variety. For instance, for the gesture "Static", we thought of it as a gesture of having our hands lying on a flat surface and not moving. However, by telling other users to perform a static motion with their hand, some of them just relax their hands and not move it midair. In view of this difference in understanding of gestures, we have decided to extend the definition of "Static" to be not moving our hands but could be at any position or any orientation. Our training data has also included data that collected with our hands in any static form as a result.

3.2.2.2 More data contributors

Even though we have a wider variation of data trained in our model, we still believe that the model is not comprehensive enough with our own comprehension of the gestures. Thus, we have found some of our friends to help collecting data without telling them the detail instructions but only telling them the name of the gestures. They then performed the gesture with their own thoughts and helped enriching our training data. This has furthermore extended the variation of our training data.

3.2.3 Data Length Changes

In this section, we will discuss how the data length affects the performance of our model. Note that our glove generates data at a frequency of around 40hz. This means that we have 40 data samples per second, and the motion data is continuously streaming into our model. Our model has a data buffer that keeps running inferencing on it. The buffer is a FIFO structure, and whenever there are newer data, the oldest data will be discarded, and the new data will be appended to the buffer.

Initially, we recorded data by wearing the glove and performing gestures. During training, we cropped the data to a length of 250 to ensure that the model had enough data to recognize the gesture. However, we found that it took around 5 to 6 seconds for the gesture to be discarded, meaning that the model's buffer was occupied for a long time. If the next gesture came while the previous one had not been discarded, the model would get confused and lower the prediction confidence.

To improve the response time, we changed the length for both cropping and the model's buffer to 100. This change reduced the time taken for the buffer to be renewed. However, we observed that the buffer was still occupied for slightly too long, and this affected the model's real time performance.

Ultimately, we changed the buffer length again and fixed it to 50, which means that for around every second, the buffer is completely renewed. This change improved the response time, and for every one second, we could perform another gesture. However, reducing the buffer length also means that we cannot have gesture that needs more than one second to complete, and this set a limitation for the gestures but no gestures should take more than one second in real world application.

3.3 Data Augmentation

To further extend the variation of gestures being able to be detected, we have decided to look into data augmentation to generate wider variations of training data. Thus, we have done research on performing time-series data augmentation.

We have then found this paper by Iwana and Uchida that proposed several methods of time series augmentations [4] . Out of all the augmentation method, we have decided to try jittering, scaling and time warping as they seemed to fit out use case the most as they can preserve the features in our data.

After applying these three augmentation methods to our dataset, we have done some tuning on the magnitude of augmentation and we have found that jittering and time warp could help, but scaling doesn't seem possible to be used. The reason for that would be there was no constant value that could be used to preserve the features in our data while having enough modification to our data. For instance, for a gesture that rotates our hand by 90 degrees, an IMU would detect acceleration of magnitude 9.81 on the axis pointing towards the ground. After the rotation is done, the IMU would output close to 0 magnitude towards the ground since the original axis is now perpendicular to the surface. In this case, if we apply constant scaling larger than one to the magnitude on every data point, the dataset will return more than 9.81 which doesn't make sense in real life for a static gesture. This is the flaw we found while developing scaling for data augmentation. Thus, we continued with only using time warp and jittering.

We have also tried performing both data augmentation method on our dataset. It turned out the resultant data could increase the data variation while preserving the important features for the gestures. The augmented dataset is then used for our final training.

3.4 Server

To efficiently process the data collected from the glove, including receiving, filtering and data transformation to higher dimensions, we need build a data flow system. There are two main options for building this system: Linux Inter-Process Communication (IPC) and Robotic Operating System (ROS). The key differences between the two are:

- Scalability: ROS is designed to be scalable and supports integration of new components, while Linux IPC is a basic communication mechanism between processes. This makes ROS more suited for projects that may change or grow over time.
- Community Support: ROS has a large and active community of developers, which provides a wealth of resources, tutorials, and support for users.
- Modular Design: ROS supports modular development, while Linux IPC requires more manual effort to separate different components of the system.
- Level of use: ROS provides a higher-level interface for building and integrating components of a robotic system, which abstracts away many of the low-level details. This makes it easier for developers to focus on building and integrating their components, without having to worry about low-level communication or synchronization details.

In conclusion, while Linux IPC can be used for inter-process communication in a robotic project, ROS provides a more comprehensive and scalable framework for building and integrating different components of a robotic system. Therefore, we have chosen ROS as the framework for our server.

3.4.1 Server Environment

To install ROS2 Foxy on our server, we would like to have an Ubuntu environment as it is natively supported by ROS2 Foxy. To reduce our development cost, instead of buying another

computer or on-board computer, we have decided to use a containerized Docker Ubuntu environment on our computer for development.

Initially, we have gone with using the traditional way of setting up a Docker environment. We found an Ubuntu Image and installed ROS2 Foxy on the environment. It worked fine and we tested it out by setting up topics, publishers, and subscribers. After which, we have encountered our first issue on the server, that was we couldn't observe whether nodes are connected to the topics. Normally, on a non-docker environment, we can use the built-in ROS graph visualization GUI rqtgraph for viewing the status of the system. However, since we only have access to command line interface on a traditional docker container, it is not possible to use GUI applications.

In order to use GUI applications and also future possible visualization applications, we decided to look into using X11. It enables us in viewing windows-based GUI and interacting them with our mouse and keyboard inputs on our Docker container. By embedding X11 to our docker image, we can now use most of the application with GUI on our Docker system and so we could use rqtgraph for debugging and testing.

3.5 Gesture Recognition Model

3.5.1 Model Architecture

During the implementation phase, we tested several architectures for both the feature extractor and fully connected neural network halves of our model. In the feature extractor half, we experimented with different hidden layer sizes ranging from 256, 128, to 64 units in the LSTM layer. All of them gave similar results in terms of accuracy. Ultimately, we chose a smaller size of 64 units to optimize the speed of prediction while maintaining accuracy.

Similarly, we conducted experiments on the hidden size of the fully connected neural network in the second half of our model. We tested double layers of 256 units down to a single layer of 64 units. Once again, we observed that all of them gave similar results in terms of accuracy. Therefore, we opted for the smallest model to reduce the computational cost.

In addition to these architectural choices, we also added layer normalization between each layer to prevent outliers in the weights, and we added L2 regularization to each layer for the same reason. We also set the LSTM parameter "return_sequences" to true, this would make the layer a “many-to-many” layer, rather than a “many-to-one” layer. This can be beneficial in cases where the input sequence contains long-term dependencies, as the LSTM layer will be able to capture those dependencies and output a sequence of features that can be fed into subsequent layers to improve classification performance.

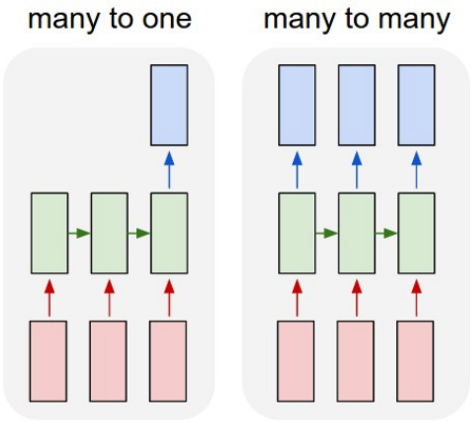


Figure 19 LSTM many-to-one vs many-to-many [5]

We also tested the model with a bidirectional LSTM layer to examine whether it could further improve the performance of the model. Initially, we believed that the bidirectional structure could make the model more robust and capable of recognizing different gestures by considering the flow of gesture data from both directions. However, we later found out that the bidirectional structure did not improve the performance of the model in practice. Therefore, we decided to remove the bidirectional layer from our model and continued the experiments with a unidirectional LSTM layer.

3.5.2 Training History

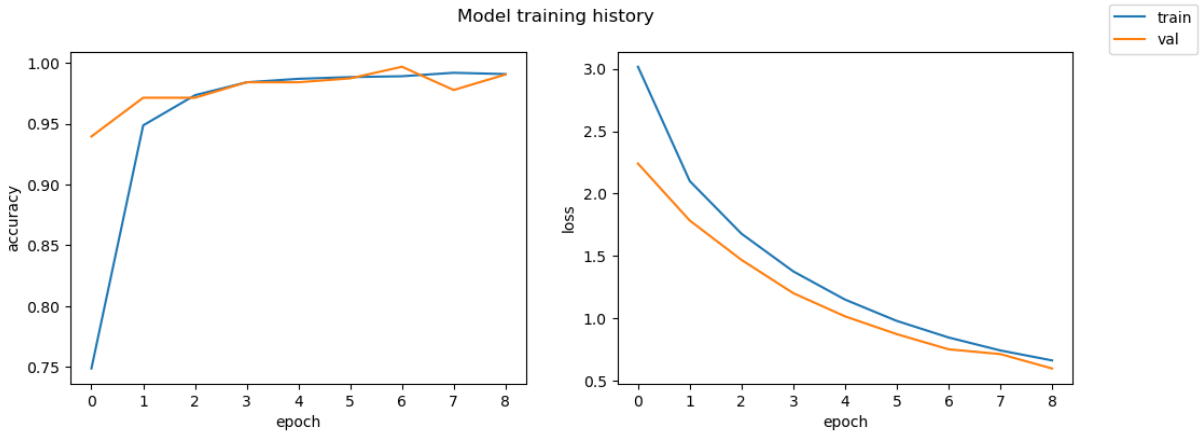


Figure 20 Training history

The training history of our gesture recognition model showed that we were able to achieve high accuracy levels in a relatively short amount of time. We conducted multiple tests and found that the model consistently achieved a training accuracy of over 0.99 within the first 10 epochs, triggering the early stopping callback. The validation accuracy also gradually increased over time and reached over 0.99 but slightly slower.

Throughout the training process, the model's loss decreased from around 3.0 to 0.66 for the training data and from 2.24 to 0.60 for the validation data. These results indicate that our machine learning model was able to classify different gestures that the user was performing.

Overall, we are pleased with the training history of our gesture recognition model. The training history shows that our model is capable of achieving high levels of accuracy while avoiding overfitting.

3.5.3 Evaluation Metrics

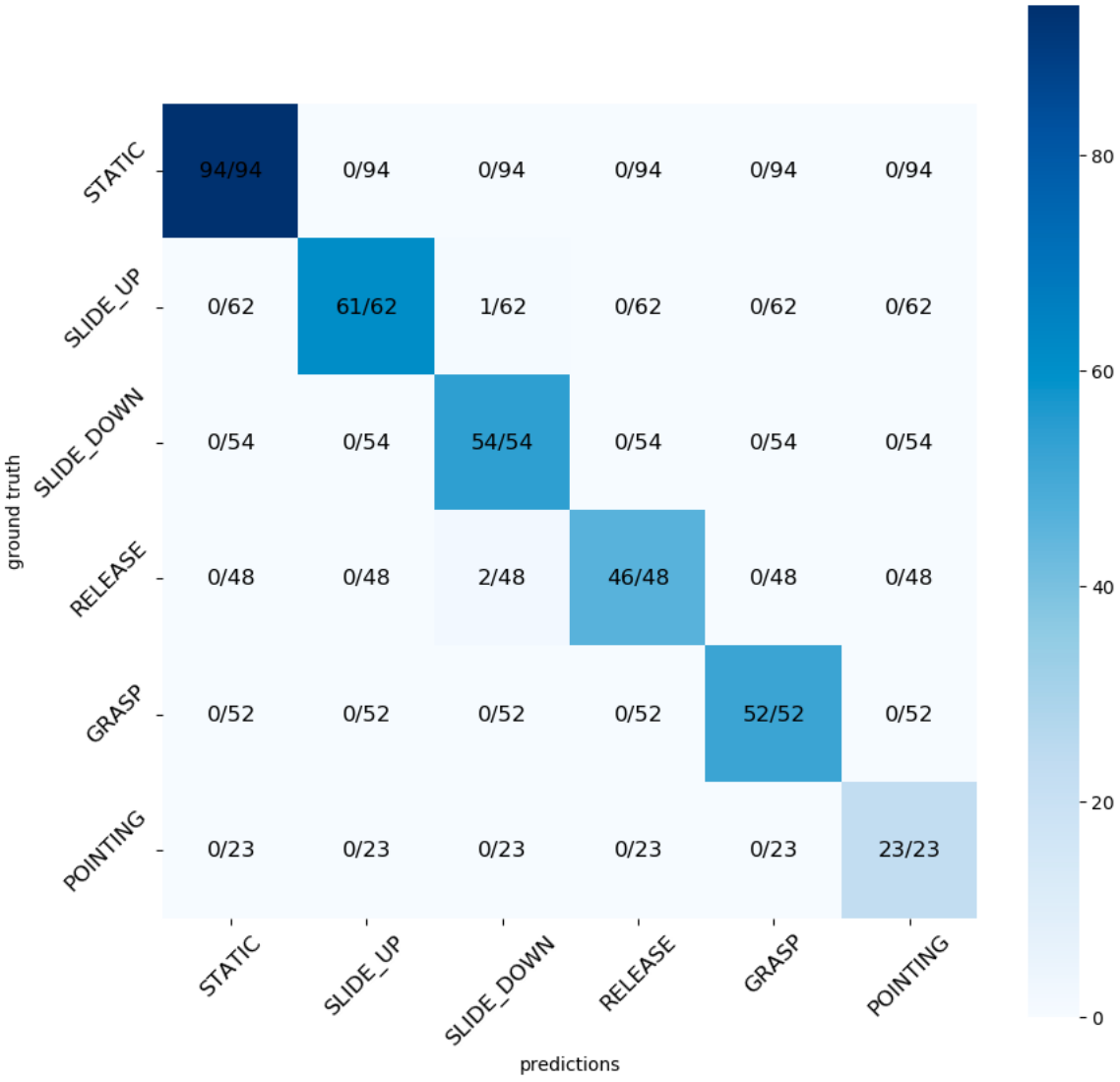


Figure 21 Confusion matrix

From the confusion matrix, we can see that in Figure 21 Confusion matrix, the majority of the samples were classified correctly. Class 1, which corresponds to the “Slide up” gesture, has one misclassification. Class 5, which corresponds to the ”Release” gesture, has two misclassifications as well. The other classes were well classified.

We also generated a classification report to further evaluate the performance of our model. The classification report (Figure 22 Classification report) provides precision, recall, and F1 scores for each class.

	precision	recall	f1-score	support
STATIC	1.00	1.00	1.00	94
SLIDE_UP	0.98	1.00	0.99	61
SLIDE_DOWN	1.00	0.95	0.97	57
RELEASE	0.96	1.00	0.98	46
GRASP	1.00	1.00	1.00	52
POINTING	1.00	1.00	1.00	23
accuracy			0.99	333
macro avg	0.99	0.99	0.99	333
weighted avg	0.99	0.99	0.99	333

Figure 22 Classification report

The precision is the ratio of true positives to the total number of predicted positives (Equation 7). The recall is the ratio of true positives to the total number of actual positives (Equation 8). The F1 score is the harmonic mean of precision and recall (Equation 9).

$$Precision = True\ Positive\ /\ (True\ Positive + False\ Positive)$$

Equation 7

$$Recall = True\ Positive\ /\ (True\ Positive + False\ Negative)$$

Equation 8

$$F1\ Score = 2 * (Precision * Recall)\ /\ (Precision + Recall)$$

Equation 9

3.5.4 Feature Importance

To evaluate the importance of each feature in our dataset, we conducted an experiment where we dropped one feature at a time and fed the validation set to the model to observe the changes on accuracy. An increase in loss implies the importance of a feature, the larger the loss increase, the more important the feature is. We computed the loss for each feature and normalized it with the loss with no discarded feature. To provide a more detailed explanation of the Normalized Importance Ratio, a value of 2.5 would indicate that dropping a particular feature would result

in a loss 2.5 times greater than the original data. The list below shows the feature importance values for our dataset:

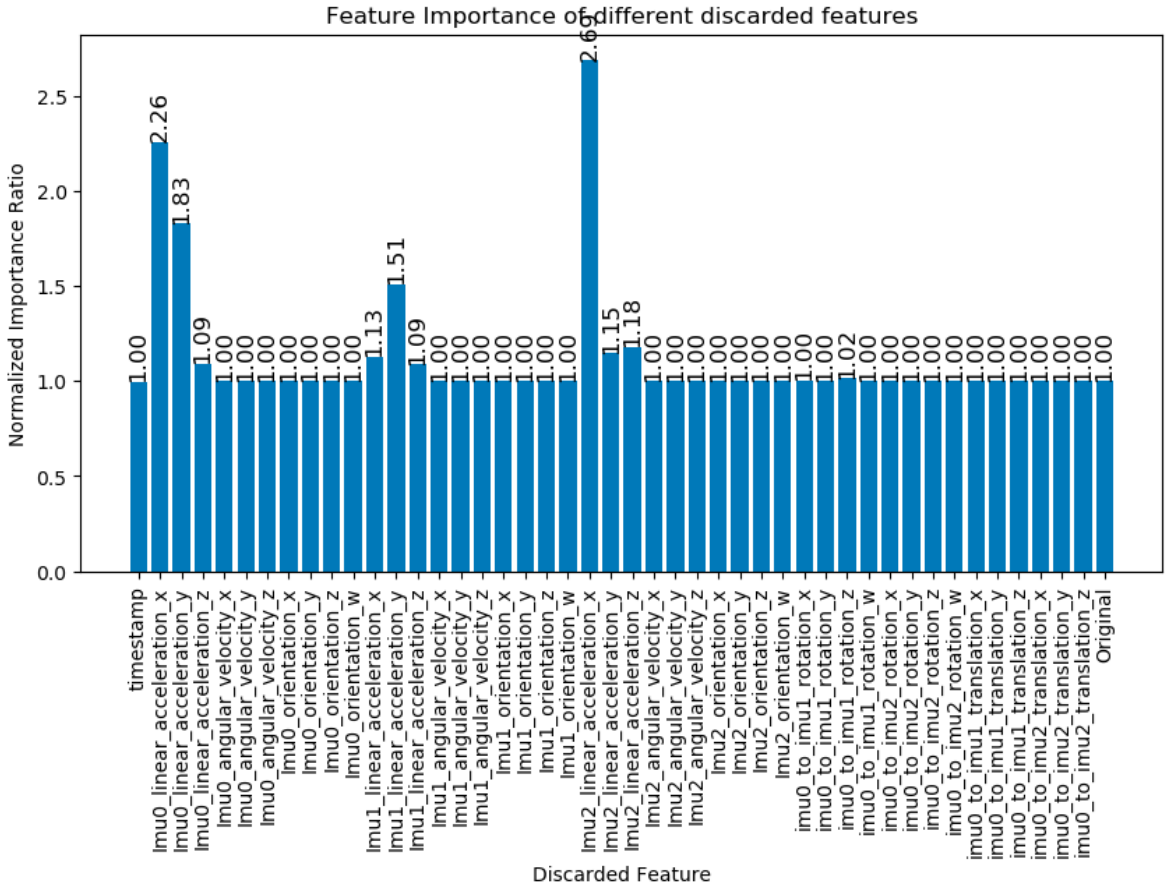


Figure 23 Feature importance (45 features)

We observed that the linear accelerations from all three IMUs were the most important features in our dataset, with feature importance values ranging from 2.69 to 1.09. The other features, such as angular velocities, orientations and transformations between IMUs, had lower importance around 1, which means dropping it or not did not affect the prediction too much.

After analyzing the feature importance, we decided to discard the features other than linear accelerations from all three IMUs. We found that these features were not contributing significantly to the performance of the model, removing them improved the inference speed and computational efficiency of the model. The updated feature list includes only the linear acceleration features from all three IMUs and is listed below:

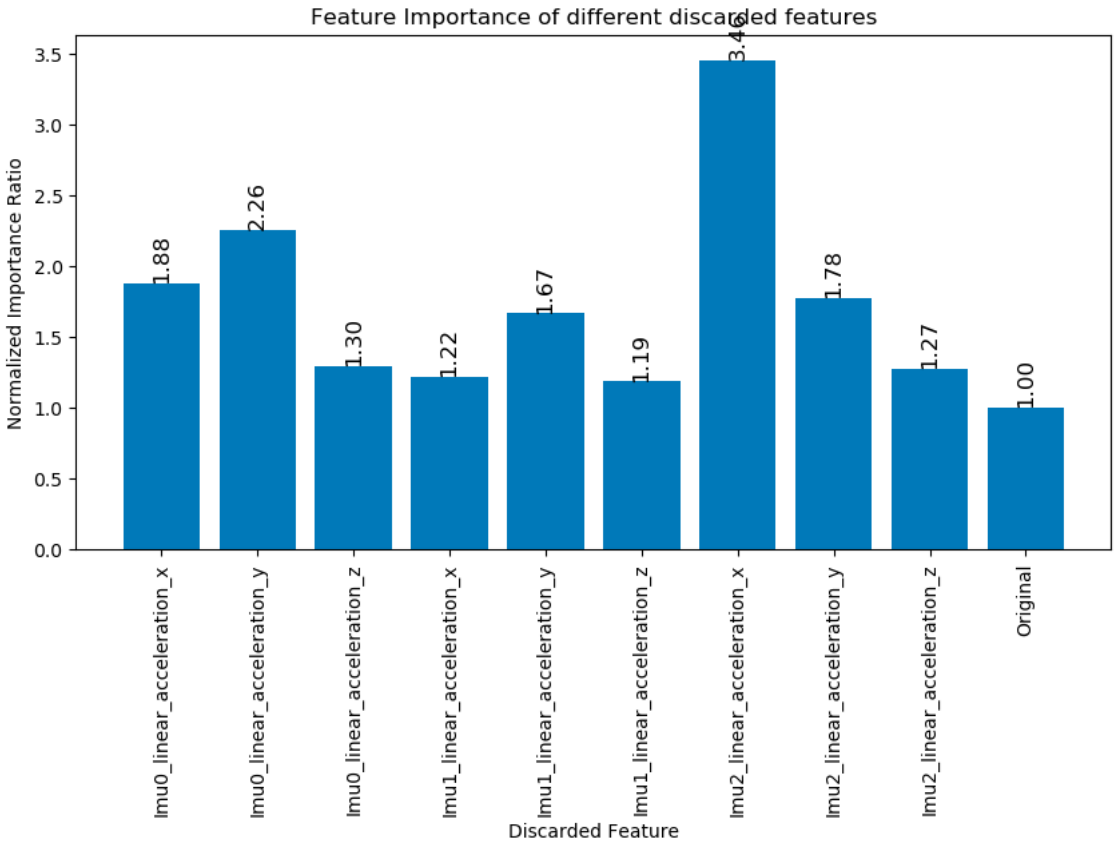


Figure 24 Feature importance (9 features)

We tested our model using this updated feature set and observed no significant drop in accuracy, but the computation time was reduced.

3.5.5 Timing Importance

To better understand how our model is making decisions, we used SHAP-like (SHapley Additive exPlanations) method to examine what the model "sees" during the gesture classification task [6]. Specifically, we conducted feature importance analysis at the time step level. For each step, we created a block to cover a segment of our time series data for all features by assigning zero to the data in the block on the time axis, we then calculated the loss for each step. After we iterate through all the timestamp of the data, we can find that the position with the highest loss was where the block covered the gesture, or at least a part of it. This allowed us to determine where in the time series the gesture appeared.

Figure 25 Timing importance shows the data and loss graph along the time axis. The red interval represents the block of length 10, while the green graph represents the loss along the time axis. We observed that the green line had a peak at where the gesture was happening, while other positions remained flat. This indicates that the model was able to identify the crucial features that contributed to the classification of the gesture.

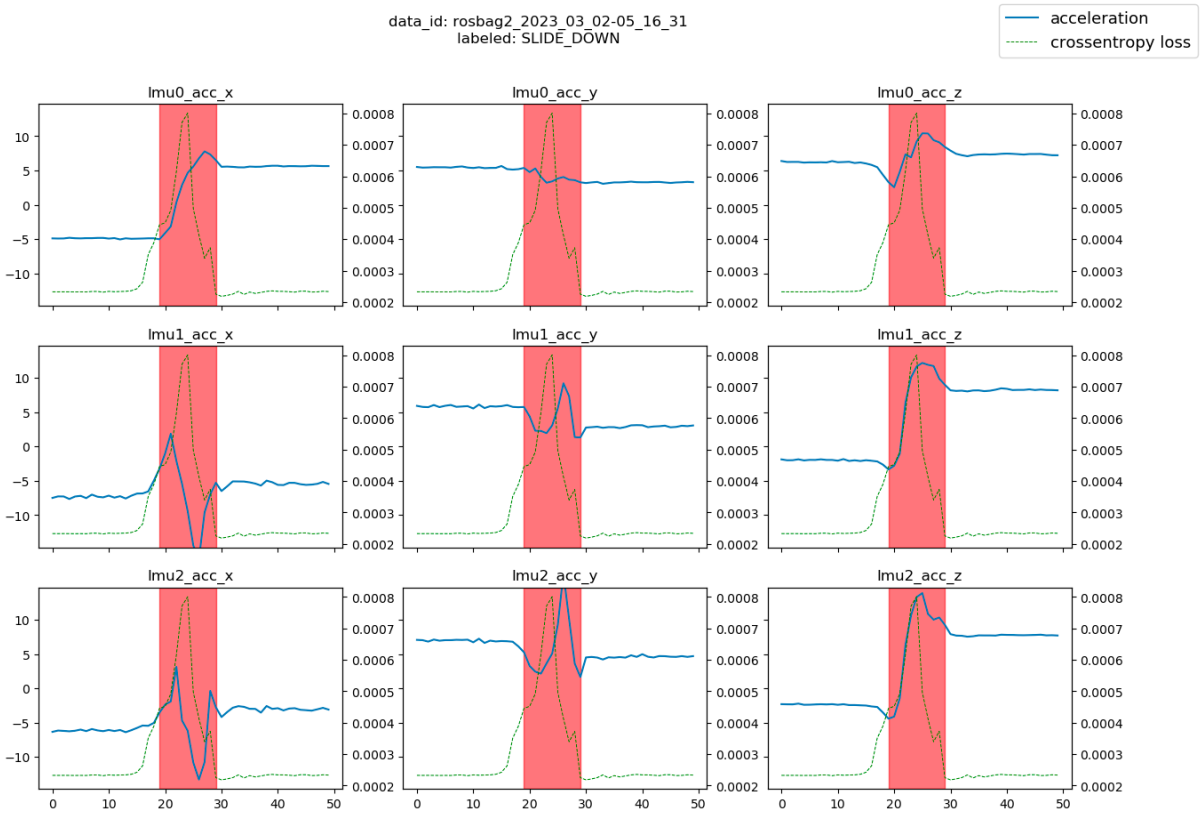


Figure 25 Timing importance

By visualizing the feature importance at the time step level, we were able to gain insights into how our model was making decisions. This information can be used to validate that our model could see and understand where the gesture is, rather than acting like a mysterious black box that humans cannot interpret.

3.5.6 Garbage Class

During the development of our gesture recognition model, we also considered including a garbage gesture to help differentiate between meaningful and meaningless recognition results. However, after careful consideration and analysis, we decided not to include a garbage gesture in our model at the end. The main reason for this decision was that it would result in too much variety, as all the gestures that do not belong to the predefined set would need to be classified as garbage gestures. Instead, we turned to set a confidence threshold to separate real gestures from any other meaningless motion. By using a confidence threshold for detecting gestures, we were able to effectively distinguish between meaningful and meaningless recognition results without the need for a garbage gesture and avoid misclassifying any random motion as the predefined set.

4 Conclusion

4.1 Summary of work conducted

To conclude what we have done for this project, it could be divided into four major parts.

1. Smart Glove
    a. Designed and implemented the hardware (MCU, sensors, PCBs) to be used on the glove
    b. Implemented the communication protocols to be used between the hardware components
    c. Implemented the wireless transmission of data collected from sensors to the server with multi-core capabilities
    d. Implemented the connection of ESP32 to personal Wi-Fi, WPA-Enterprise Wi-Fi as well as being a standalone access point
2. Off-pipeline Process
    a. Collected the dataset for gestures (around 3000 samples for 8 classes)
    b. Implemented data preprocessing to enhance the variation of data
    c. Implemented transformation of data for real-time visualization of our glove
    d. Trained the gesture recognition model, which achieved 95%+ accuracy on the training and validation set
    e. Performed feature and timing analysis to interpret the model
3. ROS Server
    a. Implemented the receiver module that receives data stream from the glove
    b. Designed the dataflow for the gesture recognition model to classify gestures in real-time.
    c. Implemented the commander module that can load pre-defined configurations and map commands to different appliances
4. Applications
    a. Implemented the Appliances Configuration Interface for enhancing configurability of devices
    b. Implemented the PowerPoint control on user's computer based on gestures received
    c. Implemented the LED control on ESP32 based on gestures received

4.2 Recommendation on the further development

Based on our analysis and experiments, we have identified some areas that could be improved for our gesture recognition system:

1. Incorporate more gestures, data, and variety to enhance the model's ability to recognize different situations. Collecting more diverse data will help train the model to recognize a wider range of gestures and improve its accuracy in real-world scenarios.
2. Explore the possibility of expanding the classification to include user-defined gestures. Instead of relying on predefined gestures, we can use the model's capability to understand the approximate meaning of the gesture to recognize user-defined gestures, similar to sentiment analysis. This approach would make the system more adaptable and flexible to the user's needs.
3. Improve the physical glove itself by finding more flexible wires or even fabric circuit. The current wires are inconvenient and can interfere with the user's experience, so using more flexible wires or fabric circuits could make the glove more comfortable to wear and use.
4. Implement authentication to our current system. The current server and applications can be controlled by any requests received with the appropriate JSON format. Authentication (such as JWT or OAuth2) should be implemented to identify the sending device.
5. Locate the user’s position: In our current method for device control based on orientation has limitations. It assumes that the user will always stay in the middle of a room, and the surrounding devices will remain in the same position. However, if the user moves away from the center or the devices' positions change, the orientation method would not work effectively. To address this issue, one potential solution is to use RSSI to determine the user's position relative to the surrounding devices.

By further development in these areas, we believe that our gesture recognition system can be further improved in accuracy, usability, and flexibility, and making it more suitable for various applications, such as virtual reality, gaming, and human-computer interaction.

5 References

[1] Y. Tam, T. Too ‘Minority Report home automation with hand gestures’, 2022.

[2] M. Kim, J. Cho, S. Lee, και Y. Jung, ‘IMU Sensor-Based Hand Gesture Recognition for Human-Machine Interfaces’, Sensors, τ. 19, τχ. 18, 2019.

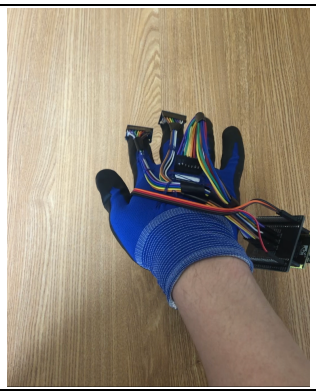
[3] A. Fu, ‘Real-time Gesture Pattern Classification with IMU Data’, 2017.

[4] B. K. Iwana and S. Uchida, “Time series data augmentation for neural networks by time warping with a discriminative teacher,” arXiv, 2020.

[5] Andrej Karpathy, “The Unreasonable Effectiveness of Recurrent Neural Networks,” karpathy.github.io. https://karpathy.github.io/2015/05/21/rnn-effectiveness/, May. 2015

[6] S. Lundberg and S.-I. Lee, “A Unified Approach to Interpreting Model Predictions,” arXiv:1705.07874 [cs, stat], Nov. 2017

6 Appendix A: Gesture Definition

Gesture	Definition	Description
Static		Hand remains still and motionless.
Slide up		Start with your palm facing down, relaxed and open. Slide your hand upwards from the bottom.
Slide down		Start with your palm facing down, relaxed and open. Slide your hand downwards from the top.

Slide left		Start with your palm facing left, relaxed and open. Slide your hand to the left from the right.
Slide right		Start with your palm facing left, relaxed and open. Slide your hand to the right from the left.
Release		Start by making a fist with your hand, end by opening your hand.
Grasp		Start with your hand open, end by making a fist.

Pointing		Start by making a fist with your hand, end with your index finger pointing forward to something.

7 Appendix B: Meeting Minutes

Minutes of the 1st Project Meeting

Date: June 10, 2022
Time: 08:00 pm
Place: Zoom
Present: Chan Cheuk Wai, LO Shih-heng
Absent: None
Recorder: LO Shih-heng

1. Report on progress
   1.1 All team members have read the instructions of the Final Year Project online and have done some research on the topic.
   1.2 All team members have done some research on the glove.

2. Discussion items
   2.1 Chan Cheuk Wai suggested that the glove could input characters by drawing in the air.
   2.2 Chan Cheuk Wai suggested that we could input characters by imaging typing on a virtual keyboard, and somehow our model could figure out the words.
   2.3 LO Shih-heng said instead of inputting explicit characters, gestures are another option for the input source.
   2.4 Chan Cheuk Wai said we could initiate commands by some explicit keyword such as “Okay Google”, “Hey Siri” or snapping fingers.
   2.5 LO Shih-heng said we could also consider using a pressure sensor to capture motion or even 5 IMUs on each finger.
   2.6 Chan Cheuk Wai raised the question of which MCU we are going to use for this project.
   2.7 LO, Shih-heng said we could install a small battery on the glove.
   2.8 Chan Cheuk Wai raised the question that do we buy or build the smart glove.

3. Goals for the coming meeting
   3.1 All members will begin to survey further use of the smart glove.
   3.2 All members will begin to think of practical end-application that our glove can apply to - for example, an AR game, a gesture language translator, or a remote control for home appliances.

Minutes of the 2nd Project Meeting

Date: Jun 21, 2022
Time: 8:00 pm
Place: Zoom
Present: Chan Cheuk Wai, LO Shih-heng, Prof. Brahim BENSAOU
Absent: None
Recorder: LO Shih-heng

1. Report on progress
   1.1 None

2. Discussion items
   2.1 The professor showed us various MCU we could use for this project. Both Chan Cheuk Wai and LO Shih-heng agreed that we would use ESP32 for its built-in Bluetooth and Wi-Fi modules.
   2.2 Professor gave us advice on the end application. We think a presentation helper is a good choice.

3. Goals for the coming month
   3.1 Both of us will begin to produce a project plan, including a list of the main tasks, who will work on each task and a GANTT chart.
   3.2 LO Shih-heng will start working on building the IMU library.
   3.3 Chan Cheuk Wai will start working on testing the communication between the ESP32 and PC.

Minutes of the 3rd Project Meeting

Date: Sep 2, 2022
Time: 8:00 pm
Place: Zoom
Present: Chan Cheuk Wai, LO Shih-heng
Absent: None
Recorder: LO Shih-heng

1. Report on progress
   1.1 LO Shih-heng had already finished the IMU library for a single IMU.
   1.2 Chan Cheuk Wai had already tested the communication between ESP32 and PC.

2. Discussion items
   2.1 LO Shih-heng suggested an overview of the project, separating the system into smaller tasks, including an Augmentation Module, a Visualization Module, a Recoding Module, a Decisioning Module, and an NLP model.
   2.2 Chan Cheuk Wai said he has experience in graphics so he will take the Visualization Module.
   2.3 LO Shih-heng said he would take the Augmentation Module and Recoding Module.
   2.4 Chan Cheuk Wai said we could use LSTM model to process the time series of the IMU raw data.
   2.5 LO Shih-heng also suggested using Self-Attention model to process the time series.
   2.6 We decided to work separately on each ML model and investigate which can bring better results.
   2.7 LO Shih-heng raised the question that if we are going to separate the system into small tasks, it will be challenging to well-organize the data and multi-program issues. As a result, we could try to use ROS2 as our system framework, which has a great ability to handle communication and multi-program coordination.

3. Goals for the coming meeting
   3.1 Survey on other possible frameworks for multi-program coordination
   3.2 Survey on possible NLP model

Minutes of the 4th Project Meeting

Date: Sep 6, 2022
Time: 4:00 pm
Place: Lab
Present: Chan Cheuk Wai, LO Shih-heng
Absent: None
Recorder: Chan Cheuk Wai

1. Report on progress
   1.1 LO Shih-heng had already finished the IMU library for a single IMU.
   1.2 Chan Cheuk Wai had already tested the communication between ESP32 and PC.

2. Discussion items
   2.1 We had an agreement on using ROS2 as our system framework
   2.2 Chan Cheuk Wai suggested that we could resize the data segment into one-second data as resizing will not change the feature of the original data.
   2.3 LO Shih-heng said that we could have a two-stage model, the first stage can be a Gesture Word Recognizer and the second stage can be a Gesture Sentence Semantic Analyzer

3. Goals for the coming meeting
   3.1 Start working on ROS2 modules
   3.2 Testing docker on different OS
   3.3 LO Shih-heng should build the Augmentation Module
   3.4 Chan Cheuk Wai should build the Visualization Module
   3.5 Chan Cheuk Wai should finish the Communication Module on the ESP32 side
   3.6 LO Shih-heng should start building the glove.

Minutes of the 5th Project Meeting

Date: Sep 20, 2022
Time: 8:00 pm
Place: Zoom
Present: Chan Cheuk Wai, LO Shih-heng
Absent: None
Recorder: LO Shih-heng

1. Report on progress
   1.1 Chan Cheuk Wai has drafted the visualization module and finished the script for setting up ROS Docker container with X11
   1.2 LO Shih-heng has finished the labeler and the augmentor module

2. Discussion items
   2.1 Demo what we have currently
   2.2 Discussed on filtering of IMU raw data
   2.3 Discussed how we can use the glove as a pointer
   2.4 Discussed the crafting of the actual glove

3. Goals for the coming meeting
   3.1 LO Shih-heng will start working out filtering on IMU raw data(investigate ROS package)
   3.2 LO Shih-heng will start building the glove
   3.3 Chan Cheuk Wai will make the bluetooth connection stabler
   3.4 Chan Cheuk Wai will finish the visualizer module and try implement it on ROS to connect with other nodes

Minutes of the 6th Project Meeting

Date: Oct 13, 2022
Time: 8:00 pm
Place: Zoom
Present: Chan Cheuk Wai, LO Shih-heng
Absent: None
Recorder: LO Shih-heng

1. Report on progress
   1.1. Chan Cheuk Wai implemented the Wi-Fi connection and sending configurations on ESP32S3
   1.2. Chan Cheuk Wai built a server written in JS for the glove to post data on for testing the ESP to perform HTTP POST request
   1.3. Chan Cheuk Wai has further implemented the visualization module to handle 3-dimensional inputs to rotate the 3D hand
   1.4. LO Shih-heng merge Chan Cheuk Wai's communication part into the main branch, and verify it works

2. Discussion items
   1.1. Discussed and reorganize the priority of our following month:
      1.1.1. Making sure all the glove functions are ready
      1.1.2. Visualization module on PC
      1.1.3. Labeling
      1.1.4. Designing the model
   1.2. Experimenting the merging of two parts of the glove code
   1.3. Discuss the format of the message to be sent to the pc

3. Goals for the coming meeting
   1.1. LO Shih-heng will try new IMU(MPU9250)
   1.2. Chan Cheuk Wai will change the routing server into Python with Flask such that it will be better compatible with ROS for the python package

Minutes of the 7th Project Meeting

Date: Oct 25, 2022
Time: 8:00 pm
Place: Zoom
Present: Chan Cheuk Wai, LO Shih-heng
Absent: None
Recorder: LO Shih-heng

1. Report on progress
   1.1. LO Shih-heng tested the new ESP32S3 and verify it works with the old code
   1.2. LO Shih-heng tested the new IMU(MPU9250)
   1.3. Chan Cheuk Wai changed the server from JS to Python with Flask
   1.4. LO Shih-heng checked Chan Cheuk Wai's communication code still works with the new server

2. Discussion items
   1.1. LO Shih-heng found out that sending the whole message seems to be too heavy for the ESP, suggested to assign the sending part and the IMU part to two different physical cores
   1.2. Chan Cheuk Wai thinks that it might be possible to deploy the entire system to a server such that the server can use static IP for ESP32 to send POST Request to

3. Goals for the coming meeting
   1.1. LO Shih-heng will finish sending the whole data
   1.2. Chan Cheuk Wai will try bridging the data with ROS

Minutes of the 8th Project Meeting

Date: Nov 25, 2022
Time: 8:00 pm
Place: Zoom
Present: Chan Cheuk Wai, LO Shih-heng
Absent: None
Recorder: LO Shih-heng

1. Report on progress
    1.1. LO Shih-heng had already finished the IMU library for a single IMU.
    1.2. Chan Cheuk Wai had already tested the communication between ESP32 and PC.
2. Discussion items
    2.1. Transfer Protocol
        2.1.1. Can try HTTP3
        2.1.2. Can try UDP with acknowledgement with timer
    2.2. Server
        2.2.1. Can be local
    2.3. IMU data collection
    2.4. NLP
        2.4.1. Build a gesture classifier for separating gesture words

Minutes of the 9th Project Meeting

Date: Feb 9, 2023
Time: 2:00 pm
Place: Professor's office
Present: Chan Cheuk Wai, LO Shih-heng, Professor Brahim
Absent: None
Recorder: LO Shih-heng

1. Report on progress
    1.1. LO Shih-heng finished TF Calculation Module
    1.2. LO Shih-heng wrote launch file
    1.3. LO Shih-heng finished glove time sync with SNTP
    1.4. LO Shih-heng fixed the recording frame dropped issue
    1.5. Chan Cheuk Wai started and LO Shih-heng implemented TF calculation algo
    1.6. We decided to not include the "end of gesture" token for the model.
2. Discussion items
    2.1. Start training models
        2.1.1. Transformer-based, Chan Cheuk Wai will have a try on this
        2.1.2. LSTM, LO Shih-heng will have a try on this
        2.1.3. CNN, LO Shih-heng will have a try on this
    2.2. Analyze the dataset
        2.2.1. Visualize the data
        2.2.2. See the similarity between same gestures
        2.2.3. See the difference between different gestures

Minutes of the 10th Project Meeting

Date: March 7, 2023
Time: 2:00 pm
Place: Professor's office
Present: Chan Cheuk Wai, LO Shih-heng, Professor Brahim
Absent: None
Recorder: Chan Cheuk Wai

1. Report on progress
   1.1. LO Shih-heng finished training the model (LSTM)
   1.2. LO Shih-heng model inferencing flow (building the RTOS buffer mechanism)
   1.3. LO Shih-heng investigate error cases of the model prediction
   1.4. Chan Cheuk Wai finished the commander module
   1.5. We changed the buffer length from 250 to 100 for faster response
2. Discussion items
   2.1. Professor Brahim said the gesture is not diverse enough (the dataset only represents my hand, but not others, for example, the model cannot recognize professor's gesture)
      2.1.1. Solution: collect data from others but not just us
   2.2. Professor Brahim said the Glove takes too long to connect to the NTP server. Possible solutions:
      2.2.1. Connect to a closer NTP server (preferably near HK)
      2.2.2. Sync the time with local machine

Minutes of the 11th Project Meeting

Date: April 11, 2023
Time: 2:00 pm
Place: Professor's office
Present: Chan Cheuk Wai, LO Shih-heng, Professor Brahim
Absent: None
Recorder: LO Shih-heng

1. Report on progress
    1.1. We changed the buffer length from 100 to 50 for even faster response
    1.2. We built the complete flow of our system working
        1.2.1. Glove connection to server
        1.2.2. Server get the data and do prediction based on data stream
        1.2.3. PowerPoint can be controlled based on the prediction result
    1.3. LO Shih-heng collected more data
    1.4. LO Shih-heng fix the slow sync issue
2. Discussion items
    2.1. Professor Brahim said that we need to make our presentation more exciting instead of showing we can control PowerPoint
    2.2. Professor Brahim said we need to make an interface for user to configure their own appliances
    2.3. Discuss how the writing of final reports
    2.4. Discuss the possible way for the glove to send command to a specific device
        2.4.1. Using Bluetooth to locate the glove in a room
        2.4.2. Using other sensors

8 Appendix C: Project Planning

8.1 Distribution of Work

Task	Chan Cheuk Wai	LO Shih-heng
Do the literature survey	●	○
Design the gestures	●	○
Make the physical glove	○	●
Develop sensors on the glove	○	●
Develop data transmission of the glove	●	○
Develop the Receiving Module	●	○
Develop the Filtering Module	○	●
Develop the Visualization Module	○	●
Develop the Labeling Module	○	●
Collect the dataset	○	●
Develop data augmentation program	●	○
Develop the gesture recognition model	○	●
Implement the model inferencing flow	○	●
Do analysis on the gesture recognition model	○	●
Develop the Decisioning Module	●	○
Develop the programs for end appliances	●	○
Write the reports	○	●

● Leader ○ Assistant

8.2 GANTT Chart

Task	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr
Do the literature survey										
Design the gestures										
Make the physical glove										
Develop sensor on the glove										
Develop data transmission of the glove										
Develop the Receiving Module										
Develop the Filtering Module										
Develop the Visualization Module										
Develop the Labeling Module										
Collect the dataset										

Develop data augmentation program										
Develop the gesture recognition model										
Do analysis on the gesture recognition model										
Develop the Decisioning Module										
Develop the programs for end appliances										

9 Appendix D: Required Hardware & Software

9.1 Hardware

1. Development PC: PC with Wi-Fi connectivity
2. Glove Control Core: ESP32S3 (1 unit)
3. Glove Sensors: MPU9250(3 units), GY26(1 unit)
4. IOT Control Core: ESP32S3 (1 unit)

9.2 Software

1. C/C++ For ESP32 programming
2. Python For NLP, visualization and recording tools
3. Docker Ubuntu20.04 image, for the whole glove system
4. ESP Driver and IDE ESP-IDF
5. Robot Operating System ROS2 Foxy, backbone of the server
6. Tensorflow/Keras For deep learning model development
7. Flask For communication between the glove and server
8. PyAutoGui For controlling PowerPoint on a PC