UNIVERSITAT POLITÈCNICA DE CATALUNYA

Barcelona East School of Engineering

Master's degree in Interdisciplinary and Innovative Engineering

Design and Development of a Telerehabilitation app

Author: Xiya Tao
Director: TORNER RIBE, JORDI
Co-Director: SERRANCOLÍ MASFERRER, GIL
Call: Fall 2022

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

Abstract

Background

Neuromusculoskeletal injuries are a common condition and after medication and surgery, most patients still suffer from physical deficits and even psychological disorders due to the lack of scientific and effective rehabilitation. However, the high cost of rehabilitation makes it impossible for many patients to receive a complete and scientific post-operative rehabilitation, so self-training at home can provide the opportunity for most patients to receive good treatment.

Objective

In order to enable patients to rehabilitate themselves at home, a new solution is proposed for telemedicine, where patients can play a serious game of simple rehabilitation with just an ordinary computer with a camera. Apart from the development phase, no further involvement of occupational therapists is required, significantly reducing the cost and complexity of rehabilitation.

Methods

This study develops a simple serious game for playing the piano based on OpenCV, MediaPipe and Unity. The game is developed with the user in mind, recording game data to facilitate the analysis and processing of user data and recording each user's progress, making the game much more beneficial and applicable.

Results

This study saves all results on the web, allows simple rehabilitation using only a computer, records patient progress saves observed behavioral information and expressions, receives positive feedback during user use and inferred from the study that this is indeed a viable solution.

Conclusion

The combined results suggest that serious game as a rehabilitation treatment is a viable solution, not only as an excellent rehabilitation solution for those who are financially disadvantaged, but also as a good complementary training for those who can afford expensive rehabilitation treatment. Overall, Serious Play can prove beneficial to rehabilitation by providing measurable data on each session and quantifying the patient's training.

Acknowledgments

I would like to take this opportunity to thank the many people who have provided me with invaluable assistance in writing this thesis.

My greatest thanks go first and foremost to Prof. TORNER RIBE and Prof. SERRANCOLÍ MASFERRER, my supervisors, who have guided me through the writing phase and research phase of this thesis. Without their enlightening guidance tips and encouragement, this thesis would not have reached its current stage.

Secondly, I would like to express my sincere gratitude to all my professors and teachers, without whom I would not have learnt the wealth of writing and research skills I have learned during my two years of postgraduate study, without their classroom paving.

I am also very grateful for the help provided by the authors and scholars mentioned in the bibliography, without whose work the literature review of my thesis would not have been possible.

Finally, I am deeply grateful to my dear parents, friends and classmates who have been supportive, brainstormed about my thesis, provided valuable insights, and were willing to help me with research tests. Their help and support have been with me through the difficult journey of this thesis and every moment of my life.

INDEX

1. Introduction

Neuromusculoskeletal injuries are a prevalent condition, ranging from neonatal injuries, brachial plexus strains in the gym, impact injuries in combat sports, accidental trauma, or strokes due to hypertension, hyperglycemia, and hyperlipidemia, all of which can cause damage to the brain, neuronal pathways and/or muscle tissues. With the development of technology and society, the cure rate for neuromusculoskeletal injuries is increasing, but after medication and surgery, most patients are still unable to return to their previous work and life due to various after-effects and psychological reasons. In the case of stroke, for example, it is estimated that up to 1/3 of stroke patients often suffer from depression, with a cumulative incidence of 55%[1]. However, the effects of depression can be greatly reduced or even eliminated if, in time, the patient starts a scientific and effective rehabilitation program after surgery.

Scientific and effective rehabilitation is particularly important in the post-operative phase for patients with neurological conditions. In many rehabilitation programs, high-definition, high-resolution cameras and various electronic devices are used to capture human body motion or muscle activation processed from the electrical activity of muscles (electromyography - EMG) at rest or during contraction to achieve more accurate results, e.g., Training of Hand Rehabilitation Using Low-Cost Exoskeleton and Vision-Based Game Interface [2] . In 2021, a wearable multimodal system based on force myography, electromyography, and inertial sensing, and two associated serious games, were introduced for stroke rehabilitation of twelve hand movements related to activities of daily living and the Fugl Meyer assessment [3] . Another example is to describe an iterative design process for developing an escape room-based serious game as a boundary object [4]. Besides, a home application, Muvity, contains games that effectively exercise and rehabilitate the shoulders, elbows and lower back of stroke victims at home [5].

The degree of depression in post-stroke depression (PSD) is highly related to financial status and medical costs. However, not everyone can afford expensive hospital-based post-operative rehabilitation, such as brain-computer interface (BCI) systems that require extensive equipment and specialist staff, nor does everyone have access to specialist laboratory rehabilitation programs. When the argument that self-training of patients at home through remote serious play can be extremely helpful in their recovery was confirmed remotely supervised self-training, which can be carried out at home, is the best option for a large number of patients [5] . Remotely supervised self-training at home can significantly reduce the cost of rehabilitation and, because it is readily available, can significantly reduce the time required for the rehabilitation process.

Most digital rehabilitation requires the human skeleton detection. Accurate real-time detection better reflects the patient's rehabilitation progress and task completion. In order to achieve accurate real-time testing, posture estimation processing is essential. The Open-Source Computer Vision Library (OpenCV), introduced by Intel in 2000, provides real-time image processing, computer vision, human recognition, and other capabilities. There are already many open-source libraries based on OpenCV, such as the OpenPose human pose recognition project[6] at Carnegie Mellon University (CMU), which is based on deep neural networks (DNN), convolutional neural networks (CNN), and supervised learning method together with Caffe framework. In 2018, Shanghai Jiao Tong University also proposed a lighter and more accurate multi-person pose estimation system, AlphaPose, which is based on You Only Look Once (YOLO) v3-spp pedestrian detection, pose key point detection, and pedestrian re-identification algorithms.

There is another open-source framework from Google, which is called MediaPipe and known for its powerful keypoint detection, face and gesture recognition, and that supports many common AI functions. MediaPipe hands is an official Google trained and open-source tool for gesture detection that can be called simply to get good results. is able to detect 21 joints in a person's hand, draw them in 3D, return the drawn bones to the location in the detected video and label them in the image, and by calculating the distance and depth between the points a simple gesture discrimination can be achieved.

Objectives

This study presents a serious hand rehabilitation demo based on OpenCV and MediaPipe[7] for hand detection, with this demo, the project expected to achieve: the patient's improved manual dexterity and improved post-operative mental health by playing a virtual piano through a regular webcam. The patient's family or therapist can use the recorded data to analyze the improvement in the patient's manual dexterity and thus adjust the patient's treatment plan and playtime. The user's facial expressions are analyzed and the results are used to adjust the duration and frequency of play. During the testing phase of this game, people of different ages and genders will be chosen to test the game. Participants will have different affinities for the video game, but due to the simplicity of the game, participants will be able to complete the game. In the user evaluation, an attempt will be made to evaluate the game individually on a one-to-one basis to make the opinions obtained more accurate.

The following efforts have been made to achieve the objectives.
1. Compare currently available algorithms to accomplish a system for hand recognition and calculation of joint angles
2. Develop a serious game aimed at restoring manual dexterity in patients with impaired manual dexterity and hand movement deficits due to neuromusculoskeletal injuries
3. Use the data obtained from the game and joint angle detection as feedback for the continuous development of new versions of the game

The rest of the report is structured as follows. In section 2, previous work on serious games and hand detection is described, in section 3 the reasons for the choice of the hand-detection algorithm, the development process, and the methodology for the development of the game are explained. The results of this research are clearly presented in section 4, some limitations of this research and ways to improve it are described in section 5, the future work is described in section 6. The report concludes with a summary of this project (section 7).

2. State-of-the-art

The term "serious games" was first coined in the 1970s by Clark Abt in his book 《*Serious Games*》[8], which was also the first to define "serious games": "Serious games are neither games nor serious, they are both. "

The first time that serious game solutions for rehabilitation were explicitly proposed by International Conference on Computer & Information Science (ICCIS)[9], where they presented a new framework for games to increase patients' motivation for treatment and to investigate the feasibility and effectiveness of a new game-based technology to support hand and leg rehabilitation. The final system was not tested on patients, but they showed that if patients do exercises in the direction and at the right angle between body movements properly, patients can recover their potential early. Innovative gaming techniques will provide more effective results and change movement therapy's direction.

With the advent of Leap Motion (Leap Motion Inc, San Francisco, CA, USA), home rehabilitation goes even further. A game that allows patients to perform real-world tasks has been developed [10], such as washing dishes using Leap Motion, which will see their 3D avatar performing the same task on-screen synchronously. Their findings suggest that while satisfying the primary purpose of motor rehabilitation, several entertaining and interactive programs designed to suit the patient's living environment will successfully shift the patient's attention from physical work to a fun challenge that becomes exciting.

Kinect has also played an important role in promoting home rehabilitation. Galna established the accuracy of the Kinect in measuring clinically relevant movements in people with Parkinson's Disease[11]. And a review summarized the most current avenues of research into Kinect-based elderly care and stroke rehabilitation systems, which provides the limitations, and issues of concern as well as suggestions for the promising future work in this direction[12].

Previous work on hand pose estimation considered the RGB information, e.g., Rehg and Kanade explored the method of developing vision-based human-computer interaction (HCI) applications[13]. But Due to a controlled environment is need for recognizing the contour edges, skin tones, and shadows, the movements could only be simple due to the minimal detection results of the technology available at the time. The use of colored gloves removed some of these detection limitations, allowing results to be presented with excellent accuracy[14]. This became

the basis for NIMBLE VR (A hand-tracking company founded in 2012 by Rob Wang, Chris Twigg, and Kenrick Kin).

But with continued research, Google Research developed an open-sourced MediaPipe, a multimedia machine learning model application framework that can be used for hand recognition. This has led to much higher detection accuracy for gesture estimation.

3. Materials and Methods

3.1. Framework for project design

In order to define the basic framework of the project, the methodology for its design first needs to be determined. As described in the related work section, the previous literature covers different methodological techniques used for the serious gaming hand therapy process, including using platforms such as Leap Motion and Kinect. This has helped this study to select more suitable algorithms, development methods, and ideas based on previous experience.

At the very beginning of the project, Agile development was chosen, which is a human-centered, step-by-step development approach. The project was first to cut into three sub-project parts: hand recognition, data transfer, and Unity build. The hand recognition and Unity build were linked by the data transfer project, but could also be run independently as three small projects, working in short iteration cycles, as the test population was simple and the project could be selected over several iterations to produce the best results. Although the initial iterations produce imperfect results and many features are not implemented, with constant updates to the requirements analysis, framework design, code rebuilding, and re-testing, the product will continue to improve and will be closer to customer needs in terms of functionality and quality. Constantly improving versions of the project in testing, one can to focus more on the user experience, analyze the risks precisely, and constantly check and adjust the design as it goes along. Agile development is the most suitable development method for this project as this project was carried out by only one person and there are no communication costs.

The most important aspect of the gesture recognition sub-project was the choice of the algorithm. In the early stages of this project, the focus was developing a multi-view guidance method that follows the convolutional pose machine architecture [15]. However, this did not easily allow for its use in the field of real-time detection, and this project focused more on real-time monitoring of data transmission. More attention was given to AlphaPose, MediaPipe, OpenPose and Openpifpaf. After the first iteration, MediaPipe gave extremely good results and Openpifpaf's experiments were removed due to its failure to establish hand keypoints, despite good detection of body parts. After training both the AlphaPose and OpenPose algorithms, AlphaPose's hand presented only two-point detection in all cases, failing to detect finger keypoints. Taking these considerations into account, MediaPipe and Openpose were selected for comparison and one of them was chosen for the development of the first part detection for this project.

For the data transfer sub-project, python was used to process the data and monitor the hand in real time, output the 3D coordinates of the first key point, and send the coordinates code to the client. UDP (User Data Protocol) was chosen over TCP (Transmission Control Protocol) as the data transfer socket. This is because UDP does not require the maintenance of complex link-state tables, the establishment of connections for data transfer, or the maintenance of connection states. Also, when using the UDP protocol, the speed of operation is limited only by the speed of the data generated by the application, the capacity of the computer, and the bandwidth of the transmission, and it only needs to grab data when it needs to be transmitted, without wasting resources.

For the Unity build sub-project, only a simple piano playing project has been implemented in this project for the time being. The virtual piano is played by lifting and landing the fingers. It will be explained in detail in Section 4.

3.2. Hand recognition algorithm

3.2.1. Hand Keypoint Detection using Mediapipe and OpenPose

MediaPipe is an open-source system from Google (Mountain View, California, USA), it was developed in 2019 [7], and MediaPipe Hands is a high-fidelity palm and finger tracking solution. MediaPipe has good real-time performance and the collaboration of multiple models in a machine learning pipeline makes detection more accurate and smoother, as well as significantly reducing the need for data augmentation.

MediaPipe's construction of the hand skeleton and the key points and their coordinates are shown in Figure 1.

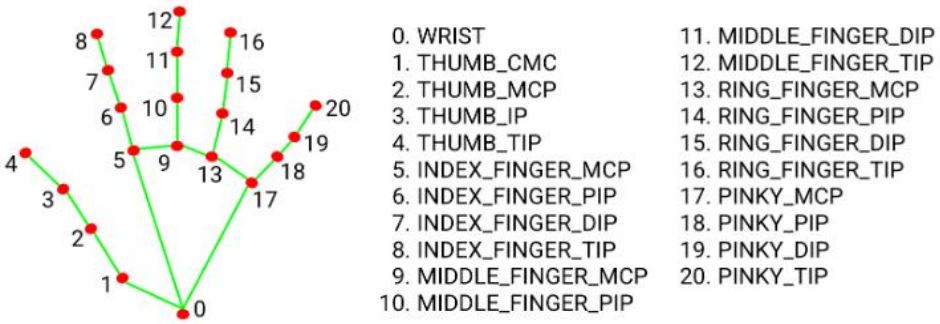


Figure 1. 21 Coordinates of hand keypoints in MediaPipe.

OpenPose [6] is a human detection system developed by CMU, which is an open-source real-time multi-person detection system that not only detects the whole body but also has independent finger detection. The independent finger detection is based on 2*21 hand keypoint recognition and the computing time depends on the number of people detected, which also means that Openpose will be our first choice if we need to detect more than two hands. It has excellent robustness and is not affected by occlusion.

OpenPose's construction of the hand skeleton and the key points and their coordinates are shown in Figure 2.

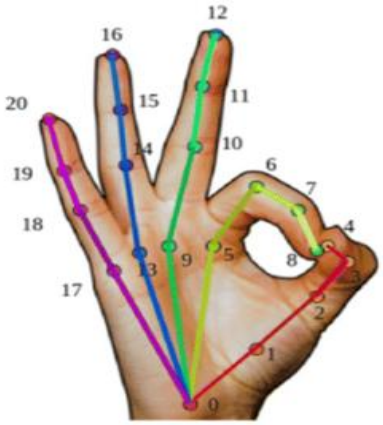


Figure 2. 21 Coordinates of hand keypoints in OpenPose.

MediaPipe and OpenPose each have their own advantages and disadvantages. MediaPipe can only be used for single person detection, whereas OpenPose can be used for multi-person detection, but MediaPipe does not require extensive downloading or configuration, whereas OpenPose requires downloading and compiling and installing *caffe* (CMU-Perceptual-Computing-Lab) before it can be used.

For this project, Visual Studio code (VScode) was chosen as the runtime environment, version 1.71.2. For MediaPipe, only VScode and python are required to build the environment, and after running, MediaPipe can be installed directly using python in VScode. To build the environment you need to download CMake (Kitware's open source cross-platform automated build system), caffe and pybind11 (a lightweight header-only library for converting interfaces between Python and C++, allowing you to create Python interface bindings for existing C++ code.) For OpenPose, you also need to download CMake, caffe and pybind11, but note that the cuDNN version must be 7.6.53 and the CUDA version must be 10.2. The above requirements are mainly because openpose does not support cuDNN version 8, and the cuda for each cuDNN is is fixed.

3.2.2. Assumptions

As this project focuses on hand recognition, all comparisons are based on hand data only and do not include the recognition of other body parts. This comparison was tested using the same video, but for the time being the video light brightness factor was not taken into account due to an insufficient data set. The chosen dataset is a sign language video [17]. This database is only used for testing MediaPipe and OpenPose and is not relevant to the build of this game.

3.2.3. Evaluation of pose-estimation algorithms

In order to compare the accuracy of MediaPipe and OpenPose, this project intercepted Chinese sign language videos and side hand videos from Fudan University [17] to conduct hypothetical experiments.

The key parameters to be compared within this project are the following:

1. Frontal hand movements
2. Lateral hand movements
3. Speed of motion detection
4. Hand overlap detection
5. kinematics analysis

The following hardware configurations were used during the tests:
Operating System: Windows 10 x64
Processor: Intel(R) Core (TM) i7-10750H CPU @ 2.60GHz 2.59 GHz
Graphics Cards: NVIDIA GeForce RTX 2070 GDDR6 8GB (256 bits)

3.3. Participants

The participants needed for this study are the end users of the application. The end-users were required to interpret user requirements and test the prototype. The researcher designs the model by understanding the user requirements upfront, and after receiving feedback from the end-users needs to collect feedback, improve the model, and subsequent technical support and updates.

For end users, this category should include both rehabilitation therapists and patients. In a clinical study, patients should be screened for previous brain injury detection and needs for manual therapy, with basic equipment to use the application (mobile phone or computer with camera). Rehabilitation therapists, on the other hand, need to have basic clinical knowledge and experience in discriminating. This study did not require screening of patients for age, or gender because of its non-differential treatment. However, due to a lack of sample, this study was only tested with the participation of four unimpaired people, the four participants were aged 12, 24, 25, and 45 years old.

3.4. Design and Development

3.4.1 Design elements

No external devices such as sensors are required to perform the test. Because it is suitable for all patients following neuromusculoskeletal injury, direct hand detection using the camera is more widely available and accessible to more people than wearable detection interactions. Patients could not wear the device due to pain or joint deficiencies. The cost of unassisted interaction would be much lower.

The hardware is portable and small. One of the goals of the study is to make rehabilitation more universal, and the fact that it can be used with only a mobile phone allows the user to use the software anywhere, anytime, and speeds up the rehabilitation process.

Low cost. Patients do not need to purchase additional equipment and only need to use a computer, which makes the program accessible to many people of modest means.

It provides entertainment, reduces the user's resistance to rehabilitation, and reduces post-operative depression.

Pianist game description.

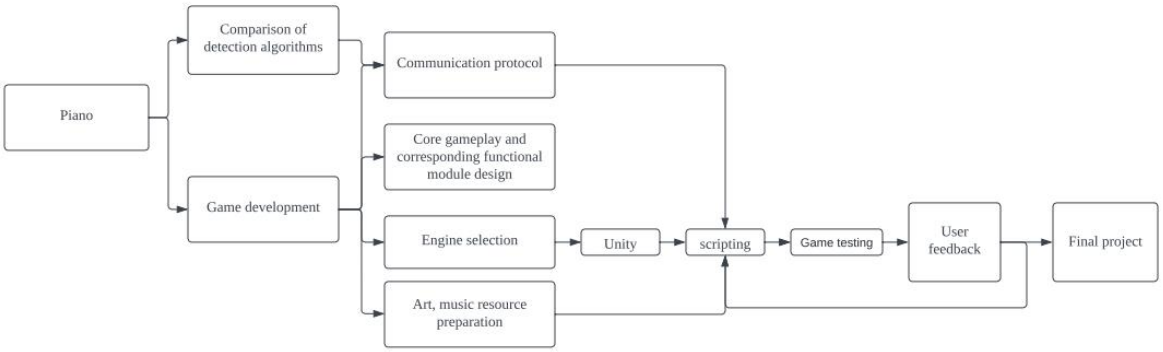


Figure 3. Workflow of the experiment design

Game objective: to complete the piano repertoire and record self-composed pieces

Game controls: The patient has to move each finger to touch the piano keys to achieve the result of completing the repertoire. A skeletal model of the patient's hand is shown in the game, which provides more visual feedback about the hand.

Game data: The hand data is imported into unity via UDRreceive and the data is saved.

Game implementation methods: The in-game piano uses a simple trigger class response. When the red dot representing the finger touches a piano key (white or black), the corresponding sound effect bound to the corresponding key is played.

3.4.2 Assessment methods

During the testing phase of the prototype, participants were taught simple scales and songs. The piece chosen was *little star* played with one hand.

When testing the prototype, we recorded the user's gameplay, facial expressions, and hand angles. The user's performance is recorded automatically during the game, the facial expressions are recorded using a separate camera and the hand angles are recorded automatically by python code. The duration of each test was set to 10 minutes in order to avoid user fatigue and boredom. [18] did pioneering work on modern facial expression recognition by studying the six basic human expressions (i.e., happiness, sadness, surprise, fear, anger, and disgust), identifying the categories of objects to be recognized, and systematically building a database of thousands of images of different facial expressions, meticulously. The database of thousands of different facial expressions was systematically created, describing in detail how each expression corresponds to a change in the face, including eyebrows, eyes, eyelids, lips and so on. This has helped immensely in the emergence of face recognition systems. Samsung AI [19] , in collaboration with

Imperial College London, then developed an AI program that tracks eight facial expressions and can determine whether a subject is in a negative or positive mood and how they are fluctuating.

For the simplicity of the project, no external sensors were used for data collection and analysis, only a webcam was used to collect the subjects' facial expressions while they were playing. In this project, only three criteria - pleasure, pain and calm - were used as a reference for the subjects' emotions, supervising the subjects to stop playing when they were in pain or bored.

The face expression recognition system [20] was used in this project. The system is capable of classifying expressions into anger, disgust, fear, happiness, sadness, surprise and calmness, and I only used disgust, happiness and calmness as the three main reference criteria.

4. Results

4.1. MediaPipe vs OpenPose

4.1.1 Frontal hand movements

First, the frontal hand movements were analyzed. Two hands were selected with the backs of the hands facing the camera completely unobstructed with all ten fingers clear and intact, with a slight overlap between one hand and the face (which may have a slight effect on detection due to the similarity of skin tones). In the Figure 4, the results show that OpenPose lost the hand key points of the overlapping hand due to the similar skin tone of the face hand, the other non-overlapping hand was detected well, MediaPipe detected both hands accurately without losing any key points and the overlapping part of the hand and face were detected accurately. MediaPipe shows a better result in frontal hand movements.

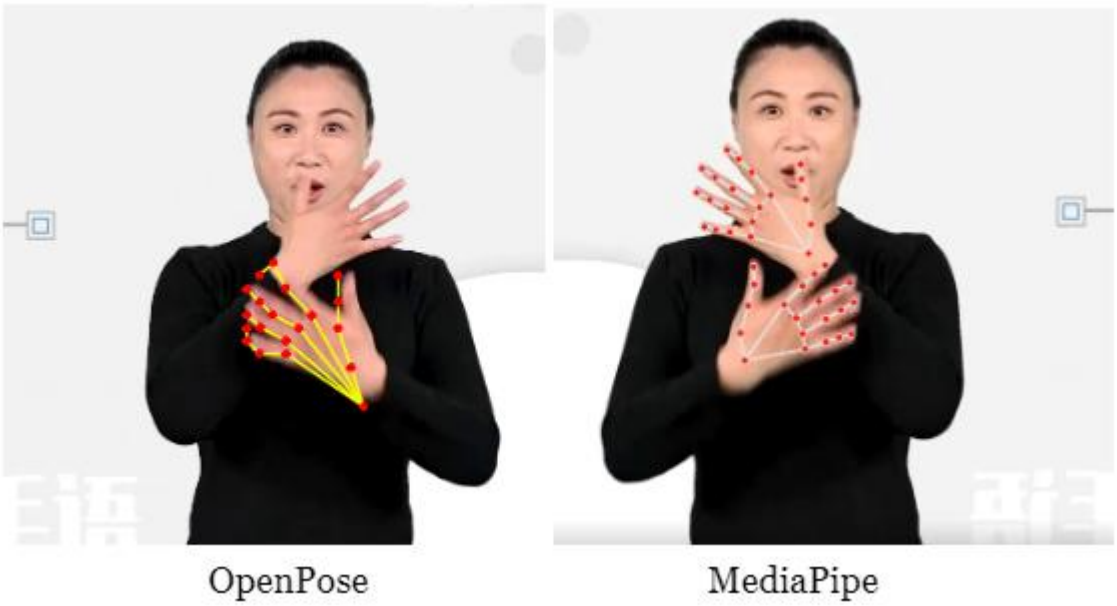


Figure 4. Comparison of frontal recognition for OpenPose and MediaPipe

4.1.2 Lateral hand movements

In the Comparison of the lateral hand movement, a one-handed pose was chosen. The palm of the hand is facing sideways towards the camera with slightly overlapping fingers.

As the figure 5 shows, when each finger is not clearly detected, Openpose does not automatically reconstruct the finger position but produces poor results. The results processed by MediaPipe, determined the position of the entire hand based on the hand points (8, 12, 16, 20) and (13, 17) ,which show in figure 6 ,and fictitiously the positions of the remaining incomplete points.

Figure 5. Comparison of lateral recognition for OpenPose and MediaPipe

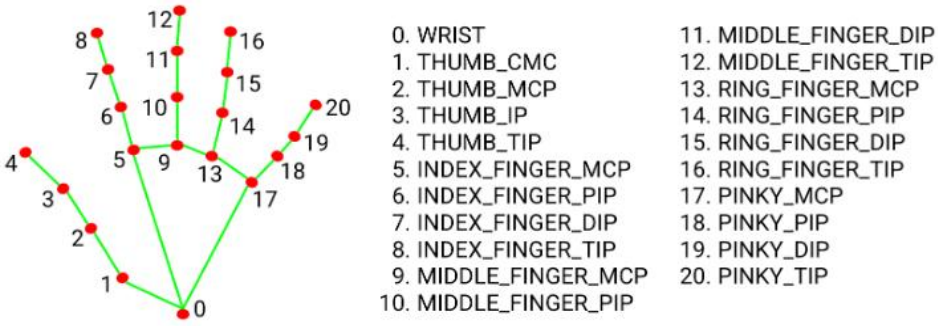


Figure 6. 21 Coordinates of hand keypoints in MediaPipe

4.1.3 Speed of motion detection

It is also important to identify the speed of the hand movements. To verify this element, three videos (all videos are 15 seconds long) were selected using only the CPU. The video will be more difficult to detect because of the blurred images that occur when the hand is in motion, and the accuracy of the algorithm will be more demanding. The faster the algorithm recognizes the hand, the more complete the hand coordinates will be, so in this section used the speed of completion of the video detection and motion blur as the criteria for differentiation.

4.1.3.1 Speed of completion of the video detection

In this project, a time function is added before and after running OpenPose and MediaPipe to record the duration used for each video analysis. The shorter the duration used for video analysis at a fixed frame rate, the higher the video frame rate and the higher the arithmetic power.

As we can see in table 1 and figure 7, MediaPipe completes detection faster than OpenPose for the same number of frames, with OpenPose taking almost 18 times longer to detect than MediaPipe.

Inspection Completion time(s)	Video 1	Video 2	Video 3
OpenPose	631.08	598.27	587.49
MediaPipe	34.38	29.56	32.53

Table 1. Inspection Completion Time of Videos

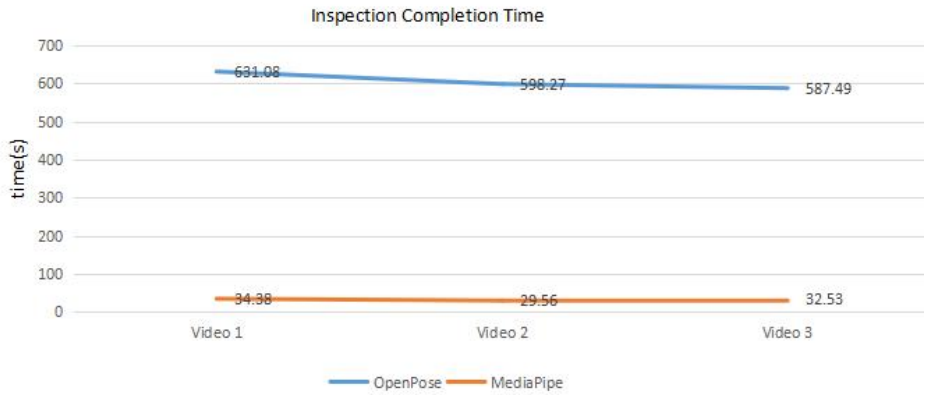


Figure 7. Visualization comparison of inspection completion time for Openpose and MediaPipe

For target detection, the detection rate is an important observation. The calculation formula is as follows:

$$Average\ FPS = Total\ number\ of\ frames\ /\ total\ number\ of\ seconds \quad \text{Eq.1}$$

In the experiment, the result is shown in table2, figure8, extremely poor results were obtained with OpenPose. It is clearly evident that MediaPipe's detection is much faster, and the results are much clearer.

Average frame rate(FPS)	Video 1	Video 2	Video 3
OpenPose	2.1	1.8	1.7
MediaPipe	38.6	36.4	30.7

Table 2. Average Frame Rate of Videos

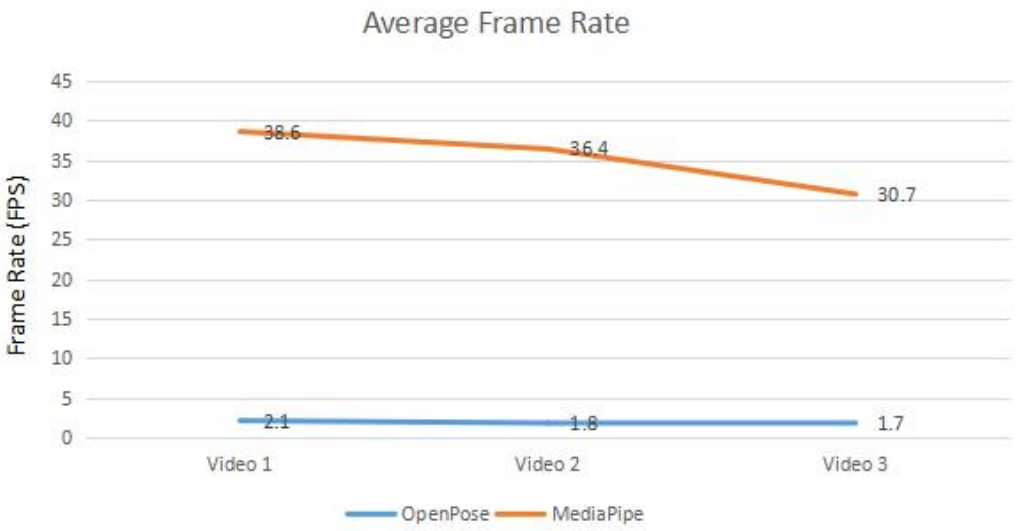


Figure 8. Visualization comparison of Average Frame Rate for Openpose and MediaPipe

4.1.3.2 Hand motion-blurred detection

Because the hand in motion is detected from start to finish, examining the fast movements is an important part of the experiment for the algorithm to determine the critical points.

I selected the part of the video where the hand is moving rapidly (the person being filmed in the video is engaged in an applause activity) and put it into Media Pipe and OpenPose separately for testing. Figure 9 was taken from the recorded test results, the OpenPose regularly loses key points or even the whole hand when the hand is moving fast, while the MediaPipe shows good stability. Some frames of the

video even appear blurred, but MediaPipe still measures the entire hand position and clearly identifies the key points.

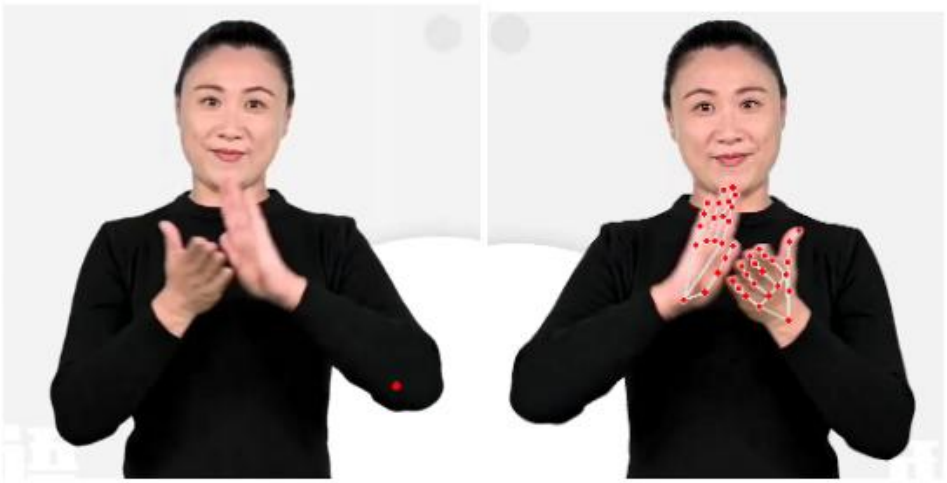


Figure 9. Detection result in the motion-blurred scenario

For motion-blurred images, no frame can be detected well for OpenPose, which is unable to identify the points and thus construct the hand points, whereas MediaPipe can accurately identify the starting point (i.e., the wrist) allowing the whole hand to be detected clearly and completely.

Hand overlap detection

For the hand overlap detection, the result shows like figure 10, OpenPose loses points of the hand, but MediaPipe worked perfectly most time. Both have significant loss of detection due to the rapid movement of people in the video, but OpenPose's is more pronounced.

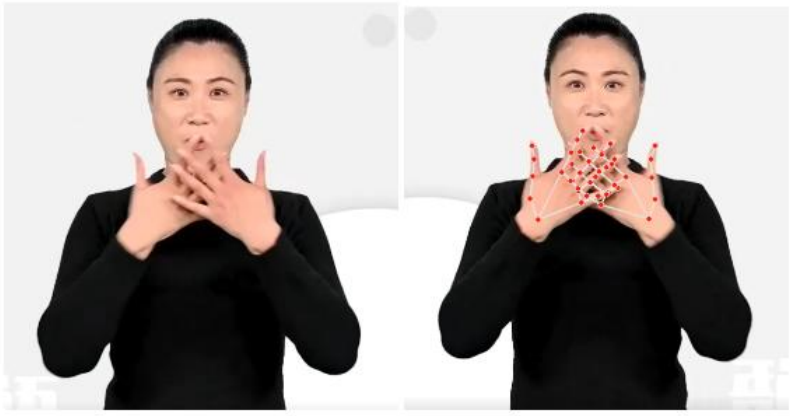


Figure 10. Detection result in the hand-overlap scenario

Analysis of these results shows that OpenPose is far worse than MediaPipe when only the CPU is used. This is most likely due to the fact that OpenPose requires extremely high CPU usage and the machine being tested is not up to its usage standards. And OpenPose did get better results when tested with the GPU. Also, OpenPose requires us to check the approximate speed of the graphics card in order to get a better detection. However, with MediaPipe, excellent results are achieved on the CPU and there is no need to configure the installation environment or enable some special settings to get faster speeds.

Kinematics analysis

For the kinematic analysis, image points that were well identified by both MediaPipe and OpenPose were selected and joint angles were compared in a time frame. The extracted images are shown in Figure 11 below.

Figure 11. Result of kinematric analysis

In the python file, angles were measured for all finger joints in point (4,3,2), (8, 7, 6), (12, 11, 10), (16, 15, 14), (20, 19, 18), (3,2,1), (7,6,5), (11,10,9), (13,14,15), (17,18,19).

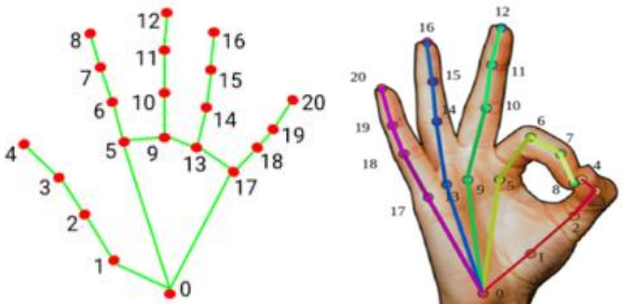


Figure 12. Applied keypoints during the kinematic analysis

The extracted joint angles are all from the right hand and the key point sequence number group being analyzed here is for OpenPose and MediaPipe are (4,3,2), (8, 7, 6), (12, 11, 10), (16, 15, 14), (20, 19, 18), (3,2,1), (7,6,5), (11,10,9), (13,14,15), (17,18,19).

As can be seen from the data in table 3, when the image detection is intact, the angles measured in coordinates are close. MediaPipe and OpenPose are comparable in terms of kinematic analysis.

Joint Angle between keypoint		
Key point group	MediaPipe	OpenPose
(2, 3, 4)	174.29°	179.32°
(6, 7, 8)	174.76°	177.48°
(10, 11, 12)	175.68°	178.53°
(14, 15, 16)	175.42°	179.6°
(18, 19, 20)	177.21°	177.56°

Table 3. Joint Angle between keypoints

Angle detection was performed on the hand video and 1683 angle data were obtained from MediaPipe and 1524 angle data were obtained from OpenPose, but some values of 0 were found in OpenPose, which required data collation and cleaning. After cleaning, the final corresponding data of 1244 was obtained. The root mean square error was applied to these data and the formula is as follows.

$$RMSE(X,h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2}$$

Eq.2

We used python to process all the data and came up with the RMSE (in unit of degree) for both sets of data. The plot of the data is shown below. We found that for angles 6, 8 and 9, the accuracy was extremely poor. By analyzing the data and video, we speculated that it might be because OpenPose could not detect the exact point and position information when the fingers were bent at an angle, resulting in the miscalculation of the angle. For angle 10, the angle data output by OpenPose has been 180 because the key point 17 has been obscured, and no real measurement of its angle has been made, while MediaPipe has accurately predicted the key point and come up with more accurate data.

Angle	Point lists
1	(4,3,2)
2	(8, 7, 6)
3	(12, 11, 10)
4	(16, 15, 14)
5	(20, 19, 18)
6	(3,2,1)
7	(7,6,5)
8	(11,10,9)
9	(13,14,15)
10	(17,18,19)

Table 4. Corresponding Keypoints group for each angle

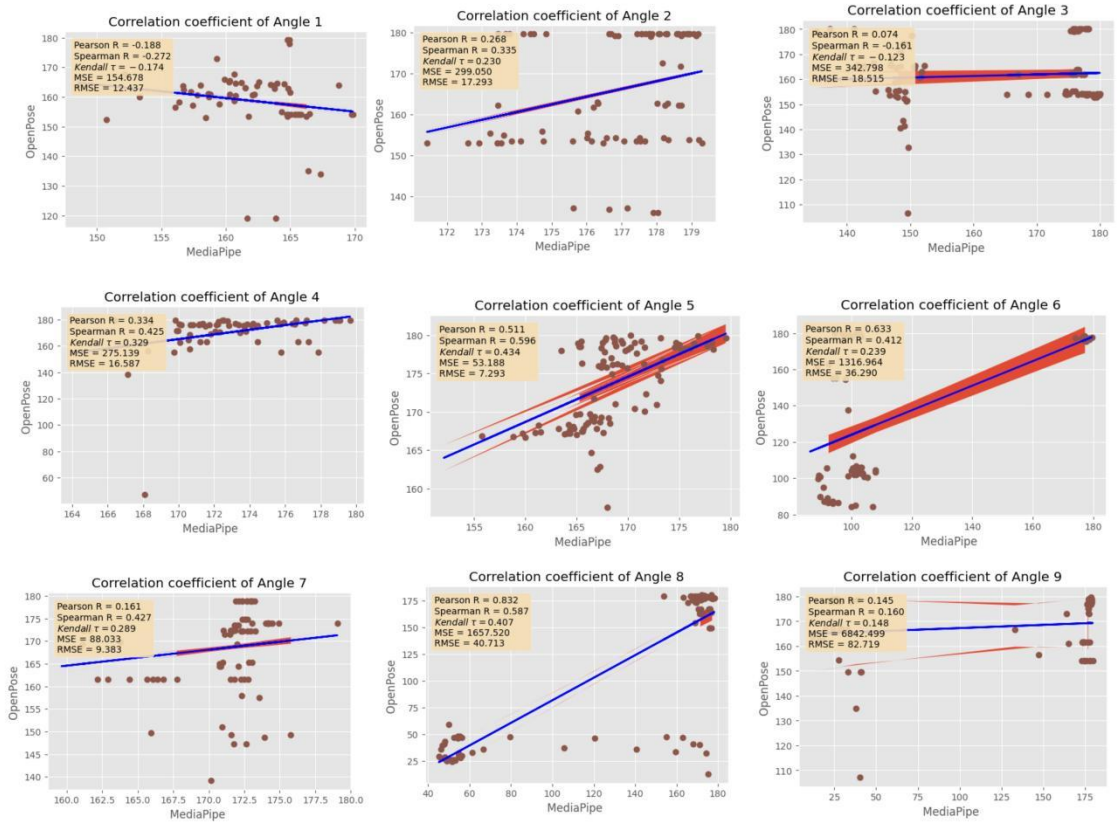


Figure 13. Scatter plots of correlation coefficient for each of angle

However, it is still important to consider the effect on recognition accuracy in the case of overlapping hands. When the area of overlapping hands is greater than 60%, even with MediaPipe, worse results are obtained. In the CPU-only-based case, the use of OpenPose is very slow compared to using MediaPipe, takes a long time to process the video, and requires the use of a device with a high level of computing power. MediaPipe would therefore be more suitable for this project.

4.2 Prototype Evaluation

Based on the results of the algorithm analysis we chose MediaPipe and OpenCV for the basic hand detection. This game only requires the use of a regular webcam to produce perfect detection results and improve the patient gaming experience. We chose to play the piano game in an attempt to exercise all the fingers and hand muscles. Because there are no rules and only simple prompts, there are no complicated messages to understand, so the game is easy to play and can be completed without instruction. The piano is played with a touch sound and does not need to be struck vigorously, ensuring that the patient's muscles and joints are safe while receiving good feedback.

The completed prototype for this project is an 88-key piano in a nice room, the game is built using Unity, version 2021.3.0f1. The camera is Logitech, and the pixels are 720p. The prototype scene is built as shown in Figures 14 and 15.

Figure 14. Scenario 1 in the game prototype

Figure 15. Scenario 2 in the game prototype

I used the hand as the object to be followed by the camera within Unity and take the approach of fixed camera following, without angle shifts, always at a distance from the hand and maintaining a fixed position in relation to the reference object (i.e., the hand).

For the piano part, a whole piano was created, but tied each key as a separate element to a specific sound. When the dots representing the fingers (red in this project) touch the keys of the piano (black or white in this project) the corresponding sound effect is played automatically, the contact of the colors simply triggering the sound effects of the piano. In view of the patient's lack of finger mobility and in order to avoid psychological barriers to rehabilitation, there is no need to hit the keys hard to produce the sound effect, the patient simply touches the piano keys with his fingers to get the desired sound. The aim of the game is to improve the patient's finger dexterity as much as possible, so there are no requirements for the intensity of the tapping.

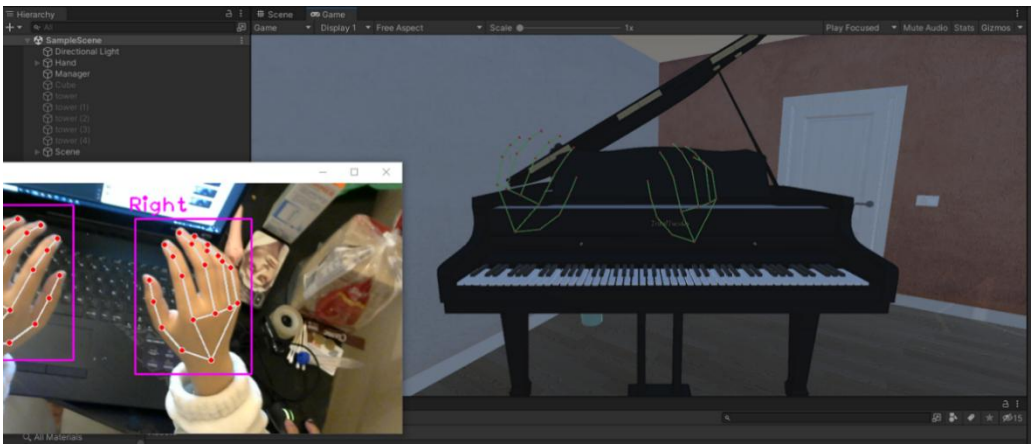


Figure 16. Real-time testing scene of the game

During the testing phase of the project, the participants completed the project well due to the simplicity and high visibility of the songs chosen. The difficulty, however, lay in teaching them scales, and it was difficult for someone without a musical foundation to discern which of the 88 keys represented the sound he wanted to play.

The size and position of the hand in the game can be changed by the distance from the camera to better suit the playing of the piano. When we lower our hand, our fingers will touch the piano and we will be able to play. Before the patient uses the game, we will declare to the patient the correct finger-playing position, i.e. raising the finger and tapping down, and expect the patient to play with a gesture close to the correct position. However, as it is possible that the patient does not have hand

dexterity, only small angular movements of the fingers are required to touch the keyboard at the beginning of the game.

In order to determine the patient's progress in recovery, the patient's maximum finger flexion angle needs to be recorded at regular intervals. This helps the family and therapist to understand the patient's recovery.

Figure 17 shows the patient's finger flexion using the MediaPipe, from which we can clearly see the change in finger flexion angle. The angle calculated here is the angle of bending during the finger movement. By measuring the MediaPipe, the coordinates of the position of the point (8, 7, 6) (shown in Fig. 18), for example, can be calculated with the help of the following formula to find out the tangent angle of the two lines and then subtract them to find the desired angle between the points (8, 7, 6).

$$\text{Angle} = \arctan(y/x) \qquad \text{Eq.3}$$

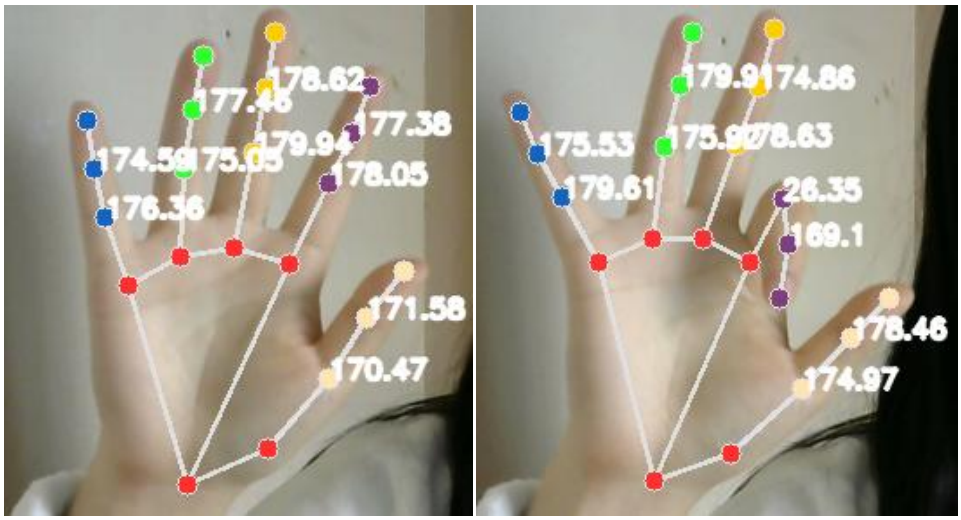


Figure 17. Finger curvature test

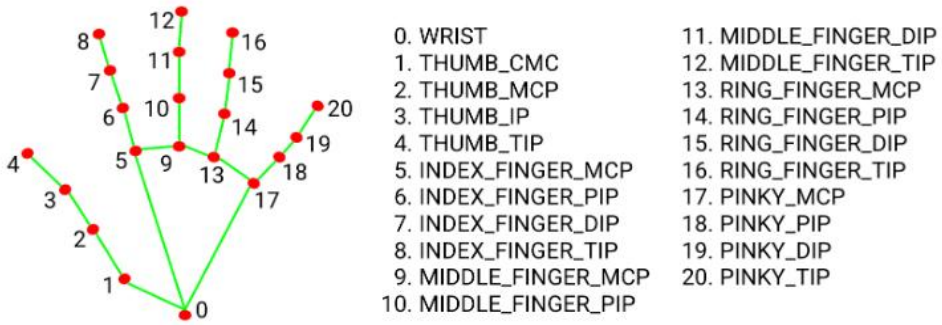


Figure 18. Hand Landmarks of MediaPipe

Individual interviews were conducted with participants to evaluate the game interface, comfort, appearance and comprehensiveness. The interviews were conducted individually on a one-to-one basis so that participants were not influenced by third parties or others that could fluctuate or change the results.

Participant	Age	Gender
P1	12	Male
P2	24	Male
P3	25	Female
P4	40	Male

Table 5. User information

Participant	Interface	Ease	Attractiveness	OverALL
P1	8	8	8	8
P2	7	7	8	7
P3	7	5	6	7
P4	8	3	8	5

Table 6. User Satisfaction Questionnaire

We have analyzed each participate scoring by asking them some question. The table shows that overall the participants were satisfied with the prototype, but for P4 the overall game experience was not good enough because the participant had more severe 3D vertigo [21] . When the P4 shakes its hand, because the camera follows the shaking of the hand, the central nervous system receives signals of rapid movement, but the receptors in the rest of the body receive only stationary signals. The body experiences an illusion of spatial orientation, causing a corresponding sense of vertigo.

So this project has developed a prototype to use a fixed-point camera aimed at the piano for frame acquisition rather than the camera following the hand. This reduces the variability of the game, but greatly enhances the adaptability of the participants and reduces the discomfort that users like P4 may experience from using a serious game.

Participant	Happy	Disgusted	Calm
P1	36%	8%	56%
P2	41%	5%	54%
P3	68%	3%	29%
P4	46%	13%	41%

Table 7. User Emotional Feedback

In terms of the feedback collected, the facial expressions of the participants were analyzed as happy and calm most of the time when they were playing in small amounts. This suggests that the subjects were not resistant to the seriousness of the

game in hand rehabilitation, and the joint point data collected from the python suggests that the subjects performed sufficient hand movement during the game to exercise the joints of the hand, which has been shown in previous studies to be an effective way to move the joints of the hand in hand rehabilitation [3].

5. Discussion

In order to provide more effective rehabilitation for post-traumatic patients, this study proposes a simple rehabilitation-based serious game that can be played at home. Preliminary results show that this game can detect the hand well and correctly detect the patient's joint curvature.

From the above experiments, it is easy to see that MediaPipe outperforms OpenPose when using MediaPipe and OpenPose for frontal hand motion, side hand motion and hand overlap detection, because in the comparison of algorithms, it is observed that OpenPose has a large number of missing key points, and in terms of motion detection speed, MediaPipe is even ten times faster than OpenPose. In the final kinematic analysis, when the hand detection is stable, the performance of MediaPipe and OpenPose are close most of the time and both can accurately measure the angle of the finger. When no hand appears on the screen, the output of OpenPose is confusing, with occasional errors at the detected key points, while MediaPipe has no output, but when the hand key points are obscured When the key point of the hand is obscured, OpenPose seems to be unable to infer the corresponding joint, which has a great impact on OpenPose's angle detection.

Based on the algorithm and the required operating environment, the reasons for the poor OpenPose results are likely to be the following 1. Insufficient training of the algorithm in the early stages of the project, resulting in less accurate recognition and frequent loss of hand keypoint coordinates. 2. The basic hardware configuration of the project was not up to the standard required by OpenPose. The project only uses a pure CPU to compare the two algorithms and the project may not have the required memory to run OpenPose. When using OpenPose for hand detection, CPU usage reached a maximum of 99%. This may have significantly limited the computing speed of OpenPose. When OpenPose is run on the GPU, the computing speed is greatly increased.

In this project, a good level of user satisfaction was obtained. In general, however, the project has a few limitations. In our user satisfaction survey, we found that musically educated and younger participants were more motivated by the game. The younger and more musically educated participants were more able to learn when taught scales and repertoire. This may have limited the age and education level of the participants. In subsequent development a cue bar will be introduced to indicate which key to press next after a key is pressed, with the length of the bar determining the duration of the key press and the bar disappearing when the user presses the exact key and the duration is reached. This will break the musical

limitations of the game and allow people who don't know how to play the piano to be able to play beautiful pieces. The project is not very open-ended and needs to be developed more, with more levels or with the appropriate tracks played and the background developed, in order to achieve immersion in the game, reduce the patient's resistance to manual rehabilitation, and shift the patient's attention from pain to enjoyment of the game [22] to the extent that this is acceptable.

The sample is inadequate as the project was conducted on normal people only, without the supervision of a therapist and as the project is not complete and does not simulate all the exercises currently covered in manual therapy, it is not a substitute for a full therapy session and can only be used as a supplement and development to a therapy session.

The project needs to be supported by more rehabilitation-related researchers and the involvement of a variety of users from different backgrounds, which will allow for more accurate user feedback and professional feedback, and from user feedback and observation of user behavior across different impairments and levels of impairment, which will allow for a more realistic user demand for the new version and more effective treatment.

Preliminary results suggest that this project is rehabilitative feasible and entertaining. As this project is only in the testing phase so far, the sample size greatly limits the generation of conclusions. However, from the results obtained so far, the feasibility of this project is high due to its simplicity and portability, allowing users to freely use this serious game at home. This greatly enhances the accessibility and universality of rehab, allowing more patients to recover and return to normal life more quickly after surgery.

6. Future Work

In the future, the project will be further improved by adding more gamification features to enhance the user experience, such as adding more levels and adding a reward system.

We will also engage real patients to participate in the design and testing of the game, so as to bring a better and more relevant version of the game to the actual treatment, to observe the effect of the game in the rehabilitation and the effective data obtained, to compare the advantages and disadvantages of the serious game alone and the combination of the serious game with the current treatment and rehabilitation and the gap between the two, so as to improve the game version and achieve a better treatment effect.

In the further future, it is expected that serious games will be integrated with traditional rehabilitation tools to form a complete system for analyzing treatment and evaluation, recording user data, and using deep learning and artificial intelligence to adjust the difficulty of the game, making it even more functional and effective.

7. Conclusion

This study is a serious hand rehabilitation game based on OpenCV and MediaPipe focusing on hand testing that can be used to promote hand rehabilitation therapy anywhere and anytime. Training finger flexion angles through piano playing and immersive music can help patients regain some hand dexterity and calm their emotions. Some progress has been made in hand testing, game integrity and data analysis.

This serious hand rehabilitation game can be played on a regular webcam, allowing for daily physical rehabilitation without the use of any expensive electronic devices, and because of its entertaining nature, patients are not resistant to the rehabilitation content. The program can record data for each exercise, making it easy for the therapist or family to access the records and analyze whether the patient is making progress in the exercises. The patient's facial expressions are also recorded each time the game is played and analyzed in the background so that the patient's time in the game can be adjusted.

The application and still has limitations in many areas. For example, the game is too one-dimensional and prolonged use may cause users to become bored with the game, and future versions will include more hand training mini-games to make the game more interesting. The project will not include therapists and patients for the time being, but will only be tested on a normal population, but an analysis of previous therapy experiences shows that serious games can be fun and effective in speeding up the recovery process for patients. There is still a lot of testing to be done before the game is actually made available to users, but the initial model of the game has been defined and stabilized. The project also requires the construction of a server to store the data obtained by the user during use and to build it into a database. This will make it easier to recall pre-treatment data, data analysis, and user profiling. The project has attempted to do this with local storage, but due to the amount of data available, when the user reaches a certain level of usage, the previous data will be overwritten, resulting in data loss, and the family or therapist will not be able to compare the data well enough to determine the user's recovery. This project is available in a single language version and may not be accessible to some users who do not understand English. In future versions, this project will implement more features and provide a better version so that more users can benefit from it.

References

1. Towfighi A, Ovbiagele B, el Husseini N, et al. Poststroke Depression: A Scientific Statement for Healthcare Professionals From the American Heart Association/American Stroke Association. *Stroke*; 48. Epub ahead of print February 2017. DOI: 10.1161/STR.0000000000000113.

2. Bouteraa Y, Abdallah I ben, Elmogy AM. Training of Hand Rehabilitation Using Low Cost Exoskeleton and Vision-Based Game Interface. *J Intell Robot Syst* 2019; 96: 31–47.

3. Song X, Tong SJ, Shankara Van De Ven S, et al. Wearable Multimodal-Serious Game System For Hand and Cognitive Rehabilitation After Stroke. Epub ahead of print 2021. DOI: 10.21203/rs.3.rs-917425/v1.

4. Terlouw G, Kuipers D, van 't Veer J, et al. The Development of an Escape Room–Based Serious Game to Trigger Social Interaction and Communication Between High-Functioning Children With Autism and Their Peers: Iterative Design Approach. *JMIR Serious Games* 2021; 9: e19765.

5. Garcia A, Mayans B, Margelí C, et al. A feasibility study to assess the effectiveness of Muvity: A telerehabilitation system for chronic post-stroke subjects. *Journal of Stroke and Cerebrovascular Diseases* 2022; 31: 106791.

6. Cao Z, Hidalgo G, Simon T, et al. OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields.

7. Lugaresi C, Tang J, Nash H, et al. MediaPipe: A Framework for Building Perception Pipelines.

8. Clark C. Abt. *Serious Games*. University Press of America, 1987.

9. Saini S, Rambli DRA, Sulaiman S, et al. A low-cost game framework for a home-based stroke rehabilitation system. In: *2012 International Conference on Computer & Information Science (ICCIS)*. IEEE, 2012, pp. 55–60.

10. Szaniawski M, Boess S, Kraan G, et al. Enhancing engagement with motor rehabilitation therapy through gamified rehabilitation interactions. 2015.

11. Galna B, Barry G, Jackson D, et al. Accuracy of the Microsoft Kinect sensor for measuring movement in people with Parkinson's disease. *Gait Posture* 2014; 39: 1062–1068.

12. Webster D, Celik O. Systematic review of Kinect applications in elderly care and stroke rehabilitation. *J Neuroeng Rehabil* 2014; 11: 108.

13. Rehg JM, Kanade T. DigitEyes: vision-based hand tracking for human-computer interaction. In: *Proceedings of 1994 IEEE Workshop on Motion of Non-rigid and Articulated Objects*. IEEE Comput. Soc. Press, pp. 16–22.

14. Wang RY, Popović J. Real-time hand-tracking with a color glove. *ACM Trans Graph* 2009; 28: 1–8.

15. Wei S-E, Ramakrishna V, Kanade T, et al. Convolutional Pose Machines. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 4724–4732.

16. CMU-Perceptual-Computing-Lab. Caffe. *caffe.berkeleyvision.org/*.

17. Ni Lan, Fudan University(China). Sign Language Supporting Video. *https://www.youtube.com/watch?v=RfSY8frrmCc*.

18. Ekman P, Freisen W v., Ancoli S. Facial signs of emotional experience. *J Pers Soc Psychol* 1980; 39: 1125–1134.

19. Toisoul A, Kossaifi J, Bulat A, et al. Estimation of continuous valence and arousal levels from faces in naturalistic conditions. *Nat Mach Intell* 2021; 3: 42–50.

20. X.Wu. Introduction and Demonstration of Facial Expression Recognition System. *https://www.bilibili.com/video/BV18C4y1H7mH/?vd_source=a6ef87ce99ada22e1324fb35d3565915*.

21. Liang Z-Q, Chen M-B, Wu C-X, et al. Effect of Viewing Angle and Distance on Motion Sickness in 3D game. In: *2020 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)*. IEEE, 2020, pp. 213–215.

22. Majid ESA, Garcia JA, Nordin AI, et al. Staying Motivated During Difficult Times: A Snapshot of Serious Games for Paediatric Cancer Patients. *IEEE Trans Games* 2020; 12: 367–375.

Annex A. Hardware and Software used

The following lists the hardware and software used to build *Piano.*

Hardware	*PC Personal*
	Web Camera
Software	*Visual Studio Code*
	Unity
	Emotion Recongnition v1.2

A.1 PC Personal

The computer used for all the tasks has been a desktop PC of the brand ASUS and model ROG Strix G17.

The characteristics of the PC are shown below.

Operating systems	Windows 10 Professional 21H1
Processor	Intel(R) Core (TM) i7-10750H CPU @ 2.60GHz 2.59 GHz
Graphics card	NVIDIA GeForce RTX 2070 GDDR6 8GB (256 bits)
Hardware RAM	16 GB
System type	x64

A.2 Web Camera

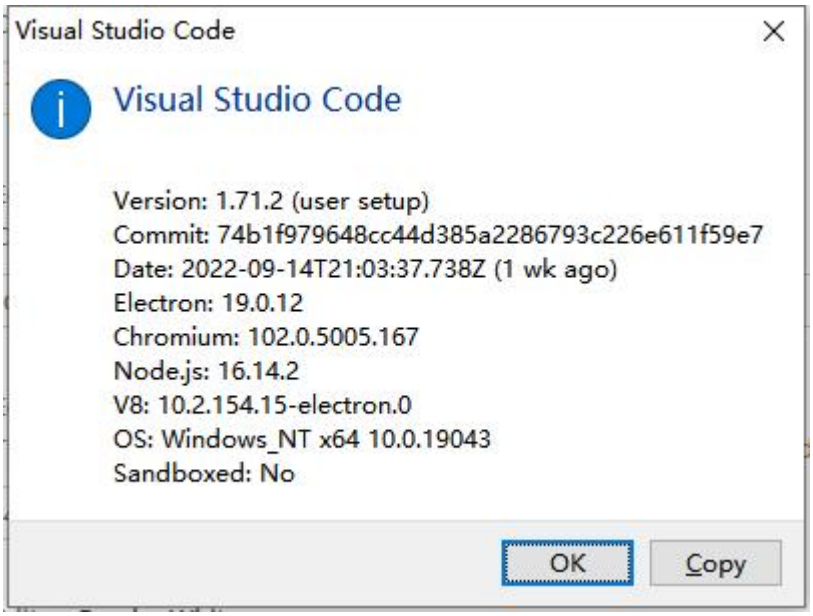
The camera used for hand detection is Logitech HD webcam C270i 720p. Officially priced at €37.99. Easy and quick access to images using the USB socket connection. For this project, it was used to obtain hand and face images.

General characteristics	
Image Sensor	1.2 Megapixel
Effective Image Range	0.4m-infinity
Best Image Range	0.4-5m Viewing
Angle Range	60°
Video Support Resolution	720p Video Frame
Rate	30 frames
Minimum Illumination	30Lux
LENS	Plastic
Bit Rate	16kHz, 24kHz, 32kHz, 48kHz

A.3 Visual Studio Code

Visual Studio Code (VScode) is a code editor redefined and optimized for building and debugging modern web and cloud applications. This project uses Visual Studio Code for Python (for hand detection) and C# (for data transmission and unity building) programming.

Version number	
Visual Studio Code	1.71.2
Python	python 3.9.13 64-bit
C#	V1.25.0



A.4 Unity

Unity is a cross-platform game engine developed by Unity Technologies. The engine has support for a wide range of desktop, mobile and virtual reality development. It is the world's most popular game development platform and is easy to use for beginner developers. The engine can be used to create both three-dimensional (3D) and two-dimensional (2D) games.

The Unity Hub vision used in this project is 3.1.0 beta.2. The editor version number used in this project is 2021.3.of1.

A.5 Emotion recongnition v1.2

The Emotion recognition software is from Xian Wu. The operating system requirements are Windows 10 and above, the integrated development environment need to be Pycharm or VScode, python version number 3.7 and above, and for the vision used in this project (v1.2) need to download a number of dependencies:

PyQT5 5.11.3
Keras 2.2.4
Tensorflow 1.13.1
Pandas 0.24.2
Scikit-learn 0.21.2
Opencv-python 4.10.25
Imutils 0.5.2

Now this software only has the Chinese version.

Annex B.1 Script "HandTrakingModule.py"

```
import cv2
import mediapipe as mp
import math


class HandDetector:

    def __init__(self, mode=False, maxHands=2, detectionCon=0.5, minTrackCon=0.5):

        self.mode = mode
        self.maxHands = maxHands
        self.detectionCon = detectionCon
        self.minTrackCon = minTrackCon

        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(static_image_mode=self.mode, max_num_hands=self.maxHands,
                                        min_detection_confidence=self.detectionCon,
                                        min_tracking_confidence=self.minTrackCon)
        self.mpDraw = mp.solutions.drawing_utils
        self.tipIds = [4, 8, 12, 16, 20]
        self.fingers = []
        self.lmList = []

    def findHands(self, img, draw=True, flipType=True):

        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        self.results = self.hands.process(imgRGB)
        allHands = []
        h, w, c = img.shape
        if self.results.multi_hand_landmarks:
            for handType, handLms in zip(self.results.multi_handedness, self.results.multi_hand_landmarks):
                myHand = {}
                ## lmList
                mylmList = []
                xList = []
                yList = []
                zList = []
                for id, lm in enumerate(handLms.landmark):
                    px, py, pz = int(lm.x * w), int(lm.y * h), int(lm.z * w)
                    mylmList.append([px, py, pz])
                    xList.append(px)
                    yList.append(py)
                    zList.append(pz)

                ## bbox
                xmin, xmax = min(xList), max(xList)
                ymin, ymax = min(yList), max(yList)
                boxW, boxH = xmax - xmin, ymax - ymin,
                bbox = xmin, ymin, boxW, boxH
                cx, cy = bbox[0] + (bbox[2] // 2), \
                         bbox[1] + (bbox[3] // 2),
                myHand["lmList"] = mylmList
                myHand["bbox"] = bbox
                myHand["center"] = (cx, cy)

                if flipType:
                    if handType.classification[0].label == "Right":
                        myHand["type"] = "Left"
                    else:
                        myHand["type"] = "Right"
                else:
                    myHand["type"] = handType.classification[0].label
                allHands.append(myHand)
```

```
            ## draw
            if draw:
                self.mpDraw.draw_landmarks(img, handLms,
                                           self.mpHands.HAND_CONNECTIONS)
                cv2.rectangle(img, (bbox[0] - 20, bbox[1] - 20),
                              (bbox[0] + bbox[2] + 20, bbox[1] + bbox[3] + 20),
                              (255, 0, 255), 2)
                cv2.putText(img, myHand["type"], (bbox[0] - 30, bbox[1] - 30), cv2.FONT_HERSHEY_PLAIN,
                            2, (255, 0, 255), 2)
    if draw:
        return allHands, img
    else:
        return allHands

def fingersUp(self, myHand):

    myHandType = myHand["type"]
    myLmList = myHand["lmList"]
    if self.results.multi_hand_landmarks:
        fingers = []
        # Thumb
        if myHandType == "Right":
            if myLmList[self.tipIds[0]][0] > myLmList[self.tipIds[0] - 1][0]:
                fingers.append(1)
            else:
                fingers.append(0)
        else:
            if myLmList[self.tipIds[0]][0] < myLmList[self.tipIds[0] - 1][0]:
                fingers.append(1)
            else:
                fingers.append(0)

        # 4 Fingers
        for id in range(1, 5):
            if myLmList[self.tipIds[id]][1] < myLmList[self.tipIds[id] - 2][1]:
                fingers.append(1)
            else:
                fingers.append(0)
    return fingers
```

```
    def findDistance(self, p1, p2, img=None):

        x1, y1 = p1
        x2, y2 = p2
        cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
        length = math.hypot(x2 - x1, y2 - y1)
        info = (x1, y1, x2, y2, cx, cy)
        if img is not None:
            cv2.circle(img, (x1, y1), 15, (255, 0, 255), cv2.FILLED)
            cv2.circle(img, (x2, y2), 15, (255, 0, 255), cv2.FILLED)
            cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), 3)
            cv2.circle(img, (cx, cy), 15, (255, 0, 255), cv2.FILLED)
            return length, info, img
        else:
            return length, info
def main():
    cap = cv2.VideoCapture(0)
    detector = HandDetector(detectionCon=0.8, maxHands=2)
    while True:
        # Get image frame
        ret, img = cap.read()
        # Find the hand and its landmarks
        hands, img = detector.findHands(img)  # with draw
        # hands = detector.findHands(img, draw=False)  # without draw

        if hands:
            # Hand 1
            hand1 = hands[0]
            lmList1 = hand1["lmList"]  # List of 21 Landmark points
            bbox1 = hand1["bbox"]  # Bounding box info x,y,w,h
            centerPoint1 = hand1['center']  # center of the hand cx,cy
            handType1 = hand1["type"]  # Handtype Left or Right



            if len(hands) == 2:
                # Hand 2
                hand2 = hands[1]
                lmList2 = hand2["lmList"]  # List of 21 Landmark points
                bbox2 = hand2["bbox"]  # Bounding box info x,y,w,h
                centerPoint2 = hand2['center']  # center of the hand cx,cy
                handType2 = hand2["type"]  # Hand Type "Left" or "Right"

        # Display
        cv2.imshow("Image", img)
        cv2.waitKey(1)


if __name__ == "__main__":
    main()
```

Annex B.2 Script “Main.py”

```
from HandTrackingModule import HandDetector
import cv2
import socket

cap = cv2.VideoCapture(0)
cap.set(3, 480)
cap.set(4, 320)
#Defining ports
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
serverAddressPort = ("127.0.0.1", 5052)

ret, img = cap.read()
h, w, _ = img.shape
detector = HandDetector(detectionCon=0.8, maxHands=2)
data = []
pTime = 0
cTime = 0

while True:
    # Get image frame
    ret, img = cap.read()
    # Find the hand and its landmarks
    hands, img = detector.findHands(img)  # with draw
    data = []

    if hands:
        # Hand 1
        hand1 = hands[0]
        lmList1 = hand1["lmList"]  # List of 21 Landmark points
        bbox1 = hand1["bbox"]  # Bounding box info x,y,w,h
        centerPoint = hand1['center']  # center of the hand cx,cy
        handType1 = hand1["type"]  # Handtype Left or Right
        pointdown1 = lmList1[0]
        pointup1 = lmList1[12]
        n1, _ = detector.findDistance(pointdown1[0:2], pointup1[0:2])
        W = 6.3

        #Finding distance
        f = 840
        d1 = (W * f) / n1
        print('d1=',d1)
        for lm1 in lmList1:
            data.extend([lm1[0], h - lm1[1], d1])

        sock.sendto(str.encode(str(data)), serverAddressPort)
            #Send data

        if len(hands) == 2:
            # Hand 2
            hand2 = hands[1]
            lmList2 = hand2["lmList"]  # List of 21 Landmark points
            bbox2 = hand2["bbox"]  # Bounding box info x,y,w,h
            centerPoint2 = hand2['center']  # center of the hand cx,cy
            handType2 = hand2["type"]  # Hand Type "Left" or "Right"
            pointdown2 = lmList2[0]
            pointup2 = lmList2[12]
            n2, _ = detector.findDistance(pointdown2[0:2], pointup2[0:2])
            W = 6.3
```

```
            #Finding distance
            f = 840
            d2 = (W * f) / n2
            print('d2=',d2)
            for lm2 in lmList2:
                data.extend([lm2[0], h - lm2[1], d2])

            sock.sendto(str.encode(str(data)), serverAddressPort)

        print(data)

        # Display
    cv2.imshow("Image", img)
    cv2.waitKey(1)
```

Annex B.3 Script “MediaPipeAngle.py”

```
import cv2
import numpy as np
import mediapipe as mp

mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_holistic = mp.solutions.holistic

joint_list = [[4,3,2],[8, 7, 6], [12, 11, 10], [16, 15, 14], [20, 19, 18], [3,2,1], [7,6,5], [11,10,9], [13,14,15], [17,18,19]]

cap = cv2.VideoCapture(0)
with mp_holistic.Holistic(
        min_detection_confidence=0.5,
        min_tracking_confidence=0.5) as holistic:
    while cap.isOpened():
        success, image = cap.read()
        if not success:
            print("Ignoring empty camera frame.")
            break
        image.flags.writeable = False
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        results = holistic.process(image)
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                                  landmark_drawing_spec=mp_drawing_styles.get_default_hand_landmarks_style())
        mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                                  landmark_drawing_spec=mp_drawing_styles.get_default_hand_landmarks_style())
        # 监测到右手，执行
        if results.right_hand_landmarks:
            RHL = results.right_hand_landmarks
            # 计算角度
            for joint in joint_list:
                a = np.array([RHL.landmark[joint[0]].x, RHL.landmark[joint[0]].y])
                b = np.array([RHL.landmark[joint[1]].x, RHL.landmark[joint[1]].y])
                c = np.array([RHL.landmark[joint[2]].x, RHL.landmark[joint[2]].y])
                # 计算弧度
                radians_fingers = np.arctan2(c[1] - b[1], c[0] - b[0]) - np.arctan2(a[1] - b[1], a[0] - b[0])
                angle = np.abs(radians_fingers * 180.0 / np.pi)  # 弧度转角度

                if angle > 180.0:
                    angle = 360 - angle

                cv2.putText(image, str(round(angle, 2)), tuple(np.multiply(b, [640, 480]).astype(int)),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE_AA)

        # cv2.imshow('MediaPipe Holistic', cv2.flip(image, 1))
        cv2.imshow('Mediapipe Holistic', image)  # 取消镜面翻转
        if cv2.waitKey(5) == ord('q'):
            break
cap.release()
```

Annex B.4 Script "Openposehand.py"

```
import cv2
import time
import numpy as np


protoFile = "pose_deploy.prototxt"
weightsFile = "pose_iter_102000.caffemodel"
nPoints = 22
POSE_PAIRS = [ [0,1],[1,2],[2,3],[3,4],[0,5],[5,6],[6,7],[7,8],[0,9],[9,10],
               [10,11],[11,12],[0,13],[13,14],[14,15],[15,16],[0,17],[17,18],[18,19],[19,20] ]

threshold = 0.2

cap = cv2.VideoCapture("D:\ApowerREC\handlang1.mp4")
hasFrame, frame = cap.read()

frameWidth = frame.shape[1]
frameHeight = frame.shape[0]

aspect_ratio = frameWidth/frameHeight

inHeight = 368
inWidth = int(((aspect_ratio*inHeight)*8)//8)


vid_writer = cv2.VideoWriter('output.avi',cv2.VideoWriter_fourcc('M','J','P','G'),
                             15, (frame.shape[1],frame.shape[0]))


net = cv2.dnn.readNetFromCaffe(protoFile, weightsFile)
k = 0
while True:
    k+=1
    t = time.time()

    hasFrame, frame = cap.read()
    frameCopy = np.copy(frame)
    if not hasFrame:
        cv2.waitKey()
        break

    # blobFromImage
    inpBlob = cv2.dnn.blobFromImage(frame, 1.0 / 255, (inWidth, inHeight),
                                    (0, 0, 0), swapRB=False, crop=False)

    net.setInput(inpBlob)

# The model can generate 22 key points, 21 of which are for the human hand and the 22nd point represents the background
output = net.forward()

print("forward = {}".format(time.time() - t))
```

```
    # Empty list to store the detected keypoints
    points = []

    for i in range(nPoints):
        probMap = output[0, i, :, :]
        probMap = cv2.resize(probMap, (frameWidth, frameHeight))


        minVal, prob, minLoc, point = cv2.minMaxLoc(probMap)

        if prob > threshold :
            cv2.circle(frameCopy, (int(point[0]), int(point[1])), 6, (0, 255, 255), thickness=-1, lineType=cv2.FILLED)
            cv2.putText(frameCopy, "{}".format(i), (int(point[0]), int(point[1])), cv2.FONT_HERSHEY_SIMPLEX, .8, (0, 0, 255), 2, lineType=cv2.LINE_AA)

            points.append((int(point[0]), int(point[1])))
        else :
            points.append(None)


    for pair in POSE_PAIRS:
        partA = pair[0]
        partB = pair[1]

        if points[partA] and points[partB]:
            cv2.line(frame, points[partA], points[partB], (0, 255, 255), 2, lineType=cv2.LINE_AA)
            cv2.circle(frame, points[partA], 5, (0, 0, 255), thickness=-1, lineType=cv2.FILLED)
            cv2.circle(frame, points[partB], 5, (0, 0, 255), thickness=-1, lineType=cv2.FILLED)

    print("Time Taken for frame = {}".format(time.time() - t))

    cv2.imshow('webcam', frame)

    key = cv2.waitKey(1)
    if key == 27:
        break

    print("total = {}".format(time.time() - t))

    vid_writer.write(frame)

vid_writer.release()
```

Annex B.5 Script "openposeangle.py"

```
import sys
import cv2
import os
from sys import platform
import argparse
import time
import math

def angle_between_points( p0, p1, p2 ):
    # 计算角度
    a = (p1[0]-p0[0])**2 + (p1[1]-p0[1])**2
    b = (p1[0]-p2[0])**2 + (p1[1]-p2[1])**2
    c = (p2[0]-p0[0])**2 + (p2[1]-p0[1])**2
    if a * b == 0:
        return -1.0

    return  math.acos( (a+b-c) / math.sqrt(4*a*b) ) * 180 /math.pi

def length_between_points(p0, p1):
    # 2点之间的距离
    return math.hypot(p1[0]- p0[0], p1[1]-p0[1])

def get_angle_point(human, hand):
    # 返回各个部位的关键点
    pnts = []
    if hand == 'thumbs_up1':
        hand_list = (1,2,3)
    elif hand == 'index_finger1':
        hand_list = (5,6,7)
    elif hand == 'middle_finger1':
        hand_list = (9,10,11)
    elif hand == 'ring_finger1':
        hand_list = (13,14,15)
    elif hand == 'little_finger1':
        hand_list = (17,18,19)
    elif hand == 'thumbs_up2':
        hand_list = (2,3,4)
    elif hand == 'index_finger2':
        hand_list = (6,7,8)
    elif hand == 'middle_finger2':
        hand_list = (10,11,12)
    elif hand == 'ring_finger2':
        hand_list = (14,15,16)
    elif hand == 'little_finger2':
        hand_list = (18,19,20)
    else:
        print('Unknown  [%s]', hand)
        return pnts


    for i in range(3):
        if human[hand_list[i]][2] <= 0.1:
            print('component [%d] incomplete'%(hand_list[i]))
            return pnts

        pnts.append((int( human[hand_list[i]][0]), int( human[hand_list[i]][1])))
    return pnts
```

```
def angle_thumbs_up1(human):
    pnts = get_angle_point(human, 'thumbs_up1')
    if len(pnts) != 3:
        print('component incomplete')
        return -1

    angle = 0
    if pnts is not None:
        angle = angle_between_points(pnts[0], pnts[1], pnts[2])
        print('thumbs_up1:%f'%(angle))
    return angle


def angle_index_finger1(human):
    pnts = get_angle_point(human, 'index_finger1')
    if len(pnts) != 3:
        print('component incomplete')
        return

    angle = 0
    if pnts is not None:
        angle = angle_between_points(pnts[0], pnts[1], pnts[2])
        print('index_finger1:%f'%(angle))
    return angle


def angle_middle_finger1(human):
    pnts = get_angle_point(human, 'middle_finger1')
    if len(pnts) != 3:
        print('component incomplete')
        return

    angle = 0
    if pnts is not None:
        angle = angle_between_points(pnts[0], pnts[1], pnts[2])
        print('middle_finger1:%f'%(angle))
    return angle


def angle_ring_finger1(human):
    pnts = get_angle_point(human, 'ring_finger1')
    if len(pnts) != 3:
        print('component incomplete')
        return

    angle = 0
    if pnts is not None:
        angle = angle_between_points(pnts[0], pnts[1], pnts[2])
        print('ring_finger1:%f'%(angle))
    return angle


def angle_little_finger1(human):
    pnts = get_angle_point(human, 'little_finger1')
    if len(pnts) != 3:
        print('component incomplete')
        return

    angle = 0
    if pnts is not None:
        angle = angle_between_points(pnts[0], pnts[1], pnts[2])
        print('little_finger1:%f'%(angle))
    return angle
```

```
def angle_thumbs_up2(human):
    pnts = get_angle_point(human, 'thumbs_up2')
    if len(pnts) != 3:
        print('component incomplete')
        return

    angle = 0
    if pnts is not None:
        angle = angle_between_points(pnts[0], pnts[1], pnts[2])
        print('thumbs_up2:%f'%(angle))
    return angle

def angle_index_finger2(human):
    pnts = get_angle_point(human, 'index_finger2')
    if len(pnts) != 3:
        print('component incomplete')
        return

    angle = 0
    if pnts is not None:
        angle = angle_between_points(pnts[0], pnts[1], pnts[2])
        print('index_finger2:%f'%(angle))
    return angle

def angle_middle_finger2(human):
    pnts = get_angle_point(human, 'middle_finger2')
    if len(pnts) != 3:
        print('component incomplete')
        return

    angle = 0
    if pnts is not None:
        angle = angle_between_points(pnts[0], pnts[1], pnts[2])
        print('middle_finger2:%f'%(angle))
    return angle


def angle_ring_finger2(human):
    pnts = get_angle_point(human, 'ring_finger2')
    if len(pnts) != 3:
        print('component incomplete')
        return

    angle = 0
    if pnts is not None:
        angle = angle_between_points(pnts[0], pnts[1], pnts[2])
        print('ring_finger2:%f'%(angle))
    return angle


def angle_little_finger2(human):
    pnts = get_angle_point(human, 'little_finger2')
    if len(pnts) != 3:
        print('component incomplete')
        return

    angle = 0
    if pnts is not None:
        angle = angle_between_points(pnts[0], pnts[1], pnts[2])
        print('little_finger2:%f'%(angle))
    return angle
```

```
    # Process and display images
    for imagePath in imagePaths:
        datum = op.Datum()
        imageToProcess = cv2.imread(imagePath)
        datum.cvInputData = imageToProcess
        opWrapper.emplaceAndPop(op.VectorDatum([datum]))

        print("Hand keypoints: \n" + str(datum.handKeypoints))

        human_count = len(datum.handKeypoints)

        for i in range(human_count):
            for j in range(25):
                print(datum.handKeypoints[i][j][0])

        for i in range(human_count):
            angle_thumbs_up1(datum.handKeypoints[i] )
            angle_index_finger1(datum.handKeypoints[i] )
            angle_middle_finger1(datum.handKeypoints[i] )
            angle_ring_finger1(datum.handKeypoints[i] )
            angle_little_finger1(datum.handKeypoints[i] )
            angle_thumbs_up2(datum.handKeypoints[i] )
            angle_index_finger2(datum.handKeypoints[i] )
            angle_middle_finger2(datum.handKeypoints[i] )
            angle_ring_finger2(datum.handKeypoints[i] )
            angle_little_finger2(datum.handKeypoints[i] )

        if not args[0].no_display:
            cv2.imshow("OpenPose 1.7.0 - Tutorial Python API", datum.cvOutputData)
            cv2.imwrite("{}.jpg".format(time.time()), datum.cvOutputData)
            key = cv2.waitKey(15)
            if key == 27: break

except Exception as e:
    print(e)
    sys.exit(-1)
```

Annex B.6 Script "UDPReceive.cs"

```
using UnityEngine;
using System;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.Threading;

public class UDPReceive : MonoBehaviour
{


    Thread receiveThread;
    UdpClient client;
    public int port = 5052;
    public bool startRecieving = true;
    public bool printToConsole = false;
    public string data;



    public void Start()
    {


        receiveThread = new Thread(
            new ThreadStart(ReceiveData));
        receiveThread.IsBackground = true;
        receiveThread.Start();
    }

    private void ReceiveData()
    {


        client = new UdpClient(port);
        while (startRecieving)
        {

            try
            {
                IPEndPoint anyIP = new IPEndPoint(IPAddress.Any, 0);
                byte[] dataByte = client.Receive(ref anyIP);
                data = Encoding.UTF8.GetString(dataByte);

                if (printToConsole) {print(data);}
            }
            catch (Exception err)
            {

                print(err.ToString());
            }
        }
    }
}
```

Annex B.7 Script "HandTraking.cs"

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

public class HandTracking : MonoBehaviour
{
    // Start is called before the first frame update
    public UDPReceive udpReceive;
    public GameObject[] handPoints;
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        string data = udpReceive.data;
        data = data.Remove(0, 1);
        data = data.Remove(data.Length-1, 1);
        print(data);
        string[] points = data.Split(',');
        print(points[0]);



        //0        1*3      2*3
        //x1,y1,z1,x2,y2,z2,x3,y3,z3

        for ( int i = 0; i < 42; i++)
        {

            float x = float.Parse(points[i * 3]) / 20;
            //Can use 3-this value is to find the boundary value to put our hand in the centre
            float y = float.Parse(points[i * 3 + 1]) / 20;
            float z = float.Parse(points[i * 3 + 2]) / 8;

            handPoints[i].transform.position = new Vector3(x, y, z);

        }


    }
}
```

Annex B.8 Script "LineCode.cs"

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

public class HandTracking : MonoBehaviour
{
    // Start is called before the first frame update
    public UDPReceive udpReceive;
    public GameObject[] handPoints;
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        string data = udpReceive.data;
        data = data.Remove(0, 1);
        data = data.Remove(data.Length-1, 1);
        print(data);
        string[] points = data.Split(',');
        print(points[0]);



        //0        1*3      2*3
        //x1,y1,z1,x2,y2,z2,x3,y3,z3

        for ( int i = 0; i < 42; i++)
        {

            float x = float.Parse(points[i * 3]) / 20;
            //Can use 3-this value is to find the boundary value to put our hand in the centre
            float y = float.Parse(points[i * 3 + 1]) / 20;
            float z = float.Parse(points[i * 3 + 2]) / 8;

            handPoints[i].transform.position = new Vector3(x, y, z);

        }


    }
}
```

Annex B.9 Script "GameManager.cs"

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using ToolManager;
public class GameManager : Singleton<GameManager>
{
    public AudioClip[] allClips;
    public AudioSource audio;
    // Start is called before the first frame update
    void Start()
    {

    }
    public void playAudio(int  index) {

        audio.clip = allClips[index];
        audio.Play();
    }
    // Update is called once per frame
    void Update()
    {

    }
}
```

Annex B.10 Script “Piano.cs”

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

public class Piano : MonoBehaviour
{
    public int id;
    public int ThisOne;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
    private void OnTriggerEnter(Collider other) {
        if (other.tag == "1") {
            GameManager.Instance.playAudio(id);
            GetComponent<MeshRenderer>().material.color = Color.red;
        }
    }
    private void OnTriggerExit(Collider other) {
        if (other.tag == "1")
        {
            if(ThisOne==0)
            GetComponent<MeshRenderer>().material.color = Color.white;
            else
            {
                GetComponent<MeshRenderer>().material.color = Color.black;
            }
        }
    }
    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.tag == "1") {
            GameManager.Instance.playAudio(id);
            GetComponent<MeshRenderer>().material.color = Color.red;
        }

    }

    private void OnCollisionExit(Collision collision)
    {
        if (collision.gameObject.tag == "1")
        {
            GetComponent<MeshRenderer>().material.color = Color.white;
        }
    }
}
```