Electronics and Computer Science
Faculty of Engineering and Physical Sciences
University of Southampton

Zaib Uddin
29/4/2025

Face and hand gesture recognition AI for embedded systems

Project supervisor: Professor David Thomas
Second examiner: Dr Nema Abdelazim

A project report submitted for the award of
Electronic Engineering with Computer Systems (MEng)

Abstract

AI (artificial intelligence) has seen a sudden and sustained bout of interest as of late, spurred on by the great success of Large Language Models (LLMs) such as ChatGPT when it was launched in 2022. However, despite the advances that have come from refining these models for applications such as creative writing and research, their vaunted progress, publicity, and potential for profit have limited the scope of AI applications. In this age of development, it is important to focus on smaller, closer-to-human applications that assist in everyday matters such as communication and interaction.

My project involves evaluating current facial and hand recognition programs, the development of an application that can recognise the faces of registered individuals and the hand gestures they make on a Raspberry Pi 4 Model B using these resources, and testing what real users think of the technology.

I will detail the research, development, and testing phases that lead me to believe that my project was a success, properly demonstrating the technologies' capabilities for HCI (human-computer interaction) and highlighting how it can be improved with respect to quantitative metrics and user feedback.

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

I have acknowledged all sources, and identified any content taken from elsewhere.
I have used open-source code produced by others.
I did all the work myself, or with my allocated group, and have not helped anyone else.
The material in the report is genuine, and I have included all my data/-code/designs.
I have not submitted any part of this work for another assessment.
My work involved human participants; ERGO/FEPS/101427.

ECS Statement of Originality Template, updated August 2018, Alex Weddell aiofficer@ecs.soton.ac.uk

Acknowledgements

I would like to thank my supervisor, Professor David Thomas, and my second examiner, Dr Nema Abdelazim, for their constant support. They went beyond their duties to aid me in every step of my project, I am truly grateful for everything that they have done and I hope to have such guidance in the future as well.

Contents

1 Introduction

1.1 Purpose

In the span of a few decades, AI has went from a field with mainly theoretical applications to an inescapable topic that permeates virtually every facet of the modern world. While AI and its nearly limitless applications have undoubtedly progressed rapidly as the entire field continues to grow and find relevancy, a disproportionate amount of this attention has gone to LLMs.

While undoubtedly useful with a myriad of applications, it can be easily seen that hyper-focusing on fine-tuning chatbots for giant corporations due to economic incentives take away resources, time, and talent from other applications that greatly benefit the general populace [1]. One such area of use is the use of AI for HCI (Human-Computer Interaction) on embedded systems. Rather than constantly upscaling resource-hungry data centres that would increasingly strain finances and deplete water, smaller energy-efficient edge systems [2] that display a wide use of specialised applications from assisting the blind [3] to exploiting embedded vision in Cyber-Physical Systems (CPS) [4]. This project will demonstrate how existing open-source software and affordable embedded devices can be used to create a reasonably advanced system that uniquely responds to each registered user while evaluating the overall capabilities and limitations of the technology used.

1.2 Aim

The primary goal of this project is to create and evaluate the performance of an HCI program meant to run on a Raspberry Pi. The program will be made of two combined scripts based of existing open-source projects, one that recognises the faces of users (displaying a box around their faces recognised Region of Interest (ROI) and being able to declare when it spots the faces of unregistered users) and another that responds to specific hand gestures made by users with unique responses for each one. These scripts would be combined to create a program capable of identifying whether the face of a user was a registered one or not, naming the user by their username (if they were registered, otherwise a default username for unknown users would be shown), displaying a box around the recognised ROI of the face, and giving custom responses for each gesture made by the user comprising of an acknowledgment of the type of gesture made and their username.

The greater details and extended goals of this project are as follows:

- Individually running and evaluating both face and gesture recognition programs.
- Combining both programs into a single script.
- Testing the script myself to evaluate against specific metrics.
- Having a group of individuals register themselves as users.
- Having each registered user test the system individually.

- As an advanced goal, configure the program to be able to respond to 2 users at once.

- As an advanced goal, having registered users test the system as pairs to see whether it can detect which gesture belongs to which user.

The main goal of this project was the successful creation and testing of the combined face and gesture recognition script, with plans of having separate individuals test the system at once as an additional milestone to evaluate its further capabilities. Below in Figure 1 is a block diagram depicting how the application would function for a single user.

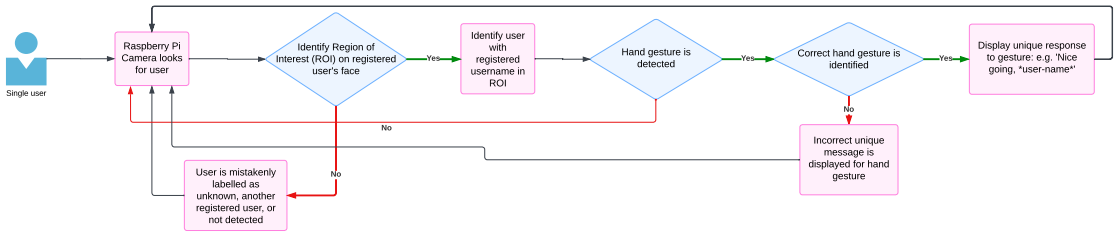


Figure 1: Block diagram of the combined application operating for a single user.

1.3 Scope

The application would be run on a Raspberry Pi 4 Model B with 4 GB of RAM and a 32 GB SD card. Additional accessories required are:

- A Raspberry Pi Camera Module 3, connected through the CSI (Camera Serial Interface) port, which will capture images to train the facial recognition task and videos for the main application.

- A pi-FAN to prevent overheating.

- A Raspberry Pi keyboard and mouse to directly control the device.

- An interchangeable monitor connected to the Raspberry Pi through an HDMI to Micro HDMI cable.

- Blu tack (a reusable adhesive material) to position the camera by attaching it to the monitor.

The following is an image of the Raspberry Pi running with the mentioned peripherals.

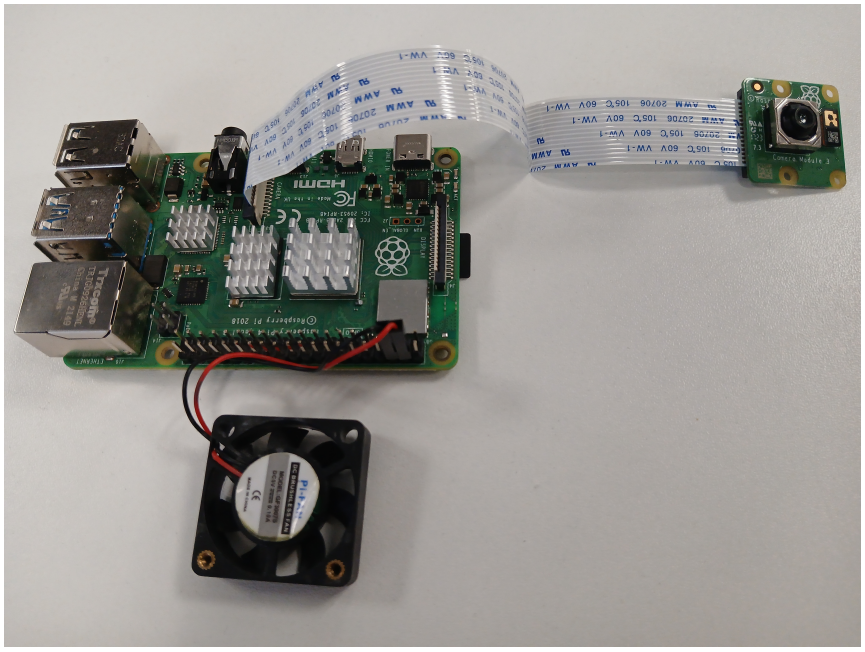


Figure 2: Raspberry Pi 4 Model B with the camera module and pi-FAN.

The application was based on online projects, those of facial [5] and hand gesture recognition [6], and open-source libraries such as OpenCV [7] and MediaPipe [8]. This project not only involves an effort to reasonably optimise the program for the hardware it is run on and analysing its quantitative performance metrics, but also the ethical enlistment of students and/or staff from the University of Southampton who may act as volunteers who can evaluate the system on their own and share insight on the experience it offers.

2 Background

After first studying the uneven influences of corporate investment in AI [1], I further looked into how AI could be run on smaller energy-efficient edge computing devices [2]. Looking further into how embedded vision platforms could be used for a wide array of applications, from spatial AI-enabled CPS [4] to Text-to-Speech converters for the blind [3], I analysed how hand tracking and gesture recognition could be used to allow for naturalistic use of software [9]. Another significant part of my research included evaluating the actual capabilities of the Raspberry Pi 4 Model B and observing how algorithms such as CNNs (Convolutional Neural Networks) could be run on it [10].

After deciding to develop an HCI system, I decided to see how other students of the University of Southampton had attempted to do the same. After reading how a system could be designed to use facial recognition for authentication purposes [11] and how embedded systems could be used for assistive purposes like using gesture recognition to translate sign language into text and speech [12], I gained a foundation on the purposes, development, and capabilities of such technology. I then researched similar systems that would give me ideas on how to structure and implement my own. This included an article on the effectiveness of CNNs in computer vision and the strength of a specific model in recognising Indian sign language and hand gestures [13]. Furthermore, I analysed other applications of CNNs, particularly how they were used in an assistive communication system that categorised sign language hand gestures into text and classified facial expressions with types of emotions so that a mute person may use it to express their feelings and communicate using speech [14].

I then studied articles that detailed how to construct such a system with my own Raspberry Pi. I heavily referred to specific online guides that detailed the equipment, code, and procedures required in order to create separate hand gesture [6] and user face recognition [15] systems. In order to successfully reproduce both experiments, I referenced relevant sources on the hardware required such as the camera module, its technical capabilities, how to operate it, and possible substitutes or newer, better versions [16]. I also regularly visited the Python Software Foundation [17] to better understand the rules and available libraries of the programming language when optimising the performance of each script and trying to combine them into an efficient single program.

I relied on literature [18] when first familiarising myself with the intricacies of the Raspberry Pi 4 Model B and its Operating System (OS), the Raspberry Pi OS. This allowed me to better configure the device for my particular computer vision applications, which was particularly useful in maintaining the OS and installing required libraries for programs such as the OpenCV library itself [7]. This was particularly useful when I discovered that a newer version of the facial recognition guide I originally used was released [5], containing updated code and detailing safer methods of constructing the recognition program in a virtual environment (VE) that I would later apply to the gesture recognition and final combined program.

Constant investigation of material related to the study was conducted at every phase. After finding that the code of the gesture recognition guide had poor performance, I sought out code that better used the MediaPipe library. I found that the MediaPipe gesture recogniser task developed by Google itself [8] to be a superior version that would allow specific gestures to be recognised from a model trained of countless images of these gestures, taking less processing power and thus working better with the facial recognition program. Even after testing how the developed HCI system operated for volunteers, other articles were useful in advising how the completed program could be tested through detection rates and Frames Per Second (FPS) [19]. The final assessment of the capabilities of the system and what could be done to improve it, such as using AI accelerators available in other embedded hardware platforms like the Jetson Xavier NX, also has relevant experimentation that supports it [20].

3 System design

3.1 Initial design

I first planned to design the base of the facial and hand gesture recognition system, which can be seen in Figure 3 below, as the Raspberry Pi 4 Model B connected to a Raspberry Pi mouse and keyboard pair, a pi FAN, and a Camera Module 3 attached to the screen of a connected display monitor through Blu tack.

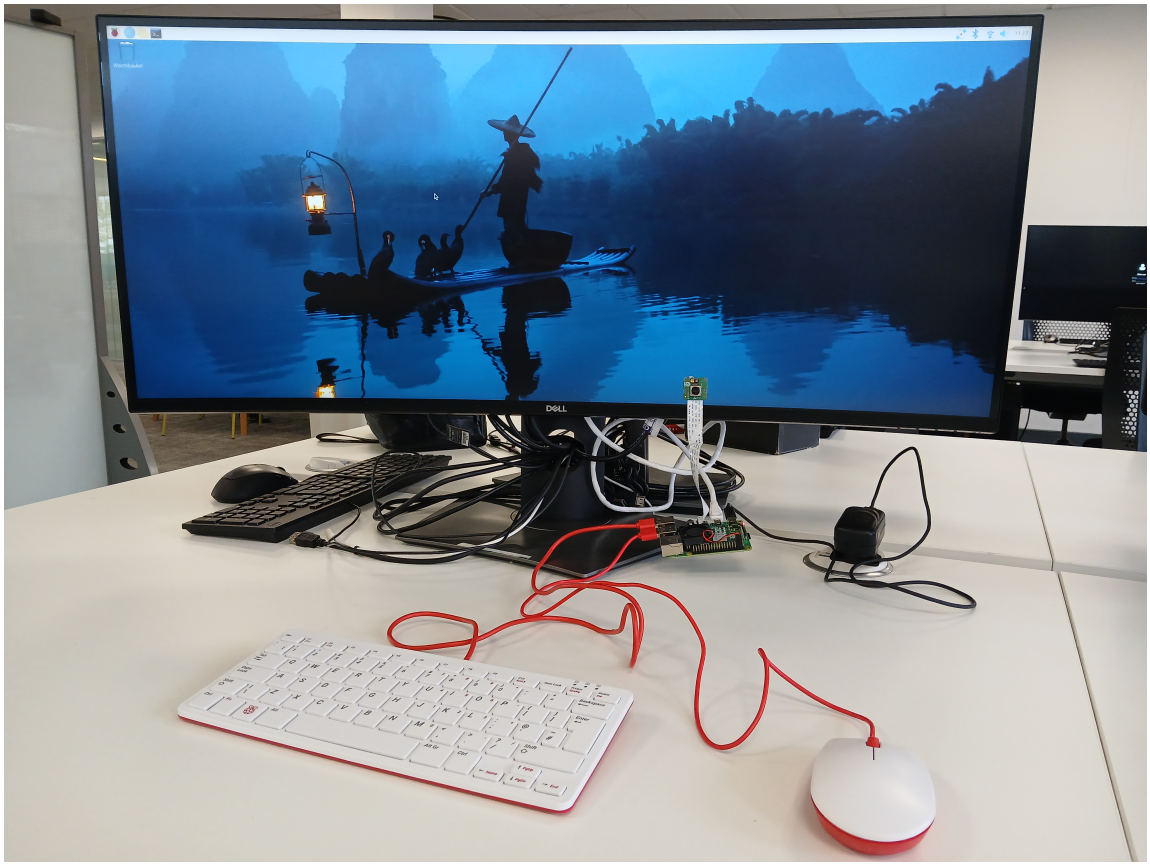


Figure 3: Physical setup of the combined recognition system.

The main program would be a single Python file, its final form is Listing 1 in Appendix B, that would only require a single click of a mouse to begin running within an integrated development environment (IDE) such as Geany or Thonny. The application would then take a few seconds to load before running the recognition task. The application would then continue to run, identifying the face of an individual who would be positioned in front of the camera and determining whether they are a registered user. The ROI of their identified face would have a blue box around it with their registered username (or a default "Unknown" name for unregistered users) above the box, simultaneously scanning for any hands in the captured camera frame and trying to associate the shape of any recognised hands with a list of specific hand gestures based on the joints, fingers, and positions of the hand. The name of the recognised gesture would then be displayed next to the hand in a message that includes the username of the user, e.g. displaying "Rock on, Zaib!" on the area of the screen above where I make a closed fist gesture. The average FPS would then be displayed at the edge of the screen,

continually being updated as the user continues to see whether their face and the gestures of up to two of their hands could be tracked and identified to give unique user response messages for each gesture until the button ”q” is pressed on the keyboard which ends the program's execution.

In order to register themselves, users would first have to train the facial recognition model that the program uses. This would require the user to submit about 9 slightly varying photos of their face, the photos in this case being captured through a Python program ”image_capture.py” and the camera module, which will then be stored in a folder sharing their selected username within a larger ”dataset” folder that stores all separate user folders. These folders would then be accessed in the ”model_training.py” script, which spends a few seconds training the model with user pictures and eventually generates the new recognition model to be used in the ”encodings.pickle” file. While the original facial recognition guide [5] emphasises that only 1 close-up photo of the user in decent lighting looking into the camera will generally be enough for basic recognition, one such image that I used to train the model being shared below in Figure 4, I used 9 pictures of my face at different angles and with varying expressions to ensure that the application would be able to recognise my face better as I moved and changed expressions. An example of a photo with a unique facial expression and angle used to train the model is shared in Figure 5.

Figure 4: Simple photo to train facial recognition model for basic recognition.

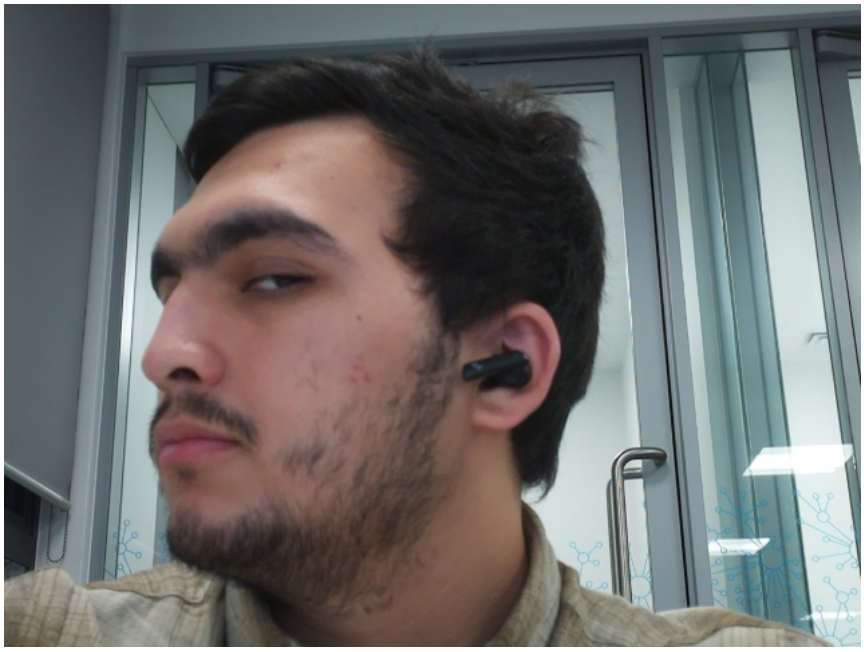


Figure 5: Photo of a serious face at a side angle to train a facial recognition model.

3.2 Altered features

As I developed the program, I decided to implement changes to its design. After testing the base gesture recognition script and attempting to configure it to recognise specific gestures, it was found to be too slow and imprecise at doing so. This caused me to instead choose the MediaPipe Gesture Recognizer task [8] as it not only ran smoother, displaying a consistently higher frame rate, but came with a set of canned/pre-trained gestures (”None” for when no gestures are recognised, ”Closed_Fist”, ”Open_Palm”, ”Pointing_Up”, ”Thumb_Down”, ”Thumb_Up”, ”Victory”, and ”ILoveYou”) and was easier to modify to work for multiple users (more than two hands in a single frame).

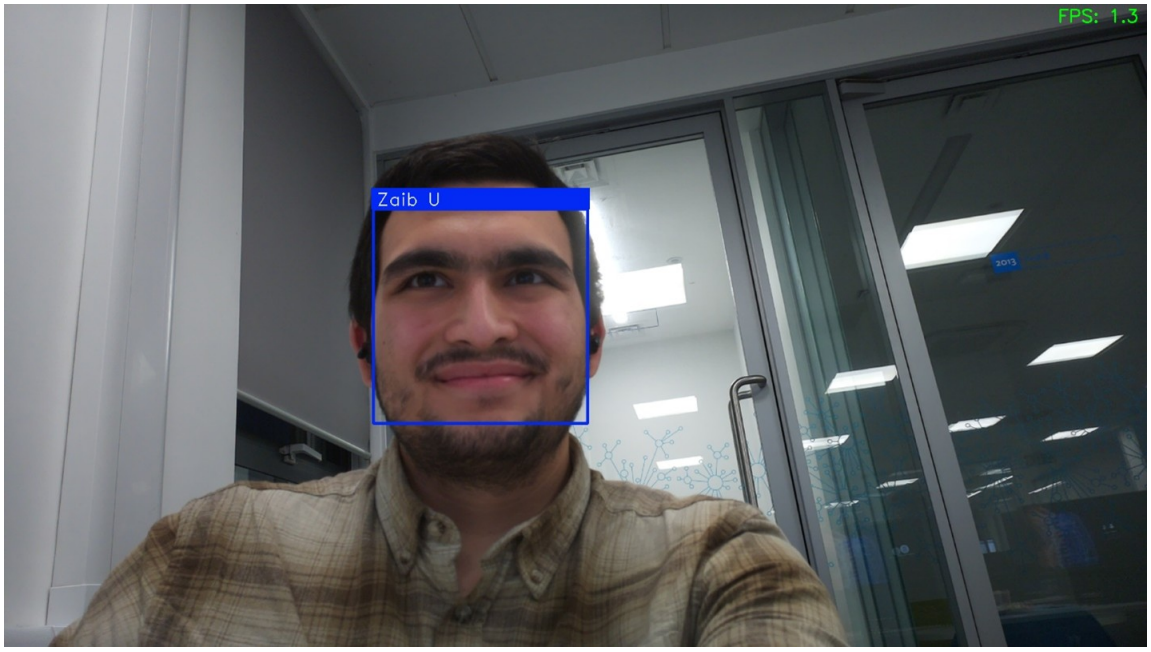


Figure 6: Original face recognition script being run.

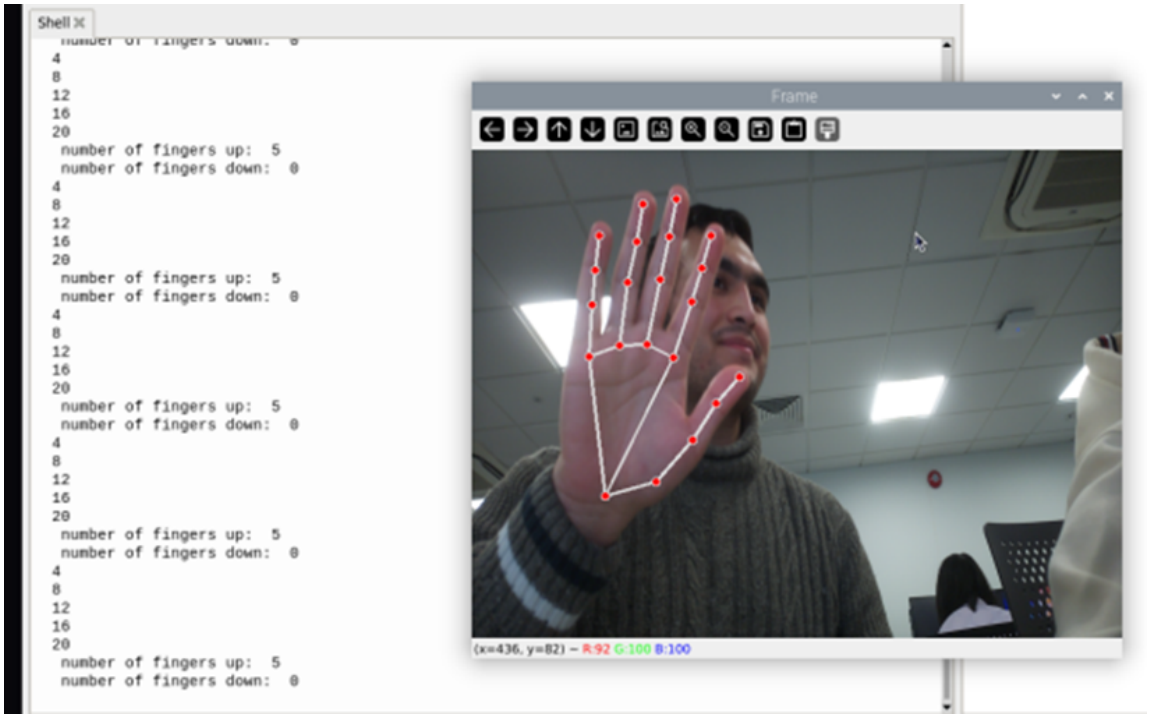


Figure 7: Original gesture recognition script being run.

Other changes have been made to the original facial and Google gesture recognition scripts. Not only was the facial recognition script updated from a newer version of the original guide [5] while still needing to be altered so that it could properly handle the OpenCV libraries that required different colour spaces than what the program already used, but the Google gesture recognition script was also continually modified in similar ways such as colour frame correction so that each program would work individually and when fused to form the combined recognition script.

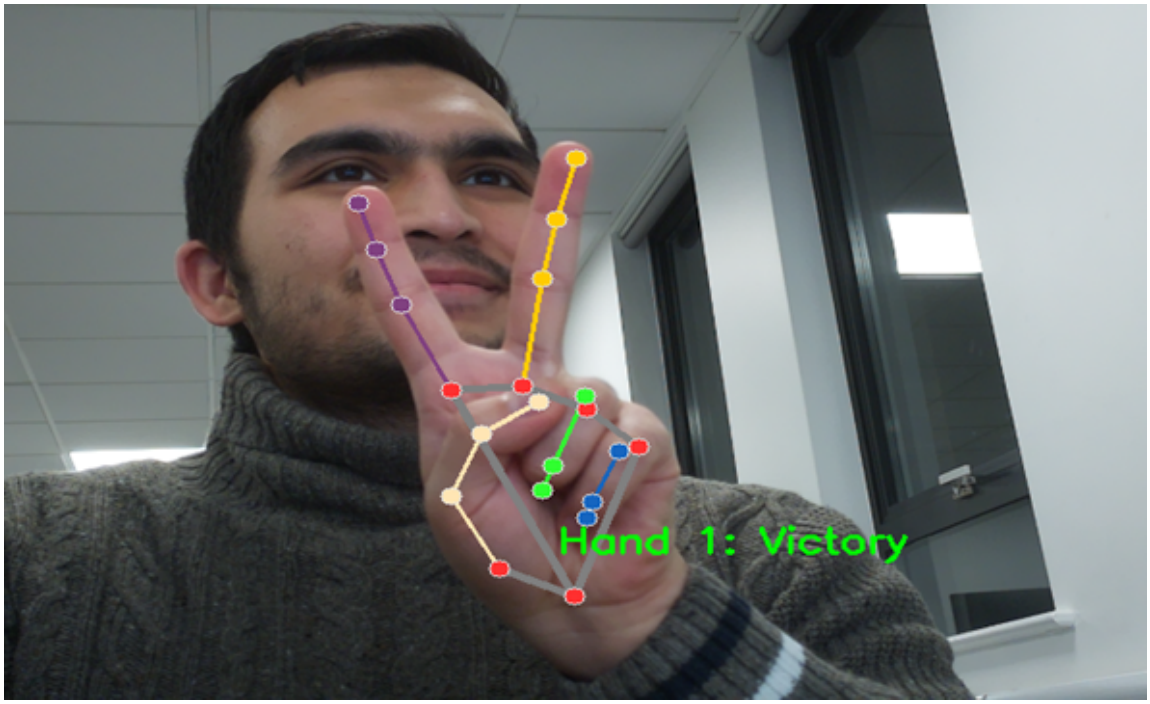


Figure 8: An instance of the Google MediaPipe gesture recogniser detecting the ”Victory” gesture.

3.3 Additional goals

After the combined recognition application had been completed and confirmed to work for me as a single user, it was modified to work for multiple users. Due to the limited capabilities of the hardware, it was decided that the program would only be modified to accommodate two users at once. The program was modified to be able to recognise the faces of two separate users and up to two hand gestures that each user makes. Using an algorithm that determines the Euclidean (shortest) distance between wrists and the centre of a user's face, the program would determine that gestures belonged to whichever user's face was closest to it, and thus their username would be incorporated in the unique gesture response message.

3.4 Testing

After completing the experiment, I decided measure the performance of the used program in quantitative metrics.

As can be seen in Appendix C, the test code is largely identical to the version used in the experiment. It has the parameter ”num_hands = 2” so that it can be configured to only expect up to two hand gestures at once as I individually test it along with added code meant to log whether a users face, hand gestures, or both would be detected each frame. The application was then run to see how well it could detect me when I was constantly making gestures and revealing my face to it over a period of 30 seconds, first testing it 0.5 meters from the camera module and then increasing my distance by 0.1 meter and keeping the time constant for each subsequent test. There is also additional code that tracks the FPS statistics over the period of each test, being inspired by how similar methods of detection

rates and FPS statics were used to evaluate the effectiveness of algorithms in detecting faces from real-time video [19]. All of these statistics have been recorded and displayed in Appendix D.

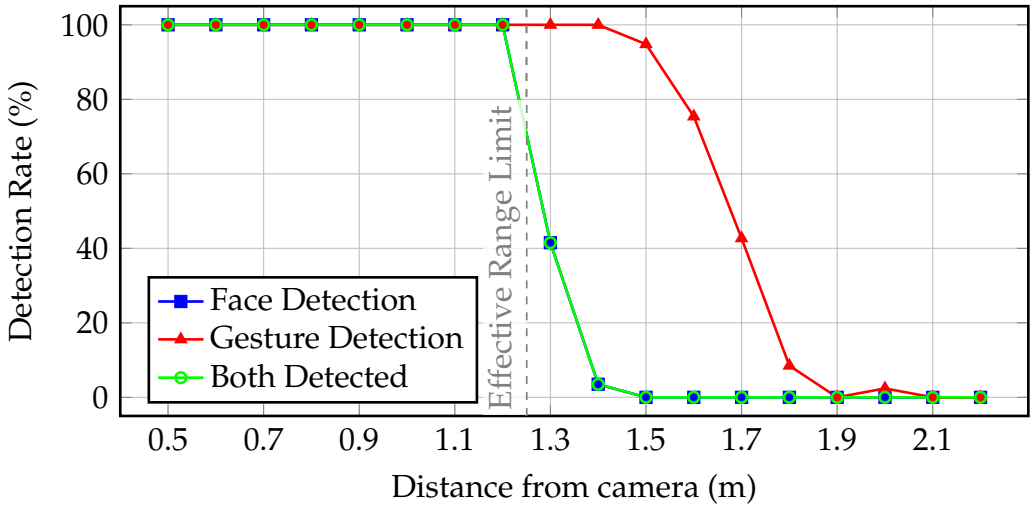


Figure 9: A graph displaying the detection rates for face, gesture, and simultaneous recognition as the user becomes more distant from the camera.

As seen in figure 6 above, the application has been determined to effectively recognise the face of a single user (over 95% detection rate) until they are 1.3 meters from the camera, at which it only has a detection rate of 41.5%, and being virtually unable to detect user faces after this point. Similarly, gestures detection rates are 100% and become more unreliable, though at a much more gradual rate than the face recognition. After 1.8 meters, at which gesture recognition rate is 8.5%, the program's ability to track gestures is negligible. This shows how the program can thus only detect a single user's face and gestures reliably up to 1.2 meters, the detection rate of both a face and gestures at a certain distance being seen to be the same as that of only faces.

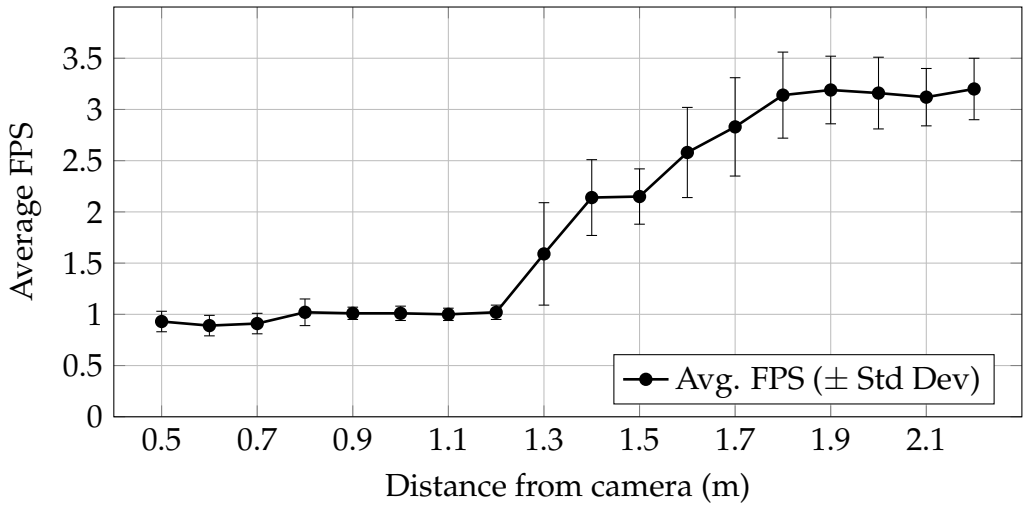


Figure 10: A graph displaying the average Frames Per Second (FPS) processed by the application as the user becomes more distant from the camera, error bars indicating standard deviation measured.

Figure 7 shows how the FPS processed by the application varies as the user's distance from the camera module does. Both the average FPS and its standard deviation of frames per second increase as distance does, notably beginning to increase after 1.2 meters when facial recognition becomes more unreliable and continuing to do so when gesture recognition rates also decrease. This exemplifies that the system has more free computational power when it does not have to use it to detect faces or gestures. It can also be observed that the standard deviation jumps from values as high as 0.13 to 0.5 at 1.3 meters, after which it maintains such high values when it has a high FPS, as opposed to having a low deviation with smaller FPS.

3.5 Technical implementation

After reviewing the available hardware, using relevant documentation [16] to understand how to properly use it, and confirming that it was fully operational, I decided to use the original Raspberry Pi components obtained from previous modules of my course. These pre-obtained components included the Raspberry Pi 4 Model B, its power cable, pi-FAN, an HDMI to Micro HDMI cable, and Blu tack. Using these, along with keyboards and mouses that were already available with monitors in building 16, saved time by allowing me to install software libraries and prepare basic scripts for the recognition programs before ordered components like the camera module, mouse, keyboard, or additional SD cards arrived by delivery.

Though I initially considered using C++ to program the application due to having more experience with the language and because the OpenCV library [7] itself was written in it, I early on decided to use Python. This was not only because I used the language for personal and academic projects before, but also due to how supporting literature confirmed its capabilities for computer vision [18]. Additionally the original facial [15] and gesture [6] recognition guides also used Python and thus were easier to import and modify in Python, with the referenced book reinforc-

ing how the language was supported by the Raspberry Pi with pre-installed IDEs such as Geany or Thonny. Whenever facing difficulties in modifying the separate recognition tasks and combining them, the official documentation [17] was consulted to achieve the desired functionality.

While developing the facial recognition algorithm, I found out that the guide for it had been updated [5]. I then followed the guide by using the newer code and following the recommended method of developing each program, this included the combined recognition application and its test variant, in its own VE. These separate virtual spaces allow each program to be run in isolation with its own set of libraries and dependencies, preventing any risk of interfering with other programs or even causing the OS itself to fail.

Continually modifying and testing the gesture recognition script revealed it was increasingly complex to accurately determine whether specific gestures are formed based on the original script's function of checking which fingers were up or down, this was made even harder due to the code having difficulties in telling whether fingers were on a left or right hand, causing it to misinterpret gestures. When the gesture recognition function was functioning properly it was incredibly slow, even without working alongside face recognition. These reasons caused me to look for more lightweight code that used the MediaPipe library in a more efficient manner, leading me to the MediaPipe Gesture Recognizer by Google [8]. Not only did this task already come pre-made with a set of specific gestures that it could detect, but it also accurately detected handedness and ran quicker than the previous version. As can be seen from the code in Appendix C, this code was easily modified to give more unique custom messages involving usernames by simply overwriting the existing response text from the "gesture-response" mapping dictionary.

Figure 11: The Google MediaPipe gesture recogniser simultaneously detecting the ”Victory” and ”Closed_Fist” gestures.

Aside from slight alterations such as making sure that the MediaPipe and OpenCV libraries received frames in their required colour spaces, parameters had to be changed to prepare both recognition scripts so that they could be used in the experiment. This meant for gesture recognition the variable ”running_mode” was set to
”vision.RunningMode.IMAGE” so that it could take live data from the camera and ”num_hands” was set to 4 to recognise up to 4 gestures while the already functioning face recognition script was fine-tuned to have a proper compromise between the accuracy of its face detection and the speed of its operation. This was done through adjusting parameters such as ”cv_scaler” which is divided against the input frame’s resolution; increasing it makes the application faster at the cost of accuracy and range of detection (it is set to ”6” for the program used in the experiment).

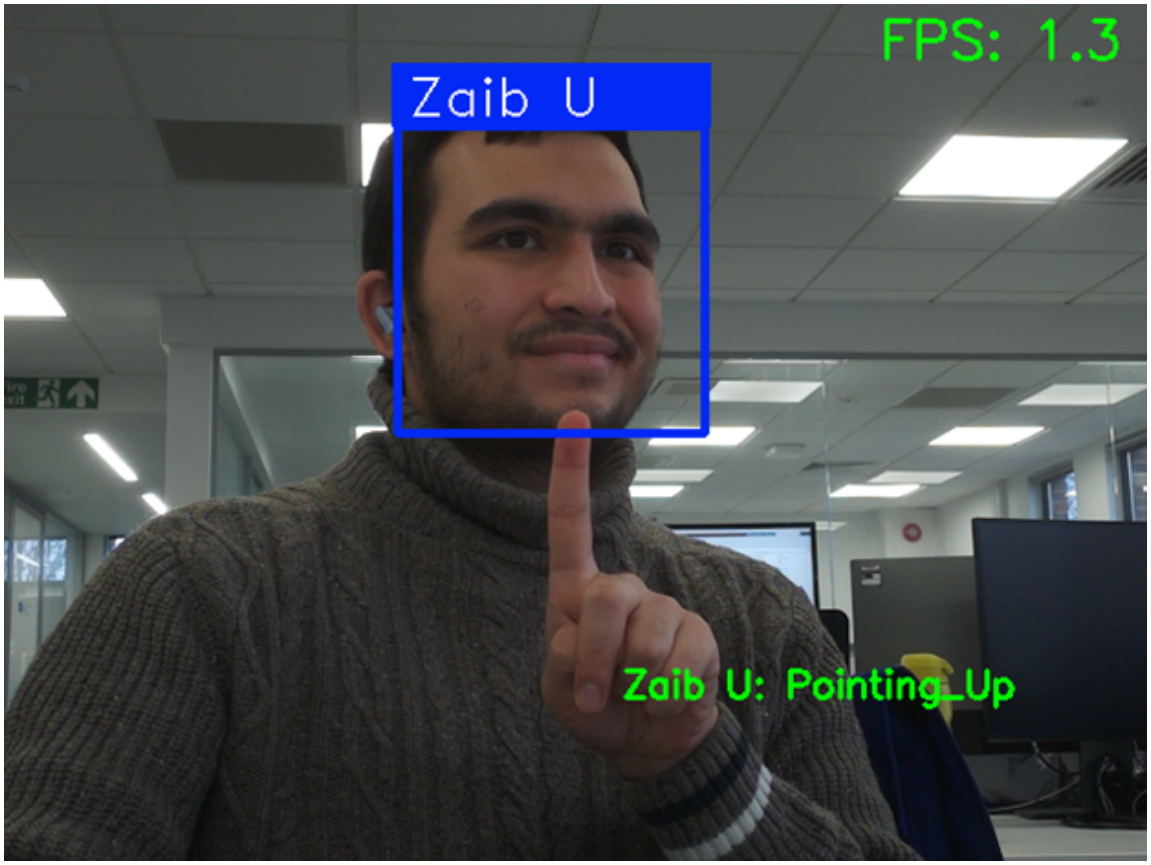


Figure 12: The combined recognition program using standard gesture response text.

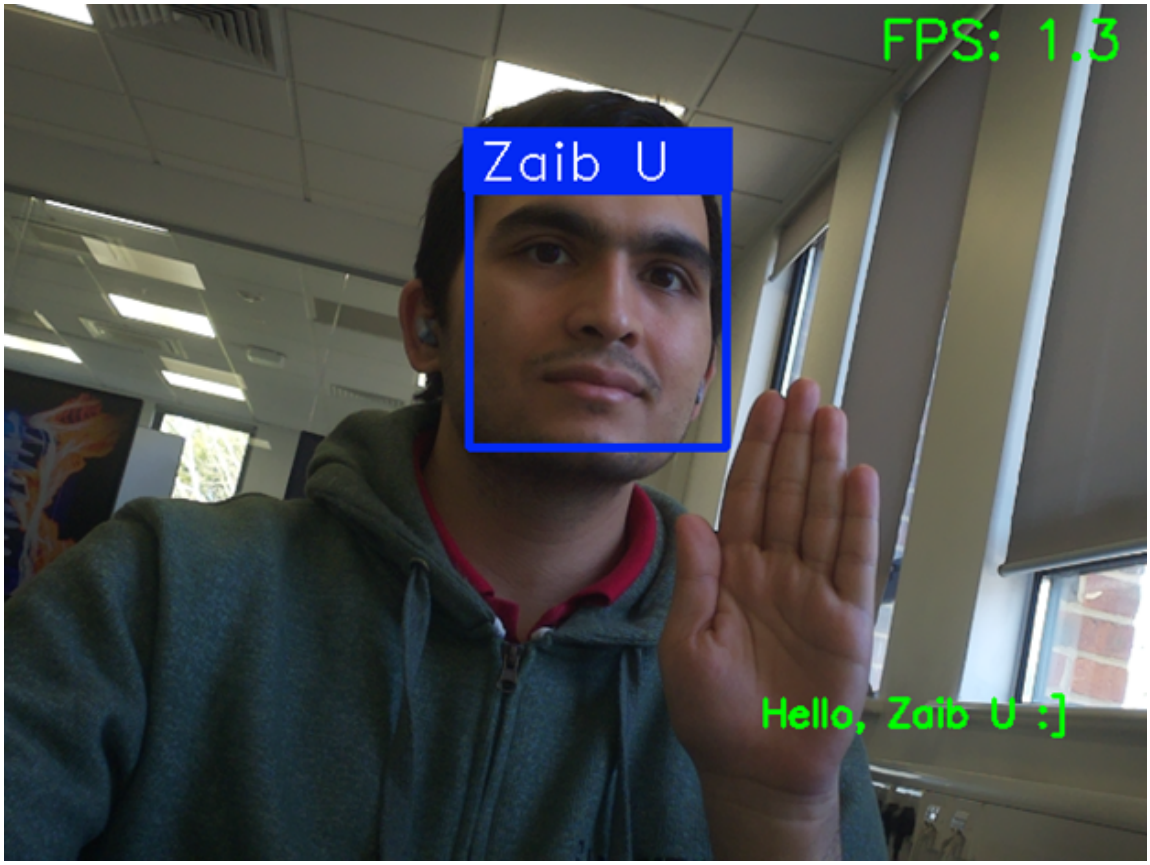


Figure 13: The combined recognition program using more engaging gesture response text.

4 Experimental setup

4.1 Original experimental model

When first planning the experiment, I made sure to study the rules and requirements laid out on the Ethics and Research Governance Online (ERGO2) website. While preparing my study ("Embedded Systems User Recognition Project") for ethics approval, I formulated the design of the experiment in which up to 6 individuals would participate. The participants would be two-handed staff or students from the University of Southampton between the ages of 18 and 60 years, capable of proper motor functions with both hands to create specific hand gestures and move them around for the combined recognition program.

After stating the nature of the application and the test about to commence, the experiment would progress as shown below:

- A participant would have their facial pictures taken and select a username for themselves.
- The model would be trained by them for a few seconds.
- The participant will see if their face is correctly recognised by the system.
- They will then test how well it recognises gestures they make from the canned gesture list with one hand.
- The participant will then observe how well the program works at recognising both of their hands.

All volunteers would be encouraged to use the system in the way that feels most natural to them, not limiting them other than informing them of the specific hand gestures that can be recognised. It would take no longer than 5 minutes for each participant to use the system, and after all participants have tested it, they would begin testing it in pairs. In pairs, participants would similarly observe how the system recognises their face and responds to the gestures they make, specifically trying to see whether the system can identify who made what specific gesture. This would take each pair less than 5 minutes, after which testing would have concluded.

Participants would then give their feedback by filling out questionnaires. They would share how they believed the system responded, how easy it was to use, and how satisfied they were with it. There was also space to share their general thoughts on the system and what could be done to improve it. The experiment will be completed after all questionnaires have been filled out. I would be available to assist volunteers and answer any questions they have regarding the experiment, taking personal notes throughout its entirety to keep a record of its happenings so that they may be evaluated later. The experiment was planned to last up to 2 hours in total.

The facial pictures and usernames and usernames of the participants would be deleted after the experiment to maintain their anonymity.

4.2 Revision

My initial ethics application was determined to be mostly sound but required a slight revision due to contentions of how data collected from the experiment would be stored. It was found that my plans to store the physical questionnaires in my own residence rather than a secure location within the University of Southampton was inappropriate, as was storing the signed consent forms on my personal laptop.

I amended this in my next submission, now planning to keep digital copies of the signed consent forms and questionnaires on university computers, shredding any physical versions of these documents after the experiment concluded. This amended version was reviewed and approved, allowing recruiting to begin.

4.3 Recruitment

As stated in my ethics application form, potential volunteers were reached out to by oral word of mouth as well as digital means like text and email. Any contacted individuals who expressed interest in joining the study would be sent an email containing a participant information sheet and a consent form to be signed.

After spending approximately three weeks contacting potential participants and discussing details of how, where, and when the experiment would take place, five individuals submitted signed consent forms and were able to confirm their availability. Not wanting to delay the experiment or risk participants becoming unavailable later, I proceeded with the experiment.

4.4 Execution

The experiment was carried out as planned. All 5 participants tested the application individually and then as pairs. For the final phase of testing, the participants were mixed to form 3 pairs, each pair testing the system before the next one did. After testing, the participants filled out and submitted their questionnaires, concluding the experiment with a duration of approximately 50 minutes. Participants' facial pictures and usernames were erased, and all paper questionnaires were digitally backed up alongside signed consent forms and personal notes taken during the experiment before being shredded, leaving no physical trace of the volunteers' identities. The following were the personal notes taken to record how the program was carried out:

"The experiment took place at 14:00 on 12th March 2025 in the level 1 laboratory of building 16, Highfield Campus, University of Southampton.

5 participants tested the system. Each first used the system individually, experiencing low frames per second (fps) such as 1.2 fps. Each user had 9 to 10 pictures of their face taken and took about 2 minutes to interact with the system.

While it was only by a few more seconds each time, training the model on participant pictures took slightly longer for every new registered user as the model had

to retrain itself on photos of already registered users along with those of newly registered ones.

After all 5 participants used the program, they tested it in 3 pairs of 2. The system could properly recognize both users and identify who made what gesture, keeping track of 4 hands at once. The system could still be tricked into thinking that a certain user's gesture belonged to another user when one placed their hands close to the other user's face and participants particularly noticed the short distance limitation of approximately 1 meter (determined by a steel ruler) after which their faces could not be reliably recognized, though what gestures were made could still be identified slightly beyond this length. Frame rates for video being processed and displayed also got particularly low for two simultaneous users with rates around 0.8 fps.

The entire experiment took around 50 minutes to complete, including the time taken to complete questionnaires."

The collected questionnaires can now be seen in Appendix E, revealing participant perceptions of the system.

5 Planning and management

5.1 Project timeline

In the earliest phases of my project I made sure to spend an adequate amount of time planning out the entire project roadmap. Considering the total workload of the project and the time that would have to be diverted to my other course modules, I planned out the phases the project would be split into, the order in which they would be addressed, and the time frame that each would be assigned. It was after the briefing on Project Management/Gantt charts that I decided I would draft and update my timeline through a Gantt chart due to its relative simplicity, ease of editing, and ability to show critical information like each phase's dependency on another in an effective visual format.

As can be seen from comparing the original Gantt chart from my progress report in Figure 14 below below with the final Gantt chart in Figure 15 that reflects the timeline of the project, some delays in progress and a minor alteration to the project schedule with a cancelled briefing have caused the timeline of the project to adapt as appropriate.

Despite each individual recognition task being developed in the expected time frame, developing the combined recognition task took longer than expected and lasted until the very end of the year. While this was not a major delay, it caused the ethics application to start later, the application process itself taking longer than expected due to delays in feedback and the time required to submit a revised edition. This itself resulted in volunteer recruitment and testing to be delayed, also effecting when the final version of the program could be tested. While each phase in this chain of slight setbacks effected each other, they did not lead to any major issues or serious project restructuring.

While the project largely follows a linear structure, especially near the end of the project timeline, I had planned for the earliest sections of the project to overlap. As can be seen in Figure 15, the timeslot for setting up the software significantly overlaps with completing a single primary recognition function. This was not only done as these two are naturally related, using the development of the recognition function as a form of testing, but also because working on them simultaneously would allow both to be completed quicker and save time for longer phases to be focused on without diverting attention.

Despite academic obligations to other modules slowing down progress from what was initially planned and a minor change in the project schedule with the Group Design Project Introduction briefing being cancelled, I rigorously followed my established action plan and managed to complete the project.

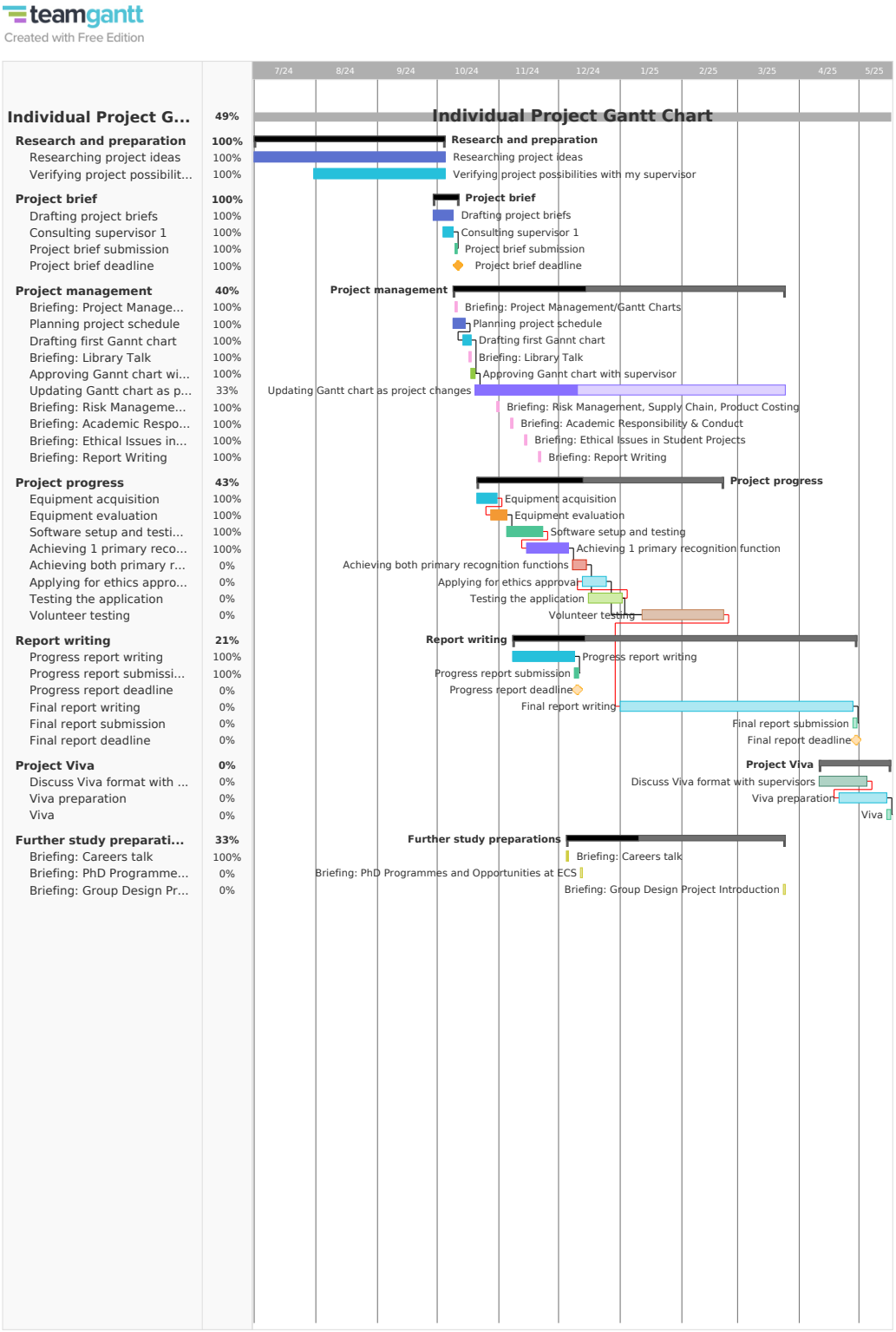


Figure 14: Initial Gantt chart first shared in the Progress Report.

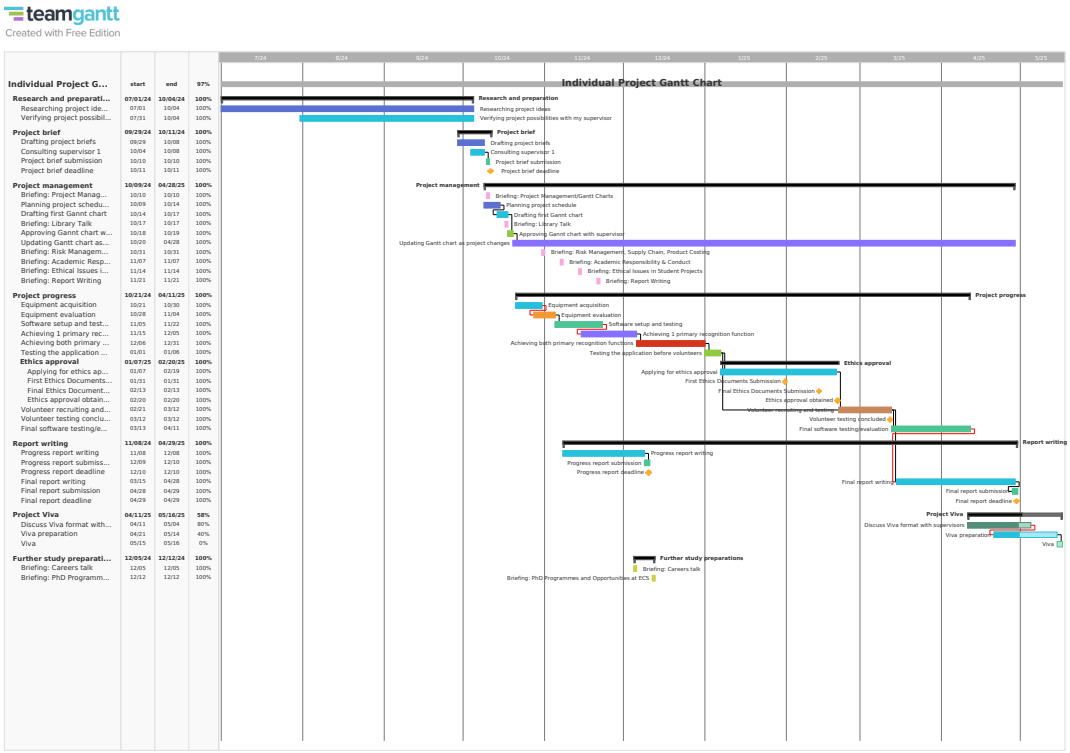


Figure 15: Final Gantt chart detailing project timeline.

5.2 Risk management

I had taken proper efforts to mitigate risks by addressing many potential points of failure early on.

Potential issues of data loss were addressed by frequently backing up the SD card of the Raspberry Pi to my personal laptop and my University OneDrive after important milestones were reached, with additional SD cards and an SD card reader being purchased in case the one currently in use was damaged or corrupted. An SD card being used early on in the project did actually get corrupted, which proved this precaution necessary, though no significant progress was lost since the basic recognition scripts being developed at that time were backed up. I also ordered a Raspberry Pi keyboard and mouse which were used with the HDMI to Micro HDMI cable I already owned to ensure that I could connect to a monitor, like the TV available at my accommodation, to interface with the Raspberry Pi outside of any University laboratory environment, allowing me to still achieve progress despite any laboratory closures, preventing the loss of time and access to the system. Whenever using any hardware, I was careful to treat it carefully and only use it for purposes permitted by official documentation [16] and reliable guides of use [18].

My supervisor was frequently consulted through various phases of the project. This allowed me to continuously share ideas of the project and how I planned to complete it, using them as a correcting force to give insight on matters such as whether phases like testing the system after the experiment would support a final

evaluation of the system. While my second examiner was also consulted whenever possible, giving a valuable perspective on expectations of the project and how the report should represent it, this was only a few times by the final phases of the project. This greatly assisted in keeping me focused on relevant additions to the project, minimising my risk of getting distracted and wasting time working on dead ends.

6 Future development and improvement

6.1 Upgrading the system

As revealed in the sentiments of users from collected questionnaires and numerical data from metric testing, the current system has notable limitations in responsivity and range. Supported by the collected FPS statistics and tested effective range limits, these are very real issues that may be improved on in future versions of the system through getting newer, faster hardware such as a Raspberry Pi 5 or even using devices that feature AI accelerators like those of Jetson Xavier NX embedded systems since similar studies indicate a lack of dedicated AI hardware is the primary bottleneck for devices like Raspberry Pi's limited performance in computer vision tasks [20].

6.2 User recommended additions

Aside from simply boosting the speed and range of the system through increasing computational power, participants also pitched other novel features that could be included to improve the user experience, as seen in their questionnaires in Appendix E. One such suggestion was adding new gestures to be recognised, something documented to be achievable with relative ease through customising the Google MediaPipe Gesture Recogniser model and training it with a dataset including new gestures [8]. Another idea involved implementing a graphical user interface (GUI) in the form a "wizard" that could be used to take photos rather than running code in an IDE, further suggesting it could be used to directly guide users on what gestures to make to interact with the system. If given proper time and resources to develop the upgraded application and acquire proper hardware to run it on, this could be developed into an interesting platform that would allow users to express themselves, getting unique responses for various gestures they make while interacting with the system in a user-friendly way similar to other systems with aesthetically pleasing yet simple GUIs to facilitate hand recognition tasks [9]. '

7 Evaluation

7.1 Project achievement

The project has been determined to be successful in achieving the primary goal of creating an HCI system using a Raspberry Pi 4 Model B as a base and testing how well it can work for registered users in an experiment with volunteers. Even the additional goal of configuring the program to run for two users and observing this in the experiment was achieved, with post-experiment testing revealing important metrics that reveal the capabilities and limits that such open-source software has when running on embedded technology like a Raspberry Pi, which is relatively cheaply available on the market.

7.2 Limitations

Despite meeting the established goals of the project, time constraints prevented me from gathering initial data that would have supported its findings. This would mainly have been in carrying out a more thorough experiment, in which a version of the test code used to extract performance statistics was used during the experiment to better analyse how the system performed for multiple users. The base experiment would have also benefited from having more volunteers. A reasonable number of users, such as 10, would provide a wider group of external perspectives that could better outline the perceived abilities of the system and how it could be improved. In the actual experiment, the number of participants was intentionally lowered due to the limited availability of students during term time and because more participants would lead to the experiment itself lasting longer, counter-productively reducing the chances of people joining the study. These restrictions also lead to using a model to identify pre-trained gestures rather than training or even programming it to identify a list of more numerous and varied gestures.

8 Conclusion

Despite any technical hurdles or logistical delays, the project was a success in displaying the capabilities of running AI for computer vision on resource-efficient embedded systems. It achieved all planned milestones without missing important deadlines, going over budget, or violating any ethical rules. It followed the proper procedure and was able to use current research to support its analysis of the application, giving a proper idea of the type of HCI system that could be constructed with a Raspberry Pi and well-documented open-source software, while outlining how the system could be modified and improved with feedback from real users and data-driven tests.

References

[1] N. Ahmed, M. Wahed, and N. C. Thompson, "The growing influence of industry in AI research," *Science*, vol. 379, no. 6635, pp. 884–886, 2023. DOI: 10.1126/science.ade2420. eprint: https://www.science.org/doi/pdf/10.1126/science.ade2420. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.ade2420.

[2] D. Prerad, Ž. Ivanović, and A. Avramović, "Edge ai acceleration using opencl framework on nxp i.mx 8m nano som," in *2024 XV International Symposium on Industrial Electronics and Applications (INDEL)*, 2024, pp. 1–5. DOI: 10.1109/INDEL62640.2024.10772692.

[3] R. Thakur, A. Thakur, R. Viral, and D. Asija, "AI based visual impairment system for blind and people with least reading capabilities," in *2024 IEEE Students Conference on Engineering and Systems (SCES)*, 2024, pp. 1–5. DOI: 10.1109/SCES61914.2024.10652439.

[4] B. Brkljač, B. Antić, and Z. Mitrović, "Potential of embedded vision platforms in development of spatial ai enabled cps," in *2022 11th Mediterranean Conference on Embedded Computing (MECO)*, 2022, pp. 1–4. DOI: 10.1109/MECO55406.2022.9797078.

[5] Jaryd. "Face recognition with Raspberry Pi and OpenCV." (2024), [Online]. Available: https://core-electronics.com.au/guides/raspberry-pi/face-recognition-with-raspberry-pi-and-opencv/ (visited on 11/23/2024).

[6] Tim. "Hand recognition and finger identification with Raspberry Pi and OpenCV." (2023), [Online]. Available: https://core-electronics.com.au/guides/hand-identification-raspberry-pi/ (visited on 09/01/2024).

[7] O. team. "Opencv." (2025), [Online]. Available: https://opencv.org (visited on 01/01/2025).

[8] G. AI. "Gesture recognition task guide." (2025), [Online]. Available: https://ai.google.dev/edge/mediapipe/solutions/vision/gesture_recognizer (visited on 01/01/2025).

[9] K. Li, J. Cheng, Q. Zhang, and J. Liu, "Hand gesture tracking and recognition based human-computer interaction system and its applications," in *2018 IEEE International Conference on Information and Automation (ICIA)*, 2018, pp. 667–672. DOI: 10.1109/ICInfA.2018.8812508.

[10] M. V. Phanindra Kumar and K. R., "Traffic sign detection and recognition with deep cnn using raspberry pi 4 in real-time," in *2023 IEEE 11th Region 10 Humanitarian Technology Conference (R10-HTC)*, 2023, pp. 24–29. DOI: 10.1109/R10-HTC57504.2023.10461824.

[11] A. Dalisay, "Using facial recognition for authentication in keyless systems," University of Southampton, 2021.

[12] B. Mashanda, "Real time sign language gesture recognition on an embedded system," University of Southampton, 2021.

[13] S. Kumar, P. Kumar, P. Mishra, and P. Tewari, "A robust sign language and hand gesture recognition system using convolutional neural networks," in *2023 5th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*, 2023, pp. 395–399. DOI: 10.1109/ICAC3N60023.2023.10541471.

[14] H. Avula, R. R, and A. S Pillai, "Cnn based recognition of emotion and speech from gestures and facial expressions," in *2022 6th International Conference on Electronics, Communication and Aerospace Technology*, 2022, pp. 1360–1365. DOI: 10.1109/ICECA55336.2022.10009316.

[15] Tim. "Face recognition with Raspberry Pi and OpenCV." (2024), [Online]. Available: https://core-electronics.com.au/guides/face-identify-raspberry-pi/ (visited on 09/01/2024).

[16] Raspberry Pi Foundation. "Raspberry Pi Documentation." (2024), [Online]. Available: https://www.raspberrypi.com/documentation/accessories/camera.html (visited on 10/01/2024).

[17] T. P. S. Foundation. "Python 3.13.3 documentation." (2025), [Online]. Available: https://docs.python.org/3/ (visited on 01/01/2025).

[18] A. Pajankar, *Raspberry Pi computer vision programming: design and implement computer vision applications with Raspberry Pi, OpenCV, and Python 3*. Packt Publishing Ltd, 2020.

[19] A. Adouani, W. M. B. Henia, and Z. Lachiri, "Comparative study of face detection methods in spontaneous videos," in *2019 International Conference on Control, Automation and Diagnosis (ICCAD)*, 2019, pp. 1–6. DOI: 10.1109/ICCAD46983.2019.9037883.

[20] R. C. Camara de M. Santos, M. Coelho, and R. Oliveira, "Real-time object detection performance analysis using yolov7 on edge devices," *IEEE Latin America Transactions*, vol. 22, no. 10, pp. 799–805, 2024. DOI: 10.1109/TLA.2024.10705971.

Appendices

9 Appendix A - Individual Project brief

Facial and hand gesture recognition AI for embedded systems

Student: Zaib Uddin
Supervisor: Professor David Thomas

Problem

AI is becoming a staple in various technologies, from those used by giant search engines to well-versed grammar checkers. Continued interest and promising results in this area have fueled each other and led to a mainstream insurgence of AI, though the focus is mainly given to large-scale applications. The discussion seems to be about what groundbreaking advancements can be achieved by tasking countless servers with complex algorithms rather than incorporating AI for smaller and more familiar tasks.

Two areas of increasing promise are facial and hand gesture recognition through AI. They have great potential to be used together and implemented in small, efficient embedded systems that can occupy several corners of a user's environment. Aside from offering an intuitive way of interfacing with devices, this technology can be developed for niche purposes such as allowing those who normally have trouble communicating verbally (e.g. deaf and/or mute) to express themselves more naturally by interpreting their facial and hand gestures.

Goals

My primary goal is to create software on a Raspberry Pi 4 Model B with 4 GB of RAM that can use a camera module to successfully identify the face of a registered user and recognise multiple gestures made with their hands. It should be able to identify a certain Region of Interest (ROI) where a registered face will be and name the user, displaying certain messages in response to specific hand gestures (such as displaying the text "Hello *user-name*" in response to an open palm gesture, using the name of the registered user).

My secondary goal would be adapting the software to reliably recognise the faces of different individuals and process their hand gestures. The program should be able to track the separate ROIs on the faces of separate individuals, each having a unique name, and recognise the hand gestures made by each individual. An

extended goal beyond this would be ensuring that the hands of each user could be identified as belonging to a specific individual and thus be used to give more personalised responses for each user's input.

Scope

The application will be based on existing facial and hand recognition projects (such as those made from OpenCV and MediaPipe). Such open-source work will be optimised to operate on the Raspberry Pi with their relevant functions combined in a single application. Another important aspect of this project is determining whether the final product can effectively allow a person, or multiple people considering the extended project, to interface with the system. Factors to consider are the speed and accuracy of recognising faces and responding to gestures.

10 Appendix B - Final facial and gesture recognition code

```
import cv2
import numpy as np
import face_recognition
import pickle
import time
import mediapipe as mp
from mediapipe.tasks import python
from mediapipe.tasks.python import vision
from picamera2 import Picamera2

# Load pre-trained face encodings
print("[INFO] loading encodings...")
with open("encodings.pickle", "rb") as f:
    data = pickle.loads(f.read())
known_face_encodings = data["encodings"]
known_face_names = data["names"]

# Initialize the MediaPipe Gesture Recognizer
base_options = python.BaseOptions(model_asset_path='
    gesture_recognizer.task')
options = vision.GestureRecognizerOptions(base_options=base_options,
    running_mode=vision.RunningMode.IMAGE, num_hands = 4)
gesture_recognizer = vision.GestureRecognizer.create_from_options(
    options)

#  Create specialised responses to gestures with a gesture-response
    mapping dictionary
gesture_text = {
     "Victory": "Peace, {username}!",
     "Thumb_Up": "Keep going, {username}!",
     "Thumb_Down": "You can do better, {username}!",
     "Open_Palm": "Hello, {username} :]",
     "Pointing_Up": "Whats up, {username}?",
     "ILoveYou": "Love you too, {username} ;]",
     "Closed_Fist": "Keep it tight, {username}!",
     "None": "I got nothing, {username}",
 }

# Initialize Raspberry Pi Camera Module 3
picam2 = Picamera2()

# preview_config = picam2.create_preview_configuration(main={"size":
    (640, 480)})
# The camera output format is set to XRGB8888 to avoid alpha channel
    issues
preview_config = picam2.create_preview_configuration( main = { "size
    ": (1920, 1000), "format": "XRGB8888"},
sensor = {"output_size": (2304, 1296), "bit_depth": 10}, encode = "
    main")
# A 16:9 aspect ratio is chosen for the camera module
# and the raw sensor output resolution is set to 2304x1296 out of
    the camera sensor's 4056x3040 pixel resolution
# This is done to achieve a wider field of view for the camera
    module
```

```
# encode = "main" allows the H.264 encoder block to be used to speed up processing
picam2.configure(preview_config)
picam2.start()

# Initialize the variables
cv_scaler = 6 # this has to be a whole number, the resolution of the input frame is reduced by
# a factor of 1/cv-scaler, the higher cv_scaler is the faster processing becomes
frame_count = 0
face_locations = []
face_encodings = []
face_names = []
start_time = time.time()
fps = 0

cv2.namedWindow('Complete Recognition', cv2.WINDOW_NORMAL)
# WINDOW_NORMAL allows the displayed window to be resized with manual sizing
cv2.resizeWindow('Complete Recognition', 1280, 720)
# Defines the size of the displayed window

try:
    while True:
        # Capture frame from Pi Camera and convert it into a BGR format
        # Ensuring that the captrued frame has only 3 channels (the alpha channel is trimmed and made contiguous), a contiguous 3-channel BGR array is obtained
        frame = picam2.capture_array()[:, :, :3].copy()

        # BGR frames are used for the OpenCV display
        bgr_frame = cv2.cvtColor(frame, cv2.COLOR_BGRA2BGR)

        # Convert the BGR frame to a RGB frame for MediaPipe processing
        rgb_frame = cv2.cvtColor(bgr_frame, cv2.COLOR_BGR2RGB)

        # Face detecting section


        # Resize the frame using cv_scaler to increase performance (less pixels processed, less time spent)
        resized_rgb_frame = cv2.resize(rgb_frame, (0, 0), fx=(1/cv_scaler), fy=(1/cv_scaler))

        # Find all the faces and face encodings in the current frame of video
        face_locations = face_recognition.face_locations(resized_rgb_frame)
        face_encodings = face_recognition.face_encodings(resized_rgb_frame, face_locations, model='small')

        face_names = []
        for face_encoding in face_encodings:
        # See if the face is a match for the known face(s)
            matches = face_recognition.compare_faces(known_face_encodings, face_encoding)
```

```
            name = "Unknown"

        # Use the known face with the smallest distance to the recognised hand
            face_distances = face_recognition.face_distance(known_face_encodings, face_encoding)
            best_match_index = np.argmin(face_distances)
            if matches[best_match_index]:
                name = known_face_names[best_match_index]
            face_names.append(name)


        # Gesture recognition section

        # Process with Gesture Recognizer
        image_mp = mp.Image(image_format=mp.ImageFormat.SRGB, data=rgb_frame)
        gesture_recognition_result = gesture_recognizer.recognize(image_mp)

        # Draw the face results
        for (top, right, bottom, left), name in zip(face_locations, face_names):
            # Scale back up face locations since the frame we detected in was scaled
            top *= cv_scaler
            right *= cv_scaler
            bottom *= cv_scaler
            left *= cv_scaler

            # Draw a box around the face
            cv2.rectangle(bgr_frame, (left, top), (right, bottom), (244, 42, 3), 3)

            # Draw a label with a name below the face
            cv2.rectangle(bgr_frame, (left -3, top -35), (right +3, top), (244, 42, 3), cv2.FILLED)
            font = cv2.FONT_HERSHEY_DUPLEX
            cv2.putText(bgr_frame, name, (left + 6, top - 6), font, 1.0, (255, 255, 255), 1)

        # Initialise an empty list to store face bounding boxes and their associated usernames
        face_boxes = []
        for (top, right, bottom, left), name in zip(face_locations, face_names):
              # Scale back up face locations since the frame we detected in was scaled
              top *= cv_scaler
              right *= cv_scaler
              bottom *= cv_scaler
              left *= cv_scaler

              face_boxes.append((left, top, right, bottom, name)) # Stores the scaled coordinates and the
                                                                  # associated username as a tuple

```

```
134         # Initialise an empty list in which wrist coordinates of
    users' hands can be stored
135         hand_position = []
136         if gesture_recognition_result.hand_landmarks:
137             for landmarks in gesture_recognition_result.
    hand_landmarks:
138                 wrist = landmarks[0] # Acquires the landmark of the
    user's wrist (this is at index 0 of Mediapipe's hand model)
139                 wrist_x = int(wrist.x * bgr_frame.shape[1]) # X and
    Y coordinates of the wrist location
140                 wrist_y = int(wrist.y * bgr_frame.shape[0]) # Covert
     normalised (between the range of 0 and 1) X and Y coordinates
    into pixel positions
141                 hand_position.append((wrist_x, wrist_y))  # Stores
    the wrist coordinates as a tuple
142 
143         # Initialise an empty list in which wrist coordinates of
    users' hands can be associated with the face closest to them and
    stored
144         hand_face_pairs = []
145         for hand_index, (hx, hy) in enumerate(hand_position):
146             closest_face = None # A null variable is assigned to the
     closest face detected
147             min_dist = float('inf')
148 
149             for (left, top, right, bottom, name) in face_boxes:
150                 face_center_x = (left + right) // 2  # The
    horizontal and vertical centre of face bounding boxes are
    calculated
151                 face_center_y = (top + bottom) // 2
152                 distance = ((hx - face_center_x)**2 + (hy -
    face_center_y)**2)**0.5 # The Euclidean (shortest) distance
    between wrists and
153                 # the centre of the face is calculated
154 
155                 if distance < min_dist: # Updates the closest face
    if the current face detected is closer
156                     min_dist = distance # The minimum distance is
    the shortest distance between the wrist and centre of the face
157                     closest_face = name
158             hand_face_pairs.append((hand_index, closest_face)) #
    Stores the closest face detected and the
159                                                                #
    associated index number as a tuple
160 
161         # Draw the gesture results
162         if gesture_recognition_result.gestures:
163            for index, (gesture_group, landmarks) in enumerate(zip(
    gesture_recognition_result.gestures,
164 
    gesture_recognition_result.hand_landmarks)):
165 
166                 top_gesture = gesture_group[0]
167                 # Access landmarks from the list of landmarks
168                 wrist = landmarks[0]
169                 text_x = int(wrist.x * bgr_frame.shape[1]) # X and Y
    coordinates of the wrist location will be used to position
    gesture text
```

```
170                 text_y = int(wrist.y * bgr_frame.shape[0]) -30 #
    Covert normalised (between the range of 0 and 1) X and Y
171                                                                 #
    coordinates into pixel positions
172                 # The hand closest to the first detected face will be
    used to respond to gestures
173                 # with usernames
174
175
176                 username = "Unknown" # The face of the user could not
    be identified
177                 if index < len(hand_face_pairs): # Executes as long as
     there are hand (wrist) associations in the list
178                     _, username = hand_face_pairs[index] # The first
    element of the tuple (index) is stored in "_", it is not used
179                 # The second element (the username) is stored from the
     tuple
180
181                 if top_gesture.category_name in gesture_text:
182                   gesture_response = gesture_text[top_gesture.
    category_name].format(username=username)
183                 else:
184                   gesture_response = f"{username}: {top_gesture.
    category_name}"
185                 # Display the response for a specific gesture of a
    specific user
186                 cv2.putText(bgr_frame, gesture_response, (text_x,
    text_y),
187                             cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0)
    , 2) # Green text will be used to identify each hand gesture
188
189             # with the username of the recognised user face
190               #    cv2.putText(bgr_frame, f"Hand {index + 1}: {
    top_gesture.category_name}", (text_x, text_y),
191                #              cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255,
    0), 2) # Green text will be used to identify each hand gesture
192
193             # Calculate and update Frames Per Second
194             frame_count += 1
195             elapsed_time = time.time() - start_time
196             if elapsed_time > 1:
197                 fps = frame_count / elapsed_time
198                 frame_count = 0
199                 start_time = time.time()
200
201             cv2.putText(bgr_frame, f"FPS: {fps:.1f}", (bgr_frame.shape
    [1] - 150, 30),
202                   cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
203
204
205             # Display everything over the video feed.
206             cv2.imshow('Complete Recognition', bgr_frame)
207
208             # Break the loop and stop the script if 'q' is pressed
209             if cv2.waitKey(1) & 0xFF == ord('q'):
210                 break
211
212 finally:
    picam2.stop()
```

```
213     cv2.destroyAllWindows()
214     gesture_recognizer.close()
```

Listing 1: Python script for face and gesture recognition tested by volunteers.

11 Appendix C - Performance test facial and gesture recognition code

```
import cv2
import numpy as np
import face_recognition
import pickle
import time
import mediapipe as mp
from mediapipe.tasks import python
from mediapipe.tasks.python import vision
from picamera2 import Picamera2

import statistics # Used to calculate metrics like mean and standard deviation

# Test configuration
# Set to "True" to run in test mode, "False" for normal operation
RUN_TEST_MODE = True
TEST_DURATION_SECONDS = 30 # For how many seconds the test should run


# Test data storage structures (acting as lists)
fps_readings = []
face_detected_log = []
gesture_detected_log = []
both_detected_log = []



# Load pre-trained face encodings
print("[INFO] loading encodings...")
with open("encodings.pickle", "rb") as f:
    data = pickle.loads(f.read())
known_face_encodings = data["encodings"]
known_face_names = data["names"]

# Initialize the MediaPipe Gesture Recognizer
base_options = python.BaseOptions(model_asset_path='gesture_recognizer.task')
options = vision.GestureRecognizerOptions(base_options=base_options, running_mode=vision.RunningMode.IMAGE, num_hands = 2)
gesture_recognizer = vision.GestureRecognizer.create_from_options(options)

#  Create specialised responses to gestures with a gesture-response mapping dictionary
gesture_text = {
     "Victory": "Peace, {username}!",
     "Thumb_Up": "Keep going, {username}!",
     "Thumb_Down": "You can do better, {username}!",
     "Open_Palm": "Hello, {username} :]",
     "Pointing_Up": "Whats up, {username}?",
     "ILoveYou": "Love you too, {username} ;]",
     "Closed_Fist": "Keep it tight, {username}!",
     "None": "I got nothing, {username}",
```

```
 }

# Initialize Raspberry Pi Camera Module 3
picam2 = Picamera2()

# preview_config = picam2.create_preview_configuration(main={"size": (640, 480)})
# The camera output format is set to XRGB8888 to avoid alpha channel issues
preview_config = picam2.create_preview_configuration( main = { "size": (1920, 1000), "format": "XRGB8888"},
sensor = {"output_size": (2304, 1296), "bit_depth": 10}, encode = "main")
# A 16:9 aspect ratio is chosen for the camera module
# and the raw sensor output resolution is set to 2304x1296 out of the camera sensor's 4056x3040 pixel resolution
# This is done to achieve a wider field of view for the camera module
# encode = "main" allows the H.264 encoder block to be used to speed up processing
picam2.configure(preview_config)
picam2.start()

# Initialize the variables
cv_scaler = 6 # this has to be a whole number, the resolution of the input frame is reduced by
# A factor of 1/cv-scaler, the higher cv_scaler is the faster processing becomes
frame_count = 0
face_locations = []
face_encodings = []
face_names = []
test_start_time = time.time() # Used to determine the time this run of the test began
fps_start_time = test_start_time # Used to calculate the rolling average FPS
fps = 0

cv2.namedWindow('Complete Recognition', cv2.WINDOW_NORMAL)
# WINDOW_NORMAL allows the displayed window to be resized with manual sizing
cv2.resizeWindow('Complete Recognition', 1280, 720)
# Defines the size of the displayed window

if RUN_TEST_MODE:
    print(f"[INFO] Test will run for {TEST_DURATION_SECONDS} seconds.")

try:
    while True:
        frame_start_time = time.time() # Start timing this frame for detailed FPS
        elapsed_time = frame_start_time - test_start_time

        # Test duration check:
        if RUN_TEST_MODE and (elapsed_time > TEST_DURATION_SECONDS):
            print(f"[INFO] Test duration completed, {elapsed_time:.2f} seconds elapsed.")
            break
```

```
93
94          # Capture frame from Pi Camera and convert it into a BGR
    format
95          # Ensuring that the captrued frame has only 3 channels (the
    alpha channel is trimmed and made contiguous), a contiguous 3-
    channel BGR array is obtained
96          frame = picam2.capture_array()[:, :, :3].copy()
97
98          # BGR frames are used for the OpenCV display
99          bgr_frame = cv2.cvtColor(frame, cv2.COLOR_BGRA2BGR)
100
101         # Convert the BGR frame to a RGB frame for MediaPipe
    processing
102         rgb_frame = cv2.cvtColor(bgr_frame, cv2.COLOR_BGR2RGB)
103
104
105         # Face detection section:
106
107
108         # Resize the frame using cv_scaler to increase performance (
    less pixels processed, less time spent)
109         resized_rgb_frame = cv2.resize(rgb_frame, (0, 0), fx=(1/
    cv_scaler), fy=(1/cv_scaler))
110
111         # Find all the faces and face encodings in the current frame
     of video
112         face_locations = face_recognition.face_locations(
    resized_rgb_frame)
113         face_encodings = face_recognition.face_encodings(
    resized_rgb_frame, face_locations, model='small')
114
115         face_names = []
116         for face_encoding in face_encodings:
117         # See if the face is a match for the known face(s)
118             matches = face_recognition.compare_faces(
    known_face_encodings, face_encoding)
119             name = "Unknown"
120
121         # Use the known face with the smallest distance to the
    recognised hand
122             face_distances = face_recognition.face_distance(
    known_face_encodings, face_encoding)
123             best_match_index = np.argmin(face_distances)
124             if matches[best_match_index]:
125                 name = known_face_names[best_match_index]
126             face_names.append(name)
127
128
129         # Gesture recognition section:
130
131         # Process with Gesture Recognizer
132         image_mp = mp.Image(image_format=mp.ImageFormat.SRGB, data=
    rgb_frame)
133         gesture_recognition_result = gesture_recognizer.recognize(
    image_mp)
134
135         # Draw the face results
136         for (top, right, bottom, left), name in zip(face_locations,
    face_names):
```

```
137             # Scale back up face locations since the frame we
    detected in was scaled
138             top *= cv_scaler
139             right *= cv_scaler
140             bottom *= cv_scaler
141             left *= cv_scaler
142 
143             # Draw a box around the face
144             cv2.rectangle(bgr_frame, (left, top), (right, bottom),
    (244, 42, 3), 3)
145 
146             # Draw a label with a name below the face
147             cv2.rectangle(bgr_frame, (left -3, top -35), (right +3,
    top), (244, 42, 3), cv2.FILLED)
148             font = cv2.FONT_HERSHEY_DUPLEX
149             cv2.putText(bgr_frame, name, (left + 6, top - 6), font,
    1.0, (255, 255, 255), 1)
150 
151         # Initialise an empty list to store face bounding boxes and
    their associated usernames
152         face_boxes = []
153         for (top, right, bottom, left), name in zip(face_locations,
    face_names):
154             # Scale back up face locations since the frame we
    detected in was scaled
155             top *= cv_scaler
156             right *= cv_scaler
157             bottom *= cv_scaler
158             left *= cv_scaler
159 
160             face_boxes.append((left, top, right, bottom, name)) #
    Stores the scaled coordinates and the
161                                                                 #
    associated username as a tuple
162 
163         # Initialise an empty list in which the wrist coordinates of
     users' hands can be stored
164         hand_position = []
165         if gesture_recognition_result.hand_landmarks:
166             for landmarks in gesture_recognition_result.
    hand_landmarks:
167                 wrist = landmarks[0] # Acquires the landmark of the
    user's wrist (this is at index 0 of Mediapipe's hand model)
168                 wrist_x = int(wrist.x * bgr_frame.shape[1]) # X and
    Y coordinates of the wrist location
169                 wrist_y = int(wrist.y * bgr_frame.shape[0]) # Covert
     normalised (between the range of 0 and 1) X and Y coordinates
    into pixel positions
170                 hand_position.append((wrist_x, wrist_y))  # Stores
    the wrist coordinates as a tuple
171 
172         # Initialise an empty list in which the wrist coordinates of
     users' hands can be associated with the face closest to them and
     stored
173         hand_face_pairs = []
174         for hand_index, (hx, hy) in enumerate(hand_position):
175             closest_face = None # A null variable is assigned to the
     closest face detected
176             min_dist = float('inf')
```

```
177
178             for (left, top, right, bottom, name) in face_boxes:
179                 face_center_x = (left + right) // 2  # The
    horizontal and vertical centre of face bounding boxes are
    calculated
180                 face_center_y = (top + bottom) // 2
181                 distance = ((hx - face_center_x)**2 + (hy -
    face_center_y)**2)**0.5 # The Euclidean (shortest) distance
    between wrists and
182                 # the centre of the face is calculated
183
184                 if distance < min_dist: # Updates the closest face
    if the current face detected is closer
185                     min_dist = distance # The minimum distance is
    the shortest distance between the wrist and centre of the face
186                     closest_face = name
187             hand_face_pairs.append((hand_index, closest_face)) #
    Stores the closest face detected and the
188                                                                 #
    associated index number as a tuple
189
190         # Draw the gesture results
191         if gesture_recognition_result.gestures:
192             for index, (gesture_group, landmarks) in enumerate(zip(
    gesture_recognition_result.gestures,
193
    gesture_recognition_result.hand_landmarks)):
194
195                 top_gesture = gesture_group[0]
196                 # Access landmarks from the list of landmarks
197                 wrist = landmarks[0]
198                 text_x = int(wrist.x * bgr_frame.shape[1]) # X and Y
    coordinates of the wrist location will be used to position
    gesture text
199                 text_y = int(wrist.y * bgr_frame.shape[0]) -30 #
    Covert normalised (between the range of 0 and 1) X and Y
200                                                                 #
    coordinates into pixel positions
201                 # The hand closest to the first detected face will be
    used to respond to gestures
202                 # with usernames
203
204
205                 username = "Unknown" # The face of the user could not
    be identified
206                 if index < len(hand_face_pairs): # Executes as long as
     there are hand (wrist) associations in the list
207                     _, username = hand_face_pairs[index] # The first
    element of the tuple (index) is stored in "_", it is not used
208                 # The second element (the username) is stored from the
     tuple
209
210                 if top_gesture.category_name in gesture_text:
211                   gesture_response = gesture_text[top_gesture.
    category_name].format(username=username)
212                 else:
213                   gesture_response = f"{username}: {top_gesture.
    category_name}"
```

```
            # Display the response for a specific gesture of a specific user
            cv2.putText(bgr_frame, gesture_response, (text_x, text_y),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2) # Green text will be used to identify each hand gesture

        # with the username of the recognised user face
          #    cv2.putText(bgr_frame, f"Hand {index + 1}: {top_gesture.category_name}", (text_x, text_y),
            #             cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2) # Green text will be used to identify each hand gesture


        # Performance measurement section

        # Measuring and recording the duration of each frame to determine the fps over the entire test period
        frame_end_time = time.time()
        frame_duration = frame_end_time - frame_start_time
        if frame_duration > 0:
            current_fps = 1.0 / frame_duration
            fps_readings.append(current_fps)

        # Detection logging for analysis of a known distance
        face_detected_this_frame = len(face_names) > 0 # True if at least one face (of a registered or unknown user) was detected this frame
        gesture_detected_this_frame = bool(gesture_recognition_result.gestures) # True if list is not empty, if MediaPipe detects a least one hand in this frame

        face_detected_log.append(face_detected_this_frame)
        gesture_detected_log.append(gesture_detected_this_frame)
        both_detected_log.append(face_detected_this_frame and gesture_detected_this_frame)


        # Calculate and update Frames Per Second, display its rolling average
        frame_count += 1
        elapsed_time = time.time() - frame_start_time
        if elapsed_time > 1:
            fps = frame_count / elapsed_time
            frame_count = 0
            frame_start_time = time.time()

        cv2.putText(bgr_frame, f"FPS: {fps:.1f}", (bgr_frame.shape[1] - 150, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

        # Display the test mode selected
        if RUN_TEST_MODE:
             cv2.putText(bgr_frame, "TEST MODE", (10, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv2.LINE_AA)
```

```
258         # Display everything over the video feed.
259         cv2.imshow('Complete Recognition', bgr_frame)
260 
261         # Break the loop and stop the script if 'q' is pressed
262         if cv2.waitKey(1) & 0xFF == ord('q'):
263             if RUN_TEST_MODE and not fps_readings: # Handles the
    case when "q" is pressed before first frame completes
264                 print("[INFO] Test ended prematurely, no FPS data
    collected.")
265             break
266 
267 finally:
268         # Display the captured performance metrics
269     if RUN_TEST_MODE and fps_readings:
270         print("\n--- Performance test results ---")
271         avg_fps = statistics.mean(fps_readings)
272         max_fps = max(fps_readings)
273         min_fps = min(fps_readings)
274         stdev_fps = statistics.stdev(fps_readings) if len(
    fps_readings) > 1 else 0
275 
276         print(f"Test duration: {TEST_DURATION_SECONDS} seconds")
277         print(f"Total frames processed: {len(fps_readings)}")
278         print(f"Average FPS: {avg_fps:.2f}")
279         print(f"Maximum FPS: {max_fps:.2f}")
280         print(f"Minimum FPS: {min_fps:.2f}")
281         print(f"Standard deviation FPS: {stdev_fps:.2f}")
282 
283         # Detection log statistics (useful for distance test
    analysis)
284         total_frames = len(face_detected_log)
285         face_detections = sum(face_detected_log)
286         gesture_detections = sum(gesture_detected_log)
287         both_detections = sum(both_detected_log)
288 
289         print("\n--- Detection log summary ---")
290         print(f"Frames with face detected: {face_detections}/{
    total_frames} ({face_detections/total_frames*100:.1f}%)")
291         print(f"Frames with gesture detected: {gesture_detections}/{
    total_frames} ({gesture_detections/total_frames*100:.1f}%)")
292         print(f"Frames with both detected: {both_detections}/{
    total_frames} ({both_detections/total_frames*100:.1f}%)")
293 
294         print("\n--- Distance test guidance ---")
295         print("To evaluate the effective recognition distance:")
296         print("1. Run this script with RUN_TEST_MODE = True.")
297         print("2. Position yourself at a known starting distance (e.
    g., 1 meter).")
298         print("3. Test whether a face, specific gesture, or both can
     be detected for the duration of the test.")
299         print("4. After the test finishes (or when 'q' is pressed),
    check the 'Detection log summary' above.")
300         print("5. Correlate the performance results with the
    physical distance tested.")
301         print("6. Analyse whether detection rates (e.g., >80-90%)
    were consistently high for the face, gestures, or both during the
     time you held that pose at that distance.")
302         print("7. Change the distance (e.g., by 0.5 meters) and
    repeat all the previous steps to analyse the performance at a
```

```
    different distance.")
303
304
305     elif RUN_TEST_MODE:
306         print("\n--- Performance test results ---")
307         print("No frames processed or test ended before completion.
    ")
308
309     picam2.stop()
310     cv2.destroyAllWindows()
311     gesture_recognizer.close()
```

Listing 2: Python script for measuring the performance of the face and gesture recognition code.

12 Appendix D - Test data

```
Distance evaluation:

0.5 meters:
[INFO] Test will run for 30 seconds.
W0000 00:00:1744713889.623998  332521 landmark_projection_calculator
    .cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only
    supported for the square ROI. Provide IMAGE_DIMENSIONS or use
    PROJECTION_MATRIX.
[INFO] Test duration completed, 30.12 seconds elapsed.

--- Performance test results ---
Test duration: 30 seconds
Total frames processed: 26
Average FPS: 0.93
Maximum FPS: 1.07
Minimum FPS: 0.55
Standard deviation FPS: 0.10

--- Detection log summary ---
Frames with face detected: 26/26 (100.0%)
Frames with gesture detected: 26/26 (100.0%)
Frames with both detected: 26/26 (100.0%)



0.6 meters:
[INFO] Test will run for 30 seconds.
W0000 00:00:1744714257.191604  339584 landmark_projection_calculator
    .cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only
    supported for the square ROI. Provide IMAGE_DIMENSIONS or use
    PROJECTION_MATRIX.
[INFO] Test duration completed, 30.13 seconds elapsed.

--- Performance test results ---
Test duration: 30 seconds
Total frames processed: 25
Average FPS: 0.89
Maximum FPS: 1.05
Minimum FPS: 0.62
Standard deviation FPS: 0.10

--- Detection log summary ---
Frames with face detected: 25/25 (100.0%)
Frames with gesture detected: 25/25 (100.0%)
Frames with both detected: 25/25 (100.0%)



0.7 meters:
[INFO] Test will run for 30 seconds.
W0000 00:00:1744714558.288278  345379 landmark_projection_calculator
    .cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only
    supported for the square ROI. Provide IMAGE_DIMENSIONS or use
    PROJECTION_MATRIX.
[INFO] Test duration completed, 30.81 seconds elapsed.

```

```
--- Performance test results ---
Test duration: 30 seconds
Total frames processed: 26
Average FPS: 0.91
Maximum FPS: 1.07
Minimum FPS: 0.56
Standard deviation FPS: 0.10

--- Detection log summary ---
Frames with face detected: 26/26 (100.0%)
Frames with gesture detected: 26/26 (100.0%)
Frames with both detected: 26/26 (100.0%)



0.8 meters:
[INFO] Test will run for 30 seconds.
W0000 00:00:1744728370.143394    8856 landmark_projection_calculator
   .cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only
   supported for the square ROI. Provide IMAGE_DIMENSIONS or use
   PROJECTION_MATRIX.
[INFO] Test duration completed, 30.31 seconds elapsed.

--- Performance test results ---
Test duration: 30 seconds
Total frames processed: 25
Average FPS: 1.02
Maximum FPS: 1.16
Minimum FPS: 0.61
Standard deviation FPS: 0.13

--- Detection log summary ---
Frames with face detected: 25/25 (100.0%)
Frames with gesture detected: 25/25 (100.0%)
Frames with both detected: 25/25 (100.0%)



0.9 meters:
[INFO] Test will run for 30 seconds.
W0000 00:00:1744728681.500383   14984 landmark_projection_calculator
   .cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only
   supported for the square ROI. Provide IMAGE_DIMENSIONS or use
   PROJECTION_MATRIX.
[INFO] Test duration completed, 30.52 seconds elapsed.

--- Performance test results ---
Test duration: 30 seconds
Total frames processed: 29
Average FPS: 1.01
Maximum FPS: 1.10
Minimum FPS: 0.73
Standard deviation FPS: 0.06

--- Detection log summary ---
Frames with face detected: 29/29 (100.0%)
Frames with gesture detected: 29/29 (100.0%)
Frames with both detected: 29/29 (100.0%)

```

```
101
102
103 1.0 meters:
104 [INFO] Test will run for 30 seconds.
105 W0000 00:00:1744729052.197748   22105 landmark_projection_calculator
       .cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only
       supported for the square ROI. Provide IMAGE_DIMENSIONS or use
       PROJECTION_MATRIX.
106 [INFO] Test duration completed, 30.64 seconds elapsed.
107
108 --- Performance test results ---
109 Test duration: 30 seconds
110 Total frames processed: 29
111 Average FPS: 1.01
112 Maximum FPS: 1.06
113 Minimum FPS: 0.65
114 Standard deviation FPS: 0.07
115
116 --- Detection log summary ---
117 Frames with face detected: 29/29 (100.0%)
118 Frames with gesture detected: 29/29 (100.0%)
119 Frames with both detected: 29/29 (100.0%)
120
121
122
123 1.1 meters:
124 [INFO] Test will run for 30 seconds.
125 W0000 00:00:1744729263.465845   26201 landmark_projection_calculator
       .cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only
       supported for the square ROI. Provide IMAGE_DIMENSIONS or use
       PROJECTION_MATRIX.
126 [INFO] Test duration completed, 30.78 seconds elapsed.
127
128 --- Performance test results ---
129 Test duration: 30 seconds
130 Total frames processed: 29
131 Average FPS: 1.00
132 Maximum FPS: 1.05
133 Minimum FPS: 0.73
134 Standard deviation FPS: 0.06
135
136 --- Detection log summary ---
137 Frames with face detected: 29/29 (100.0%)
138 Frames with gesture detected: 29/29 (100.0%)
139 Frames with both detected: 29/29 (100.0%)
140
141
142
143 1.2 meters:
144 [INFO] Test will run for 30 seconds.
145 W0000 00:00:1744729447.647382   29800 landmark_projection_calculator
       .cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only
       supported for the square ROI. Provide IMAGE_DIMENSIONS or use
       PROJECTION_MATRIX.
146 [INFO] Test duration completed, 30.79 seconds elapsed.
147
148 --- Performance test results ---
149 Test duration: 30 seconds
150 Total frames processed: 30
```

```
Average FPS: 1.02
Maximum FPS: 1.06
Minimum FPS: 0.71
Standard deviation FPS: 0.07

--- Detection log summary ---
Frames with face detected: 30/30 (100.0%)
Frames with gesture detected: 30/30 (100.0%)
Frames with both detected: 30/30 (100.0%)



1.3 meters:
[INFO] Test will run for 30 seconds.
W0000 00:00:1744729757.860114   35798 landmark_projection_calculator
   .cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only
   supported for the square ROI. Provide IMAGE_DIMENSIONS or use
   PROJECTION_MATRIX.
[INFO] Test duration completed, 30.42 seconds elapsed.

--- Performance test results ---
Test duration: 30 seconds
Total frames processed: 41
Average FPS: 1.59
Maximum FPS: 2.07
Minimum FPS: 0.70
Standard deviation FPS: 0.50

--- Detection log summary ---
Frames with face detected: 17/41 (41.5%)
Frames with gesture detected: 41/41 (100.0%)
Frames with both detected: 17/41 (41.5%)



1.4 meters:
[INFO] Test will run for 30 seconds.
W0000 00:00:1744730170.554396   43736 landmark_projection_calculator
   .cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only
   supported for the square ROI. Provide IMAGE_DIMENSIONS or use
   PROJECTION_MATRIX.
[INFO] Test duration completed, 30.13 seconds elapsed.

--- Performance test results ---
Test duration: 30 seconds
Total frames processed: 57
Average FPS: 2.14
Maximum FPS: 2.56
Minimum FPS: 0.96
Standard deviation FPS: 0.37

--- Detection log summary ---
Frames with face detected: 2/57 (3.5%)
Frames with gesture detected: 57/57 (100.0%)
Frames with both detected: 2/57 (3.5%)



1.5 meters:
```

```
[INFO] Test will run for 30 seconds.
W0000 00:00:1744730598.111163   51953 landmark_projection_calculator
    .cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only
    supported for the square ROI. Provide IMAGE_DIMENSIONS or use
    PROJECTION_MATRIX.
[INFO] Test duration completed, 30.14 seconds elapsed.

--- Performance test results ---
Test duration: 30 seconds
Total frames processed: 58
Average FPS: 2.15
Maximum FPS: 2.54
Minimum FPS: 1.45
Standard deviation FPS: 0.27

--- Detection log summary ---
Frames with face detected: 0/58 (0.0%)
Frames with gesture detected: 55/58 (94.8%)
Frames with both detected: 0/58 (0.0%)



1.6 meters:
[INFO] Test will run for 30 seconds.
W0000 00:00:1744704587.002192  155193 landmark_projection_calculator
    .cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only
    supported for the square ROI. Provide IMAGE_DIMENSIONS or use
    PROJECTION_MATRIX.
[INFO] Test duration completed, 30.25 seconds elapsed.

--- Performance test results ---
Test duration: 30 seconds
Total frames processed: 69
Average FPS: 2.58
Maximum FPS: 3.39
Minimum FPS: 0.97
Standard deviation FPS: 0.44

--- Detection log summary ---
Frames with face detected: 0/69 (0.0%)
Frames with gesture detected: 52/69 (75.4%)
Frames with both detected: 0/69 (0.0%)


1.7 meters:
[INFO] Test will run for 30 seconds.
W0000 00:00:1744705228.959289  167481 landmark_projection_calculator
    .cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only
    supported for the square ROI. Provide IMAGE_DIMENSIONS or use
    PROJECTION_MATRIX.
[INFO] Test duration completed, 30.26 seconds elapsed.

--- Performance test results ---
Test duration: 30 seconds
Total frames processed: 75
Average FPS: 2.83
Maximum FPS: 3.37
Minimum FPS: 1.58
Standard deviation FPS: 0.48
```

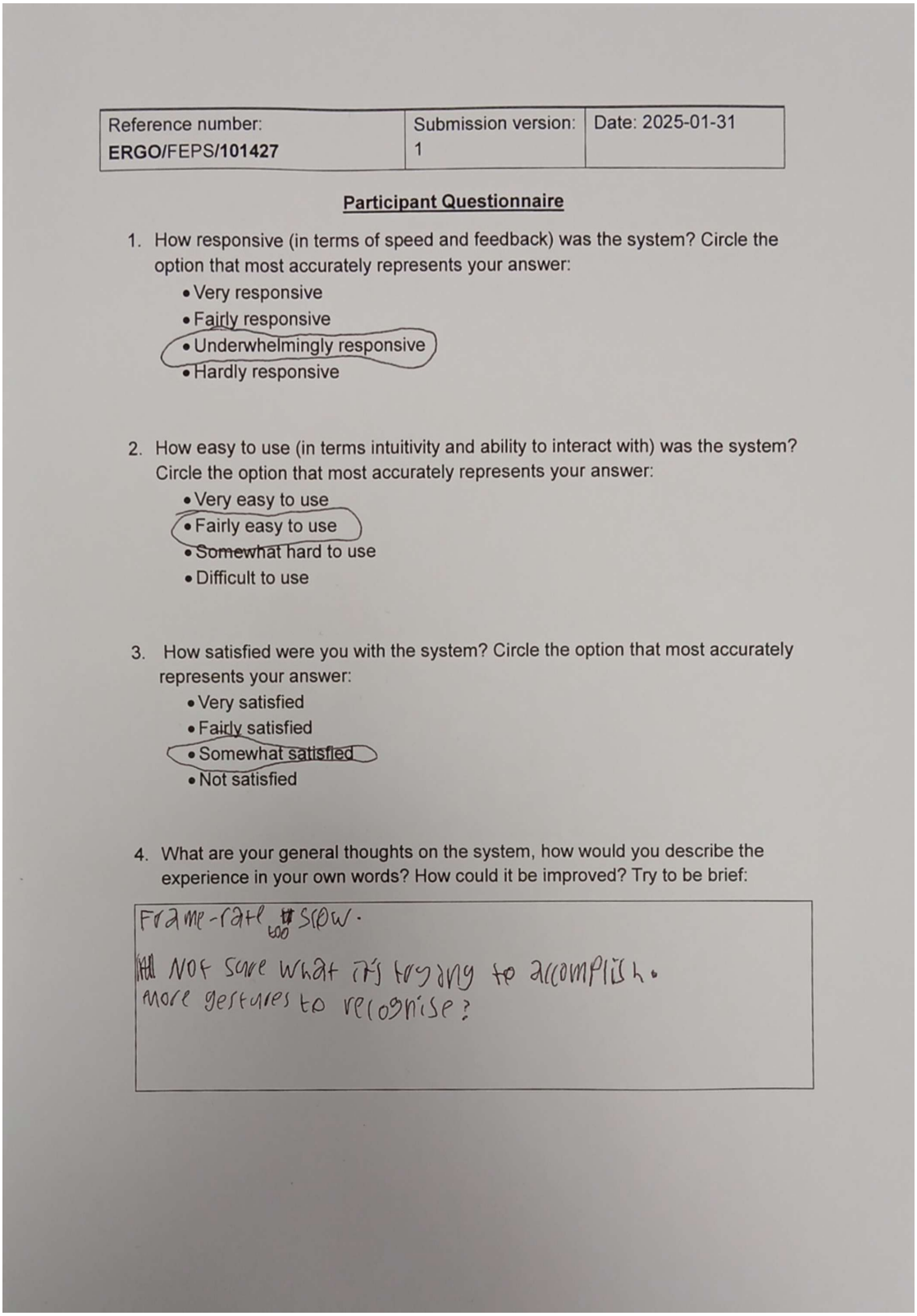
```
254
255 --- Detection log summary ---
256 Frames with face detected: 0/75 (0.0%)
257 Frames with gesture detected: 32/75 (42.7%)
258 Frames with both detected: 0/75 (0.0%)
259
260
261
262 1.8 meters:
263 [INFO] Test will run for 30 seconds.
264 W0000 00:00:1744706017.824568  182607 landmark_projection_calculator
       .cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only
       supported for the square ROI. Provide IMAGE_DIMENSIONS or use
       PROJECTION_MATRIX.
265 [INFO] Test duration completed, 30.18 seconds elapsed.
266
267 --- Performance test results ---
268 Test duration: 30 seconds
269 Total frames processed: 82
270 Average FPS: 3.14
271 Maximum FPS: 3.36
272 Minimum FPS: 0.91
273 Standard deviation FPS: 0.42
274
275 --- Detection log summary ---
276 Frames with face detected: 0/82 (0.0%)
277 Frames with gesture detected: 7/82 (8.5%)
278 Frames with both detected: 0/82 (0.0%)
279
280
281
282 1.9 meters:
283 [INFO] Test will run for 30 seconds.
284 W0000 00:00:1744706231.803641  186736 landmark_projection_calculator
       .cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only
       supported for the square ROI. Provide IMAGE_DIMENSIONS or use
       PROJECTION_MATRIX.
285 [INFO] Test duration completed, 30.24 seconds elapsed.
286
287 --- Performance test results ---
288 Test duration: 30 seconds
289 Total frames processed: 85
290 Average FPS: 3.19
291 Maximum FPS: 3.40
292 Minimum FPS: 1.40
293 Standard deviation FPS: 0.33
294
295 --- Detection log summary ---
296 Frames with face detected: 0/85 (0.0%)
297 Frames with gesture detected: 0/85 (0.0%)
298 Frames with both detected: 0/85 (0.0%)
299
300
301 2.0 meters:
302 [INFO] Test will run for 30 seconds.
303 W0000 00:00:1744710924.343150  275874 landmark_projection_calculator
       .cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only
       supported for the square ROI. Provide IMAGE_DIMENSIONS or use
       PROJECTION_MATRIX.
```

```
[INFO] Test duration completed, 30.28 seconds elapsed.

--- Performance test results ---
Test duration: 30 seconds
Total frames processed: 84
Average FPS: 3.16
Maximum FPS: 3.37
Minimum FPS: 1.29
Standard deviation FPS: 0.35

--- Detection log summary ---
Frames with face detected: 0/84 (0.0%)
Frames with gesture detected: 2/84 (2.4%)
Frames with both detected: 0/84 (0.0%)



2.1 meters:
[INFO] Test will run for 30 seconds.
W0000 00:00:1744711672.173986  290213 landmark_projection_calculator
    .cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only
    supported for the square ROI. Provide IMAGE_DIMENSIONS or use
    PROJECTION_MATRIX.
[INFO] Test duration completed, 30.00 seconds elapsed.

--- Performance test results ---
Test duration: 30 seconds
Total frames processed: 84
Average FPS: 3.12
Maximum FPS: 3.26
Minimum FPS: 1.25
Standard deviation FPS: 0.28

--- Detection log summary ---
Frames with face detected: 0/84 (0.0%)
Frames with gesture detected: 0/84 (0.0%)
Frames with both detected: 0/84 (0.0%)



2.2 meters:
[INFO] Test will run for 30 seconds.
W0000 00:00:1744713472.810298  324528 landmark_projection_calculator
    .cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only
    supported for the square ROI. Provide IMAGE_DIMENSIONS or use
    PROJECTION_MATRIX.
[INFO] Test duration completed, 30.28 seconds elapsed.

--- Performance test results ---
Test duration: 30 seconds
Total frames processed: 86
Average FPS: 3.20
Maximum FPS: 3.39
Minimum FPS: 1.11
Standard deviation FPS: 0.30

--- Detection log summary ---
Frames with face detected: 0/86 (0.0%)
Frames with gesture detected: 0/86 (0.0%)
```

```
Frames with both detected: 0/86 (0.0%)
```

Listing 3: Results obtained from running the test version of the code in Appendix C over varying distances.

13 Appendix E - Participant feedback



Reference number: **ERGO/FEPS/101427**	Submission version: 1	Date: 2025-01-31

Participant Questionnaire

1. How responsive (in terms of speed and feedback) was the system? Circle the option that most accurately represents your answer:
   - Very responsive
   - Fairly responsive
   - Underwhelmingly responsive
   - Hardly responsive

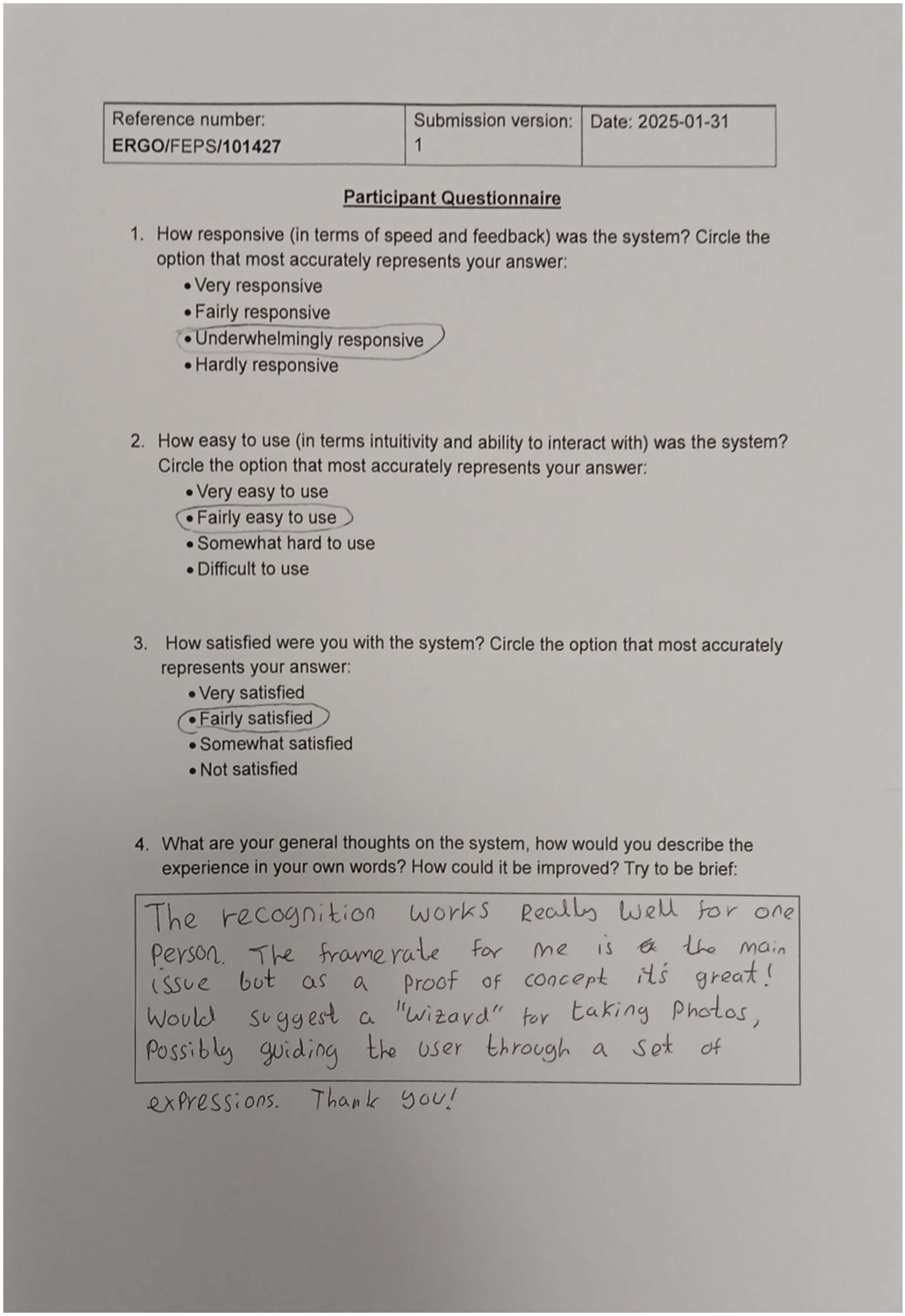
2. How easy to use (in terms intuitivity and ability to interact with) was the system? Circle the option that most accurately represents your answer:
   - Very easy to use
   - Fairly easy to use
   - Somewhat hard to use
   - Difficult to use

3. How satisfied were you with the system? Circle the option that most accurately represents your answer:
   - Very satisfied
   - Fairly satisfied
   - Somewhat satisfied
   - Not satisfied

4. What are your general thoughts on the system, how would you describe the experience in your own words? How could it be improved? Try to be brief:

Frame-rate too slow.
Not sure what it's trying to accomplish.
More gestures to recognise?

Figure 16: Participant feedback questionnaire - Page 1 of 5.



Reference number: ERGO/FEPS/101427	Submission version: 1	Date: 2025-01-31

Participant Questionnaire

1. How responsive (in terms of speed and feedback) was the system? Circle the option that most accurately represents your answer:
   - Very responsive
   - Fairly responsive
   - Underwhelmingly responsive
   - Hardly responsive

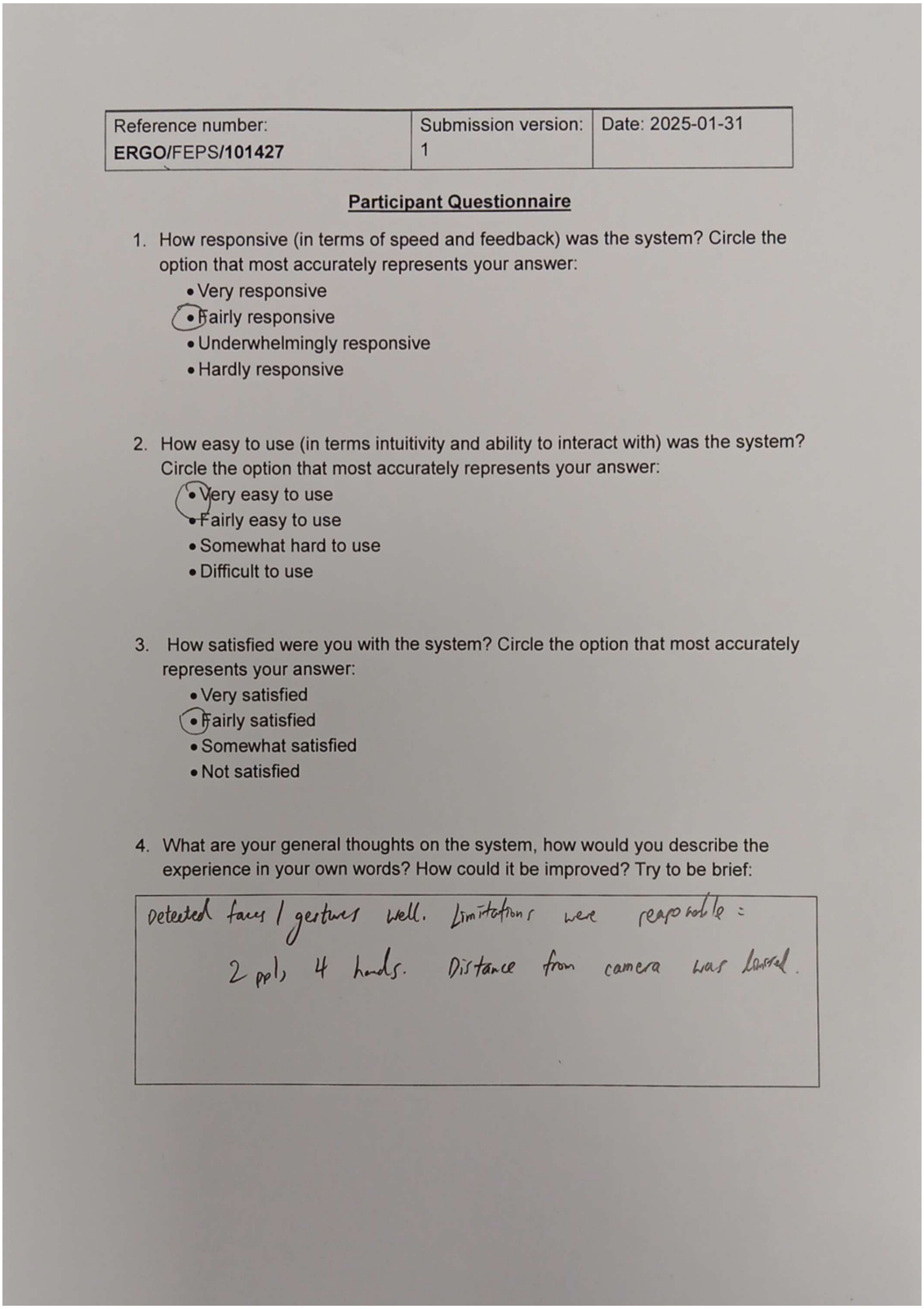
2. How easy to use (in terms intuitivity and ability to interact with) was the system? Circle the option that most accurately represents your answer:
   - Very easy to use
   - Fairly easy to use
   - Somewhat hard to use
   - Difficult to use

3. How satisfied were you with the system? Circle the option that most accurately represents your answer:
   - Very satisfied
   - Fairly satisfied
   - Somewhat satisfied
   - Not satisfied

4. What are your general thoughts on the system, how would you describe the experience in your own words? How could it be improved? Try to be brief:

The recognition works really well for one person. The framerate for me is ~~a~~ the main issue but as a proof of concept it's great! Would suggest a "wizard" for taking photos, possibly guiding the user through a set of expressions. Thank you!

Figure 17: Participant feedback questionnaire - Page 2 of 5.



Reference number: **ERGO/FEPS/101427**	Submission version: 1	Date: 2025-01-31

Participant Questionnaire

1. How responsive (in terms of speed and feedback) was the system? Circle the option that most accurately represents your answer:
   - Very responsive
   - Fairly responsive
   - Underwhelmingly responsive
   - Hardly responsive

2. How easy to use (in terms intuitivity and ability to interact with) was the system? Circle the option that most accurately represents your answer:
   - Very easy to use
   - Fairly easy to use
   - Somewhat hard to use
   - Difficult to use

3. How satisfied were you with the system? Circle the option that most accurately represents your answer:
   - Very satisfied
   - Fairly satisfied
   - Somewhat satisfied
   - Not satisfied

4. What are your general thoughts on the system, how would you describe the experience in your own words? How could it be improved? Try to be brief:

Detected faces / gestures well. Limitations were reasonable: 2 ppl, 4 hands. Distance from camera was limited.

Figure 18: Participant feedback questionnaire - Page 3 of 5.

Reference number: ERGO/FEPS/101427	Submission version: 1	Date: 2025-01-31

Participant Questionnaire

1. How responsive (in terms of speed and feedback) was the system? Circle the option that most accurately represents your answer:
   - Very responsive
   - (•) Fairly responsive
   - Underwhelmingly responsive
   - Hardly responsive

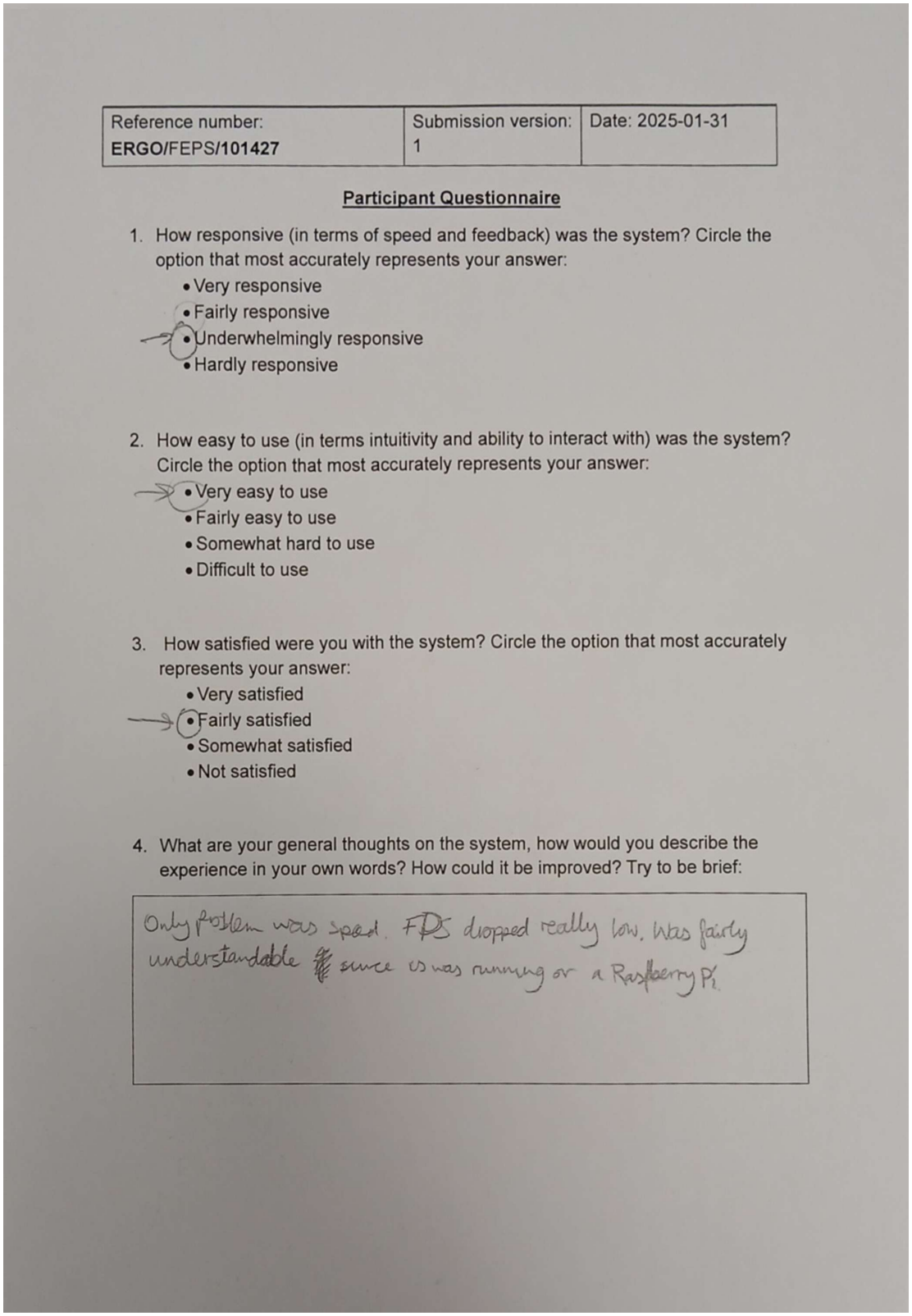
2. How easy to use (in terms intuitivity and ability to interact with) was the system? Circle the option that most accurately represents your answer:
   - (•) Very easy to use
   - Fairly easy to use
   - Somewhat hard to use
   - Difficult to use

3. How satisfied were you with the system? Circle the option that most accurately represents your answer:
   - Very satisfied
   - (•) Fairly satisfied
   - Somewhat satisfied
   - Not satisfied

4. What are your general thoughts on the system, how would you describe the experience in your own words? How could it be improved? Try to be brief:

Accurately detected face and hand gestures. Less accurate when distance from camera sensor is increased.

Figure 19: Participant feedback questionnaire - Page 4 of 5.



Reference number: ERGO/FEPS/101427	Submission version: 1	Date: 2025-01-31

Participant Questionnaire

1. How responsive (in terms of speed and feedback) was the system? Circle the option that most accurately represents your answer:
   - Very responsive
   - Fairly responsive
   - Underwhelmingly responsive
   - Hardly responsive

2. How easy to use (in terms intuitivity and ability to interact with) was the system? Circle the option that most accurately represents your answer:
   - Very easy to use
   - Fairly easy to use
   - Somewhat hard to use
   - Difficult to use

3. How satisfied were you with the system? Circle the option that most accurately represents your answer:
   - Very satisfied
   - Fairly satisfied
   - Somewhat satisfied
   - Not satisfied

4. What are your general thoughts on the system, how would you describe the experience in your own words? How could it be improved? Try to be brief:

Only problem was speed. FPS dropped really low. Was fairly understandable since is was running on a Raspberry Pi.

Figure 20: Participant feedback questionnaire - Page 5 of 5.

14 Appendix F - Design and data archive

The following details the contents of the submitted design and data archive:

- ”Ethics_Submission_2”, a collection of documents submitted to gain ethics approval for the experiment; this includes the application form, participant information sheet, questionnaires for participants and other required documents.
- ”ExperimentRecords”, containing results obtained from the experiment; this includes signed participant consent forms, questionnaires with their feedback, and personal notes taken during the experiment.
- ”RaspberryPiSDBackup”, an early SD card backup of the Raspberry Pi after basic setup work has been done, e.g. required libraries being downloaded.
- ”RaspberryPiSDBackup_1”, a backup after individual recognition tasks have been configured.
- ”RaspberryPiSDBackup_2”, a backup after the combined recognition task has been completed.
- ”RaspberryPiSDBackup_3(TEST_VERSION)”, a backup after the test code has been completed and run.
- ”face_and_gesture_rec_code”, a folder containing the code, datasets, and related files for the combined recognition task.
- ”face_rec_code”, a folder containing the code, datasets, and related files for the facial recognition task.
- ”gesture_rec_code”, a folder containing the code, datasets, and related files for the original gesture recognition task.
- ”google_gesture_rec_code”, a folder containing the code, datasets, and related files for the Google version of the gesture recognition task.
- ”TEST_face_and_gesture_rec_code”, a folder containing the code, datasets, and related files for the test version of the combined recognition task.

15 Appendix G - Hardware costs

Supplier	Quantity	Stock code	Description	Price in GBP (£)	Total price in GBP (£)
Mouser Electronics	8	358-SC1628	Memory Cards A2 class micro-SD cards	34.2386	110.618
Mouser Electronics	1	358-SC0872	Raspberry Pi Camera Module 3	23.712	
Mouser Electronics	1	485-2143	Flex Cable for Raspberry Pi Camera or Display - 1 meter	3.744	
Mouser Electronics	1	358-SC0166	RPi-KYB UK (Red/White)	16.128	
Mouser Electronics	1	358-SC0442	RPi-Mouse (Red/White) - UMEC01531	7.584	
Mouser Electronics	1	545-U452-000-SD-A	USB C/USB A SD/MSD CARD READER	25.212	

Figure 21: Cost breakdown of items purchased from the project budget.