INVESTIGATE THE FEASIBILITY OF CAPTURING MIRCROMOVENTS ON FINGERS FOR SENSOR FUSION IN VIRTUAL KEYBOARD

YINAN. CHEN.

3rd Year Project Report

Department of Electronic &
Electrical Engineering

UCL

Supervisor: Arsam Nasrollahy Shiraz

4 April 2024

I have read and understood UCL's and the Department's statements and guidelines concerning plagiarism.

I declare that all material described in this report is my own work except where explicitly and individually indicated in the text. This includes ideas described in the text, figures and computer programs.

I acknowledge the use of ChatGPT-3.5[OpenAI],[http://chat.openai.com] to check grammar.

This report contains 18 pages (excluding this page and the appendices) and 5644 words (excluding this page ad the appendices).

Signed: __Yinan Chen__________________ Date: ______4th April________

(Student)

Investigating the feasibility of capturing micro-movements on fingers for sensor fusion in virtual keyboards

Yinan. Chen.

Abstract

This study explores the potential of optical sensors in capturing finger micro movements for virtual keyboard applications, aiming to overcome current limitations in accurately detecting finger movements. Two wearable devices, equipped with three infrared reflection sensors each—one worn as a ring and the other on the palm—were developed and tested. The keyboard design, based on the bending angle of the Proximal Interphalangeal (PIP) joint, resulted in a 16-key keyboard achieving 80% accuracy with the ring device. In a second version comprising 7 keys, the palm-worn device achieved 92% accuracy during testing with participants typing on a surface. Furthermore, the study investigated the relationship between Distal Interphalangeal (DIP) and PIP angles, expressing them as formulas, and explored the feasibility of capturing tiny movements. Results also revealed the device's ability to recognize high-frequency trembling, up to 25Hz. While infrared sensors show promise for virtual keyboards, further advancements such as sensor fusion techniques and improved detection reliability are essential to enhance usability and effectiveness. This underscores the need for future research to focus on enhancing detection capabilities and exploring applications in virtual reality environments.

1.Introduction

1.1 Introduction

1.1.1 Human Machine interface

Human-machine interfaces (HMIs) have undergone rapid development in recent decades, with hand gesture recognition emerging as one of the most prominent approaches, offering a high degree of control[1]. Various methods for hand gesture recognition exist, including camera-based[2][3], bio-signal[4][5][6][7], and data glove[8][9] systems.Camera-based solutions provide high resolution but may struggle with detecting hand gestures when hands are out of frame or when multiple hands are present. Bio-signal devices, typically worn on the forearm or wrist, detect muscle movements but often have lower resolution and limited gesture recognition capabilities[10]. Data gloves, which are equipped with strain sensors, offer high resolution but can be cumbersome and uncomfortable[11]. Those methods have also been employ in the medical field, researchers have explored the use of hand gesture recognition in studying Parkinson's tremor signals[12][13], employing technologies such as data gloves[14][15] and cameras[16][17].

1.1.2 Virtual keyboard

Hand gesture recognition has found applications in virtual keyboards, which are among the most common HMIs used for human-machine communication. In 2016, over 89% of households in the United States owned a laptop[18], a number that continues to grow due to economic and technological advancements, as well as the increased use of computers prompted by COVID-19 work-from-home policies[19]. Virtual keyboards allow users to reduce weight during travel[20] and personalize the typing experience to maximize speed and comfort[21]. Nowadays, virtual keyboards are primarily developed based on camera technology, such as Apple Vision Pro, enabling typing through hand gestures captured by cameras in headsets. However, a limitation of such systems is their ability to capture only one finger for each hand during typing[22]. Besides typing by hands, Electrooculogram (EOG)-based virtual keyboards are used by tracking eye movement for typing[23], but they also have limited capacity to process information, capturing only one finger movement per hand or one letter input at a time, resulting in inefficiency and usability challenges.

1.1.3 Virtual keyboard with sensor fusion

To address this limitation, research has focused on sensor fusion approaches. Some studies propose using high-precision cameras in conjunction with tracking marker gloves worn on the hands[24], while others suggest employing multiple cameras, including infrared (IR) cameras to track IR markers worn on the hands[25]. Sensor fusion approaches enable simultaneous capture of all hand movements but often require specialized equipment. For instance, tracking marker gloves, although effective, are bulky and commonly used in medical robotics for surgical procedures. Similarly, multi-camera setups may not be suitable for everyday use due to their complexity and impracticality in daily life scenarios.

1.1.4 Optical sensor in hand movement detection

In addition to the common methods mentioned above, alternative approaches using optical sensors have been explored for gesture detection. For instance, a system employ small fish-eye cameras worn on the hand for hand gesture recognition[26]. However, these systems may encounter challenges when fingers go out of frame, leading to task failure, and they often require heavy computational resources. Another innovative method involves the use of small IR sensors, as proposed in [27], which utilize wearable rings equipped with IR sensors to track finger movements. These rings analyze IR reflections on the skin to measure the angle between finger segments. This approach holds promise for sensor fusion applications, particularly in virtual keyboards, where optical sensors may be overly sensitive to minor finger movements. Similarly, a technique utilizing optical sensors placed on the back of the hand can detect hand gestures by sensing skin deformation [28]. However, there has been limited research on enhancing the detection of finger bending solutions in degrees besides from data glove, which represents a potentially crucial area for further investigation. Exploring micro movements, such as light tapping or detecting gestures in stroke patients, could yield valuable insights and applications.

1.1.5 Project aim

In this report, we aim to investigate the feasibility of using optical sensors to capture micro movements in fingers for sensor fusion applications in virtual keyboards. However, it's important to note that IR-based systems may encounter interference from external sources like sunlight or neighboring IR sensors, which can affect detection accuracy. Therefore, experiments will be conducted to mitigate environmental influences and enhance detection reliability.

1.2 Background and theory

1.2.1 The Choice of Photo-Reflective Sensor

Photo-reflective sensors have been employed in various fields, such as facial expression recognition[29]. In one instance, photo-reflective sensors placed inside a glass were able to detect eight different expressions. Additionally, they have been utilized in recognizing finger movement while touching the back of the hand[30], which can serve as a panel for interfacing with other screen devices. These applications highlight the efficacy of photo-reflective sensors in detecting human movement. In such cases, phototransistors are used as part of the IR reflector due to their high sensitivity and linear response to light intensity[31].

1.2.2 Calculation of Bending Angles for Joints

In the report, camera-based recognition is employed for labeling finger bending, and MediaPipe is utilized to label the bending angles for each joint while the participant wears the designed device. MediaPipe is an open-source library capable of labeling hands as skeletons from a camera, with the segments for fingers saved as vectors. The angles for each joint are computed based on the hand's skeleton by recognizing joint positions in the image. For each segment of a finger, a vector is employed. The angle can be calculated using the equation:

$$\theta = \sin^{-1}\left(\frac{|\vec{a} \times \vec{b}|}{|\vec{a}||\vec{b}|}\right)$$

where $\vec{a}$ is a vector representing middle phalanges, and $\vec{b}$ can be a vector of proximal phalanges, θ is therefore representing the bending angle of PIP.

1.2.3 Relationship between distal interphalangeal (DIP) and proximal interphalangeal (PIP) joints

With the device developed in the report, we utilize one IR reflector to monitor both DIP and PIP angles. We aim to rely on machine learning to identify patterns since we cannot control the movement of DIP angles. Through the use of a data glove, the relationship between PIP and DIP angles has been identified, as depicted in Figure A.

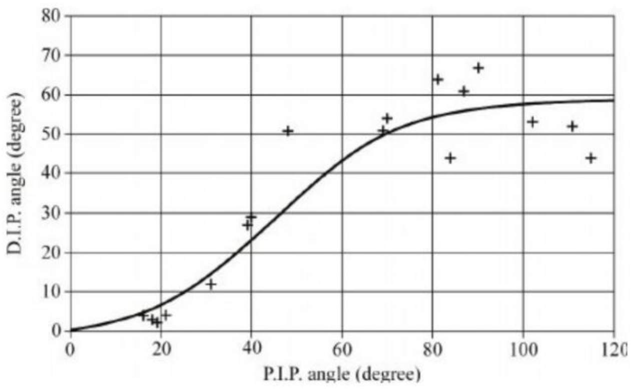


Figure A: Experimental data (crosses) and analytical fit (line)of D.I.P.–P.I.P. flexion angles for a V-plus finger[32]

1.2.4 Machine Learning Model Selection

With optical devices, there exists a correlation between different sensors, making it impractical to rely solely on the linear relationship between sensor readings and finger bending angles. To classify movement, machine learning is necessary. In this report, XGBoost has been chosen over other models such as SVM and MLP, as it has demonstrated superior performance [33][34]. XGBoost was selected for its exceptional processing speed and its ability to handle noise effectively due to its decision tree structure. In this structure, each tree acts as a weak guesser, but when combined, their weighted contributions help to mitigate the impact of noise. As a result, XGBoost is adept at discerning relevant features while disregarding noise[35] .

1.3 Literature review

1.3.1 Micro-movement capture and calibration

The data glove stands out as the most convenient and reliable method for capturing subtle hand movements. Numerous studies have demonstrated its ability to measure joint bending angles with high resolution, achieving approximately 2 degrees of accuracy[36][37]. These studies often calibrate the glove using wooden blocks positioned at different slopes, ensuring precise measurement of bending angles. However, optical devices like our proposed device face challenges with this calibration method, as the blocks placed between proximal and middle phalanges can block light and hinder accurate detection.Alternatively, some studies propose using advanced depth cameras like the Leap Motion for hand skeleton capture[38], offering high accuracy in capturing hand movements.

1.3.2 IR interference

Despite the calibration, environmental interference remains a concern when using infrared devices. IR thermometers, for instance, rely on IR emitted from the skin to measure body temperature but can be affected by sunlight, temperature fluctuations, and humidity, leading to increased uncertainties[39]. Moreover, differences in skin color can impact emissivity, with darker skin reflecting less light[40].To address interference and noise in IR devices, recurrent neural networks have been applied in noise reduction, such as in audio processing which similar to our analog data output [41]. Additionally, the placement of sensors in close proximity on the device may lead to interference between readings from different sensors. Inspired by devices like ThumbTrak[42], which capture the relationship between the thumb and other fingers, this interference can be leveraged as valuable additional information for hand gesture recognition.

1.3.3 Adaption to VR headset

Considering sensor fusion approaches in virtual keyboards, which integrate camera-based systems from headsets with glove-based systems[24][25], the device developed in this report could offer a new solution for virtual keyboards integrated with VR headsets. This integration could enhance the user experience and usability of virtual keyboards in VR environments.

2.Methods and Material

2.1 Hardware

In this report, a simple circuit comprises one IR emitter(VSMY2940G 940nm) and one IR phototransistor(SFH 3600-Z) is used for detecting a single bending angle between the proximal interphalangeal (PIP) joint. An input voltage of 3.3V is supplied to the IR emitter in the form of a pulse wave. The circuit constitutes a single sensing unit. (refer to Figure B(Left)) By integrating two sensing units, a ring is created to be worn on each finger. One sensing unit is utilized to measure the distance between the ring (located at the base of the finger) and the middle phalanges (middle segment of a finger), and even to the fingertip. As the sensing unit points upward, with an increase in the PIP angle, the distance between the sensing unit and the middle phalanges would increase, leading to an increase in the intensity of the IR phototransistor. Additionally, apart from pointing upward, another sensing unit placed on the side can be used to measure the spread of the finger to its neighboring finger (refer to Figure B(Right)).

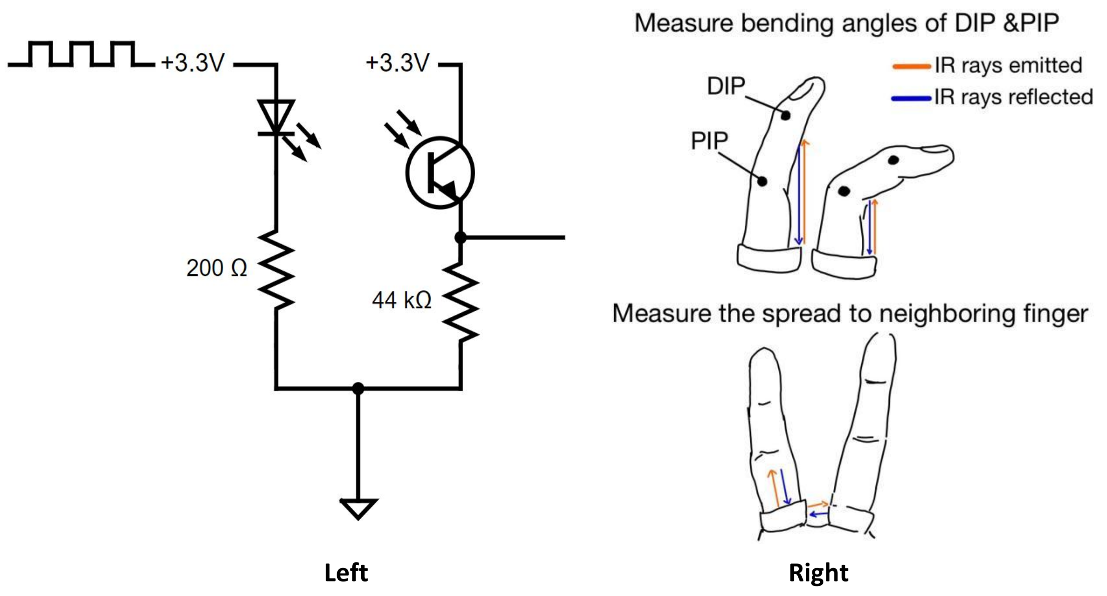

+3.3V
+3.3V
200 Ω
44 kΩ
Measure bending angles of DIP &PIP
DIP
PIP
IR rays emitted
IR rays reflected
Measure the spread to neighboring finger
Left
Right

Figure B: (Left): Circuit diagram for one sensing unit. (Right): Measurement on bending and spread by different placement of sensing unit.

With four sensing units together, it can gather information for the index, middle, ring, and little finger. In the following discussion, we only focus on the sensing unit pointing upward, as we only investigate the bending angle of the PIP and distal interphalangeal (DIP) joints. All readings are sent to the ESP32 with a flashing LED at 40Hz, and the sampling frequency is also set to 40Hz. For real life application, the system is wireless which can communicate with laptop by WIFI, and with a 9V battery for power supply. The following figure C present the working flow of the device.

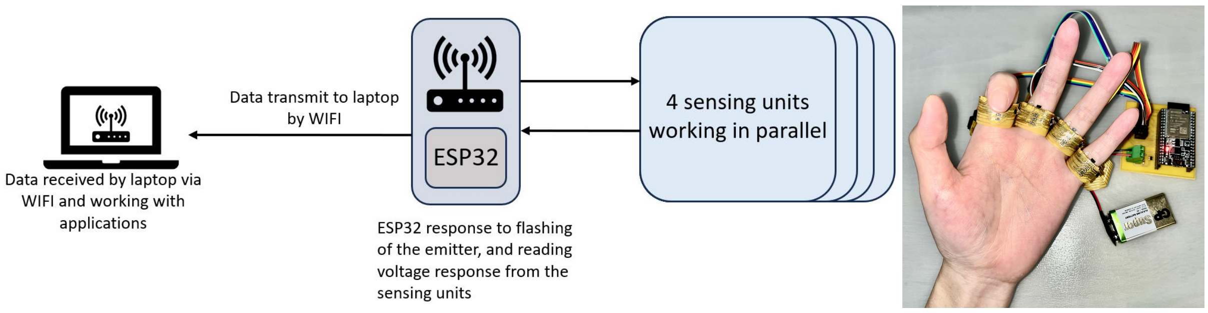

Data transmit to laptop by WIFI
Data received by laptop via WIFI and working with applications
ESP32
ESP32 response to flashing of the emitter, and reading voltage response from the sensing units
4 sensing units working in parallel

Figure C: Working principal of the device and actual setup.

With the serial data received from the ESP32 to the laptop, we would like to process the data with labels, indicating the bending angles of the PIP and DIP joints for each value obtained from our sensing unit. To achieve this, a camera-based hand gesture recognition system is used for labeling.

2.2 Recognition system with machine learning

2.2.1 Parallel recording

In order to do the labeling, we construct our system with the following the flowchart figure D, code can be found on appendix A.

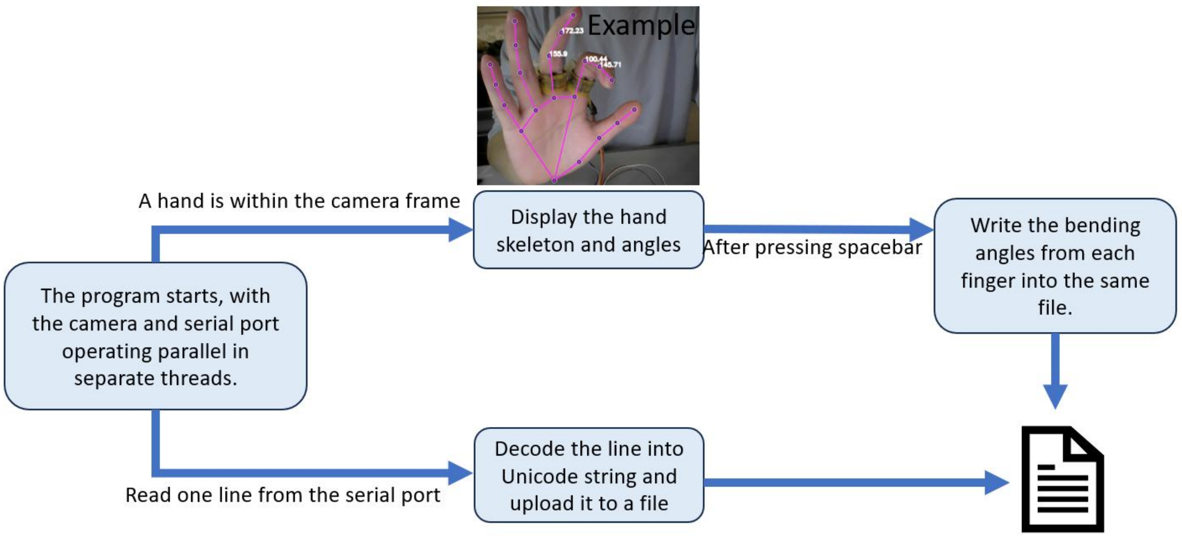

Example
A hand is within the camera frame
Display the hand skeleton and angles
After pressing spacebar
Write the bending angles from each finger into the same file.
The program starts, with the camera and serial port operating parallel in separate threads.
Read one line from the serial port
Decode the line into Unicode string and upload it to a file

Figure D: Flowchart on parallel reading for serial and camera hand gesture data collection. Using parallel threads, it captures hand gestures via the camera and reads voltage responses from the device via the serial port. Both sets of data are then combined and written into a single file, streamlining the data collection process.

In camera reading, ,ediapipe used, which shown on Figure D, as pink skeleton, and the number labels indicate the bending angles for each joints which can be calculated based on the spacial relationship which have been explain in background.The camera is operating at 30fps, and therefore the angles representing bending angles for each join angle can be printed out every 33ms. Inside the file, the serial data coming from the ESP32 are written in lines, with each line consisting of 8 values. The first 4 values represent the IR readings, which reflect the intensity from the finger when the IR LED is on. The last 4 values represent the IR readings from the environment when the IR LED is off, serving as the reference level. Upon pressing the spacebar, camera information such as the orientation of the detected hand and the bending angle of each finger are written following each line of serial data. However, it's important to note that the data may not always be synchronized due to hardware limitations[43]. Therefore, manual checking is required to determine any shift between the serial port data and camera information. At a low frequency of 10Hz, the camera tends to write data slower, resulting in a delay of approximately 2-5 lines. Conversely, at higher frequencies such as 50Hz, the serial port may become slow, causing a delay of about 6-9 lines as the serial port information buffers during the import stage.

2.2.2 Machine learning with XGBoost

After synchronizing the data, we can proceed with machine learning. In this case, we opted for XGBoost due to its rapid training speed and exceptional performance in noisy environments. To effectively utilize the model, ample training data is essential. In this report, we have amassed over 3000 samples for training. Within the dataset, the initial 8 readings are sourced from the serial port, representing the voltage response and serving as input features. The subsequent 8 values denote the recorded bending angles for each joint, extracted from camera data and labeled as output. To prevent overfitting during classification, we train on a single model for all bending angles. Numerous parameters can be adjusted, with the chosen values summarized in the table below:

Learning_rate	Max depth	alpha	Number of estimators
0.42	8	30	100

A relatively high learning rate was selected, reflecting the relatively straightforward nature of the recognition task. Additionally, the model training process can accommodate the reading of one or more lines of serial inputs at the outset. This approach helps mitigate sudden jumps or recording errors, thereby minimizing inaccuracies and enhancing overall accuracy.

For further insight, the detailed code can be referenced in Appendix B. Upon completion of training, the model is exported for potential validation in alternative scenarios.

3.Experiment with device

3.1 Plotting classified and label values verse time

With the model created, we were able to classify finger bending for each joint, code for plotting can be found on Appendix C. During the experiment, a participant sat in front of the camera while wearing the device. The participant was then instructed to move all fingers freely for 30 seconds, with no limitation on speed. However, it was crucial to ensure that the hand wearing the device remained at a relatively constant distance from the camera and that the entire hand remained within the frame throughout the recording period. The error can be calculated by comparing the absolute difference between classified values and the recorded values from the camera, and the accuracy of the devices can also be obtained through this comparison.

3.2 First version keyboard

In addition to comparing classification and camera plotting, a virtual keyboard was employed in the initial version of the experiment. Appendix D contains the code used for implementing the virtual keyboard functionality.The recognition system relied on classified values from a previously trained model. Each finger was assigned four keys for selection, determined by the bending angle of the DIP joint. The selection range was evenly divided into four intervals: 180-165, 165-140, 140-115, and 115-90 degrees. During the experiment, participants randomly bent their fingers in the air, and each gesture lasted for 4 seconds. The recorded gestures were then compared with the classified results, with a total validation time of 2 minutes. For example, when the finger bending resembled the state depicted in Figure E (left), the virtual keyboard would produce output corresponding to the keys shown on the right. In our setup, each row represents the bending angle of a specific finger: the index finger, middle finger, ring finger, and little finger, in respective order. The bending angles are indicative of the keyboard selection, with higher angles corresponding to a movement towards the right on the virtual keyboard. It's important to note that the actual finger bending angles were not directly classified during the experiment but were determined based on visual observation by participant.

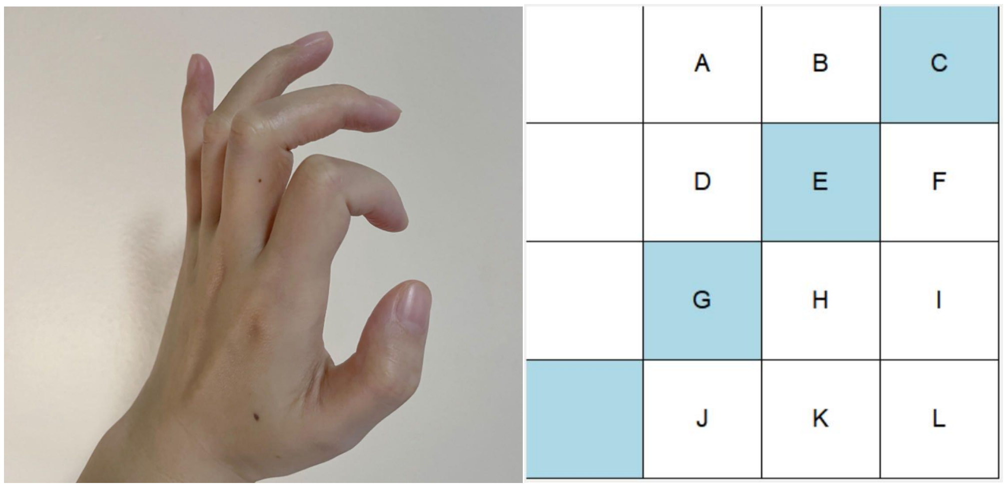


Figure E: On the left side, the illustration depicts the hand gesture corresponding to the output shown on the virtual keyboard on the right. The recognition system identifies the index finger as bent within the range of 115-90 degrees, the middle finger within 140-115 degrees, the ring finger within 165-140 degrees, and the little finger within 180-165 degrees.

3.3 second version keyboard

To further enhance the experiment, another virtual keyboard has been designed, where user is need to type on a surface. This keyboard offers a more scientifically robust scale as it mitigates potential camera errors. The machine learning model is based on the virtual keyboard itself, simplifying the classification process to only identify the key pressed.In the virtual keyboard setup, each finger is capable of selecting one out of 3 keys, except for the ring finger, which can select only 2 keys. Taking into account a participant's hand size of 17cm, measured from the end of the palm to the tip of the middle finger, and a length of the index finger of 6.7cm, we design our keyboard with radius of 0.7cm circle. The keys are not evenly spread but follows the shape outlined below, which enables the most comfortable way of pressing keys while only the fingers are moving. The participant hand size is draw in the background, the position of keys and joints position can also be observed. The figure F printed and work as a keyboard.

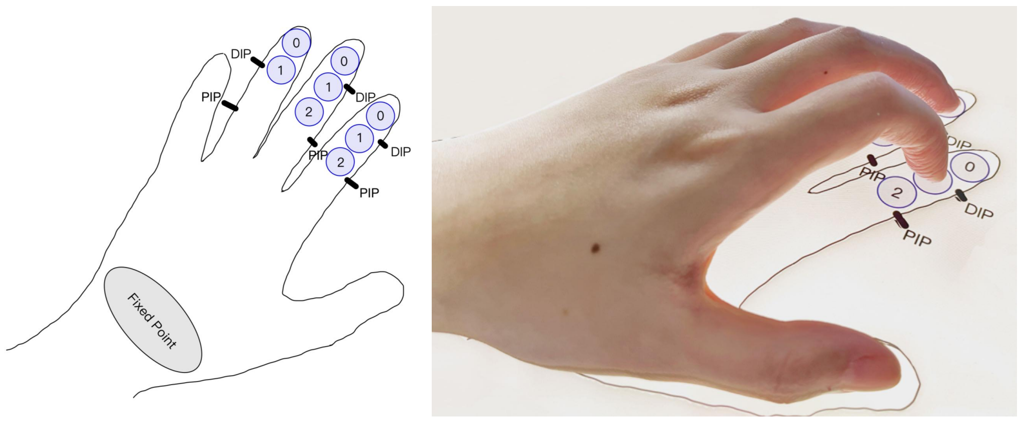


Figure F： On the left side, it shows the keyboard layout and the size of participant hand with label of joints positions. On the right side size shows how participant use the keyboard to pressed key 1 on index finger.

During the experiment, the carpal bones (located at the end of the palm, close to the wrist) of the participant were positioned on the fixed point marked on the picture. This fixed point served as a reference in the image capture process. To ensure that only one finger movement was recorded at a time, the other fingers were held in a fixed position (labeled as position 0) while one finger was actively moving. This precaution helped isolate the movement of the selected finger and minimize any interference from adjacent finger movements. The following figure shows the expectation of the experiment.

3.4 With sensing unit wearing on palm

The wearing device was redesigned with a new setup which can refer to figure G: instead of wearing the sensors as individual rings on each finger, they were positioned together on the palm. This setup allowed for a different approach to capturing finger movements. In this new configuration, each sensor could capture more interference from neighboring fingers due to the wider view from sensor, the intensity of the interference signals was relatively stronger compared to the previous setup. The aim was to convert this interference from noise into useful information for measuring the bending of both fingers simultaneously. The performance was test by using the second version keyboard.

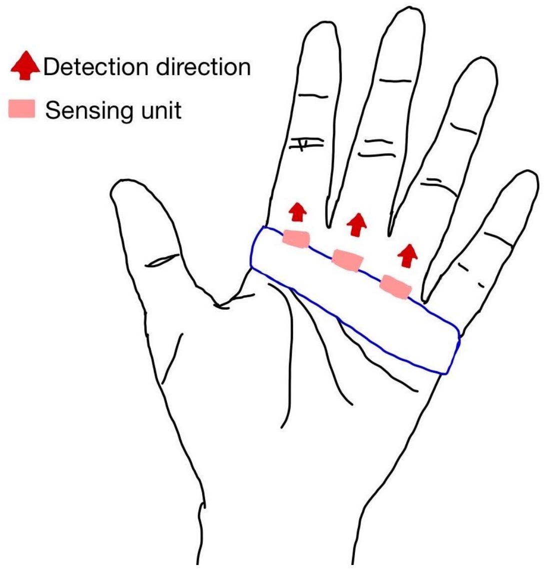


Figure G： New setup of the device, the device is wearing on palm and with sensing units facing toward each finger.

3.5 Test on trembling

Trembling is a common occurrence in Parkinson's patients, yet there is currently no device capable of fully studying Parkinson's tremor signals. To simulate this phenomenon reliably, a massage gun was utilized. The massage gun was programmed to target the PIP joint with a frequency of 25Hz, and the resulting data was recorded for analysis.

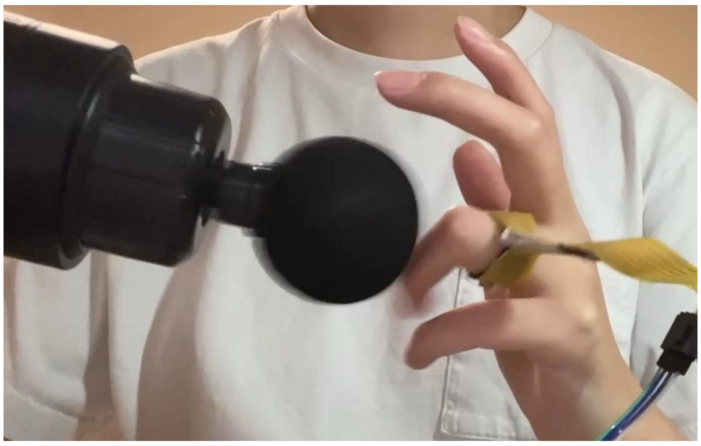


Figure H: How massage gun knock on finger

4.Result

4.1 Result of real time plotting

The results for real-time plotting are presented below, with only a 2-second recording shown for clear presentation. The overall accuracy for all fingers' classification, with an absolute error within 15 degrees, is 81%. The average absolute error for all classification is within 9 degrees.

Accuracy table for each joints	Index	Middle	Ring	Little
DIP	0.8000	0.7333	0.8133	0.8800
PIP	0.8533	0.7467	0.7867	0.8667

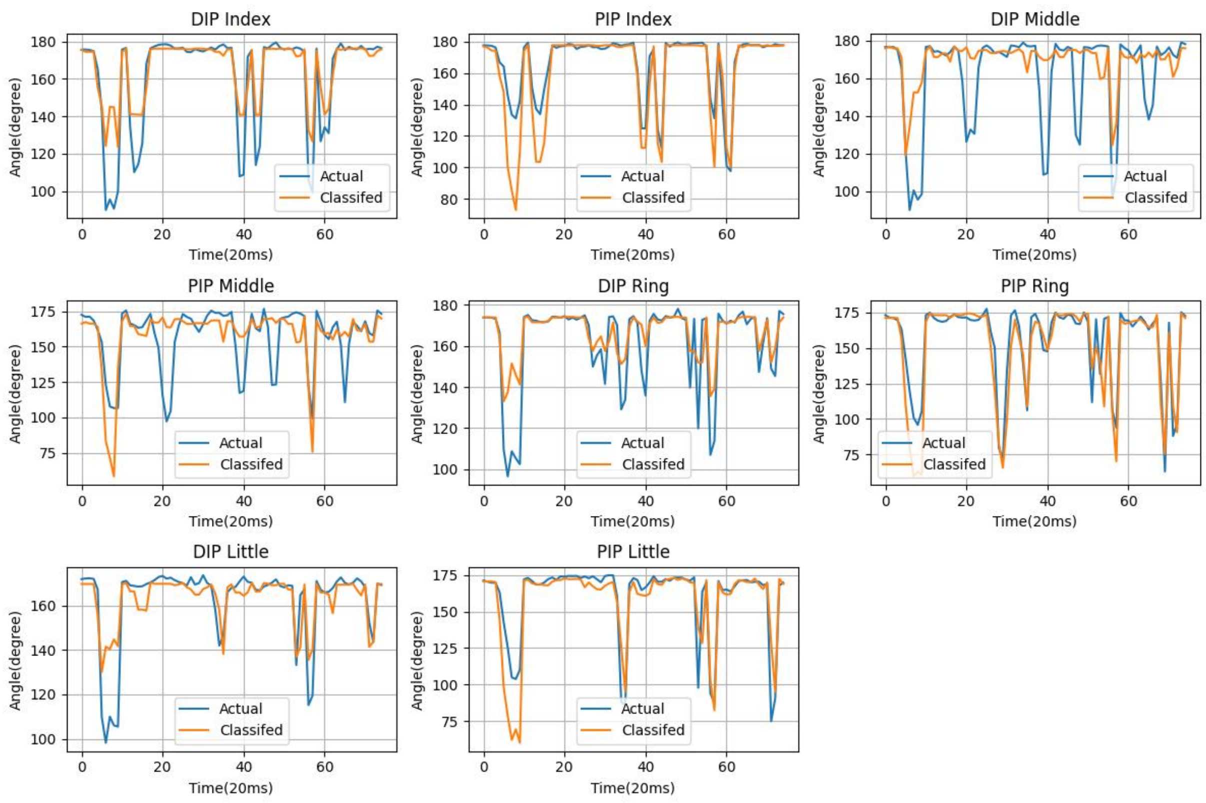
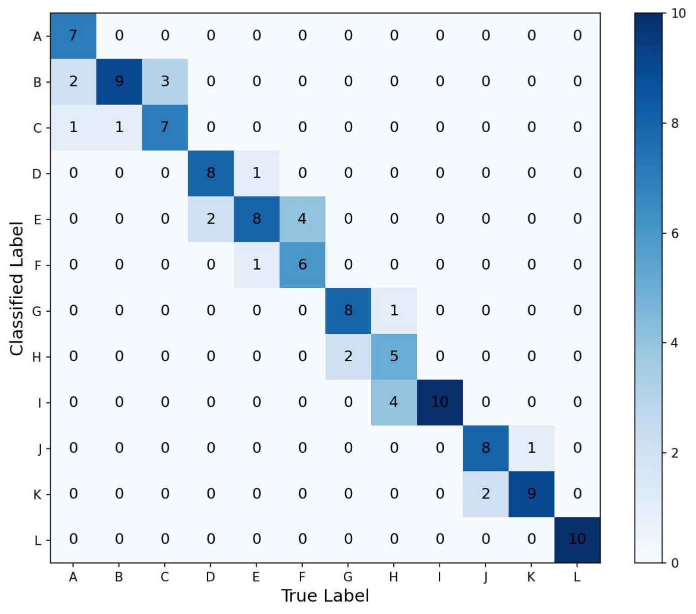


Figure I: The classified bending values for each joints verse values record from camera.

Beside from the accuracy, there is a flaw in the prototype, as the sensors are not always pointing toward the same direction, as it is easy to bend on a flexible PCB, the model need to retrain each time before using.

4.2 First version keyboard result

The accuracy of this method is approximately 80 percent by identify total 30 gestures without recording the empty key where multiple keys are pressed. The confusion matrix:



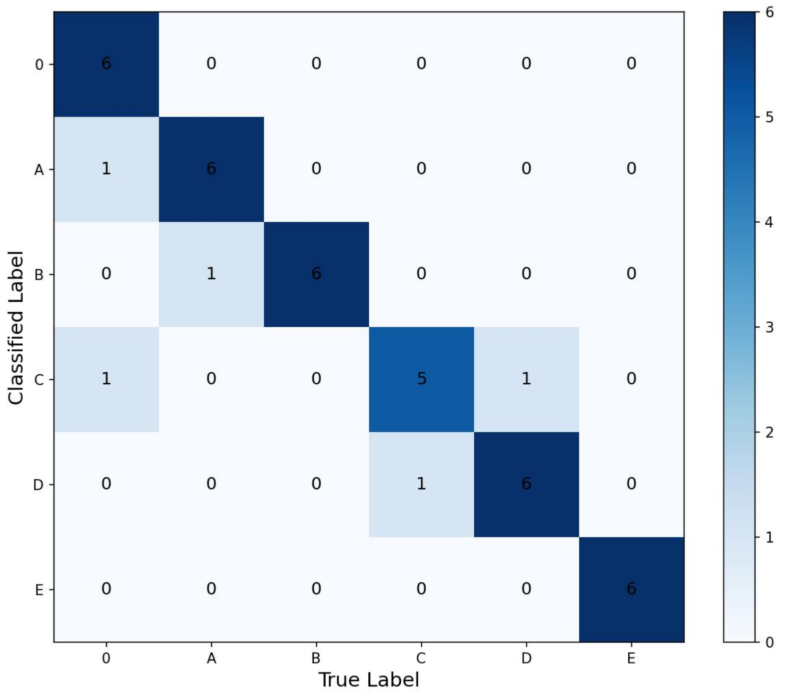
	A	B	C	D	E	F	G	H	I	J	K	L
A	7	0	0	0	0	0	0	0	0	0	0	0
B	2	9	3	0	0	0	0	0	0	0	0	0
C	1	1	7	0	0	0	0	0	0	0	0	0
D	0	0	0	8	1	0	0	0	0	0	0	0
E	0	0	0	2	8	4	0	0	0	0	0	0
F	0	0	0	0	1	6	0	0	0	0	0	0
G	0	0	0	0	0	0	8	1	0	0	0	0
H	0	0	0	0	0	0	2	5	0	0	0	0
I	0	0	0	0	0	0	0	4	10	0	0	0
J	0	0	0	0	0	0	0	0	0	8	1	0
K	0	0	0	0	0	0	0	0	0	2	9	0
L	0	0	0	0	0	0	0	0	0	0	0	10

Figure J: The confusion matrix of testing on first version keyboard.

From the result, the model is trying to minimizing the overall error by guessing middle values. For instance, the model tends to classification values closer to the middle of the range. For large bending values, such as 90 degrees, the model tends to classify values slightly higher, around 100 degrees. Similarly, for a bending angle of 180 degrees, the model tends to classify a value closer to 170 degrees. Additionally, the movement of adjacent fingers, particularly when the little finger is bending, can influence the readings of other fingers. This phenomenon results in inaccurate measurements, prompting the exclusion of the little finger from the analysis.

4.3 Second version keyboard result

The accuracy of this method has notably risen to 87.5 percent. In this keyboard setup, each finger's action corresponds to specific outputs. When all fingers are either pressed on key 0 or in the air, it signifies the output as 0. If the index finger is pressed, resulting in either 1 or 2, it denotes outputs A or B respectively. Similarly, the middle finger's action indicates outputs C or D, and for the ring finger, pressing 1 signifies output E. Confusion matrix shown below with total 1500 data for training. The validation test on 40 samples.



Classified \ True	0	A	B	C	D	E
0	6	0	0	0	0	0
A	1	6	0	0	0	0
B	0	1	6	0	0	0
C	1	0	0	5	1	0
D	0	0	0	1	6	0
E	0	0	0	0	0	6

Figure K: The confusion matrix of testing on second version keyboard.

When using the keyboard setup, typing occurs on a surface, and individuals have varying preferences regarding the placement and height of their palms. These differences leads to significant errors if they deviate from the training data. Therefore, the participant was need to be in a fix posture during training and validation.

4.4 Result of changing sensor arrangement with second version keyboard

Placing sensors on the palm results in correlated voltage readings across different fingers. Specifically, the readings of middle finger movement can be validated by three sensors simultaneously. In Figure L below, we observe the impact of device positioning on these readings.

The following image displays the voltage response over time when bending the index finger alone, bending the middle finger alone, and bending both fingers simultaneously in a time series. It illustrates these responses for two different setups.

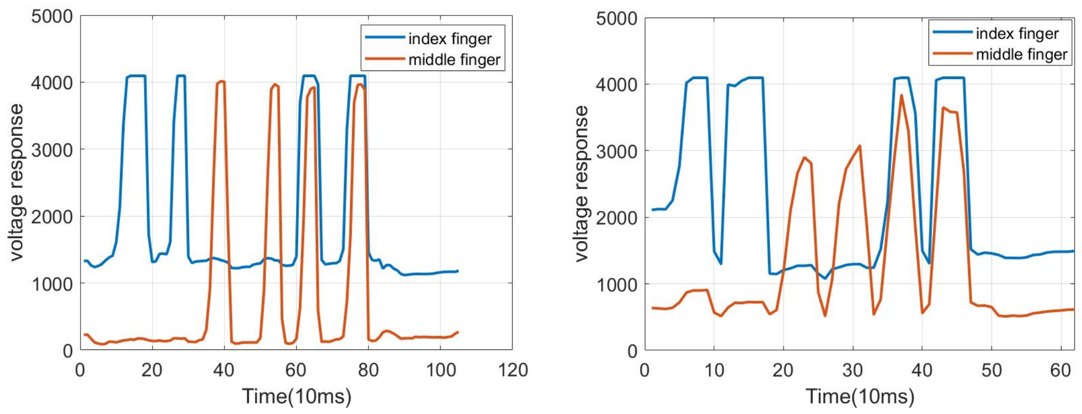


Figure L: On the left, it demonstrates the interaction between two finger movements with the original setup. On the right, it showcases the response when wearing a device on the palm.

Analyzing Figure L reveals that changes in the sensing unit's position lead to readings that may incorporate information from neighboring fingers. For instance, when the index finger or middle finger bends individually, there is a corresponding increase in response from another sensor unit.

With the device wearing on palm, we test its performance with second version virtual keyboard.

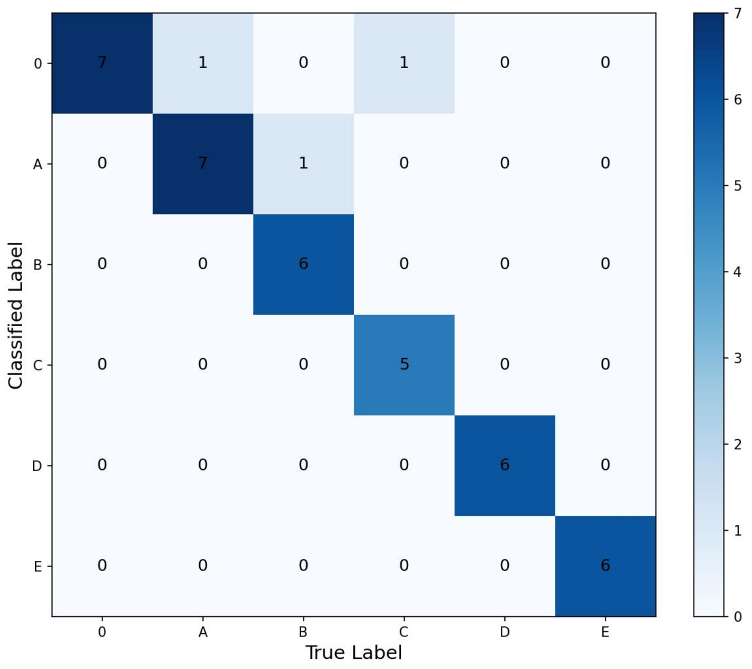


Figure M: The confusion matrix of testing on second version keyboard with new setup.
With the new setup, the result can give up to 92% accuracy with keeping the same procedure when we test on the ring device.

4.5 Test result of trembling recognition

During the simulation trembling test, the device is capable capture fast movement and predict the angles of trembling. The real time plotting is pretty slow, as the sampling frequency is too big for running it in real time while we are using machine learning model for classification. In figure N, it shows how model being able to record when finger is trembling in 25Hz.

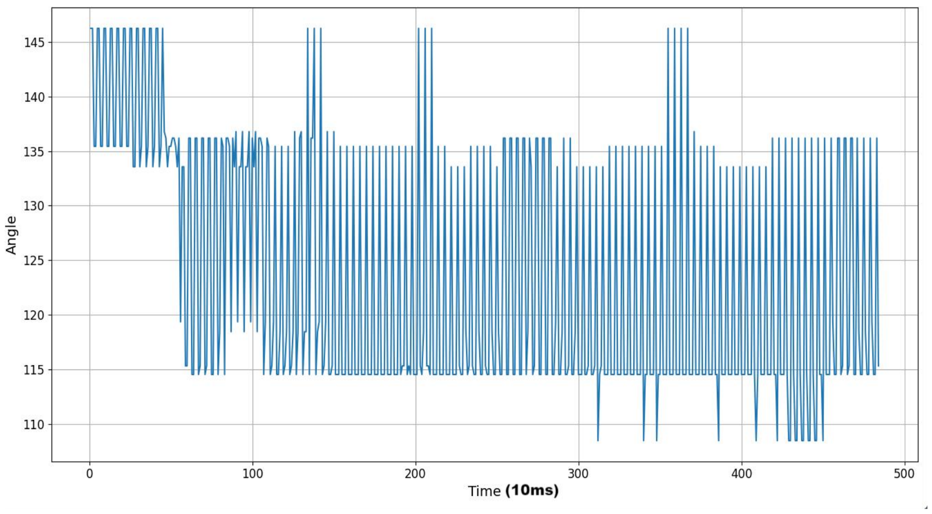
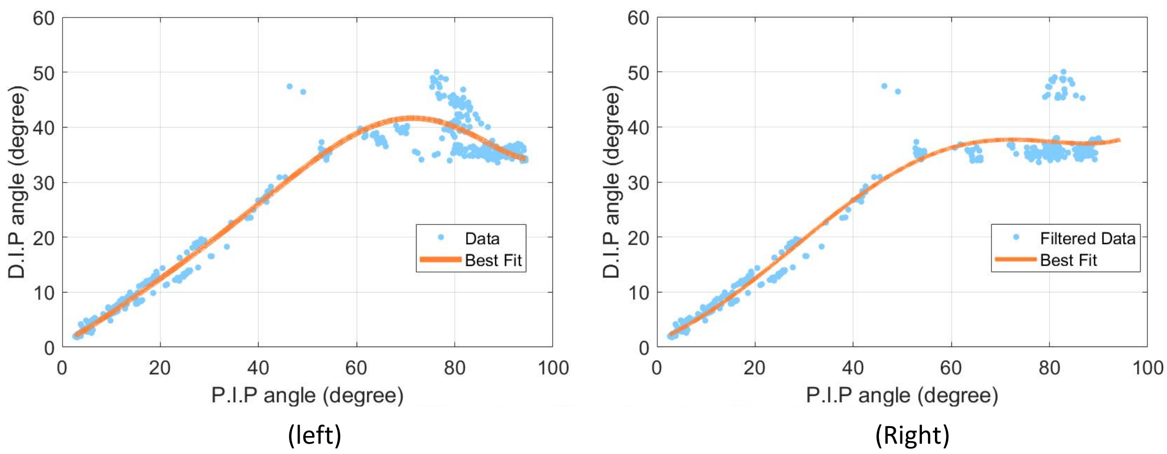


Figure N: The plotting of trembling by the device.

5.Discussion

5.1 Discussion of real time plotting result

The classification for the PIP angles are generally accurate in machine learning with XGBoost, but for the DIP angles, the error is significant. This discrepancy arises because we only have one sensor for recognizing the bending, which includes both the bending from the DIP and PIP. Additionally, the camera shows poor performance in terms of angle bending calculation. Generally, the error is less than 5 degrees, but it can sometimes be substantial, especially when bending fingers beside the index finger. For instance, when bending my middle finger to 90 degrees, it shows the bending angle as 65 degrees, and the DIP angle is also affected due to the mismatch of hand skeleton.To reduce the error of the PIP angle, followed the methodology[32], relationship between the PIP and DIP can be found as they follows a certain pattern. By plotting the DIP versus PIP angles for the index finger (as shown in Figure A), we observed a similar relationship to that described in the paper after removing several data points that could be considered mistakes in camera-based recognition. Which indicate there are great error with the use of camera in hand gesture when PIP angle is big.



(left) (Right)

Figure O: (Left) Best fit for raw data of DIP verse PIP getting from camera and raw data of index finger. (Right)Best fit for filtered data of DIP verse PIP getting from camera and raw data finger

5.1.1 Relationship between DIP and PIP

Following the same steps, we were able to obtain the bending angle of the DIP for all four fingers using their own functions related to the PIP. The relationship between the DIP and PIP for the index finger, based on the best fit, can be expressed as:

$$Y=2.45 \times 10^{-8}x^5 - 3.84\times 10^{-6}x^4 + 5.6 \times 10^{-5}x^3 + 0.00814x^2 + 0.408x + 1.1306$$

where x is the bending angle of the PIP and Y is the bending angle of the DIP.Other fingers follow the following equation,

Middle finger DIP angle=

$$4.06\times 10^{-8}x^5 - 4.73\times 10^{-6}x^4 - 0.000363x^3 + 0.0567x^2 - 0.899x + 6.4989$$

Ring finger DIP angle=

$$-3.66 \times 10^{-9}x^5 + 9.43 \times 10^{-7}x^4 - 4.52\times 10^{-5}x^3 - 0.00572x^2 + 0.67x - 4.5042$$

Little finger DIP angle=

$$8.09\times 10^{-8}x^5 - 2.43\times 10^{-5}x^4 + 0.00237x^3 - 0.0764x^2 + 1.02x + 2.9233$$

In conclusion, to classification of DIP angle, it will be more accurate by directly employ the equations above, as the relationship between DUP and PIP is almost fix.

5.1.2 Low accuracy in middle finger

Beside from the DIP error, we can also observe the error in detecting middle finger is huge compare with other finger. The guessing on on with the low accuracy is that the reading of middle finger can be affect by 2 fingers, index and ring, which provide interference to the linear relationship between bending angle and voltage response. Even thought ring also between 2 fingers, but the movement of little finger is almost synchronize to ring finger, therefore for ring finger, there aren't reading too many interference as middle finger do.

5.2 Keyboard comparison:

Both keyboard setups exhibit decent accuracy. However, the first keyboard's recognition method relies on the bending angle of the DIP joints, which may not be suitable for typing as it requires users to adapt to a new typing technique. This method is better suited for tasks such as sliding control, such as adjusting volume. Beside, from the unreliable camera based in calculating the bending angle introduce too many error in data training.

On the other hand, the second version of the keyboard closely resembles a traditional keyboard, despite having fewer keys and an unconventional layout. The limit number of keys is due to difficulties in recognition of whole hands movement with the optical sensors alone. To enhance performance and work in a standard keyboard layout, incorporating a camera or inertial momentum units to monitor the entire hand movement could be beneficial. However, it's worth noting that using infrared technology and typing on a surface may introduce noise, particularly from reflective surfaces like tabletops, leading to discrepancies between the actual input and the classified value.

5.3 Palm setup

By transitioning the device to be worn on the palm, sensor readings become more reliable due to the interrelated nature of the signals. This setup offers increased freedom of movement for the fingers. Further improvements in reliability can be achieved by adding two additional sensors: one positioned adjacent to the index finger and another adjacent to the ring finger. With this configuration, all finger movements can be validated using multiple sensors.

5.4 Micro-movement Capture:

The results demonstrate the device's capability to accurately capture high-frequency trembling, making it a valuable tool for keyboard usage. This feature enables users to type rapidly, as micro-movements are clearly captured by the optical sensor. Beyond typing applications, the device can also serve as a monitoring tool for Parkinson's patients, facilitating early detection and intervention.

6.Conclusion:

The relationship between the DIP and PIP angles for each finger have been discovered, providing a reference for freely bending the hand without external force. In the experiment of virtual keyboards, the voltage response was clear for human interpretation when only a few keys were involved.Therefore, no machine learning needed is it follow linear regression relationship. However, in second version keyboard, typing on a surface introduces challenges as surface properties can affect readings due to reflection. Additionally, individuals have varying palm-raising preferences, necessitating personalized adjustments to mitigate significant reading variations among users, the complexity might need more data for analysis. For the first version keyboard, without any blocking on the sensor, it is working fine in most of the scenario with a 80% accuracy, but it is hard to make an scientific experiment while typing on air. To see the keyboard, the experiment might need to interact with an actual VR keyboard.

Furthermore, the prototype's sensor positions are not always constant due to bending, leading to inconsistent readings and reduced voltage response resolution. This variability requires the model to be retrained before each use, which presents inconveniences.

Despite these challenges, wearable optical sensors still hold promise for virtual keyboards, especially when complemented with camera-based assistance. Cameras can aid in hand and finger movement detection, while the established DIP and PIP angle relationship serves as a valuable reference for hand position detection. However, the current virtual keyboard development is limited to capturing index finger selections, potentially hindering efficiency, as users typically employ multiple fingers per hand during typing. Overall, wearable optical devices offer improved functionality, but additional enhancements are necessary to optimize their usability and effectiveness.

7.Future work:

To further develop the palm-wearing device prototype, several improvements are necessary. Firstly, ensuring the fixed position of sensors is crucial, as it guarantees consistent readings. Additionally, incorporating at least two more sensors, dedicated to providing detailed information about the bending of the index and ring fingers, is essential. With only three sensors, validating the movements of these two fingers becomes challenging, particularly when distinguishing between bending on the middle finger or slight movements on the index finger.

In the future, enhancing camera labeling techniques by zooming in and focusing on a single finger can improve stability. Currently, finger obstruction by others and unstable angle readings pose challenges. Furthermore, integrating Leap Motion technology, which utilizes advanced posture sensing techniques involving two types of cameras, could enhance accuracy and reliability.

Moreover, to function as an actual virtual keyboard with VR headsets, feasibility testing with both devices is necessary. Achieving sensor fusion for hand gesture recognition while typing on a virtual keyboard will be crucial for seamless integration and optimal performance.

Reference:

[1]

L. Guo, Z. Lu and L. Yao, "Human-Machine Interaction Sensing Technology Based on Hand Gesture Recognition: A Review," in IEEE Transactions on Human-Machine Systems, vol. 51, no. 4, pp. 300-309, Aug. 2021, doi: 10.1109/THMS.2021.3086003.

[2]

M. Oudah, A. Al-Naji, and J. Chahl, "Hand Gesture Recognition Based on Computer Vision: A Review of Techniques," Journal of Imaging, vol. 6, no. 8, p. 73, Jul. 2020, doi: https://doi.org/10.3390/jimaging6080073.

[3]

M. Neff, "State of the Art in Hand and Finger Modeling and Animation," Computer Graphics Forum, Jan. 2015, Accessed: Mar. 26, 2024. [Online]. Available: https://www.academia.edu/102032252/State_of_the_Art_in_Hand_and_Finger_Modeling_and_Animation

[4]

M.-K. Liu, Y.-T. Lin, Z.-W. Qiu, C.-K. Kuo, and C.-K. Wu, "Hand Gesture Recognition by a MMG-Based Wearable Device," IEEE Sensors Journal, vol. 20, no. 24, pp. 14703–14712, Dec. 2020, doi: https://doi.org/10.1109/jsen.2020.3011825.

[5]

M. T. Tarata, "Mechanomyography versus Electromyography, in monitoring the muscular fatigue," BioMedical Engineering OnLine, vol. 2, no. 1, p. 3, 2003, doi: https://doi.org/10.1186/1475-925x-2-3.

[6]

S. S. Lin, N. M. Gamage, K. Herath, and A. Withana, "MyoSpring: 3D Printing Mechanomyographic Sensors for Subtle Finger Gesture Recognition," Sixteenth International Conference on Tangible, Embedded, and Embodied Interaction, Feb. 2022, doi: https://doi.org/10.1145/3490149.3501321.

[7]

S. Jiang et al., "Feasibility of Wrist-Worn, Real-Time Hand, and Surface Gesture Recognition via sEMG and IMU Sensing," in IEEE Transactions on Industrial Informatics, vol. 14, no. 8, pp. 3376-3385, Aug. 2018, doi: 10.1109/TII.2017.2779814., "S. Jiang et al., 'Feasibility of Wrist-Worn, Real-Time Hand, and Surface Gesture Recognition via sEMG and IMU Sensing,' in IEEE Transactions on Industrial Informatics, vol. 14, no. 8, pp. 3376-3385, Aug. 2018, doi: 10.1109/TII.2017.2779814."

[8]

J. Connolly, J. Condell, K. Curran, and P. Gardiner, "Improving Data Glove Accuracy and Usability Using a Neural Network When Measuring Finger Joint Range of Motion," Sensors, vol. 22, no. 6, p. 2228, Mar. 2022, doi: https://doi.org/10.3390/s22062228.

[9]

L. Dipietro, A. M. Sabatini and P. Dario, "A Survey of Glove-Based Systems and Their Applications," in IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 38, no. 4, pp. 461-482, July 2008, doi: 10.1109/TSMCC.2008.923862.,

[10]

Jong Man Kim, B. Koo, Y. Nam, and Y. Kim, “sEMG-Based Hand Posture Recognition Considering Electrode Shift, Feature Vectors, and Posture Groups,” Sensors, vol. 21, no. 22, pp. 7681–7681, Nov. 2021, doi: https://doi.org/10.3390/s21227681.

[11]

Bariya et al., “Glove-based sensors for multimodal monitoring of natural sweat,” Science Advances, vol. 6, no. 35, Aug. 2020, doi: https://doi.org/10.1126/sciadv.abb8308.

[12]

J. Fujikawa et al., “Diagnosis and Treatment of Tremor in Parkinson’s Disease Using Mechanical Devices,” Life, vol. 13, no. 1, p. 78, Dec. 2022, doi: https://doi.org/10.3390/life13010078.

[13]

N. Kostikis, D. Hristu-Varsakelis, M. Arnaoutoglou, and C. Kotsavasiloglou, “A Smartphone-Based Tool for Assessing Parkinsonian Hand Tremor,” IEEE Journal of Biomedical and Health Informatics, vol. 19, no. 6, pp. 1835–1842, Nov. 2015, doi: https://doi.org/10.1109/jbhi.2015.2471093.

[14]

T. Su et al., “3-D motion system (‘data-gloves’): application for Parkinson’s disease,” vol. 52, no. 3, pp. 662–674, Jun. 2003, doi: https://doi.org/10.1109/tim.2003.814702.

[15]

Y. Li et al., “Learning Hand Kinematics for Parkinson’s Disease Assessment Using a Multimodal Sensor Glove,” Advanced Science, vol. 10, no. 20, May 2023, doi: https://doi.org/10.1002/advs.202206982.

[16]

G. Güney et al., “Video-Based Hand Movement Analysis of Parkinson Patients before and after Medication Using High-Frame-Rate Videos and MediaPipe,” Sensors, vol. 22, no. 20, p. 7992, Oct. 2022, doi: https://doi.org/10.3390/s22207992.

[17]

S. Islam et al., “Using AI to measure Parkinson’s disease severity at home,” npj digital medicine, vol. 6, no. 1, Aug. 2023, doi: https://doi.org/10.1038/s41746-023-00905-9.

[18]

C. Ryan, Computer and internet use in the United States: 2016, Available: https://www.census.gov/history/pdf/comp-internetuse2016.pdf

[19]

Chung, H. Seo, S. Forbes, and H. Birkett, “Changing preferences and the future of work,” 2020. Available:https://www.birmingham.ac.uk/Documents/college-social-sciences/business/research/wirc/epp-working-from-home-COVID-19-lockdown.pdf

[20]

Y. Zhang, W. Yan and A. Narayanan, "A virtual keyboard implementation based on finger recognition," 2017 International Conference on Image and Vision Computing New Zealand (IVCNZ), Christchurch, New Zealand, 2017, pp. 1-6, doi: 10.1109/IVCNZ.2017.8402452.

[21]

C.Topal, B. Benligiray and C. Akinlar, "On the efficiency issues of virtual keyboard design," 2012 IEEE International Conference on Virtual Environments Human-Computer Interfaces and Measurement Systems (VECIMS) Proceedings, Tianjin, China, 2012, pp. 38-42, doi: 10.1109/VECIMS.2012.6273205.

[22]

Apple Inc., "Apple Vision Pro," Apple Support. Retrieved from https://support.apple.com/zh-cn/guide/apple-vision-pro/tana14220eef/visionos

[23]

A.B. Usakli and S. Gurkan, "Design of a Novel Efficient Human–Computer Interface: An Electrooculagram Based Virtual Keyboard," in IEEE Transactions on Instrumentation and Measurement, vol. 59, no. 8, pp. 2099-2108, Aug. 2010, doi: 10.1109/TIM.2009.2030923.

[24]

J. Dudley, H. Benko, D. Wigdor and P. O. Kristensson, "Performance Envelopes of Virtual Keyboard Text Input Strategies in Virtual Reality," 2019 IEEE International Symposium on Mixed and Augmented Reality (ISMAR), Beijing, China, 2019, pp. 289-300, doi: 10.1109/ISMAR.2019.00027.

[25]

S.A. Faleel, M. Gammon, K. Fan, D. -Y. Huang, W. Li and P. Irani, "HPUI: Hand Proximate User Interfaces for One-Handed Interactions on Head Mounted Displays," in IEEE Transactions on Visualization and Computer Graphics, vol. 27, no. 11, pp. 4215-4225, Nov. 2021, doi: 10.1109/TVCG.2021.3106493.

[26]

Liwei Chan, Yi-Ling Chen, Chi-Hao Hsieh, Rong-Hao Liang and Bing-Yu Chen, "2015. CyclopsRing: Enabling Whole-Hand and Context-Aware Interactions Through a Fisheye Ring", Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST '15), pp. 549-556.

[27]

Yamamoto, T. Matsumoto, T. Sudo, M. Miyashita, and T. Kondo, “Quantitative measurement of finger usage in stroke hemiplegia using ring-shaped wearable devices,” Journal of Neuroengineering and Rehabilitation, vol. 20, no. 1, Jun. 2023, doi: https://doi.org/10.1186/s12984-023-01199-4.

[28]

Sugiura, F. Nakamura, W. Kawai, T. Kikuchi and M. Sugimoto, "Behind the palm: Hand gesture recognition through measuring skin deformation on back of hand by using optical sensors," 2017 56th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE), Kanazawa, Japan, 2017, pp. 1082-1087, doi: 10.23919/SICE.2017.8105457.

[29]

Katsutoshi Masai, Yuta Sugiura, Masa Ogata, Kai Kunze, Masahiko Inami and Maki Sugimoto, "Facial Expression Recognition in Daily Life by Embedded Photo Reflective Sensors on Smart Eyewear", Proceedings of the 21st International Conference on Intelligent User Interfaces (IUI '16), pp. 317-326, 2016.

[30]

Kei Nakatsuma, Hiroyuki Shinoda, Yasutoshi Makino, Katsunari Sato and Takashi Maeno, "Touch interface on back of the hand" in SIGGRAPH 2011 Emerging Technologies (SIGGRAPH '11), New York, NY, USA, 2011.

[31]

Obodoeze Fidelis Chukwujekwu and Obiokafor Ifeyinwa Nkemdilim, “Technical Report: Comparative Analysis of Photodetectors for Appropriate Usage in Optical Communication Applications,” Zenodo (CERN European Organization for Nuclear Research), vol. 14, no. 2278–4861, Sep. 2021, doi: https://doi.org/10.5281/zenodo.5518079.

[32]

Zwieten, K. J., Schmidt, Klaus, Bex, Geert, Lippens, Peter, & Duyvendak, Wim. (2015). An Analytical Expression for the D.I.P. - P.I.P. Flexion Interdependence in Human Fingers. Acta of Bioengineering and Biomechanics / Wroclaw University of Technology, 17, 129-135. https://doi.org/10.5277/ABB-00078-2014-02.

[33]

Embark, A.S., Haggag, R.Y., & Saleh, S.A.F. (2022). "A Framework for Prediction Banking Risk Using Machine Learning Techniques." *Journal of Theoretical and Applied Information Technology*, 100(20), 6150. ISSN: 1992-8645. E-ISSN: 1817-3195.

[34]

W.Zhang, X. Zhao and Z. Li, "A Comprehensive Study of Smartphone-Based Indoor Activity Recognition via Xgboost," in IEEE Access, vol. 7, pp. 80027-80042, 2019, doi: 10.1109/ACCESS.2019.2922974.

[35]

Chen and C. Guestrin, “XGBoost: a Scalable Tree Boosting System,” Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD ’16, pp. 785–794, 2016, doi: https://doi.org/10.1145/2939672.2939785.

[36]

Wu, C., Wang, K., Cao, Q., Fei, F., Yang, D., Lu, X., Xu, B., Zeng, H., & Song, A. (2021, June 30). Development of a Low-Cost Wearable Data Glove for Capturing Finger Joint Angles. *Micromachines (Basel)*, 12(7), 771. doi: 10.3390/mi12070771.

[37]

Connolly, J., Condell, J., Curran, K., & Gardiner, P. (2022). Improving Data Glove Accuracy and Usability Using a Neural Network When Measuring Finger Joint Range of Motion. Sensors, 22(6), 2228. https://doi.org/10.3390/s22062228

[38]

Jiang, L., Xia, H., & Guo, C. (2019). A Model-Based System for Real-Time Articulated Hand Tracking Using a Simple Data Glove and a Depth Camera. Sensors, 19(21), 4680. https://doi.org/10.3390/s19214680

[39]

J. F. Spindel, S. Pokrywa, N. Elder, and C. Smith, “The Environment Has Effects on Infrared Temperature Screening for COVID-19 Infection,” American Journal of Infection Control, Aug. 2021, doi: https://doi.org/10.1016/j.ajic.2021.08.002.

[40]

Church, J.S., Hegadoren, P.R., Paetkau, M.J., Miller, C.C., Regev-Shoshani, G., Schaefer, A.L., Schwartzkopf-Genswein, K.S. (2014). Influence of environmental factors on infrared eye temperature measurements in cattle. Research in Veterinary Science, 96(1), 220-226. ISSN 0034-5288. doi: 10.1016/j.rvsc.2013.11.006.

[41]

A. I. R. Maas, Q. V. Le, T. M. O'Neil, Oriol Vinyals, P. Nguyen, and A. Y. Ng, "Recurrent neural networks for noise reduction in robust ASR," Sep. 2012, doi: https://doi.org/10.21437/interspeech.2012-6.

[42]

Sun, W., Li, F. M., Huang, C., Lei, Z., Steeper, B., Tao, S., Tian, F., & Zhang, C. (2021). ThumbTrak: Recognizing Micro-finger Poses Using a Ring with Proximity Sensing. In *Proceedings of the 23rd International Conference on Mobile Human-Computer Interaction (MobileHCI '21)* (pp. 1–9, Article 2). Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3447526.3472060

[43]

Malakhov, A. (2016). "Composable Multi-Threading for Python Libraries." In Proceedings of the 15th Python in Science Conference (SciPy 2016), pp. 15-19. DOI: 10.25080/Majora-629e541a-002.

Appendices

Appendices A (Parallel recording)

```python
#libraries need to be installed in adavance
#pip install mediapipe opencv-python numpy pyserial
import mediapipe as mp
import cv2
import numpy as np
import serial
import time
import threading
from datetime import datetime

mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands
data_to_write = ""
recording = False  # Initialize recording as False
camera_data = []  # List to store camera data
hand_data = []    # List to store hand data
# Function to draw finger angles and return angle information
def draw_finger_angles(image, results, joint_list):
    angle_info = []  # List to store angle information
    for hand in results.multi_hand_landmarks:
        for joint in joint_list:
            a = np.array([hand.landmark[joint[0]].x, hand.landmark[joint[0]].y, hand.landmark[joint[0]].z])
            b = np.array([hand.landmark[joint[1]].x, hand.landmark[joint[1]].y, hand.landmark[joint[1]].z])
            c = np.array([hand.landmark[joint[2]].x, hand.landmark[joint[2]].y, hand.landmark[joint[2]].z])
            cross_product = np.cross(a - b, c - b)
            dot_product = np.dot(a - b, c - b)
            radians = np.arctan2(np.linalg.norm(cross_product), dot_product)
            angle = np.abs(radians * 180.0 / np.pi)
            if angle > 180.0:
                angle = 360 - angle
            cv2.putText(image, str(round(angle, 2)),
                        tuple(np.multiply([b[0], b[1]], [640, 480]).astype(int)),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE_AA)
            angle_info.append((f"Joint {joint[1]}", round(angle, 2)))
            angle = round(angle, 2)
    return image, angle
# Function to toggle recording
```

```
def toggle_recording():

    global recording

    recording = not recording

    if recording:

        print("Recording started")

    else:

        print("Recording stopped")

# Function to read serial data

def read_serial_data(output_file):

    with serial.Serial('COM5', 115200, timeout=1) as ser:

        print("Serial port opened")

        while True:

            if ser.in_waiting:

                data = ser.readline().decode('utf-8').strip()

                #print("Received data:", data)  # Debugging print statement

                data = data[0:-1]

                parts = data.split(',')

                if all(part for part in parts if part):

                    current_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

                    output_file.write(f"{data},{data_to_write}\n")

                    #print("Data written to file:", f"{current_time},{data},{data_to_write}")  # Debugging print
statement

# Open camera

cap = cv2.VideoCapture(0)

# Define joint_list

joint_list_index1 = [[8, 7, 6]]

joint_list_index2 = [[7, 6, 5]]

joint_list_middle1 = [[12, 11, 10]]

joint_list_middle2 = [[11, 10, 9]]

joint_list_ring1 = [[16, 15, 14]]

joint_list_ring2 = [[15, 14, 13]]

joint_list_pinky1 = [[20, 19, 18]]

joint_list_pinky2 = [[19, 18, 17]]
```

```
# Create file for saving serial data

with open('serial_data.txt', 'w') as output_file:

    # Create and start thread for reading serial data

    read_thread = threading.Thread(target=read_serial_data, args=(output_file,))

    read_thread.daemon = True  # Set as daemon thread so it exits when the main program exits

    read_thread.start()
```

```
# Initialize camera and hand tracking
with mp_hands.Hands(min_detection_confidence=0.8, min_tracking_confidence=0.5) as hands:
    while cap.isOpened():
        ret, frame = cap.read()
        # BGR 2 RGB
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        # Flip on horizontal
        image = cv2.flip(image, 1)
        # Set flag
        image.flags.writeable = False
        # Detections
        results = hands.process(image)
        # Set flag to true
        image.flags.writeable = True
        # RGB 2 BGR
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
        # Rendering results
        if results.multi_hand_landmarks:
            for num, hand in enumerate(results.multi_hand_landmarks):
                mp_drawing.draw_landmarks(image, hand, mp_hands.HAND_CONNECTIONS,
                                        mp_drawing.DrawingSpec(color=(121, 22, 76), thickness=2, circle_radius=4),
                                        mp_drawing.DrawingSpec(color=(250, 44, 250), thickness=2, circle_radius=2),
                                        )
                # Render left or right detection
                if results.multi_handedness:
                    hand_label = results.multi_handedness[num].classification[0].label
                    if hand_label == 'Left':
                        image, info1 = draw_finger_angles(image, results, joint_list_index1)
                        image, info2 = draw_finger_angles(image, results, joint_list_index2)
                        image, info4 = draw_finger_angles(image, results, joint_list_middle1)
                        image, info5 = draw_finger_angles(image, results, joint_list_middle2)
                        image, info6 = draw_finger_angles(image, results, joint_list_ring1)
                        image, info7 = draw_finger_angles(image, results, joint_list_ring2)
                        image, info8 = draw_finger_angles(image, results, joint_list_pinky1)
                        image, info9 = draw_finger_angles(image, results, joint_list_pinky2)
                        data_to_write = f"Left,{info1},{info2},{info4},{info5},{info6},{info7},{info8},{info9}"
                        hand_data.append(data_to_write)
                    elif hand_label == 'Right':
                        image, info1 = draw_finger_angles(image, results, joint_list_index1)
                        image, info2 = draw_finger_angles(image, results, joint_list_index2)
```

```
                            image, info4 = draw_finger_angles(image, results, joint_list_middle1)

                            image, info5 = draw_finger_angles(image, results, joint_list_middle2)

                            image, info6 = draw_finger_angles(image, results, joint_list_ring1)

                            image, info7 = draw_finger_angles(image, results, joint_list_ring2)

                            image, info8 = draw_finger_angles(image, results, joint_list_pinky1)

                            image, info9 = draw_finger_angles(image, results, joint_list_pinky2)

                            data_to_write = f"Right,{info1},{info2},{info4},{info5},{info6},{info7},{info8},{info9}"

                            hand_data.append(data_to_write)

            # Toggle recording

            if recording:

                line = ""

                #timestamp_ms = int(time.time() * 1000)

                #timestamp_readable = time.strftime("%H:%M:%S", time.localtime(timestamp_ms / 1000))

                camera_data.append((line))

            cv2.imshow('Hand Tracking', image)

            key = cv2.waitKey(10)

            if key == ord('q'):

                break

            elif key == ord(' '):  # Toggle recording when spacebar is pressed

                toggle_recording()

# Release the camera

cap.release()

cv2.destroyAllWindows()

# Writing hand data to list

hand_file_path = 'hand_data.txt'

with open(hand_file_path, 'w') as hand_file:

    for data in hand_data:

        hand_file.write(f"{data}\n")
```

Appendices B (Machine learning model)

```python
from sklearn.model_selection import train_test_split

import numpy as np

import xgboost as xgb

import joblib


def scale_column(column, min_val, max_val):

    min_col = np.min(column)

    max_col = np.max(column)

    scaled_column = min_val + ((column - min_col) * (max_val - min_val)) / (max_col - min_col)

    return scaled_column

# Load the data skipping the sixth column

data = np.loadtxt('train.txt', delimiter=',', usecols=[0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16])

# Extract the input features (first eight columns)

X = data[:, :8]

# Extract the output targets

y = data[:, -8:]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a single XGBoost model for all output values

model = xgb.XGBRegressor(learning_rate=0.42, max_depth=8, alpha=30, n_estimators=100)

model.fit(X_train, y_train)

# Save the trained model

joblib.dump(model, 'xgboost_model.pkl')

# Predict using the trained model

y_pred = model.predict(X_test)

# Print the predictions and actual values

for i in range(len(y_pred)):

    print(f"Case {i+1}:")

    print(f"Predicted values: {y_pred[i]}")

    print(f"Actual values: {y_test[i]}")

    print()

# Calculate absolute errors for each output

absolute_errors = np.abs(y_pred - y_test)

# Calculate average absolute errors for each output

average_absolute_errors = np.mean(absolute_errors, axis=0)

# Print average absolute errors for each output

for i, error in enumerate(average_absolute_errors):

    print(f"Average Absolute Error for Output {i+1}: {error:.4f}")

# Calculate accuracies for each output
```

```
accuracies = np.sum(absolute_errors < 15, axis=0) / len(absolute_errors)

# Print accuracies for each output

for i, accuracy in enumerate(accuracies):

    print(f"Accuracy for Output {i+1}: {accuracy:.4f}")
```

Appendices C (Plotting for validation)

```
from sklearn.model_selection import train_test_split

import numpy as np

import xgboost as xgb

import matplotlib.pyplot as plt


# Load the training data

train_data = np.loadtxt('4finger.txt', delimiter=',', usecols=[0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16])

# Extract the input features (first eight columns)

X_train = train_data[:, :8]

# Extract the output targets

y_train = train_data[:, -8:]

# Split the training data into training and validation sets

X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

# Load the testing data from another file

test_data = np.loadtxt('4finger1.txt', delimiter=',', usecols=[0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16])

# Extract the input features for testing

X_test = test_data[:, :8]

# Extract the output targets for testing

y_test = test_data[:, -8:]

# List to store trained models

models = []

# Train XGBoost models for each output value

for i in range(y_train.shape[1]):

    model = xgb.XGBRegressor(learning_rate=0.41, max_depth=8, alpha=30, n_estimators=100)

    model.fit(X_train, y_train[:, i])

    models.append(model)

# Predict using the trained models on the testing data

y_pred = np.array([model.predict(X_test) for model in models]).T

# Calculate absolute errors for each output

absolute_errors = np.abs(y_pred - y_test)

# Calculate average absolute errors for each output

average_absolute_errors = np.mean(absolute_errors, axis=0)

# Print average absolute errors for each output

for i, error in enumerate(average_absolute_errors):

    print(f"Average Absolute Error for Output {i+1}: {error:.4f}")

# Calculate accuracies for each output

accuracies = np.sum(absolute_errors < 15, axis=0) / len(absolute_errors)

# Print accuracies for each output
```

```python
for i, accuracy in enumerate(accuracies):

    print(f"Accuracy for Output {i+1}: {accuracy:.4f}")

plt.figure(figsize=(12, 8))

for i in range(8):

    plt.subplot(3, 3, i + 1)

    plt.plot(y_test[:, i], label='Actual')

    plt.plot(y_pred[:, i], label='Classifed')

    plt.xlabel('Time(20ms)')

    plt.ylabel('Angle(degree)')

    if i==0:

        plt.title(f'DIP Index')

    elif i==1:

        plt.title(f'PIP Index')

    elif i==2:

        plt.title(f'DIP Middle')

    elif i==3:

        plt.title(f'PIP Middle')

    elif i==4:

        plt.title(f'DIP Ring')

    elif i==5:

        plt.title(f'PIP Ring')

    elif i==6:

        plt.title(f'DIP Little')

    elif i==7:

        plt.title(f'PIP Little')

    plt.grid()

    plt.legend()

plt.tight_layout()

plt.show()
```

Appendices D (First version keyboard)

```python
import numpy as np

import serial

import joblib

import tkinter as tk


# Load the pretrained XGBoost model

model = joblib.load('xgboost_model.pkl')  # Adjust the filename as needed

# Connect to the serial port

ser = serial.Serial('COM4', 115200)  # Adjust the port and baud rate as needed

# Define the number of features per line

num_features = 8

# Create a Tkinter window

root = tk.Tk()

root.title("Real-time Recognition")

root.geometry("400x400")  # Set the initial size of the window

# Create label widgets to display the predicted labels

predicted_label_vars = [tk.StringVar() for _ in range(4)]

predicted_label_labels = [tk.Label(root, textvariable=var, font=("Arial", 16)) for var in predicted_label_vars]

for label in predicted_label_labels:

    label.pack()  # Add labels to the window

# Create a canvas widget for drawing the grid

canvas = tk.Canvas(root, width=400, height=400, bg="white")

canvas.pack()

grid_content = [

    ["", "A", "B", "C"],

    ["", "D", "E", "F"],

    ["", "G", "H", "I"],

    ["", "J", "K", "L"]

]

# Function to draw the grid

def draw_grid():

    # Clear previous drawings

    canvas.delete("grid")

    canvas.delete("labels")

    # Define grid parameters

    rows = 4

    columns = 4

    cell_width = 100
    cell_height = 100

    # Draw grid lines

    for i in range(columns + 1):

        x = i * cell_width

        canvas.create_line(x, 0, x, rows * cell_height, fill="black", tags="grid")

    for i in range(rows + 1):

        y = i * cell_height

        canvas.create_line(0, y, columns * cell_width, y, fill="black", tags="grid")

    # Add tags to cells for easier identification and custom labels

    for row in range(rows):

        for col in range(columns):

            canvas.create_rectangle(col * cell_width, row * cell_height,

                                    (col + 1) * cell_width, (row + 1) * cell_height,

                                    outline="black", fill="white", tags=("grid", f"cell_{row}_{col}"))

            canvas.create_text(col * cell_width + cell_width/2, row * cell_height + cell_height/2,

                               text=grid_content[row][col], font=("Arial", 16), tags="labels")

# Read lines from the serial port for real-time recognition

def read_serial():

    # Check if there is data available to read

    if ser.in_waiting > 0:

        # Read a line of data from the serial port

        line = ser.readline().strip()

        try:

            # Extract features from the line

            data = np.fromstring(line.decode(), dtype=float, sep=',')

            file_features = data[:num_features]

            # Perform prediction using the model

            predictions = model.predict([file_features])[0]  # Accessing Prediction 1

            # Loop through the predictions array and print each odd-numbered prediction

            for i in range(1, len(predictions), 2):

                # Update the predicted label on the screen

                if i==1:

                    predicted_label_vars[i // 2].set("Index Bending angle " + ": " + str(predictions[i]))

                elif i ==3:

                    predicted_label_vars[i // 2].set("Middle Bending angle " + ": " + str(predictions[i]))

                elif i ==5:

                    predicted_label_vars[i // 2].set("Ring Bending angle " + ": " + str(predictions[i]))

                # Change color of specific cell based on conditions

                elif i ==7:

                    predicted_label_vars[i // 2].set("little Bending angle " + ": " + str(predictions[i]))
                if i == 1:
                    canvas.itemconfig(f"cell_0_0", fill="White")
                    canvas.itemconfig(f"cell_0_1", fill="White")
                    canvas.itemconfig(f"cell_0_2", fill="White")
                    canvas.itemconfig(f"cell_0_3", fill="White")
                    if predictions[i] >= 160:
                        canvas.itemconfig(f"cell_0_0", fill="light blue")  # First row, first column
                    elif 120<predictions[i] <= 160:
                        canvas.itemconfig(f"cell_0_1", fill="light blue")  # First row, second column
                    elif 100<predictions[i] <= 130:
                        canvas.itemconfig(f"cell_0_2", fill="light blue")  # First row, third column
                    elif predictions[i] <= 100:
                        canvas.itemconfig(f"cell_0_3", fill="light blue")  # First row, fourth column
                elif i == 3:
                    canvas.itemconfig(f"cell_1_0", fill="White")
                    canvas.itemconfig(f"cell_1_1", fill="White")
                    canvas.itemconfig(f"cell_1_2", fill="White")
                    canvas.itemconfig(f"cell_1_3", fill="White")
                    if predictions[i] >= 160:
                        canvas.itemconfig(f"cell_1_0", fill="light blue")  # Second row, first column
                    elif 120<predictions[i] <= 160:
                        canvas.itemconfig(f"cell_1_1", fill="light blue")  # Second row, second column
                    elif 100<predictions[i] <= 130:
                        canvas.itemconfig(f"cell_1_2", fill="light blue")  # Second row, third column
                    elif predictions[i] <= 100:
                        canvas.itemconfig(f"cell_1_3", fill="light blue")  # Second row, fourth column
                elif i == 5:
                    canvas.itemconfig(f"cell_2_0", fill="White")
                    canvas.itemconfig(f"cell_2_1", fill="White")
                    canvas.itemconfig(f"cell_2_2", fill="White")
                    canvas.itemconfig(f"cell_2_3", fill="White")
                    if predictions[i] >= 160:
                        canvas.itemconfig(f"cell_2_0", fill="light blue")  # Third row, first column
                    elif 120<predictions[i] <= 160:
                        canvas.itemconfig(f"cell_2_1", fill="light blue")  # Third row, second column
                    elif 100<predictions[i] <= 130:
                        canvas.itemconfig(f"cell_2_2", fill="light blue")  # Third row, third column
                    elif predictions[i] <= 100:
                        canvas.itemconfig(f"cell_2_3", fill="light blue")  # Third row, fourth column
                elif i == 7:
```

```python
                    canvas.itemconfig(f"cell_3_0", fill="White")

                    canvas.itemconfig(f"cell_3_1", fill="White")

                    canvas.itemconfig(f"cell_3_2", fill="White")

                    canvas.itemconfig(f"cell_3_3", fill="White")

                    if predictions[i] >= 160:

                        canvas.itemconfig(f"cell_3_0", fill="light blue")  # Fourth row, first column

                    elif 120<predictions[i] <= 160:

                        canvas.itemconfig(f"cell_3_1", fill="light blue")  # Fourth row, second column

                    elif 100<predictions[i] <= 130:

                        canvas.itemconfig(f"cell_3_2", fill="light blue")  # Fourth row, third column

                    elif predictions[i] <= 100:

                        canvas.itemconfig(f"cell_3_3", fill="light blue")  # Fourth row, fourth column

        except ValueError as e:

            # Skip lines with incorrect format

            pass

    # Schedule the function to run again after a delay (adjust the delay as needed)

    root.after(100, read_serial)

# Start reading from the serial port

read_serial()

# Draw the grid

draw_grid()

# Start the Tkinter event loop

root.mainloop()

# Close the serial port

ser.close()
```

Appendices E (Second version keyboard)

```python
import numpy as np

import serial

import joblib

import time


# Load the pretrained model

model = joblib.load('xgboost_model.pkl')  # Replace 'your_model.pkl' with your model file

# Connect to the serial port

ser = serial.Serial('COM4', 115200)  # Adjust the port and baud rate as needed

# Read lines from the serial port for real-time recognition

def read_serial():

    start_time = time.time()

    while time.time() - start_time < 20:  # Run for 10 seconds

        # Check if there is data available to read

        if ser.in_waiting > 0:

            # Read a line of data from the serial port

            line = ser.readline().strip()

            try:

                # Preprocess the data (if needed)

                # Example: data = preprocess_data(line)

                # Convert the data to a NumPy array

                data = np.fromstring(line.decode(), dtype=float, sep=',')

                # Perform prediction using the model

                prediction = model.predict([data])  # Make prediction

                # Print the predicted result

                print("Predicted Result:", prediction)

            except ValueError as e:

                # Skip lines with incorrect format

                pass

        else:

            # Wait for a short time before checking again

            time.sleep(0.01)

# Start reading from the serial port

read_serial()

# Close the serial port (optional)

ser.close()
```