COMP 4801 Final Year Project [2023/24]

FYP 23008

Final Report

Exploring the Application of Machine Learning Models to Achieve Advanced Character Control and Improve the Naturalness and Immersion of Virtual Characters in Games.

Yu Ching Lok

Supervised by Dr. Choi, Yi King

Abstract

Currently, AI-based avatar control methods usually result in imprecise body movements and limited emotional expression, leading to a lack of natural interaction between the user and the avatar. While to achieve precise control of avatars, motion capture systems and devices are usually required. To enhance the interaction between users and avatars without the need for extensive equipment, we propose to utilize multiple artificial intelligence models for advanced avatar control. Our application combines these models to enable precise control of avatars. It consists of two programs: one dedicated to AI-related tasks such as landmark detection, emotion recognition, and gesture recognition; and the other focusing on controlling avatars using Unity's built-in functionality. During runtime, these programs run concurrently and communicate through pipeline. The results of the AI detection were used to control the avatar's body movements, facial expressions, emotional expressions, and to trigger specialized actions. We conducted research and development to create these AI models and avatar control mechanism and integrated it into a preliminary pipeline application providing precise control of virtual characters for naturalistic and realistic interactions. However, there are limitations due to the characteristics of the AI model used in the approach. Regardless of its limitations, our project demonstrates the potential of AI models for controlling virtual characters in games and other virtual environments. This advancement paves the way for more immersive and realistic experiences using AI models to control virtual characters.

Abbreviations

Abbreviation	Definition
AI	Artificial Intelligence
EAR	Eye Aspect Ratio
HaGRID	Hand Gesture Recognition Image Dataset
HSEmotion	High-Speed face Emotion recognition
MAR	Mouth Aspect Ratio
MLP	Multi-Layer Perceptron
ROI	Region Of Interest
TCP	Transmission Control Protocol
VCC program	Virtual Character Control Program
VRM	Virtual Reality Modeling Language
VTuber	Virtual Youtubers

Table of Contents

1. Introduction

As Artificial Intelligence (AI) technology continues to advance, its applications have expanded to various fields, including entertainment. One notable application of AI in entertainment is the emergence of virtual YouTubers (VTubers). These virtual characters are controlled by the user through landmark detection technology that captures facial expressions and body movements through camera input (Singh, 2023).

However, current methods of controlling avatars using artificial intelligence have certain limitations. Existing landmark detection models only allow for basic body movements and simple facial expressions. Expensive motion capture systems are required to enable more advanced control of character movements (Gank Content Team, 2023). Additionally, the process of changing an avatar's emotional expressions currently relies on manual input (Gank Content Team, 2023), resulting in somewhat unnatural expressions. These limitations restricted the interaction and naturalness between users and avatars.

To address these limitations, our project aims to explore how to integrate different AI models to control virtual characters, rather than just relying on landmark detection models. Specifically, we intend to combine emotion detection and gesture recognition models for more advanced control. By combining these AI models, we hope to provide users with a more unique and natural experience when controlling virtual characters.

We have developed an application that offers advance control over virtual characters. The application captures the user's movements and intentions and accurately translates them into movements of the virtual character. It enables efficient and automatic movement and action of the virtual character based on the user's intent. The backend of the application utilizes multiple AI models, each of which is responsible for controlling specific functions of the virtual character. In the front-end, we use Unity to facilitate the control of virtual characters in a virtual 3D environment. Notably, our approach eliminates the need for an expensive motion capture system, as the application only requires a single camera for input. Prioritizing affordability while maintaining a high level of control, we aim to provide a better user experience without significantly increasing costs.

The report will be divided into several sections, starting with a comprehensive overview of the background of the project. This will be followed by an overview of the objectives of the project and a clear statement of the deliverables to be achieved. In the subsequent sections there will be an in-depth discussion of the methodology adopted to develop the application. This will include a step-by-step explanation of the approach taken, highlighting the key processes and techniques used. Next, the report will present the results achieved by the project and analyze the overall results. At the end of the report, a comprehensive summary will be presented, outlining the key findings and outcomes of the project. In addition, possible future work and areas for further exploration will be discussed, paving the way for continued development and improvement.

2. Project Background

There are various ways to control virtual characters. Traditionally, virtual characters are controlled by triggering preset actions or animations using input devices such as a mouse, keyboard, joystick, or paddle controller (Lu, 2012). These methods usually involve mapping different keystrokes to specific character actions. However, this approach has its limitations as it relies on physical input from the user and restricts the range of actions and maneuvers that a virtual character can perform based on predefined mappings.

As technology advances, alternative methods have emerged that use sensors to track user actions and reflect them into virtual characters. Body tracking sensors, such as motion capture suits or wearable devices, capture the position and orientation of body parts, thus enabling virtual characters to accurately reflect the user's movements. However, using such sensor devices is both expensive and inconvenient (Wu et al., 2019).

With the rise of artificial intelligence, the technology for controlling virtual characters is also advancing. A notable example is VTuber, who use motion capture technology to control virtual characters. Motion trackers record user’s movements and behaviors, which are then mapped to the animated character in real time. This allows the virtual character to mimic the movements, expressions, and mouth movements of the virtual presenter. Technologies such as face tracking, hand tracking and motion capture all contribute to this process (Regis et al., 2022).

For 2D virtual character control, the Landmark detection model utilizes camera input to capture the user's facial features and basic body movements. Through simple facial tracking and motion capture, the acquired facial expressions and body movements can be applied to the virtual character, thus enabling the virtual character to reflect most of the user's facial expressions and basic body movements. However, this approach still relies heavily on pre-programmed commands to enhance expressiveness. Landmark detection models can only capture a limited amount of information about facial features and body movements, resulting in limited control over the facial expressions and movements of the virtual character, limiting the range of actions and behaviors that can be accurately reflected by the AI model. Users may need to physically trigger predefined animations to accomplish additional actions and manually select the desired emotional expression for the virtual character, as the AI model does not update these directly. This suggests that the landmark detection

model only enables partial control of the virtual character and that there is room for further improvement (Regis et al., 2022).

In addition, the previously mentioned methods may not be sufficient when controlling 3D virtual characters. Control in 3D often requires the use of additional motion capture devices (Regis et al., 2022). For example, Leap Motion controllers utilize infrared sensors and cameras to track hand movements and recognize gestures (Figure 1) (Ultraleap), Mocopi which equipped with accelerometer and gyroscope sensors that, combined with artificial intelligence, can accurately detect, and predict the user's 3D position and posture (Figure 2) (Sony). These devices provide additional information about the user's body movements. Combining them with facial tracking via landmark detection technology enables control of 3D virtual characters. This highlights the fact that landmark detection technology alone is not sufficient to control 3D virtual characters and that other technologies must be integrated to capture the user's body movements more accurately and precisely.

Moreover, traditional motion capture systems (including studio facilities and full-body suits with markers are still necessary for advance and precise control of 3D virtual characters (Figure 3). Unfortunately, such setups are costly and require multiple pieces of equipment to achieve the desired level of control (Regis et al., 2022).

To address the problem of expensive equipment requirements while enabling advanced control of virtual characters, we aim to explore the application of machine learning models (including deep learning) to eliminate the need for such expensive setups. By utilizing these models, we aim to enable advanced control of virtual characters without the need to rely on expensive equipment, relying only on a combination of AI models and camera input.

3. Project Objectives and Deliverables

This project has three primary objectives. First, the project aims to investigate how AI models can be applied to enhance the control of avatars in a natural and unique way. Second, the project aims to evaluate the impact of AI models on avatar control, user interaction and overall user experience. Finally, the project aims to provide advanced control of avatars without the need to rely on expensive motion capture systems, thus providing an economical means of achieving enhanced control of avatars.

To achieve these objectives, the project uses a range of AI models specializing in different aspects of avatar control. These include a landmark detection model for tracking body movements and facial expressions, a gesture classification model for recognizing gestures, and an emotion recognition model for recognizing user emotions. During runtime, these avatar-specific functions are combined to enable advanced control of the virtual character. At the end of the development process, a comprehensive evaluation is conducted to assess the impact of these models on avatar control, considering their impact on user interaction and overall user experience. Through these evaluations, the project seeks to determine the effectiveness of utilizing AI models to provide a more natural and unique avatar control experience.

The outcome of this project will be an application that integrates a variety of AI models to enable the user to have advanced control over the avatar and enhance the overall interaction between the avatar and the user. The application is expected to provide the following functionality:

1. Precise control of the avatar's body movements
2. Manipulation of the avatar's facial expressions
3. Express various emotions through the avatar
4. Execution of specialized actions by the avatar

By implementing these features, the application aims to give users a more immersive and engaging experience when interacting with virtual characters.

The following sections will focus on the practical details of the application, first describing the overall structure and workflow of the application, and then detailing what the program does and how it works.

3.1 Application

The application we developed consists of two main parts. The first part involves running an AI model to capture or predict the required information, while the second part focuses on projecting this information onto the virtual character.

In the AI model part, our main goal was to implement real-time motion capture, face tracking and hand tracking using camera input. Multiple AI models are run simultaneously to generate the necessary 3D landmark information for the body, hands, and face, as well as gesture and emotion information. This output information will be used for further mapping and analysis.

In another part, we will focus on utilizing the obtained information and mapping it onto a virtual character. We perform transformations and computations based on the acquired information. This allows us to generate various control signals which are then applied to the virtual character. As a result, the virtual character can move, perform actions, and accurately express emotions based on user movements and intentions captured by the AI model.

3.1.1 Application workflow

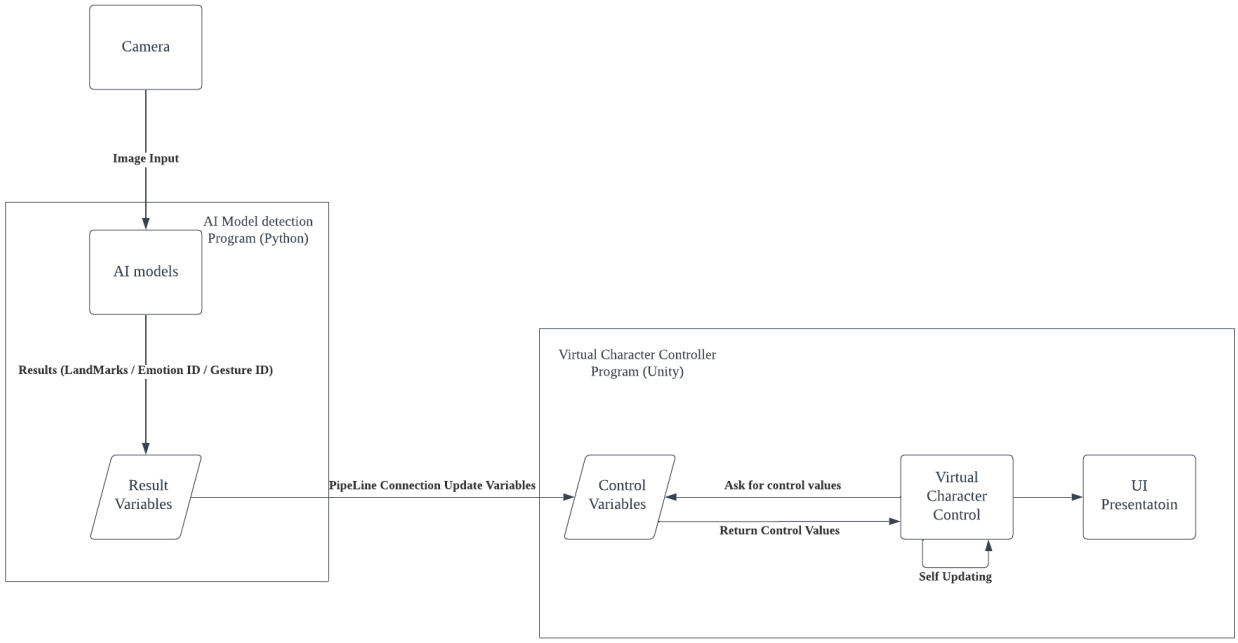


Figure 4: General workflow of application.

The application is developed as two separate programs: the AI program and the Virtual Character Control program (VCC program). The AI program, written in Python, is responsible for running the AI model, while the VCC program, written in C# with Unity being the backbone engine, is responsible for controlling the virtual characters.

During runtime, the two programs run in parallel and communicate with each other through pipeline via a server-client connection, with the VCC program acting as a server that continuously listens for incoming data in the pipeline and the AI program acting as a client that sends packets containing the results of the AI model to the server through the pipeline. The packets are encoded in JSON format. The AI program encodes the required information as JSON and passes it to the pipeline, while the VCC program decodes the received JSON file to obtain the required information.

The AI program continuously listens to the camera data stream during execution. When a new image input is received, it is provided as input to the AI model for detection. The results of the detection process are encoded and transmitted to the VCC program in the form of data packets. Upon receiving these packets, the VCC program calculates and updates its control signals, which affect and trigger different features of the virtual character (Figure 4).

3.1.2 AI Model Detection Program

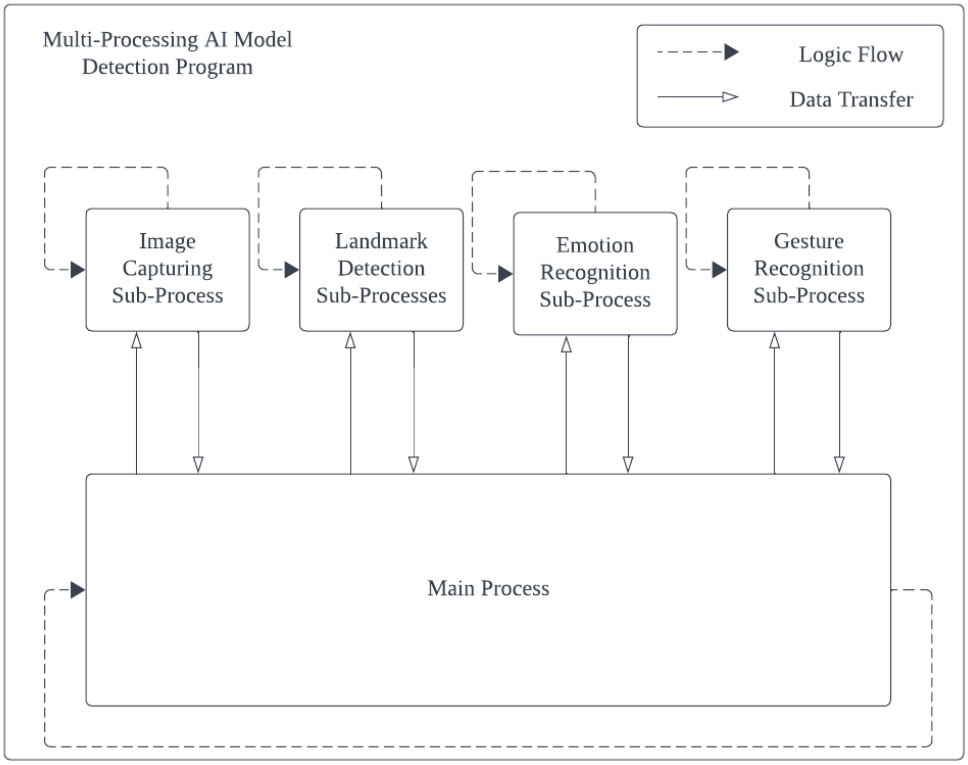


Figure 5: General structure of AI Model Detection Program.

The AI program is implemented in Python and focus only on tasks related to AI model detection. Python was chosen as the programming language because it is widely used and has good support for AI models development with extensive libraries and frameworks (Beklemysheva, 2022). The program was designed to run as a multi-process to ensure fast update rates for AI model detection and data transfer. The application is designed to reflect captured user actions and intentions onto virtual characters in real time.

In the AI program, a main process is responsible for the entire logic loop. It retrieves the raw input and transfers it to the sub-processes for detection. It also needs to retrieve the detection results from the sub-processes. The communication between the sub-processes and the master process is done through a queue and the AI detection results are encoded and transmitted to the server through a pipeline. Different AI models have different sub-processes, including camera image capture, emotion recognition, gesture recognition and landmark detection (Figure 5).

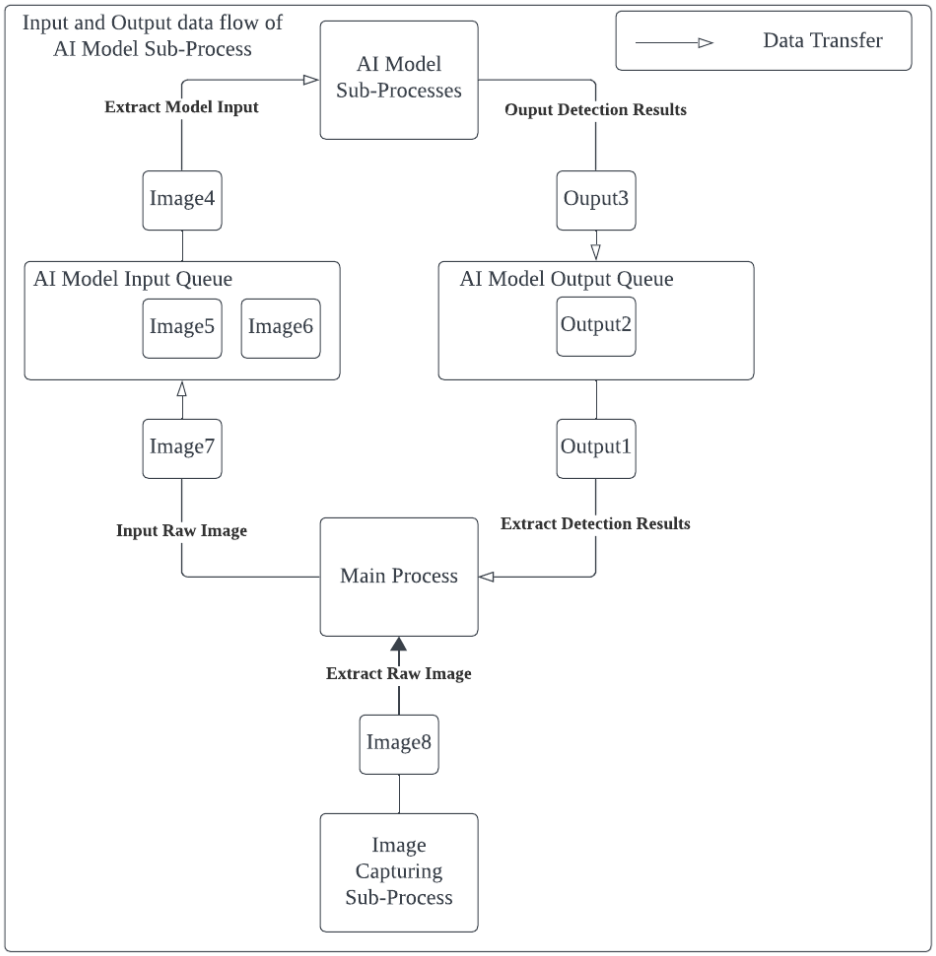


Figure 6: Input and Output data flow of AI Model Sub-Process.

The general idea behind the implementation is to optimize the update rate, considering that the inference time of an AI model might not be very short. Waiting for the previous detection to finish before initiating a new one could significantly decrease the overall update rate. To solve this problem, the camera capture process runs continuously on its own, capturing new images and placing them in a queue of finite size. When the queue is full, new input images crowd out the old ones, ensuring that the queue is constantly filled with newly captured images.

The main process then acquires images from the camera capture sub-process and transfers them to the landmark detection sub-processes via a shared queue. Each landmark detection sub-process continuously waits for input from the queue. After receiving a new image from the queue, it pops it out and performs the detection. When the detection is complete, the results are returned to another queue exclusively for the main process to obtain detection results. The main process checks the IDs of the obtained results to determine if they are from previous images. If so, they are

ignored; otherwise, the landmark detection results are obtained for further processing. This approach ensures that new images are fed into the AI model for detection as soon as they are acquired, eliminating, or minimizing any blocking operations, thus ensuring a fast update rate. Ideally, each image is directly integrated into the AI model without significant delays. In this scenario, the only limitations on the update rate would be determined by the hardware capabilities and the image capture rate of the camera (Figure 6).

The same approach applies to emotion and gesture recognition with the same principle. After executing all the AI models and obtaining the results, the relevant information is extracted and encoded into JSON packets, which are then passed into a pipeline for transmission to the server side. Once this logical loop is complete, it starts again. This parallelization approach allows for all process to be updated and run at the same time, achieving update speeds of about 30 frames per second or faster when running multiple AI models in parallel (Figure 5).

3.1.3 Virtual Character Control Program

The VCC program was written in C# using Unity as the main engine. It is responsible for controlling and displaying the 3D avatar. The decision to use Unity as the platform and C# as the programming language was made primarily because of Unity's advanced support for 3D avatar control with its built-in animation system, character control system and physic engine (NeuroSYS, 2023). The program consists of three main parts, the first is the pipeline communication with the AI program, the second is the calculation of control signals based on the AI model detection results, and the last is the control of the virtual avatar.

The VCC program acts as a server for the AI program. This server continuously listens for the latest results sent by the AI program through pipeline. The control signals of the avatar are then computed and adjusted based on these received results. The modification of the control signals relies heavily on the results detected by the AI model, which help determine the adjustments required by the avatar. For example, the rotation vector responsible for the rotation of the avatar's skeleton is derived from landmark data results. In addition, emotion IDs were mapped to specific modifications of facial features, while gesture IDs trigger corresponding animations. These aspects will be further elaborated later in this report. This iterative process ensures that captured actions and intentions are accurately transferred from the AI model to the VCC program. By continually adjusting control values and updating the avatar, the body

movements, facial expressions, and emotions of user can be accurately and efficiently reflected on the virtual character, ensuring consistency with the intended image (Figure 4).

To implement virtual character control, we first need to obtain a 3D humanoid model. In our program, we use an application called “Vroid Studio”, which allows the user to create custom 3D models of humanoid avatars. The main reason we chose to use “Vroid Studio” is that it is free, easy to use, and allows us to customize and export the 3D humanoid models we want. It has a user-friendly interface that allows us to easily modify and create the 3D humanoid models (Pixiv Inc.). The generated model will be exported in the Virtual Reality Modeling Language (VRM) format, a platform-independent file format designed for 3D characters and avatars in modern VR environments (VRM consortium Inc.) (Figure 7).

To import the humanoid model into Unity, we use a specialized extension package called UNI-VRM. This package helps to import VRM models into Unity, converting them into Unity assets equipped with existing functionality and scripts. Using this package, we were able to control the movement, animation, and facial expressions of the humanoid models within the Unity environment, utilizing the built-in functionality. (GitHub - VRM-C/UNIVRM) (Figure 8).

After importing the 3D humanoid model into Unity, we will utilize Unity’s built-in features to control the virtual avatar. For skeletal movement, we use the Animated Rigging package to apply rigging to the humanoid model to achieve realistic skeletal movement (Figure 9).

Additionally, for facial and emotional expression, we utilized the skin mesh rendering functionality inherent in humanoid modeling assets. Utilizing this feature of Unity, we were able to dynamically deform the skin mesh of the avatar at runtime. Unity then renders the modified mesh, allowing us to effectively control the avatar’s facial movements and emotions (Figure 10).

By using these extensions and taking full advantage of their functionality, we should be able to effectively control the movements, facial expressions, and emotion expressions of our avatars to create immersive and realistic virtual character experiences.

3.1.4 Summary of Application Structure and Workflow

To sum up, our application aims to enhance control over avatars by utilizing AI models and providing a unique user experience. By integrating multiple AI models, we can provide precise and naturalistic control over avatars. The application we developed consists of two main programs: a multiprocessing Python program that runs the AI models in parallel, and a Virtual Character Control (VCC) program written in C# using Unity. the Python program handles the AI model detection, emotion recognition, gesture recognition, camera image capture, and data transfer. And the VCC program uses the AI model results received from the Python program to control virtual characters in Unity-powered virtual worlds. This integration creates an immersive and engaging experience for the user as they interact with the virtual characters.

4. Project Methodology

The implementation of effective control mechanisms is essential to achieve advanced control over virtual avatar and to enhance the immersive user experience. Seamless interaction between the user and the avatar can have a significant impact on the overall experience for both the user and the audience. Furthermore, since the application is real-time, a fast update rate is required to ensure smooth rendering of the avatar's movements. Therefore, the application must accurately capture the user's movements and intentions from the camera and reflect them effectively in the avatar. This emphasizes the key factors that need to be considered when researching and developing AI models and control mechanisms for avatars, such as responsiveness, consistency, and reliability. In this section, we present methods for implementing advanced control of avatars considering the above factors. We will divide our discussion into two main sections: AI modeling and the virtual character control mechanism.

4.1 Methodology of AI Modelling

In our project we had to implement face tracking, hand tracking and motion capture. To implement these features, we used artificial intelligence modeling in the form of landmark detection models, emotion recognition models, and gesture recognition models. This section will provide an in-depth discussion of the selected models to enable advanced control of the avatar. We will discuss the reasons for the selection of these models, the development methodology, the functionality of these models, and how they can be effectively utilized to achieve the desired functionality.

4.1.1 Landmark Detection

To achieve advanced control of avatar movements and facial expressions, we perform precise motion capture and facial tracking through landmark detection AI models. Specifically, we focus on detecting hand, pose, and facial landmarks. We have primarily chosen the Mediapipe holistic model for this task, and several factors influenced our decision.

First, our application relies exclusively on a stream of camera images as input. Therefore, we needed a landmark detection model that could process the images efficiently, and Mediapipe fit the bill perfectly.

Second, our project requires AI models to detect hand, pose, and facial landmarks. Running separate models for each individual landmark detection puts a heavy workload on the hardware and may affect the inference time of each model. Therefore, it is critical to consider the workload of different AI models, especially when multiple models are running simultaneously. Fortunately, Mediapipe provides a pipeline detection model that integrates all three landmark detection models into one lightweight model. This integration greatly reduces the computational load and processing time of the device.

At the same time, Mediapipe has proven to be both lightweight and efficient, recognizing facial, postural, and hand landmarks in real-time, even on average-sized devices and web browsers. For example, it achieved a frame rate of 20 FPS on devices such as the Samsung S9+ (Grishchenko & Bazarevsky, 2020), demonstrating consistent and efficient performance in providing accurate and reliable landmark detection results.

By utilizing the Mediapipe holistic model, we can efficiently detect face, hand, and body landmarks using a single lightweight efficient pipeline model. This approach significantly reduces the computational effort and processing time of the device while still providing accurate and reliable landmark detection results.

The Mediapipe holistic model is an integrated pipeline that combines separate models optimized for posture, face, and hand components to enable advanced detection and tracking capabilities. The detection process consists of several key steps.

First, the model utilizes BlazePose's pose detector and an associated landmark model to estimate human pose, which is captured by a set of landmarks. Based on the pose landmarks, the model identifies three regions of interest (ROIs) for each hand and each face. To improve the accuracy of the ROIs, the model can adjust the cropping of the input frames. Once the ROIs are identified, the model crops the input frames based on these regions and applies task-specific face and hand models. This allows for accurate estimation of facial landmarks (including 468 landmarks) and hand landmarks (21 landmarks per hand).

By combining pose, facial and hand landmarks, the model obtains a comprehensive set of 543 landmarks. This set includes 33 postural landmarks, 468 facial landmarks, and 21 hand landmarks for each hand. Importantly, these landmarks are represented in 3D space and contain a visibility variable indicating the visibility of

a specific body part (Figure 11).

After obtaining 3D landmarks and their visibility information, we can post-process and map these landmarks to effectively control the movements and expressions of the virtual character. Pose landmarks can be used for motion capture, enabling us to capture real-world body movements of user and reflect them on the virtual character. On the other hand, facial landmarks are used for facial tracking, enabling us to control and animate facial expressions. In addition, hand landmarks play a crucial role in gesture recognition and help in recognizing and interpreting gestures. Details about mapping landmark features to control values to manipulate virtual characters will be discussed in later chapters.

4.1.1.1 Instability of Landmark Features Estimation

The Mediapipe holistic model provides three-dimensional landmark data, including x, y, and z-axis coordinates. Z-axis specifically represents the depth between the body and the camera. While Mediapipe is effective at recognizing landmarks with a relatively fast update rate, the stability of detected landmarks is sometimes inconsistent.

One notable issue is that landmarks detected on all axes are jittery. Even if the human body remains stationary in the image, there are slight variations in the landmarks detected each time. While these differences may not be noticeable when drawing the image, they have a significant impact on controlling the movements and expressions of the avatar.

This instability becomes especially problematic when the rotation vector is computed directly from the landmark detection results. Changes in landmarks can introduce fluctuations in the rotation vector, which can lead to unnatural and unusual body movements of the avatar (Figure 12). Similarly, when controlling facial expressions such as eye and mouth movements, fluctuations in landmark detection can have a direct impact on metrics such as mouth aspect ratio (MAR) and eye aspect ratio (EAR). As a result, avatars may constantly blink, open their eyes and move their mouths, further exacerbating the problem (Figure 13).

These small fluctuations in landmark detection are projected onto the control signals, leading to the previously mentioned problem of jitter during avatar motion.

This inconsistency contradicts the natural and smooth motion we wish to achieve when controlling avatars.

Unfortunately, since the model is not open source, it cannot be modified or fine-tuned to enhance stability. However, one way to mitigate this problem is to apply filters to the obtained landmark results. By filtering out noise and fluctuations, we can obtain more stable landmark data. This helps to reduce wobbling and provides smoother control values for virtual characters.

4.1.1.2 Filters

During the execution of landmark detection in real time, we obtain a series of continuous landmark data that can be considered as time series data suitable for the application of filters. In this case, the low-pass filter and the sliding window filter seem to be two suitable filters. The principle is to use previous landmark data to smooth out fluctuations in future results.

A low-pass filter allows low-frequency signals to pass through while attenuating high-frequency signals (GlobalSpec.). By using this filter, rapid changes in landmark data can be minimized, resulting in a smoother and more stable landmark data output. On the other hand, a sliding window filter is a digital signal processing technique used to analyze data within a moving window or time interval (Lesti & Spiegel). By combining and averaging previous and current landmark data, small fluctuations in the time series data can be eliminated, resulting in more stable landmark detection results.

Both filters aim to achieve similar results by reducing fluctuations, but the sliding window filter proved to be more suitable for our case (Figure 14). The main reason for this is its flexibility. While a low-pass filter can handle a variety of situations by adjusting the cutoff frequency, our time-series data typically stayed in the frequency range around 30 Hz, making it challenging to find an optimal cutoff frequency. Setting the cutoff frequency too low filters out all signals and gives the impression that the landmarks are not constantly and rapidly updated. Setting it too high has no effect. The optimal position is around 15 Hz, but there is little difference in overall performance between 10 and 20 Hz, with other frequencies having a relatively large negative impact.

In contrast, the sliding window filter is more effective. The main tuning parameter for this filter is the window size, and our best solution is to set the window size to 3 to 5. Increasing the window size further will average out the fluctuations better, or even

eliminate them completely. However, setting the window size too large also averages out important variations in the landmark data, which is undesirable. a window size of 3 to 5 effectively smooths out most of the fluctuations while retaining important variations. Furthermore, each window size in this range has a different impact on the final visualization.

Taking these factors into account, we believe that the sliding window filter is better suited to this situation, and it is now the default filter in our application. However, we also provide users with the option to choose between no filter, low-pass filter or sliding window filter, allowing them to customize the filtering method to their specific requirements. By applying filters to the acquired landmark data, we can obtain a more stable control signal and eliminate the problem of fluctuating landmark data (Figure 15).

4.1.1.3 Inaccurate and inconsistent Z-axis estimation

In addition to the instability of landmark feature estimates, another important issue with Mediapipe is the inaccuracy and inconsistency of z-axis estimates. While the overall model predicts fairly accurate x- and y-axis results, the estimation of the z-axis is problematic. While it can estimate z-axis values for different landmarks to some degree, its accuracy and consistency cannot be guaranteed. Fluctuations in this axis are particularly problematic, and the raw values themselves are inconsistent.

Applying this Z-axis data directly to calculate control values is not an ideal solution, as the instability and inconsistency of the estimated values can lead to strange motion control of the virtual character. However, we can still extract useful information from the Z-axis depth approximations from different landmark results. The exact values may not be reliable, but the overall magnitude is correct.

To solve this problem, we scaled the Z-axis values of all landmark data by a factor of 0.1. This reduces the size of the values and makes fluctuations and inconsistencies less prominent. This preserves the z-axis information while minimizing the negative impact. By testing these scaled values in conjunction with x-axis and y-axis values for controlling virtual characters, we found the overall performance to be stable and satisfactory. The scaled z-axis values did not result in the strange movements that occur with direct integration, and the z-axis variations of the different body parts were still reflected in the virtual character.

While this solution may offset some of the Z-axis variations to a certain extent,

the overall changes can still be reflected in the virtual character. Considering its stability, the smoothing phenomenon is within acceptable limits and negligible.

4.1.2 Gesture Recognition

Traditionally, triggering specific preset inputs is achieved physically by pressing buttons or using other external devices. These input devices are mapped to trigger different preset animations or actions so that the user can select different actions performed by the virtual character according to his or her intentions. While mapping landmark data to control a virtual character's body movements enables most actions or maneuvers, there may still be specific movements in a game or virtual world that are difficult to replicate in real life, such as cartwheels or backflips.

To provide a more comprehensive control system for virtual characters, we can explore the use of gesture recognition. The idea is to map different preset movements and animations to specific gestures captured by an AI model. In this way, the user can control the virtual character to perform the preset actions or animations simply by making the corresponding gestures, without the need for additional physical devices to trigger them. This approach allows users to seamlessly integrate their own movements and actions with the control of the virtual character without having to interrupt the original action to trigger a preset action (Figure 16).

By incorporating gesture recognition, we can provide a more intuitive and immersive control experience for users, who can trigger specialized preset actions by capturing their own intent through the camera. This approach enhances the control of virtual characters and provides users with a wider range of possibilities beyond the limitations of purely physical interactions.

We are implementing gesture recognition using a classification model and after considering various options, we have chosen to use a multilayer perceptron (MLP) model with hand landmarks as input. The model can classify 18 gesture categories. Our decision to use this architecture was based on a balance of performance and workload.

Since our application involves running multiple AI models at the same time, it is important to ensure efficiency, accuracy, and lightweight to prevent the device from being overloaded with work. Our goal was to maintain a balance between performance and workload, as heavy models reduce the update rate of the application,

which is undesirable in real-time scenarios. In addition, we want the AI model to accurately recognize target gestures. The MLP model we employ effectively achieves this balance.

By using hand landmark data as input, we aim to avoid the complexity of analyzing and making predictions based on raw hand images. Using raw images requires deeper and wider layers, such as convolutional layers with maximum pooling, to extract the underlying features and then perform MLP classification based on these extracted features. However, this approach leads to heavier models and higher computational requirements, which runs counter to our goal of pursuing efficient and lightweight designs.

So, to achieve an efficient and lightweight gesture recognition model, we explore the direct use of landmark data as input features to the MLP neural network. This approach allows us to obtain accurate and efficient gesture recognition without complex image analysis. Compared to images, landmark data is less complex, while MLP models have fewer parameters and lower computational requirements, resulting in faster inference times. Our main concern was whether the limited feature information provided to the neural network would lead to poor classification. However, we found that the MLP model was able to recognize gestures accurately and efficiently, validating the effectiveness of landmark data in this regard.

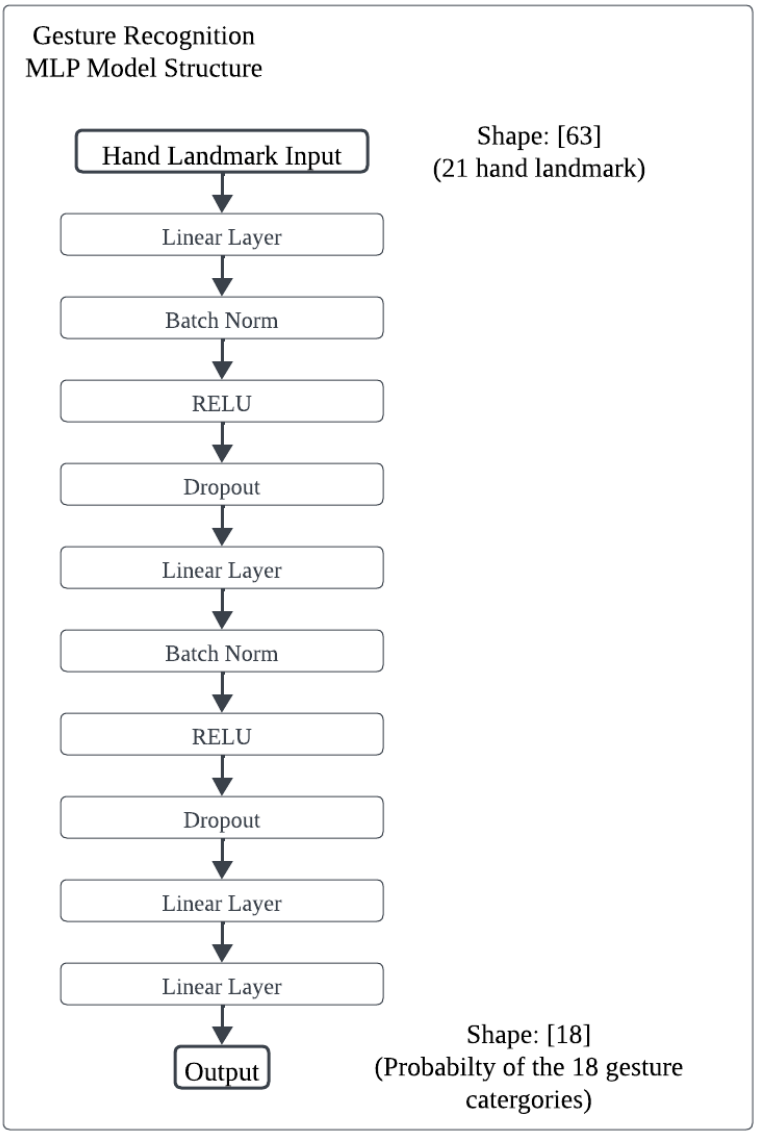


Figure 17: Structure of Gesture Recognition MLP model

Our MLP classification model consist of a 4-layers neural network. To improve the generalization and performance of this model on unseen data, we include Dropout Layers and Batch Normalization Layers. During training, the Dropout layer randomly deactivates neurons to prevent overfitting, while the Batch Normalization layer reduces internal covariate bias and stabilizes the training process.

During training, evaluation, and testing, we used the gesture recognition image dataset (HaGrid). This dataset contains 553,991 gesture images of the right and left hands, classified into 18 different gestures. The dataset contains images from 37,563 unique individuals between the ages of 18 and 65. These images were captured under a variety of lighting conditions, including artificial and natural light, reflecting the environment in which our application operates (Kapitanov, 2022). This diverse dataset

provides a range of suitable gesture images for training a model that meets our specific requirements (Figure 18).

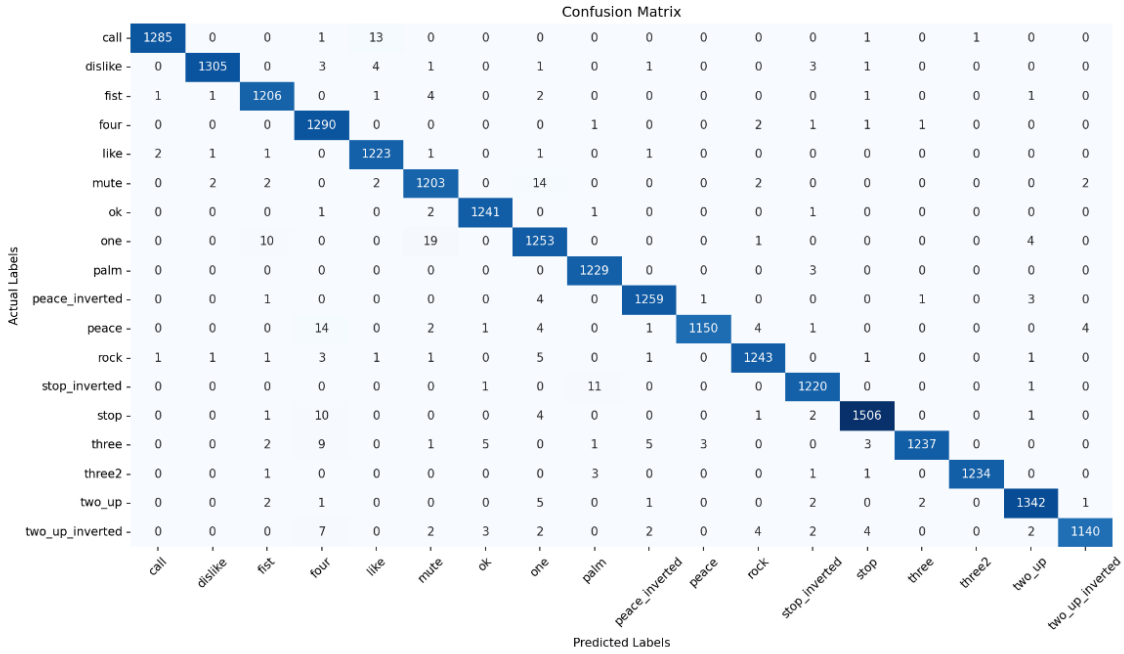


Figure 19: Confusion matrix of lefthand gesture recognition model.

.

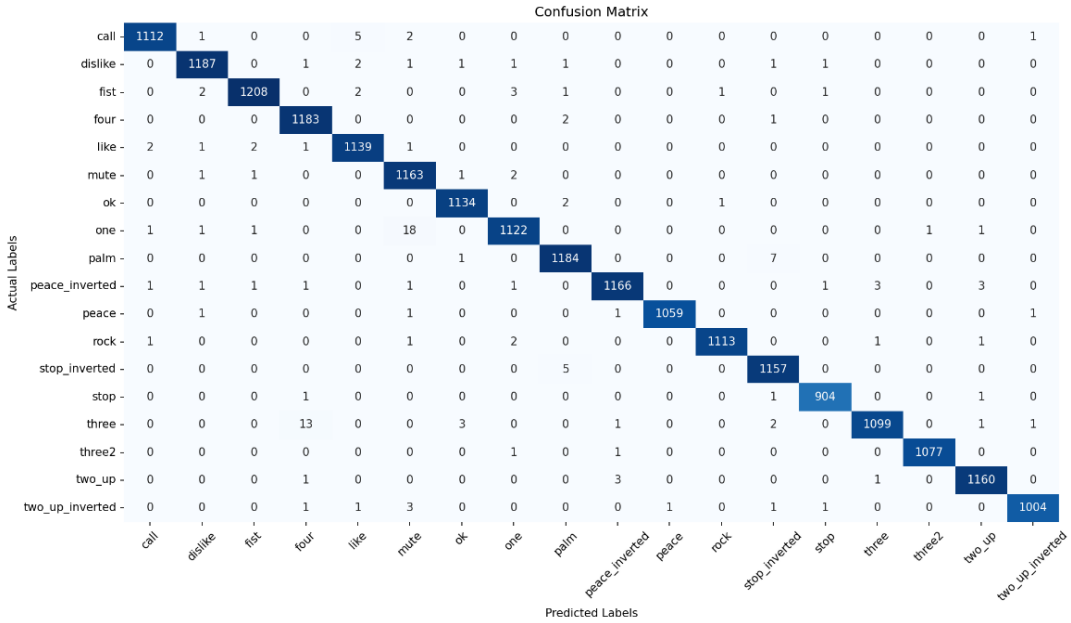


Figure 20: Confusion matrix of righthand gesture recognition model.

After extensive training, evaluation and testing, our model achieved impressive results. When running the gesture recognition model independently, the average

processing time per hand landmark detection was approximately 4.67e-05 seconds. The overall accuracy of the test dataset reached about 99%, with F1 scores, precision and recall values of about 0.99 for both hands (Figure 19 and 20). We also evaluated the real-time detection capability using hand landmark data extracted from actual web camera images. The results consistently show that the model can reliably recognize hand gestures with a high update rate. These results indicate that our gesture recognition model is lightweight, ensuring fast processing times while providing reliable and stable performance, and is therefore well suited for our application.

Using the self-developed MLP gesture recognition model, we can map the resulting gesture IDs to predefined animations or actions of the virtual character. Through this mapping, we can associate a specific gesture with a corresponding action that the virtual character can perform. In this way, we can trigger specific actions of the avatar through gestures. The integration of gesture recognition with virtual character behavior adds an interactive and immersive element to the user experience, enhancing the realism and interactivity of the virtual environment.

4.1.3 Emotion Recognition

Another important aspect of realizing a naturally realistic virtual character is controlling its emotional expression. Traditionally, in games, emotional expressions are either triggered by the system according to different scenarios or controlled by the user by selecting the desired emotional expression through a mapped input device. However, these approaches can be less convenient when it comes to real-time projection of user intentions and actions.

In the case of Vtubers, for example, they may experience a variety of emotions when engaging in a dialogue with an audience and want the virtual character to reflect these emotions. However, this usually requires the user to manually select the desired expression of the emotion and make changes in the virtual character accordingly (Regis et al., 2022). Unfortunately, this process usually results in an unnatural overall performance, as the user needs time to transition to the target emotion, resulting in a delayed emotional change. Furthermore, if the user fails to change the emotional expression, the user may show one emotion through their actions and speech while their virtual character's facial expression shows another emotion, resulting in a sense of inconsistency.

To address these challenges, we propose to automate the process by tracking

user emotions in real-time and automatically detecting current emotions. This can be achieved by utilizing an emotion recognition model that changes the virtual character's emotional expression accordingly. By tracking user emotions in real-time, we can ensure that the virtual character's emotional expression is aligned with the user's expected emotions, thus creating a more seamless and natural interaction.

With the success in utilizing landmark data for gesture classification via a simple MLP model, we proceeded to apply the same approach to emotion recognition. In this case, we used facial landmark data as input and connected it to the MLP model, which has the same structure as the model used for gesture recognition. However, the results for emotion recognition were not as satisfactory as for gesture recognition. The complexity of capturing and interpreting emotional expressions through facial landmarks seems to pose an additional challenge, despite using the same approach.

We utilize a scaled-down version of the Affectnet dataset to train and test our emotion recognition model. The original Affectnet dataset consists of approximate 400,000 manually labelled images representing eight facial expressions as well as emotions and arousal intensity (Mollahosseini et al., 2017). However, due to limited access to the full dataset, we chose the open-source version on Kaggle, which contains about 30,000 images. Of these, about 8,000 images were retained for testing and validation, and the remaining images were used for training.

We chose this dataset because it provides a wide range of head RGB images with different emotions captured under various environmental conditions and therefore suited our specific requirements. To generate the required facial landmark data, we used the Mediapipe holistic model and stored the data in CSV files. Post-processing steps, such as normalization and standardization, were performed on the data before it was used for training, evaluation, and testing.

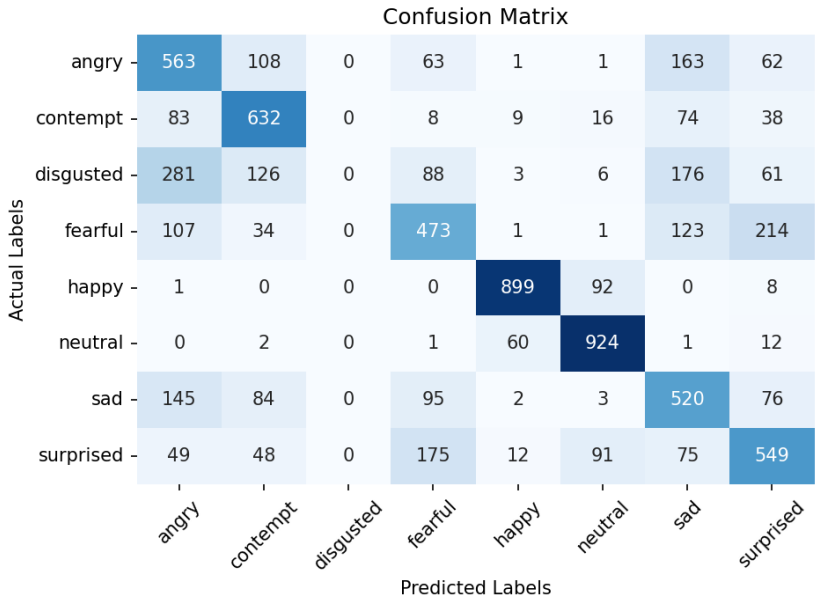


Figure 21: Confusion matrix of MLP emotion recognition model.

	precision	recall	f1-score	support
0	0.46	0.59	0.51	961
1	0.61	0.73	0.67	860
2	0.00	0.00	0.00	741
3	0.52	0.50	0.51	953
4	0.91	0.90	0.90	1000
5	0.81	0.92	0.87	1000
6	0.46	0.56	0.51	925
7	0.54	0.55	0.54	999
accuracy			0.61	7439
macro avg	0.54	0.59	0.56	7439
weighted avg	0.56	0.61	0.58	7439

Figure 22: Performance of MLP emotion recognition model.

After extensive training, evaluation, and testing, our MLP emotion recognition model achieved an overall accuracy of about 60%. However, performance varies across emotion categories, with emotions such as Happy and Neutral being better recognized with F1 scores, Precision, Recall and Accuracies above 0.8. Other emotions perform relatively poorly, with average scores of about 0.5. It is noteworthy that the emotion 'Disgust' is frequently misclassified. The complexity of the dataset poses a challenge for accurate emotion classification, as evidenced by the state-of-the-art performance on Affectnet, which also hovers around 60% accuracy (Figure 21, 22 and 28). Given that our reduced dataset is incomplete, it is difficult to directly compare our results with those from state-of-the-art datasets. Nonetheless, the complexity of the dataset

poses a challenge for simple MLP models to accurately classify emotions based on the acquired facial landmark data.

Given the complexity of the dataset, we recognized that relying solely on facial landmark data to model emotion recognition may not be sufficient. For this reason, we turned to CNN-based models to analyze cropped head images rather than focusing solely on facial landmarks. We believe that this approach allows the network to better understand the underlying facial structures and patterns, thus extracting more meaningful features and improving classification results. While prioritizing model performance, we must also consider workload and inference time, as we aim to achieve real-time applications. Therefore, we chose the well-established CNN models ResNet and MobileNet, which strike a balance between computational efficiency and performance, in the hope of achieving more desirable results. We fine-tuned the pre-trained ResNet and MobileNet provided by pytorch with our dataset and modified them into emotion recognition models.

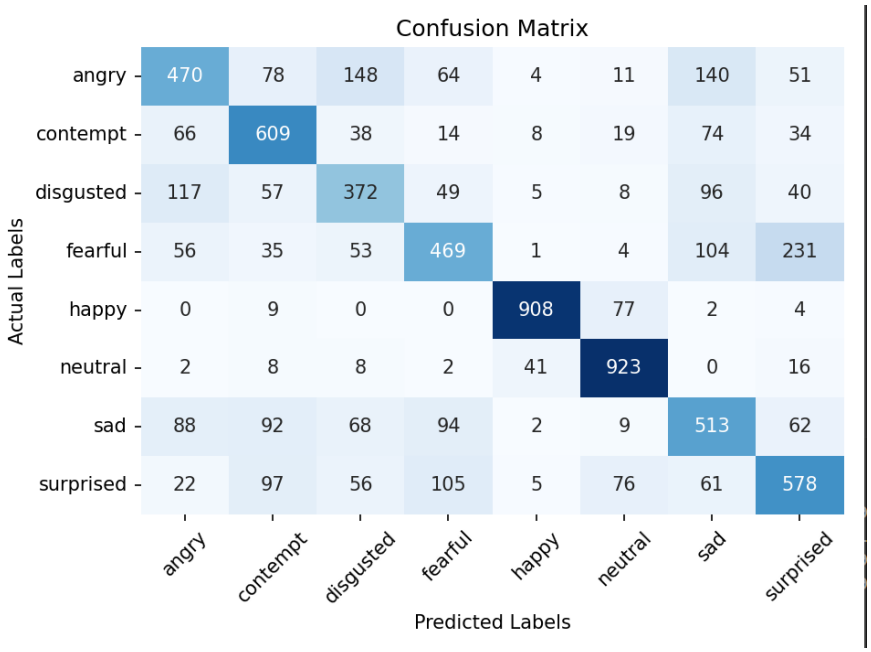


Figure 23: Confusion matrix of MobileNet emotion recognition model.

	precision	recall	f1-score	support
0	0.57	0.49	0.53	966
1	0.62	0.71	0.66	862
2	0.50	0.50	0.50	744
3	0.59	0.49	0.54	953
4	0.93	0.91	0.92	1000
5	0.82	0.92	0.87	1000
6	0.52	0.55	0.53	928
7	0.57	0.58	0.57	1000
accuracy			0.65	7453
macro avg	0.64	0.64	0.64	7453
weighted avg	0.65	0.65	0.65	7453

Figure 24: Performance of MobileNet emotion recognition model.

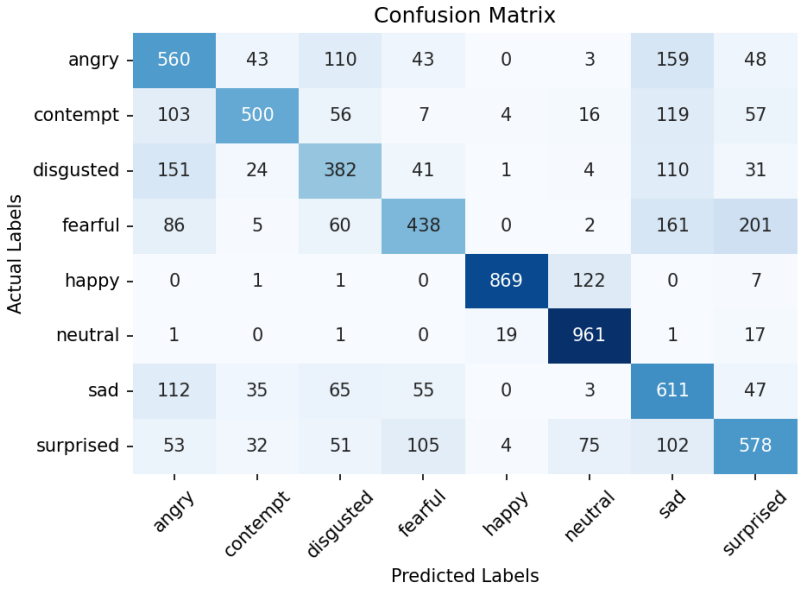


Figure 25: Confusion matrix of ResNet emotion recognition model.

	precision	recall	f1-score	support
0	0.53	0.58	0.55	966
1	0.78	0.58	0.67	862
2	0.53	0.51	0.52	744
3	0.64	0.46	0.53	953
4	0.97	0.87	0.92	1000
5	0.81	0.96	0.88	1000
6	0.48	0.66	0.56	928
7	0.59	0.58	0.58	1000
accuracy			0.66	7453
macro avg	0.66	0.65	0.65	7453
weighted avg	0.67	0.66	0.66	7453

Figure 26: Performance of ResNet emotion recognition model.

After extensive training, evaluation and testing, the results show that the CNN models do outperform the simple MLP model that relies only on facial landmarks. However, the magnitude of the improvement was not as dramatic as initially expected; the CNN model performed slightly better at recognizing emotions other than disgust, and significantly better at recognizing disgust itself. Specifically, the model achieved F1 scores, recall, and precision of 0.8 or better for emotions such as happy and neutral. However, the remaining emotions performed relatively poorly, with average scores around 0.5. The recognition rate of the “disgust” emotion improved to about 0.6 points. Overall, the model achieved an accuracy of about 66%. Although the improvement is small, it is still a significant improvement compared to the MLP model based on facial landmarks only. The results show that the CNN model is better at classifying emotions, but the phenomenon of more accurate recognition of specific emotions remains. This suggests that both models encounter similar challenges in fully understanding the underlying structure and patterns of emotions in the dataset (Figure 23, 24, 25 and 26).

In addition to testing the models on datasets, we also conducted real-time testing using a webcam for our application. Initially, we employed the Mediapipe holistic model to obtain facial landmark data. For the MLP emotion recognition model, we used these landmarks to detect the user's emotion. On the other hand, for the CNN-based emotion recognition model, we utilized the facial landmark data to identify the region of interest, crop out the head region, and post-process it into a 224x224 image. This image was then normalized and standardized before being fed into the CNN model for emotion recognition. This real-time process enabled continuous emotion detection as new webcam images were captured.

However, during testing, we encountered significant performance issues with all the models when using webcam images. They all intensively suffered from the same problem of only being able to recognize specific emotions while struggling with the recognition of other emotions. For instance, the ResNet model continuously recognized happy and neutral emotions, often misclassifying other emotions as anger. This phenomenon was also observed with the MobileNet model and our MLP model, where specific emotions were recognized while others were consistently misclassified as a particular emotion. The emotions that all models struggled to recognize include anger, contempt, disgust, fear, surprise, and sadness.

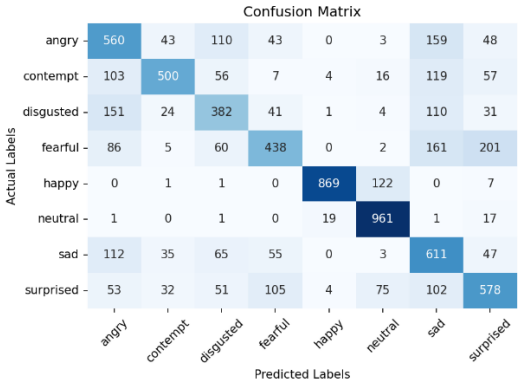
Figure 27: Images of different emotions detected by ResNet emotion recognition model.

Further investigation revealed that the models' relatively accurate recognition of “neutral” emotions could be attributed to capturing the contour of a typical human face without detecting any distinct features at that moment. Similarly, the more accurate recognition of “happy” could be attributed to the models capturing the movement and shape of the mouth when the user grin. The models also appeared to detect the opening of the mouth and movement of the eyebrows, as manipulating these facial features resulted in the models recognizing different emotions. However, despite capturing these features, the models were unable to accurately classify emotions based on them. The detected emotions were mostly incorrect or consistently misclassified as a specific emotion, such as anger (Figure 27).

Additionally, the models struggled to capture small variations in facial features, as manipulating these features did not result in the models detecting different emotions. This indicates that while the models could capture features from the face images, they struggled to classify emotions correctly when faced with subtle variations in facial expressions.

Overall, the real-time testing revealed significant limitations in the models' ability to accurately recognize a wide range of emotions from webcam images. The models captured some prominent facial features and larger variations in expressions but struggled to effectively utilize these features for accurate emotion classification, particularly with smaller variations in facial expressions.

Based on the results obtained, we believe that one of the reasons for the poor model performance is the domain bias of the dataset. Although the dataset we chose provides many RGB head images capturing different emotions in different environmental conditions, these images do not specifically represent the domain of the head images captured by the webcam we used at runtime. We suspect that there is a bias between our actual inputs and the dataset, which affects the accuracy and correctness of the model classification. In addition, we acknowledge that pre-training weights may introduce bias. Since the pre-trained weights are trained on ImageNet, they are not specific to the face input. While these weights help the model capture base features such as contours, they may introduce bias since they are trained on a different domain. However, pre-training a dedicated image classification model focused on faces is very challenging due to time and face image dataset availability constraints. Therefore, we decided to choose publicly available emotion recognition models rather than developing them from scratch. Ultimately, we chose the “High Speed Face Emotion Recognition” (HSEmotion) model. We chose HSEmotion for two main reasons. Firstly, it is one of the state-of-the-art models that has achieved good results in emotion classification tasks (Figure 28). Second, compared to other state-of-the-art models, which typically utilize attention mechanisms or visual transformers and weigh a considerable amount. HSEmotion employs a lightweight CNN model, which fully meets our requirement of achieving high classification accuracy while minimizing computational effort and processing requirements. According to the documentation, when using EfficientNet-B0 as the backbone, HSEmotion has an average inference time of about 59 ms and a variance of about 26 ms (Av-Savchenko & Andrey, 2022). Furthermore, it achieves an accuracy of 62% on the complete AffectNet dataset, which demonstrates its ability to classify emotions with reasonable accuracy and efficiency (Figure 29). With this in mind, we carried out testings and achieved satisfactory results.



	precision	recall	f1-score	support
0	0.53	0.58	0.55	966
1	0.78	0.58	0.67	862
2	0.53	0.51	0.52	744
3	0.64	0.46	0.53	953
4	0.97	0.87	0.92	1000
5	0.81	0.96	0.88	1000
6	0.48	0.66	0.56	928
7	0.59	0.58	0.58	1000
accuracy			0.66	7453
macro avg	0.66	0.65	0.65	7453
weighted avg	0.67	0.66	0.66	7453

Figure 30: Confusion matrix and Performance of HSEmotion model.

Figure 31: Images of different emotions detected by HSEmotion model.

We initially tested this on our dataset and the results were very similar to those of the fine-tuned CNN model. We achieved scores of around 0.8 for the emotions 'happy' and 'neutral', while the rest of the emotions scored around 0.5. What sets the model apart, however, is its performance in the real-time test. Overall, it performs significantly better than our fine-tuned CNN model. As shown in figure 30 and 31, the model accurately captures most of the emotions in the webcam images. It recognizes

subtle changes in facial features without significant changes. For example, a simple smile can trigger happy without the need for a grin, which is necessary in our own model. In addition, instead of associating open mouths only with anger, the model considers other facial features, allowing it to accurately distinguish between expressions of anger, surprise, and disgust. The overall performance of the model on webcam images greatly exceeds that of our previous model.

HSEmotion is an implementation of the article titled "Adaptive Frame Rate Facial Expression Recognition Based on Multiple Test Corrections". The core idea behind this implementation is to improve the performance of facial expression recognition while maintaining a balance between performance and accuracy. This is achieved by addressing the high computational complexity of video-based facial expression recognition by dynamically adjusting the frame rate at which video frames are processed.

By processing fewer frames of simpler videos at lower frame rates and more frames of complex videos at higher frame rates, the decision-making process can be accelerated without compromising accuracy. To determine the frame rate at which inference is reliable enough, this paper uses the Benjamin Hochberg procedure from multiple comparison theory. This procedure helps to control the false discovery rate and ensures the credibility of the decision-making process. With this adaptive frame frequency approach and the application of multiple testing corrections, a balance between performance and accuracy is maintained while improving performance (Figure 32) (V.Savchenko, 2023).

It is worth noting that the methodology presented in the article is applicable to any network, in this case, the authors utilized EfficientNet as the backbone network. In addition, they pre-trained the network using the VGGFace2 dataset, which is a dataset used for face recognition that contains a variety of variations including pose, age, lighting, ethnicity, and occupation (e.g., actors, athletes, politicians) (Qiong et al., 2018).

We believe that by using the adopted methodology and a network pre-trained on the face recognition dataset, the trained model can mitigate the potential domain bias mentioned earlier. Furthermore, by exploiting the ability of the adopted method to select high-quality data images, the model can self-learn and update itself in a much closer to optimal situation. In this way, state-of-art performance can be obtained even when using a lightweight CNN model as the backbone. This approach fits our

requirements very well and enables accurate facial emotion recognition using a lightweight model, so we finalize with HSEmotion as our emotion recognition model.

Finally, we use this consistent and reliable emotion recognition model to associate the generated emotion IDs with predefined blend shape weights representing the different emotions of the avatar. When the emotion recognition model detects a certain emotion, it generates the corresponding emotion ID, which is then transmitted to the VCC (Virtual Character Control) program, which then adjusts the blend shape weights according to the desired emotion, allowing the avatar to smoothly transition to the desired emotion. This mapping allows the virtual character to always display the same emotion as the user's facial expression, resulting in a more natural and realistic act of the virtual character.

4.2 Methodology of Virtual Character Control

Having accomplished facial tracking, hand tracking and motion capture, the next crucial step is to map the captured movements and expressions onto the virtual character. This process requires focusing on several key features: body movements, facial expressions, and emotions, as well as actions or animations of virtual character. Each feature requires the control of multiple body parts of the virtual character.

By combining the humanoid avatars provided by UNI-VRM and Unity's built-in features such as Animation Rigging, Skinned Mesh Renderer, and Animator. This allows us to control the virtual character by applying different control values to specific body parts or by directly triggering overall movement or animation. In this section, we'll discuss how we calculate the control signals used to control our avatars and outline the ways in how we use Unity's built-in features to implement body movements, facial expressions, emotional expressions, and animation triggers for our avatars. We'll also explore how these techniques allow us to achieve advanced control over our avatars.

4.2.1 Body Movement

To achieve control over the body movement of the humanoid model, we utilize the “multi-aim constraints” feature provided by the Animated Rigging extension. This feature will rotate a specified source target to point at the target game object. Using this feature, we can control the rotation and alignment of the humanoid model’s skeleton by aligning specific target game objects (Figure 33).

Our implementation consists of several steps. First, we assign the specified target game objects to each skeletal component of the humanoid rig. Next, we compute rotation vectors for each skeletal component based on pose landmark data obtained from the AI model detection program. These rotation vectors are indicators of desired rotation and movement, these rotation vectors are indicators of desired rotation and motion and are control signals that control the movement of the body (Figure 34).

To compute the rotation vectors, we measured the vectors between specific landmarks, such as between the left shoulder and the left elbow, to compute the rotation vectors of the left upper arm bone. By normalizing and scaling these vectors using predetermined coefficients, we obtained the necessary rotation vectors.

These rotation vectors are then applied to the corresponding target game objects. This results in a translation of the target game object relative to its original position, which further results in a rotation of the corresponding bones to point at the target game object, so we can effectively rotate and align the relevant bones of the humanoid model (Figure 35).

By utilizing “Multi-Aim Constraints” features and pose landmark detection results, we can accurately implement the necessary bone rotations and transformations on the actual humanoid model. Through this process, we can accurately transform the captured motions and body postures from the landmark detection model to the virtual humanoid model, ensuring that the virtual humanoid model effectively reflects the detected landmarks and faithfully reproduces the expected body postures and motions.

Although we can achieve necessary bone rotations and transformations on the humanoid model using pose landmark data, the sensitivity of the virtual character's body movements to variations of values in the landmark data can lead to fluctuations. We solved this problem by applying filters to reduce the fluctuations. However, in another scenario, landmark data can also fluctuate significantly and become inconsistent and unreliable.

This occurs when specific body parts are not clearly visible in the image, which requires the model to estimate their locations based on other visible body features. Since the landmark data for the non-visible body parts are estimated or predicted based on other features, the results become unreliable and inconsistent (Figure 36). If we include this data in the calculation of the rotation vector and applied it onto the

virtual character, it will directly affect the virtual character's body motion, resulting in undesirable fluctuations.

To overcome this challenge, we introduced a visibility threshold for the landmark data provided by the model. The visibility of the landmark data must exceed the threshold before the rotation vector is updated and calculated based on the current data and applied to the virtual character. By setting this threshold, the generated unreliable landmark data is ignored even if specific body parts are not clearly visible, thus preventing any strange movements of the virtual character. This approach ensures smooth control of the virtual character's body movements and enhances the overall experience.

Using the above techniques, we can control the movement and rotation of the virtual character's limbs and body to perform most user’s movements. However, due to the characteristics of facial landmark data, we need to adopt a different approach when controlling the movement and rotation of the head.

Unlike the limbs and body, which can be treated as a joystick-like case where they rotate and move along specific axes without translation, the head requires a different approach. The facial landmark data we obtain focuses on facial features rather than the center of the head. If we use the facial landmark data to compute the rotation vectors, there will be a bias as the center point is not accurately represented. This will lead to less than satisfactory results in controlling head movement and rotation.

When dealing with face landmark data, we have a much larger set of points including over 468 2D and 3D coordinates. This allows us to address this problem using the Perspective-N-Point (PnP) method, a computer vision technique for estimating the pose of a calibrated camera (Figure 37). In our case, the face landmark data provides both the necessary 3D points in the world and their corresponding 2D projections in the image, making it suitable for solving as a PnP problem.

By processing the face landmark data as a PnP (perspective point) problem, we can compute the rotation and translation vectors that transform the 3D points from the object coordinate system to the camera coordinate system. In our scenario, we are particularly interested in rotation vectors that can be extracted and further decomposed into pitch, yaw and roll angles. These angles represent rotations along the three axes of the head and can be used to control the head components directly using multi-rotation constraints.

Using “multi-rotation constraints” feature provided by the Animated Rigging extension, we can instruct the specified source target to mimic the rotation of the target game object. In this case, we can apply the computed rotation vector directly to the target game object so that the virtual character's head follows the same rotation. By utilizing multiple rotation constraints, we can establish a link between the computed rotation vector and the virtual character's head to ensure that it accurately reflects the rotation (Figure 38).

In fact, the complex math involved in calculating head pitch, yaw and roll can be simplified using the existing functions provided by the Python OpenCV library. These functions provide convenient solutions for obtaining the desired values with relative ease. Using these preset values, we can simplify the process of calculating rotation angles.

In addition, it makes sense to perform the calculation and coding of the roll, pitch, and yaw values in a Python program, considering that there are no similar libraries in Unity and that further importing of the libraries would be required, which can be very cumbersome. This approach allows for efficient calculations using existing resources and libraries. The calculated values can then be encoded into packets and seamlessly transferred to the VCC program to control the head movements of the virtual character.

To solve this PnP problem, we can take advantage of the cv2.solvePnP() function provided by the Python OpenCV library. With this function, we can estimate the object pose based on the 3D-2D point correspondence. In our case, since it is challenging to ensure the accuracy of the camera matrix and distance coefficients parameters in different scenarios, we use pre-determined values. The only variables that change are the object points representing the 468 3D landmark data and the image points representing the corresponding 2D landmark data. By supplying this data to the cv2.solvePnP() function, we can obtain the rotation matrix. Based on this, we can convert the rotation matrix into a rotation vector (Figure 39) using cv2.Rodrigues() function and perform an RQ decomposition to extract the x, y, and z-axis angles representing pitch, yaw, and roll, respectively by using cv2.RQDecomp3x3() function. Finally, the calculated head rotation can be transferred to the VCC program and used directly to control the head rotation of the avatar through multi-rotation constraints and the results are indeed quite smooth, consistent, and reliable. This shows that this method is feasible for controlling the head rotation of the virtual characters (Figure 40).

By integrating the PnP and animation rigging techniques, we can attain complete control over the 3D virtual characters' entire bodies. This fusion empowers users to execute desired actions seamlessly, ensuring a faithful and consistent reflection of their intentions onto the virtual characters.

4.2.2 Facial and Emotion Expression

To accurately control the facial and emotion expression of the avatar, we took advantage of Unity's built-in capabilities. First, we imported the humanoid model into Unity using the "UNI-VRM" extension and converted it into a Unity asset. The asset includes the humanoid 3D model as well as functions and scripts for modifying the model and applying animations or actions. We then use Unity's built-in functionality "Skinned Mesh Renderer" to control the facial and emotion expression of that Unity asset.

For facial and emotion expression control, we rely on the predefined Skin Mesh Renderer component of the imported humanoid model. Using this built-in feature, we can create deformable meshes and render them. The component sets several blend shape weight values that can be adjusted to modify the facial expression of the humanoid model dynamically. In addition, special blend shape weight value can be used to deform the entire face mesh to a specific expression. By manipulating these blend shape weight value, we can customize the avatar's facial and emotion expressions (Figure 41).

In terms of facial expressions, our focus is on controlling the opening of the eyes and mouth. To do this, we need to measure the degree of opening of these two features. A practical way to do this is to use aspect ratios, such as the "Mouth Aspect Ratio (MAR)" and "Eye Aspect Ratio (EAR)" shown in Figure 42. These ratios can be derived by calculating the distance between specific facial landmarks associated with the eyes and mouth. Typically, we consider landmarks such as the inner corner of the eye, the outer corner of the eye, the corner of the mouth, and the midpoints of the upper and lower lips.

In our case, we utilized the facial landmarks data generated by the Mediapipe holistic model. This provided us with the landmark data needed to calculate these aspect ratios, allowing us to determine the degree of eye and mouth opening. Once the EAR and MAR values are calculated, they are mapped to the appropriate weights of the blend shapes. These blend shapes control the degree of opening of the eyes and

mouth of the avatar, allowing us to dynamically deform the skin mesh based on the captured user expression.

While EAR and MAR can effectively measure mouth and eye opening, there is a challenge associated with the calculated values. Because the required landmark data are concentrated in specific regions of the image, the landmark values obtained tend to be very similar and do not show significant variation, especially for the landmarks involved in calculating EAR. This similarity of values results in the aspect ratio being less sensitive to changes and remaining within a limited range, rather than utilizing the entire 0 to 1 range.

To address this issue, we scaled the aspect ratios to amplify the values and make the results more sensitive to change. Through testing, we determined that multiplying the aspect ratio by a factor of 1.5 resulted in satisfactory performance. It is important to note that the scaling factor and bias factor may vary in different environments, but in general, a scaling factor of around 1.5 produces desirable results. This scaling factor strikes a balance between preventing excessive sensitivity, which can lead to persistent blinking (Figure 13), and able to accurately reflect the opening and closing of the eyes and mouth.

After obtaining the desired EAR and MAR values, we need to map them to a range of 0 to 100, which corresponds to the weighting values needed to control the blend shape. However, it is important to consider that there are practical limits to the weight values to avoid extreme deformations of the blend shape. Therefore, we used clipping or biasing to ensure that the aspect ratio is within an acceptable range to make visual sense.

Based on our observations, we found that the MAR is highly sensitive to mouth opening, so the weight values range from about 60 to 140. To limit the weight values to the desired range of 10 to 90, we applied a bias of -0.5 to the aspect ratio. This gives a good representation of the mouth tensor.

On the other hand, the EAR is less sensitive, with weights typically ranging from 30 to 80. However, the resulting visual output shows unusual eyelid movements. To address this issue, we biased the weighting values to -10 to ensure that the weighting values were between 20 and 70, thus effectively representing eye and mouth opening.

By applying these scaling and biasing techniques, we succeeded in limiting the

computed aspect ratios to an acceptable range. This approach ensures that the visualization of mouth and eye movements accurately reflects the opening and closing movements of the eyes (Figure 43).

Similarly, when it comes to emotion expression, we manipulate the weight values to convey the desired emotion. In this case, the emotion IDs generated by the emotion recognition model are passed to the VCC program and mapped to the appropriate blend of shape weight values corresponding to the emotion. Once a specific emotion is detected, the skin mesh is deformed to accurately depict the corresponding emotion on the virtual character's face (Figure 44).

When it comes to weight values related to emotions, the goal is to trigger specific expressions corresponding to different emotions. In this case, the resulting weight values do not necessarily have to be in the range of 0 to 100. Instead, we can directly assign a weight value of 100 to the target emotion weights, thus immediately changing the avatar's facial expression to match the emotion captured by the AI model.

However, when a new emotion is detected, a sudden transition to a weight value of 100 may appear unnatural and stiff. To solve this problem, we adopt a logic whereby when a new emotion is detected, the weight value is gradually increased to 100 over a short period of time. This gradual increase allows for smooth transitions between facial expressions, resulting in a more natural and visually pleasing transition from one specific emotion to another.

By employing this logic, we can observe a seamless transition of facial expressions between different emotions. The gradual increase in weight values is not an instantaneous change, but a smoother and more realistic representation of the changing emotional state.

By continuously calculating the EAR, MAR, and emotion ID for each frame and mapping them to the corresponding blend shape weights, we ensure that the skin mesh accurately reflects the expressions captured by the AI model. This allows for precise control of the avatar’s facial and emotional expression.

4.2.3 Specialized Action Triggering of Virtual Avatar

To enable users to perform actions on virtual characters that are difficult to perform in real life and to increase the number of ways to control the virtual character.

Here, we utilize gestures captured by the AI model to trigger specialized actions of the virtual character based on the user's intention.

To trigger specified action of the virtual Avatar, we started by acquiring the animation assets for the humanoid models. We acquired these animations from Mixamo, an online platform provided by Adobe that offers a wide range of pre-made 3D character models, animations, and rigging solutions (Adobe Systems Incorporated) (Figure 45). We then applied these downloaded animations to the imported humanoid models in the Unity environment.

To manage the transitions and playback of these animations, we used Unity's built-in animation system, known as "Animator". By using Animator, we built a finite state machine that is responsible for controlling the animation of the avatar. Each state in the state machine corresponds to a specific animation. By defining transitions between states, we can pinpoint when each animation is triggered based on the current state of the avatar. This approach allows us to seamlessly control the animations and synchronize them with the avatar's behavior and movements (Figure 46).

Next, we develop a controller program for the finite state machine that maps the gesture IDs generated by the gesture recognition model to the corresponding animation trigger IDs. When a specific gesture is detected, the controller program receives the animation trigger ID and changes the state of the animator to the corresponding state associated with the desired animation. As a result, the corresponding animation is triggered, and the avatar can perform the specialized action associated with the detected gesture.

By mapping a specific gesture to the corresponding animation, the user can trigger a specialized action by simply executing the defined gesture. This approach allows the user to seamlessly integrate their actions and behaviors with the control of the virtual character without the need to interrupt the original action to trigger a preset action, thus enhancing the control of the virtual character and providing the user with a wider range of possibilities beyond the limitations of purely physical interactions.

5. Results and Findings

Utilizing the mentioned methodology, we have successfully developed an application that allows advanced control of avatars by simply inputting camera images. After the development of the application was completed, we conducted thorough testing to evaluate its effectiveness in achieving advanced control of the avatar under different scenarios. Our testing aimed to determine the capabilities achieved by the approach we adopted and to reveal any challenges or issues associated with the current application. This section summarizes the findings and results obtained during our testing phase.

5.1 Natural Movement of Virtual Avatar

An important aspect of real-time controlling a virtual humanoid avatar is ensuring that its movements are natural. The movements of the character must be very similar to those of a real person, as any deviation will result in an unnatural visual presentation. This, in turn, would lead the audience to perceive the avatar as unrealistic, contradicting our goal of controlling the avatar to naturally reflect the user's movements.

Achieving "natural" movement in avatars encompasses several aspects, but it is challenging to consider each aspect holistically. Therefore, we focused on a few key aspects, particularly body movements, facial and emotion expression, and action triggering.

5.1.1 Body Movement

There is notable difference between 2D and 3D environments when it comes to controlling the avatar's body movements. In the case of 2D body movements, attempting to map the user's movements directly onto a 2D virtual character is not ideal because the user's movements occur in 3D space while the virtual character exists in a 2D realm. Direct mapping can lead to awkward movements. Therefore, this approach usually involves triggering different predefined actions to approximate the desired body movements, the focus is not on directly mirroring the actual user's body movements, but on depicting general body movements such as tilting the torso or limb movements.

This approach is also how 2D VTuber operate. In 2D body movements, only tilt movements are usually mapped, while other aspects are often ignored. It is challenging to accurately apply motion capture data of user movements to virtual characters, especially when they exist in different dimensions. Motion capture is mainly used to control general body movements rather than to achieve accurate mapping (Regis et al., 2022).

However, this approach also poses a significant challenge: it imposes restrictions on the movement of the avatar. Only specific features, such as tilting the body, are employed, while other aspects are not considered and need to be triggered manually, e.g., by pressing a mapping key to perform a hand movement or other action. As a result, the overall control over the virtual character is limited, only partially representing the user's actions, and resulting in restricted movement. This limitation greatly reduces the naturalness of the virtual character's behavior, which is not a desirable outcome. Based on these limitations and the desire to overcome them, we decided to abandon the implementation of 2D control of virtual characters in favor of a 3D approach.

When it comes to 3D body motion of a virtual character, the captured motion data can be mapped directly onto the virtual character. The quality of this mapping and the transformation of the acquired motion data largely affects the performance. It is also critical to ensure that the captured motion data is stable and consistent to achieve effective mapping of motion.

Traditionally, achieving accurate reflections has required the use of expensive motion capture technologies such as motion capture suits, wearable devices, or even specialized motion capture studios (Regis et al., 2022). These technologies are necessary to capture user movements in a stable and reliable manner. However, in our approach, we utilize AI models for motion capture. This allows us to some extent accurately capture 3D motion and map it to the movement of the virtual character, achieving natural 3D motion of virtual character.

5.1.1.1 Achievements on 3D Motion Mapping

We performed several tests to evaluate the movements of the virtual characters controlled by the application. These tests included inputting videos of dancers performing, as well as using the application to capture our own movements in real time. We specifically chose dance videos because they involved active movements and complex dance routines. Our goal was to assess the extent to which our application

could handle large-scale complex movements, as it may be more challenging for an AI model to capture these movements and reflect them onto a virtual character.

Even for complex dance movements, the results were promising. Most of the movements were effectively captured and reflected onto the virtual characters. The torso and limb movements of the virtual characters closely followed those of the dancers in the video, showing smooth transitions with no noticeable fluctuations or sudden strange movements. Essentially, the virtual character dances in sync with the dancer, demonstrating our application's ability to accurately capture the user's movements, even complex ones, and reflect them consistently on the virtual character. This highlights the application's ability to capture user movements and transfer them to the virtual environment accurately and reliably (Figure 47).

Additionally, we had positive results in real-time testing, where our application was able to capture our movements in real-time. The application achieved an update rate of 30 frames per second or higher, and the actions captured by the AI model were quickly reflected in the virtual character with no noticeable delay. As a result, whenever a user makes a specific movement and it is captured by the AI model, that movement is reflected in the virtual character almost immediately. These results demonstrate that it is possible to achieve effective, consistent, and reliable advanced 3D motion control of virtual characters using our application.

5.1.1.2 Restriction with Utilization of AI Models on Motion Capture

While our application shows good overall performance, it is still challenging to accurately capture user movements and reflect them on the virtual character in specific situations. One such situation is when certain body parts of the user are not clearly visible in the captured camera image. As mentioned earlier, our landmark detection model relies entirely on image inputs, and thus its performance is largely affected by image quality. If the user's body parts are not clearly visible, it will result in inaccurate landmark detection data for blurred body parts. Since the computed rotation vector controlling the virtual character depends on the accuracy of the landmark data, the inaccuracy is reflected in the virtual character's movements.

While our approach includes the use of visibility prediction to mitigate some of the strange movements caused by unclear body parts, it is challenging to eliminate all such problems. Some strange movements may still occur, albeit at a reduced frequency. However, this approach also poses another problem. If a body part remains unclear in subsequent image inputs, its movement may be ignored in subsequent frames,

causing that body part to appear idle until it is visible again. This problem becomes more pronounced if a large portion of the user's body is not captured in the image, resulting in that body part appearing to move unnaturally or remaining idle for an extended period. This situation is not ideal as it can disrupt the natural flow of movement (Figure 36).

Unfortunately, the limitations and challenges described here are inherent to the approach we employ, which relies heavily on landmark detection models that utilize only image input. As a result, our application faces significant limitations in providing fully naturalistic movements for avatars.

5.1.2 Facial Expression

Natural facial expression control relies on reliable and consistent facial tracking technology, which tracks the user's facial features and applies them to the virtual character's face. To modify the virtual character's face real-time, a procedure known as model rigging are required, it involves dynamically rigging a 3D model of the virtual character in real-time to align its face with the facial movements captured during the facial tracking process. Our approach is like the mainstream industry practice of controlling the facial features of virtual characters, which utilizes facial landmark detection models.

In our case, we analyze the facial landmark data obtained from the model and compute the necessary control values. These values are then applied to the virtual character and the model is dynamically adjusted to achieve the desired facial expression. We conducted tests to evaluate the performance of this approach for real-time facial expression capture. During the tests, we continuously changed our facial expressions to evaluate how accurately the virtual character reflected our movements.

5.1.2.1 Achievement on Facial Expression Control

The facial expression control method we used yielded consistent and reliable results. The landmark detection model accurately captured the opening of the eyes and mouth, allowing us to compute accurate aspect ratios to control the avatar's facial movements. The opening and closing of the avatar's mouth was very smooth, with no noticeable fluctuations or blinking problems. While the opening and closing of the avatar's eyes was also relatively smooth, its performance was not as strong as that of the mouth.

During comprehensive testing, we explored different scales and biases to fine-tune the EAR and MAR calculations. We found that the limitations of eye landmark detection can be attributed to the landmark detection model itself. While the Mediapipe holistic model generates reasonably accurate results for most facial landmarks, the accuracy and consistency of eye landmarks is questionable. Through various tests on different facial features, we found that the model is not very sensitive to the degree of closure of the user's eyes. The generated facial landmark data does not accurately reflect the state of eye closure; instead, it often shows the eyelids half-closed. This insensitivity, coupled with the problem of having similar values for the eye landmark data, severely impacted the performance of the facial expression control.

Fortunately, by carefully tuning the scale and bias parameters of the EAR calculation, we developed a version that is more sensitive to eye closure while minimizing the apparent fluctuation and blinking problems. This improved version effectively reflects the user's eye-opening and eye-closing movements, resulting in effective and natural control of the virtual character's facial expressions (Figure 43).

5.1.2.2 Restriction on Facial Expression Control

Like the challenges faced by body movement control, facial expression control also relies heavily on landmark detection models. As a result, the problem of unclear face display can lead to unstable performance. However, facial landmark detection typically performs better compared to pose estimation. We discover that even if almost half of the faces are not clearly visible, the generated landmark data still provides an accurate approximation. However, there is another obvious problem with the face landmark detection model compared to the pose landmark detection model.

Through our real-time testing using a web camera image with a resolution of 640 width and 480 heights, we observed a notable decline in the performance of facial landmark detection when the user appears relatively small in the image. This is mainly since the user is far away from the camera, resulting in the face in the image becoming smaller and less clear. Consequently, the generated landmark data values become more similar and inconsistent, leading to reduced sensitivity and unreliable aspect ratio calculation, especially in the case where input image is not with high resolution (Figure 48). As a result, when the user is too far away from the camera, it becomes difficult for the AI model to capture facial features consistently and reliably. This limitation prevents the effective and natural application of facial features to virtual characters. Unfortunately, this problem is caused by the inherent characteristics of the model and is therefore difficult to solve. And it greatly restricts the performance of our

application in achieving natural control over virtual characters, especially when the user is far away from the camera.

5.1.3 Emotion Expression

In addition to controlling facial expressions, emotions also influence the overall facial features of virtual characters. For example, when someone is happy, their mouth tends to form a smile, while anger or curiosity may result in a scowl. Emotions play a significant role in facial expression. However, current virtual character control mechanisms in the market lack emotion tracking, relying mostly on manual triggering by users (Regis et al., 2022). This approach leads to less natural control over the facial expressions of virtual characters.

Taking VTubers as an example, when they interact with their viewers, they experience various emotions. To reflect these emotions accurately, they must manually trigger the desired emotion themselves. This introduces a delay as users need time to change the emotion expression of the virtual character to match the desired one. Consequently, the visual representation of virtual characters often appears delayed, and if users forget to change or trigger the correct emotion expression, it further diminishes the naturalness of the virtual characters.

To address this challenge, we utilize AI models to achieve emotion tracking and automate the process. By integrating AI models into the system, we can track emotions in real-time and ensure a more seamless and natural transition of emotion expression for virtual characters.

5.1.3.1 Achievement on Emotion Expression Control

In evaluating the effectiveness of our emotion tracking application, we recognized the importance of assessing its performance in real-time scenarios, rather than focusing solely on the evaluation of the emotion detection model itself. Our goal was to examine how well the application combines facial expression control with an accurate representation of the user's emotions.

In real-time testing, we performed various combinations of emotions and facial expressions real-time to evaluate the performance of the application. The results were satisfactory. Our app demonstrated the ability to accurately detect and track emotions, effectively adapting to transitions between different emotional states. The emotion

recognition model effectively captured and reflected the user's emotions onto the virtual character, ensuring seamless integration.

In addition, our app successfully combines facial expressions with emotional expression control. It preserves the natural movements of the mouth and eyes and blends them harmoniously with emotional expression (Figure 43 and 44). This approach makes the user's facial features appear more realistic and natural on the avatar. By controlling multiple facial features at the same time, our application goes beyond the traditional approach of mapping and controlling only specific facial attributes and enables naturally realistic facial and emotional expression control.

5.1.3.2 Restrictions on Emotion Expression Control

Again, like facial expression tracking, our emotion tracking relies heavily on emotion detection models that utilize only image input. Therefore, it encounters the same challenge when the user is too far away from the camera. When the user is far away, facial features become indistinct and it is difficult for the AI model to capture useful features, leading to inaccurate and unreliable estimates of emotional expression (Figure 48). This potentially resulting in incorrect emotion expression at inappropriate times. As a result, the expression of emotions and facial expressions becomes unnatural, preventing the effective and natural application of facial features to avatars.

Unfortunately, this limitation is an inherent feature of the model and a significant challenge to overcome. It severely limits the performance of our application in achieving natural control over the emotions of the avatar, especially when the user is away from the camera. Furthermore, since emotional expression also affects the representation of facial features, this situation can have a considerable impact on the overall naturalness of the avatar's facial expressions.

5.1.4 Summary of Achievements and Restrictions on Natural Movement of Virtual Character

In summary, the methods we have discussed allow us to achieve natural and realistic control of body movements, facial expressions, and emotional expressions in most cases. However, it is important to note that certain limitations arise from the nature of the AI models we employ. These limitations restrict our application and

prevent us from achieving a completely natural and realistic control of the body movements of the avatars in all situations.

5.2 Real-Time Control of Virtual Characters

Our developed application and methodology offer a groundbreaking capability for real-time control of virtual characters. By projecting captured facial expressions, emotions, and body movements onto the virtual character, we enable the virtual character to mirror the user's movements and expressions, resulting in more realistic and lifelike behavior in the virtual world. This approach goes beyond traditional methods that rely on predefined animations or movements, as it enables dynamic and realistic actions that cannot be fully predetermined.

This breakthrough opens new possibilities for controlling virtual characters in games and other virtual environments. Unlike traditional methods that rely on predefined animations or complex motion capture systems and virtual reality devices, our approach requires only a camera as input. This eliminates the need for expensive setups and makes full-body control more accessible to a wider range of people. Since most computer devices are already equipped with a webcam, more people can now experience real-time control of their virtual characters, making the technology plebeian and no longer limit to professional users such as Vtubers.

The impact of this advancement extends beyond gaming. It opens a huge potential for the development of meta-universes, where individuals can use this technology to control their virtual characters, socialize, learn, collaborate, and play in virtual worlds (Tucci & Moore, 2024). In addition, this technology can revolutionize the gaming experience by allowing users to control their characters with their entire body and interact with the gaming environment in completely new ways. The possibilities in a variety of virtual world-related fields are vast.

However, it is important to recognize the limitations of the AI models we currently use. While we can achieve real-time control of virtual characters through landmark detection models, gesture recognition models, and emotion recognition models, these models all have limitations that restrict the performance of our applications. This raises concerns about whether similar techniques employed in meta-universes or games will encounter similar

problems and thus negatively affect the overall user experience. We therefore have a long way to go to improve these techniques to ensure consistent and optimal performance. This will make it feasible for widespread public use and enhance the user experience in a meaningful manner.

5.3 Workload and Processing Demand

For the testing of our application, we used the following hardware: an AMD Ryzen 9 7845HX CPU with Radeon Graphics (3.00 GHz with 12 cores and 24 logical processors) and an NVIDIA GeForce RTX 4060 laptop GPU. the operating system used was Windows. The operating system used is Windows. As mentioned earlier, our application consists of two separate programs running concurrently. The Python program uses the CPU to handle landmark detection, while the gesture and emotion recognition models run on the GPU. The Unity program uses both the CPU and the GPU.

The Python program had an update rate of about 13 Hz when all tasks were processed sequentially by a single process. This update rate was too slow for a real-time application, resulting in noticeable lag and delayed movements of the virtual characters. To address this issue, we conducted tests and found that using four sub-processes for landmark detection and two sub-processes for gesture and emotion recognition increased the update rate to about 30 frames per second (fps). Further increasing the number of handlers did not significantly improve the update rate, suggesting that the frame rate of the camera may be a limiting factor. 30 fps is suitable for real-time applications and ensures that the virtual character's movements are updated quickly, without noticeable delay or lag.

On the other hand, the Unity program itself runs at about 400 fps, with an inference time of about 2.0ms for the main thread and 0.2ms for the rendering thread. this suggests that the Unity program has a very fast update rate and can instantly reflect the results of any updated AI model to the virtual character. As a result, the physical movements of the virtual character appear smooth and natural, with the only limitation being the update speed of the AI model.

However, it is worth noting that our application imposes a significant workload on the hardware devices, especially the CPU. Figures 49 and 50 show that the CPU is utilized at 100% while the GPU is used at about 30%. In addition, the CPU we used has a built-in AMD Radeon (TM) graphics GPU, which has a

utilization rate of 77%. This shows that our application puts a heavy workload on the device when running AI models and game engines at the same time.

This raises concerns about the workload and processing demands of this technology. To effectively utilize this technology, users may need to equip powerful CPUs and GPUs to ensure consistent and reliable application performance. Without a GPU, the workload of the CPU alone may become excessive, resulting in a significant reduction in overall performance or even the inability to run the application due to excessive workload. Similarly, if the CPU lacks a high clock frequency, a large number of cores and logical processor, the application may still not run fast enough, resulting in a low update rate that can significantly impact the overall user experience.

These observations highlight the high workload and processing demands of our technology on the average computer. It necessitates a well-configured system for the application to run smoothly and consistently.

6. Conclusion and Possible Future Works

In summary, existing methods of controlling avatars through landmark detection models suffer from imprecise body movements and limited emotional expression, resulting in a lack of natural interaction between users and avatars. While motion capture devices allow for more precise control of avatars, they also require the purchase and installation of specialized systems and equipment. To address these limitations, our project aims to utilize multiple AI models to enable advanced control of avatars, providing users with a unique and more natural experience.

The main goal of our project is to develop an application that combines multiple AI models to control avatars. The application consists of two programs: one written in Python to handle AI-related tasks and the other written in C# on the Unity platform to control the avatar. The workflow consists of transmitting a stream of live camera data to the main process, which then assigns the necessary inputs to the corresponding AI model sub-processes for detection. The results obtained from the sub-processes are extracted and transmitted to the VCC program via pipeline. The VCC program uses this data to update its control signals, which in turn affects the various features and animations of the virtual character.

The application will provide several key features. First, it can accurately and efficiently capture the user's body movements and facial expressions using the Mediapipe holistic model. These captured movements and expressions will be directly reflected on the virtual character, ensuring that the avatar's movements are consistent with the user's intentions and body movements. In addition, the application employs self-designed, trained gesture recognition models and the open-source emotion recognition model “HSEmotion”. these models enable the avatars to accurately express emotions and initiate specific actions based on recognized gestures, resulting in more naturalistic and realistic movements and behaviors of the avatars, which greatly enhances the overall user experience. To achieve precise control over the humanoid model, the application utilizes the Animation Rigging extension and Unity's built-in features such as the Animation Maker and Skin Mesh Renderer. By integrating these components with the AI model, the app can precisely control the movements of the avatar, resulting in smooth and realistic avatar animations.

Although we have achieved real-time control of virtual characters and made significant progress in adapting body movements and expressions to user intent, we can only achieve a certain degree of natural movement and expression due to the limitations of the AI model itself. These limitations greatly affect the potential of this approach. In addition, the heavy workload and high processing demands associated with this approach present challenges. Still, this project shows its potential and opens new possibilities for using AI models to control virtual characters in games and other virtual environments, leading to more immersive and realistic virtual character control.

6.1 Possible Future Works

Given the limitations highlighted, our proposed methodology offers several areas for further research and development, focusing on the improvement of AI models and avatar models' control.

6.1.1 Improvements on AI Models

Regarding potential improvements to AI models, we focus on two main aspects: performance and workload. Currently, our approach employs three independent AI models: the Mediapipe holistic model for landmark detection, a self-designed MLP model for gesture recognition, and a CNN-based emotion recognition model "HSEmotion". Currently, all models operate independently. However, one idea for improvement is to integrate these models into a single integrated model.

This concept consists of creating a unified backbone model that is used to analyze the input data and generate relevant features. Subsequently, sub-models responsible for separate tasks (e.g., emotion recognition, gesture recognition, and landmark detection) can utilize these features. This approach eliminates the need for three separate models to analyze their respective inputs from scratch, thus significantly reducing the workload. In addition, merging models has the potential to improve overall performance because they can share information and learn from each other rather than analyzing inputs independently.

It is important to recognize that this approach also has the potential to lead to performance degradation, as the effectiveness of the architecture depends on the structure and design of the chosen backbone and individual detection/classification models. However, by carefully considering and choosing the architecture, for example by utilizing vision converters or MLP hybrids rather than CNNs, it is possible to improve the ability of the model to learn and understand the underlying patterns in the input image. By employing a deeper and broader backbone model, deeper and more informative features can be learned and subsequently used for classification and detection tasks, potentially producing better results. As a result, this allows for superior performance to be achieved with a single unified pipeline model, rather than relying on multiple AI models analyzing from scratch.

It is worth noting that this integrated AI model requires further research and testing. At this stage we cannot guarantee its effectiveness, but it has the potential to improve performance while reducing workload. Therefore, exploring the development of a self-designed integrated AI model that improves the performance of multiple tasks while reducing workload may be a viable direction for future work.

6.1.2 Improvements on Virtual Character Control

While our current application can control most body parts of the virtual avatar, there are still limitations when it comes to facial expressions. Currently, we can control how much the virtual character's eyes and mouth open, but these are not the only human expressions. Various factors, such as the shape of the mouth, the movement of the eyebrows and the movement of the iris in different scenes, are yet to be realized. The use of a skin blend renderer may help to address these limitations.

Currently, only the blend shape of the mouth, the expression of emotions, and the opening of the mouth and eyes can be adjusted, as these are the only facial features provided by the original Vroid humanoid model. However, the Skin Mesh Renderer is available for mesh deformation, so we should be able to utilize it to adjust facial features such as eyebrow and iris movement. Further development work could explore how to create additional blend shapes to control other facial expressions not provided by the original Vroid humanoid, including the eyebrow and iris expressions mentioned above.

Once facial mesh rendering is developed, we can utilize the facial landmark data generated by the landmark detection model to obtain the necessary information to compute different control values to manipulate these facial features. For example, we can determine the location of the user's iris, whether the user is frowning, and other relevant facial details. In this case, we can obtain richer and more realistic facial expressions.

6.1.3 Summary on Possible Future Works

In summary, possible future improvements in AI modeling include research and development of an integrated model that combines landmark detection, emotion recognition, and gesture recognition into a single model. Additionally, in terms of avatar control, the focus will be on the development of new blend shapes utilizing skin mesh renderers to control more facial features. These advances are aimed at achieving more consistent, reliable, and lightweight AI models and enhancing the fidelity and control of virtual character expressions (especially facial expressions).

References

1. Adobe Systems Incorporated. (n.d.). Mixamo. https://www.mixamo.com/#/
2. Av-Savchenko, & Andrey, V. (2022). AV-Savchenko/face-emotion-recognition: Efficient face emotion recognition in photos and videos. Face-Emotion-Recognition. https://github.com/av-savchenko/face-emotion-recognition
3. Beklemysheva, A. (2022, December 27). Why use python for AI and machine learning?. SteeKiwi. https://steelkiwi.com/blog/python-for-ai-and-machine-learning/
4. Gank Content Team. (2023, July 8). What is a vtuber? The Ultimate Guide to Virtual YouTubers!. Gank Blog. https://ganknow.com/blog/vtuber/
5. GlobalSpec. (n.d.). Long pass filters and short pass filters selection guide: Types, ... https://www.globalspec.com/learnmore/optics_optical_components/optical_components/long_short_pass_filters
6. Grishchenko, I., & Bazarevsky, V. (2020, December 10). MediaPipe holistic – simultaneous face, hand and pose prediction, on device. MediaPipe Holistic — Simultaneous Face, Hand and Pose Prediction, on Device. https://blog.research.google/2020/12/mediapipe-holistic-simultaneous-face.html
7. Kapitanov, A. (2022). Hukenovs/hagrid: Hand gesture recognition image dataset. GitHub. https://github.com/hukenovs/hagrid
8. Leap motion controller 2. Ultraleap. (n.d.). https://leap2.ultraleap.com/leap-motion-controller-2/
9. Lesti, G., & Spiegel, S. (n.d.). A sliding window filter for time series stream. https://www.researchgate.net/publication/319987593_A_Sliding_Window_Filter_for_Time_Series_Streams
10. Lu, W. (2012, August 21). Evolution of video game controllers. https://web.archive.org/web/20120821172337/http://www.stanford.edu/group/htgg/cgi-bin/drupal/sites/default/files2/wlu_2003_1.pdf
11. Mollahosseini, A., Hasani, B., & H. Mahoor, M. (2017, October 9). AffectNet: A Database for Facial Expression, Valence, and Arousal Computing in the Wild. https://www.researchgate.net/publication/319121606_AffectNet_A_Database_for_Facial_Expression_Valence_and_Arousal_Computing_in_the_Wild

12. NeuroSYS. (2023, October 4). Unity Engine – is it bad & what is it good for? https://neurosys.com/blog/why-people-say-unity-engine-is-bad
13. PapersWithCode. (n.d.). Papers with code – fer2013 dataset. FER2013 Dataset | Papers With Code. https://paperswithcode.com/dataset/fer2013
14. Pixiv Inc. (n.d.). Vroid Studio. https://vroid.com/en/studio
15. Qiong, C., Li, S., Xie, W., Omkar , M. P., & Andrew , Z. (2018). VGGFACE2 dataset. Visual Geometry Group - University of Oxford. https://www.robots.ox.ac.uk/~vgg/data/vgg_face2/
16. Regis, R. D. D., Ferreira, J. C. V., Diniz, G. R., & Gonçalves, P. (2022, October). (PDF) VTUBER concept review: The New Frontier of Virtual Entertainment. https://www.researchgate.net/publication/372442701_VTuber_concept_review_The_new_frontier_of_virtual_entertainment
17. Singh, J. (2023, May 21). What is a vtuber, and how do you become one?. Cointelegraph. https://cointelegraph.com/news/what-is-a-v-tuber
18. Sony Corporation. (n.d.). Sony Corporation - mocopi: About mocopi. https://www.sony.net/Products/mocopi-dev/en/documents/Home/Aboutmocopi.html
19. Tucci, L., & Moore, J. (2024, March 22). What is the metaverse? an explanation and in-depth guide. What is the metaverse? An explanation and in-depth guide. https://www.techtarget.com/whatis/feature/The-metaverse-explained-Everything-you-need-to-know
20. VRM Consortium Inc. (n.d.). GitHub – VRM-C/UNIVRM. https://github.com/vrm-c/UniVRM
21. V.Savchenko, A. (2023, June 15). *Facial Expression Recognition with Adaptive Frame Rate based on Multiple Testing Correction*. OpenReview. https://openreview.net/forum?id=DH11pt7S2t
22. Wu, Y., Wang, Y., Jung, S., Hoermann, S., & W. Lindeman, R. (2019, November). Exploring the use of a robust depth-sensor-based Avatar Control System and its effects on communication behaviors. https://dl.acm.org/doi/fullHtml/10.1145/3359996.3364267

Appendix

Figure 1: Image of utilizing Leap motion controller for gesture recognition

Figure 2: Image of utilizing Mocopi for Virtual character control

Figure 3: Image of Motion Capture System

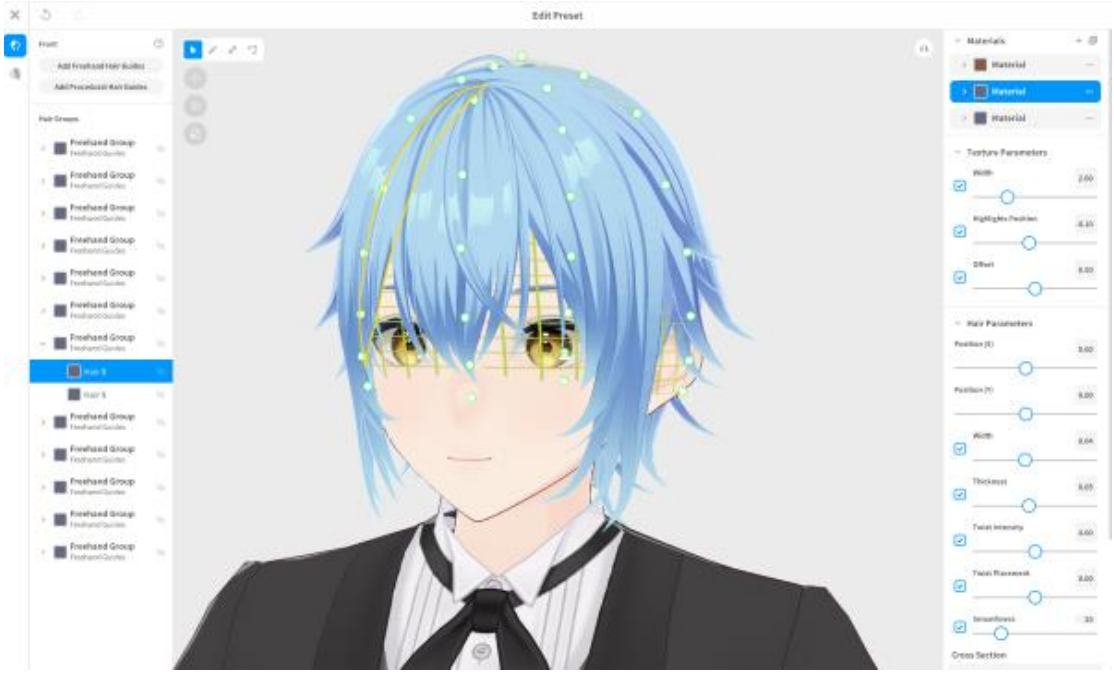


Figure 7: Vroid Studio.

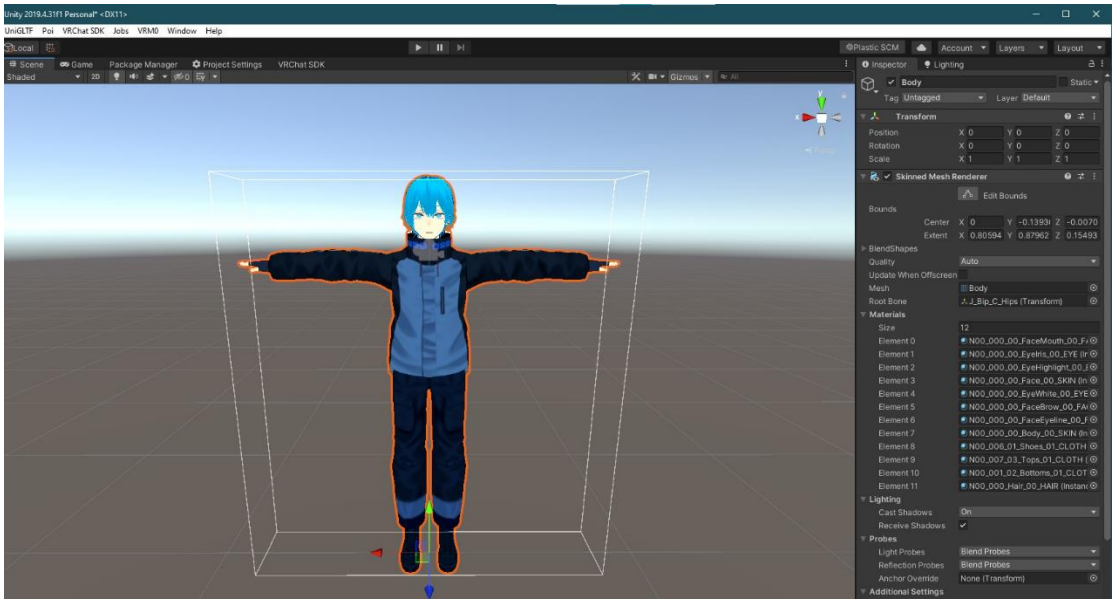


Figure 8: Unity assets imported with UNI-VRM.

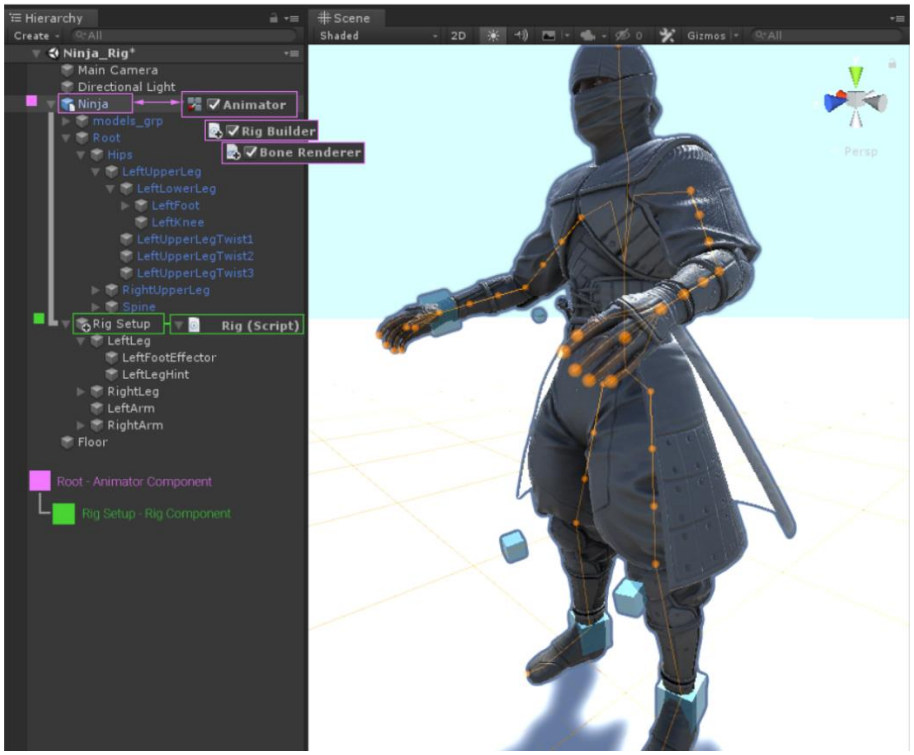


Figure 9: Animation Rigging.

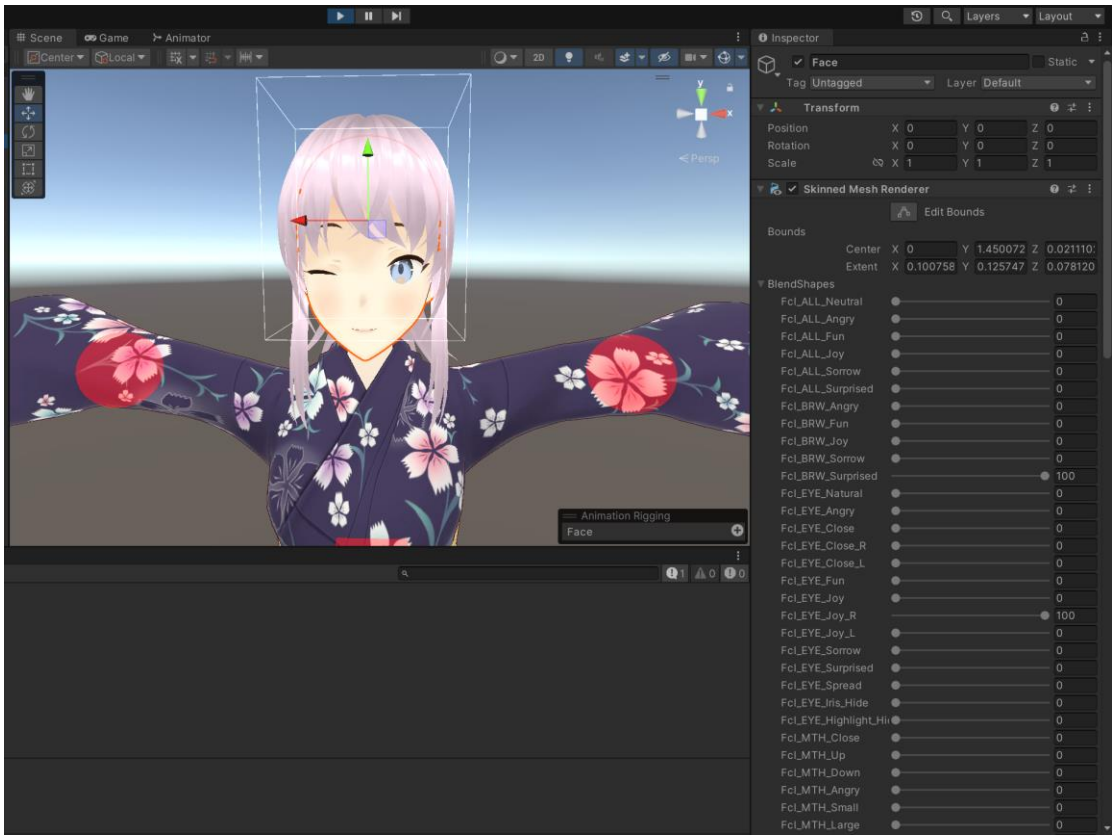


Figure 10: Skinned Mesh Renderer.

Figure 11: Mediapipe Holistic Model detection results

Figure 12: Example of fluctuation of Rotation Vector (shaking problem).

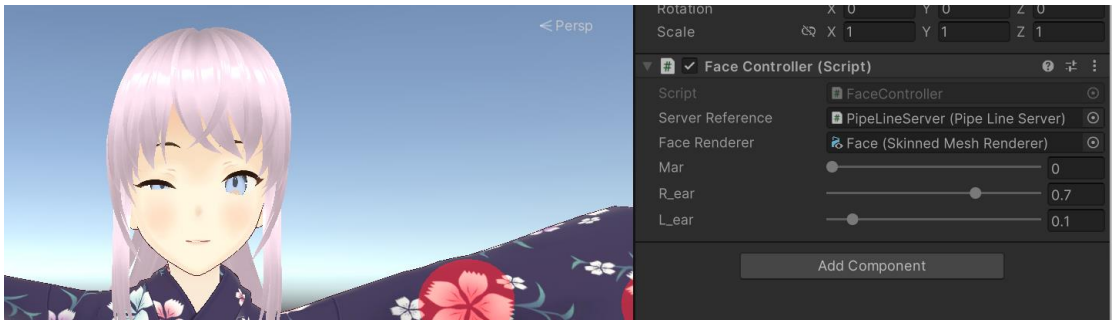


Figure 13: Example of fluctuation of MAR and EAR (blinking problem).

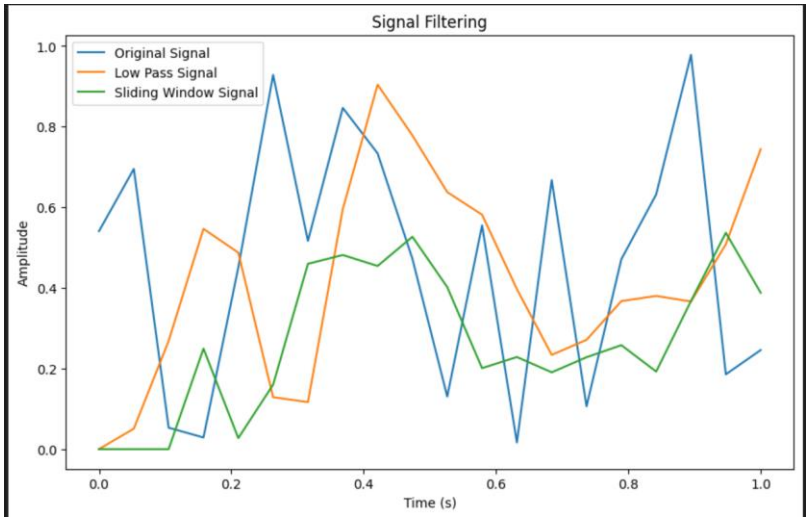


Figure 14: Example of filtering applied on random signal.

Figure 15: Example of filtering applied on landmark data (Filtered data on the right).

Figure 16: Example of action triggering through gesture recognition

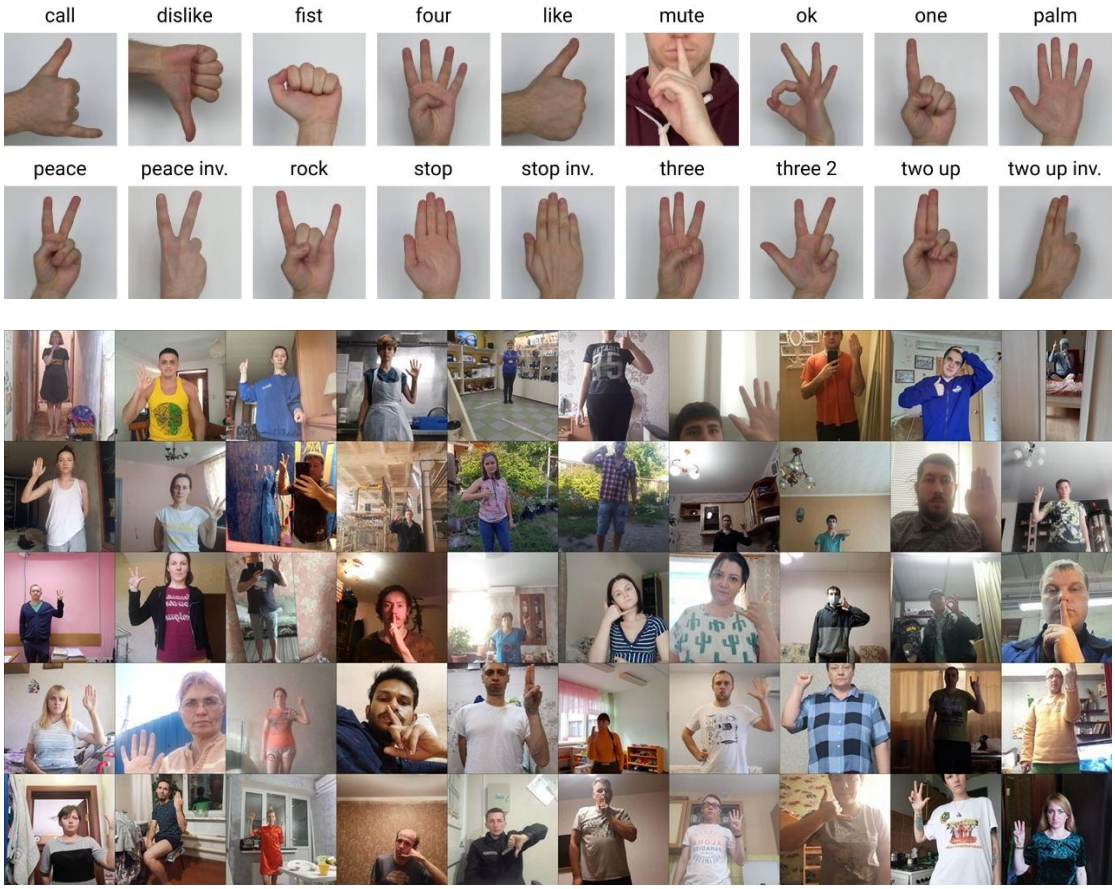


Figure 18: Example of HaGRID Dataset.

Rank	Model	Accuracy (8 emotion)	Accuracy (7 emotion)	Extra Training Data	Paper	Code	Result	Year	Tags
1	**DDAMFN**	64.25	67.03	×	A Dual-Direction Attention Mixed Feature Network for Facial Expression Recognition			2023	Attention
2	**POSTER++**	63.77	67.49	×	POSTER++: A simpler and stronger facial expression recognition network			2023	
3	**S2D**	63.06	67.62	×	From Static to Dynamic: Adapting Landmark-Aware Image Models for Facial Expression Recognition in Videos			2023	
4	**Multi-task EfficientNet-B2**	63.03	66.29	×	Classifying emotions and engagement in online learning based on a single facial expression recognition neural network			2022	
5	**MT-ArcRes**	63		×	Expression, Affect, Action Unit Recognition: Aff-Wild2, Multi-Task Learning and ArcFace			2019	
6	**Vit-base + MAE**	62.42		×	Emotion Separation and Recognition from a Facial Expression by Generating the Poker Face with Vision Transformers			2022	ViT-B
7	**DAN**	62.09	65.69	×	Distract Your Attention: Multi-head Cross Attention Network for Facial Expression Recognition			2021	ResNet-18
8	**SL + SSL in-panting-pl** (B0)	61.72		×	Using Self-Supervised Auxiliary Tasks to Improve Fine-Grained Facial Representation			2021	
9	**Distilled student**	61.60	65.4	✓	Leveraging Recent Advances in Deep Learning for Audio-Visual Emotion Recognition			2021	
10	**Multi-task EfficientNet-B0**	61.32	65.74	✓	Facial expression and attributes recognition based on multi-task learning of lightweight neural networks			2021	EfficientNet
11	**SL + SSL puzzling** (B2)	61.32		×	Using Self-Supervised Auxiliary Tasks to Improve Fine-Grained Facial Representation			2021	
12	**SL + SSL puzzling** (B0)	61.09		×	Using Self-Supervised Auxiliary Tasks to Improve Fine-Grained Facial Representation			2021	

Figure 28: State-of-art models on emotion classification on AffectNet dataset

Model	AffectNet (8 classes)	AffectNet (7 classes)	AFEW	VGAF	LSD	MTL	Inference time, ms	Model size, MB
mobilenet_7.h5	-	64.71	55.35	68.92	-	1.099	16 ± 5	14
enet_b0_8_best_afew.pt	60.95	64.63	59.89	66.80	59.32	1.110	59 ± 26	16
enet_b0_8_best_vgaf.pt	61.32	64.57	55.14	68.29	59.72	1.123	59 ± 26	16
enet_b0_8_va_mtl.pt	61.93	64.94	56.73	66.58	60.94	1.276	60 ± 32	16
enet_b0_7.pt	-	65.74	56.99	65.18	-	1.111	59 ± 26	16
enet_b2_7.pt	-	66.34	59.63	69.84	-	1.134	191 ± 18	30
enet_b2_8.pt	63.03	66.29	57.78	70.23	52.06	1.147	191 ± 18	30
enet_b2_8_best.pt	63.125	66.51	56.73	71.12	-	-	191 ± 18	30

Figure 29: Performance of HSEmotion models

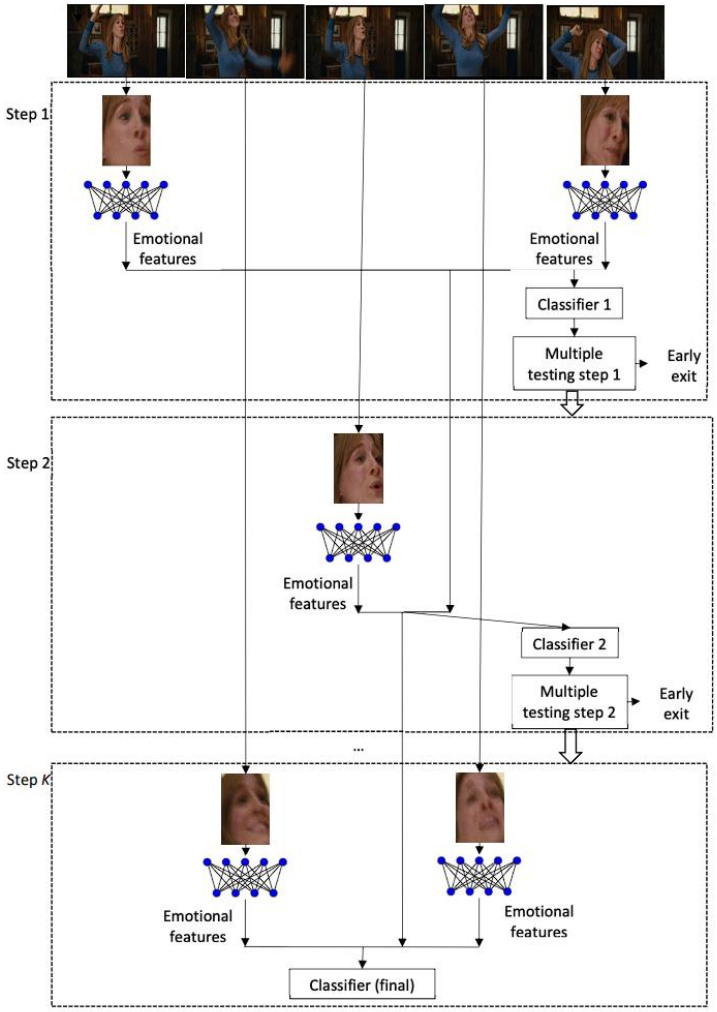


Figure 32. The overview of the approach adopted in HSEmotion with adaptive frame rate.

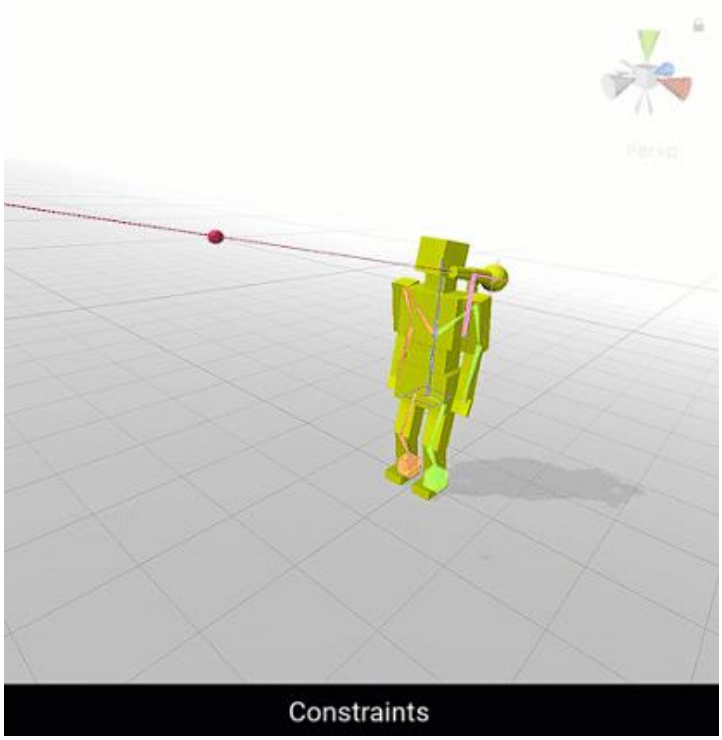


Figure 33: Example of multi-aim constraint

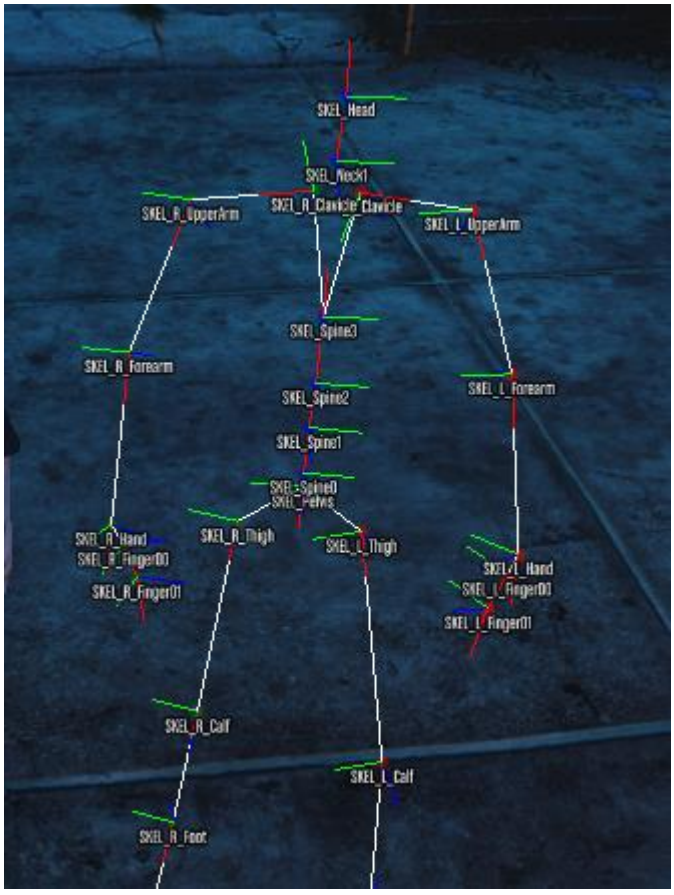


Figure 34: Example of Rotation Vector.

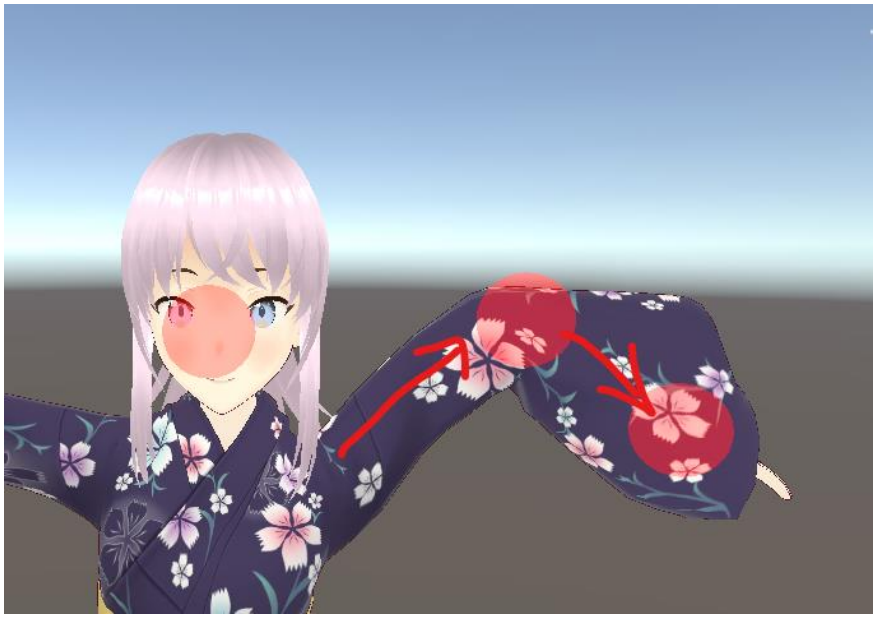


Figure 35: Example of Multi-Aim Constraint applied on avatar’s arm.

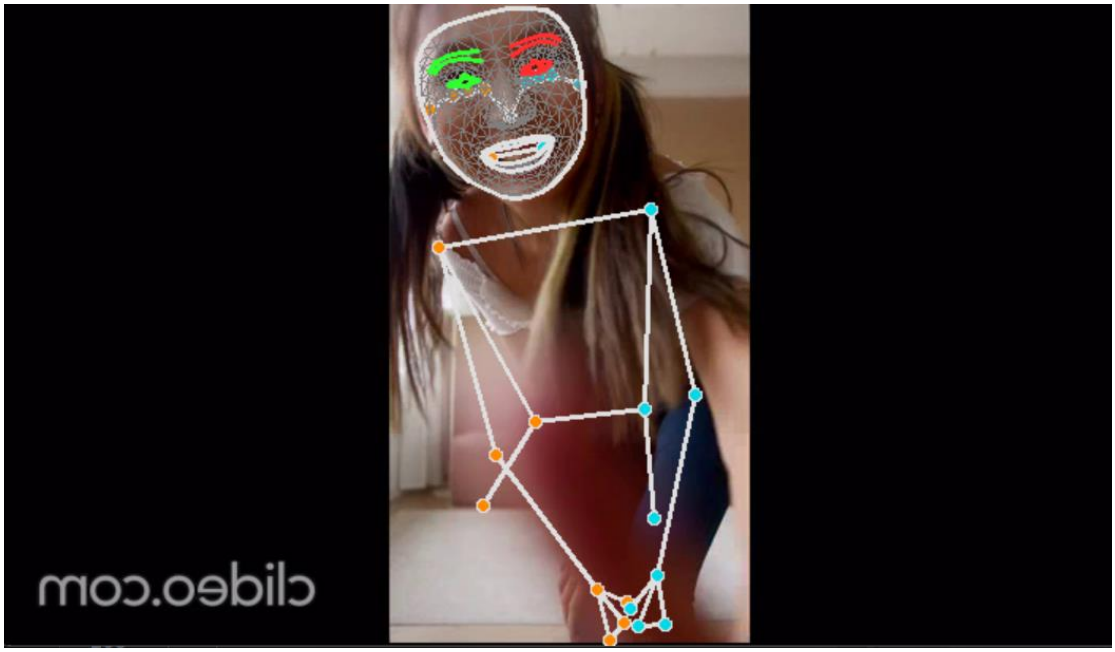


Figure 36: Example of unclear body landmark estimated by Mediapipe holistic model.

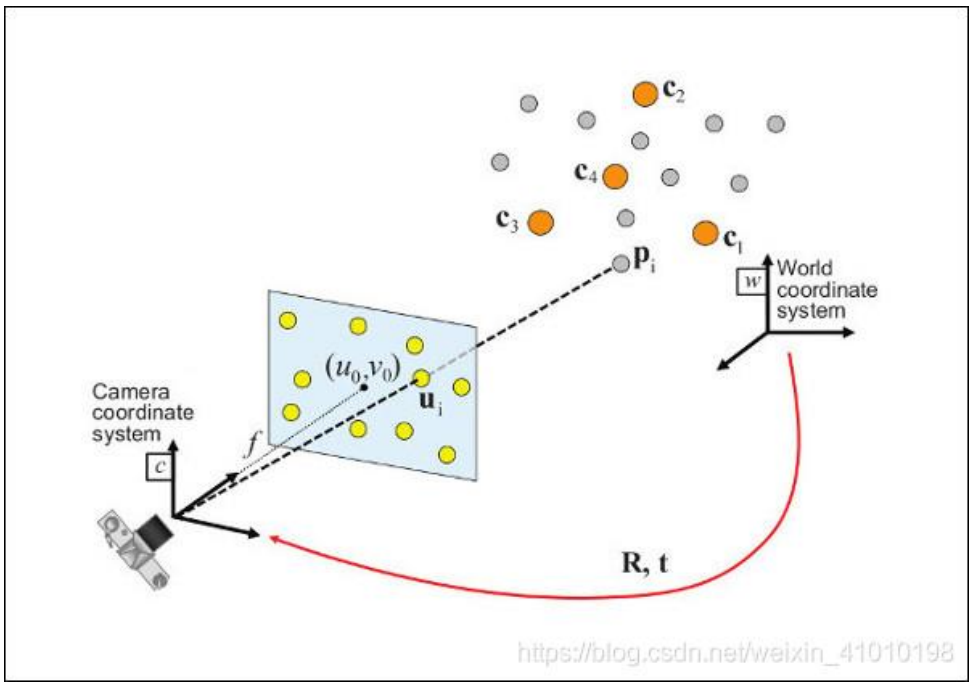


Figure 37: Image representation of PnP problems

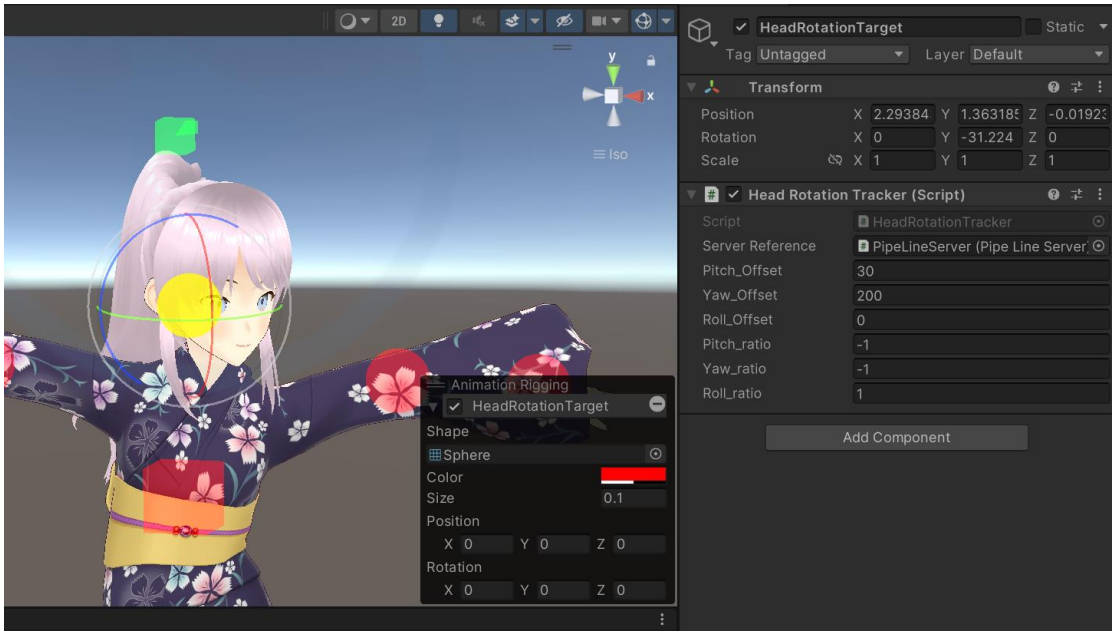


Figure 38: Image of rotation vector applied onto the head of virtual character through multi-rotation constraint.

将旋转向量转换为旋转矩阵：

$$R = \cos\theta I + (1 - \cos\theta)nn^{\mathbf{T}} + \sin\theta n^{\wedge}$$

将旋转矩阵转换为旋转向量：

$$tr(R) = \cos\theta tr(I) + (1 - \cos\theta)tr(nn^{T}) + \sin\theta tr(n^{\wedge})$$

因此：

$$\theta = arccos(\frac{tr(R-1)}{2})$$

Figure 39: Image of the mathematical formula for converting a rotation matrix into a rotation vector

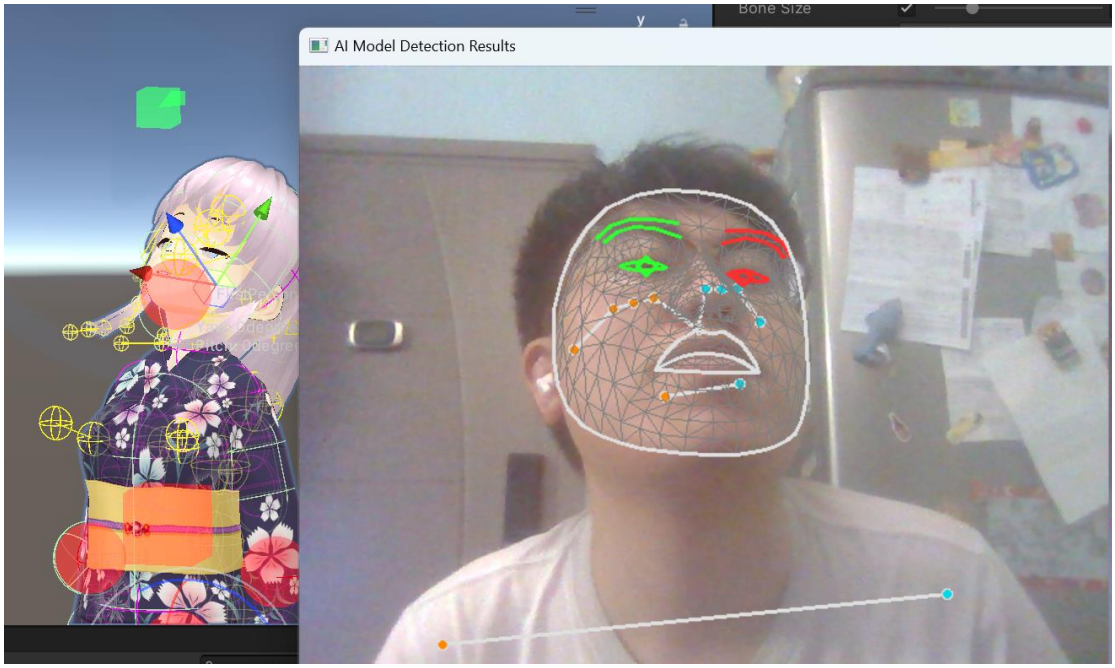


Figure 40: Image of calculated rotation vector of user's head applied onto virtual character's head.

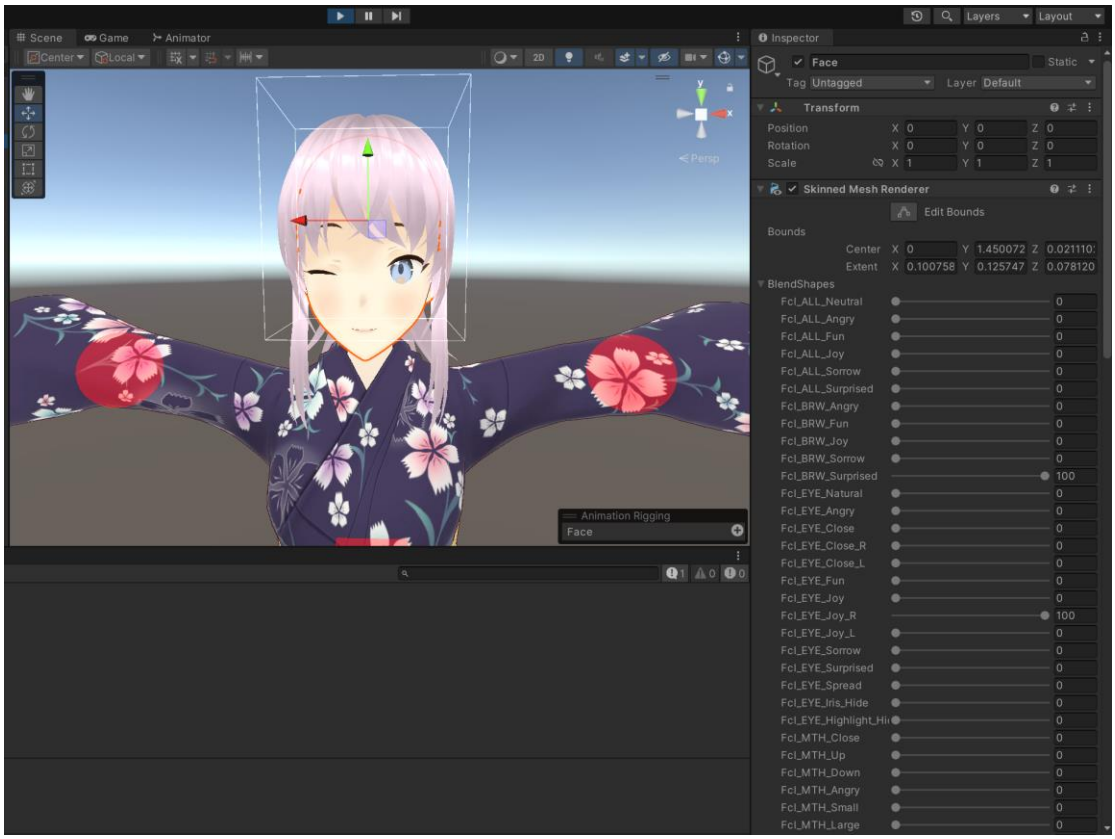


Figure 41: Example of Skinned Mesh Renderer.

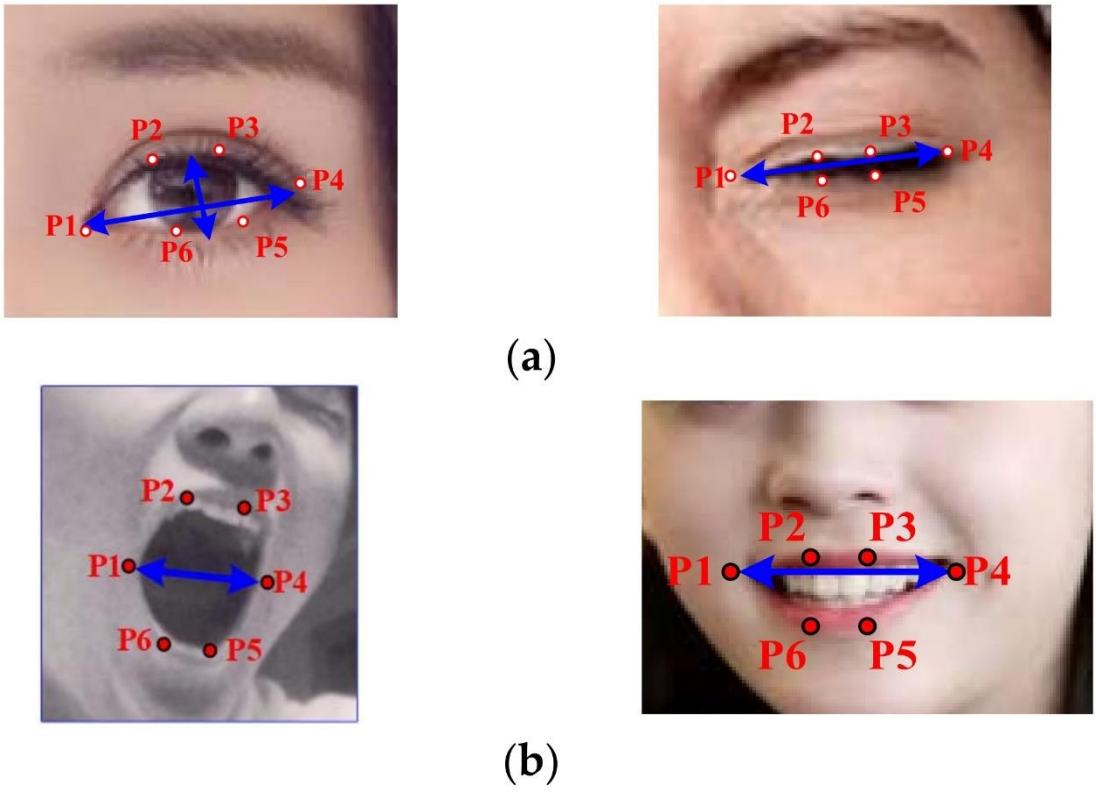


Figure 42: EAR (a) and MAR (b).

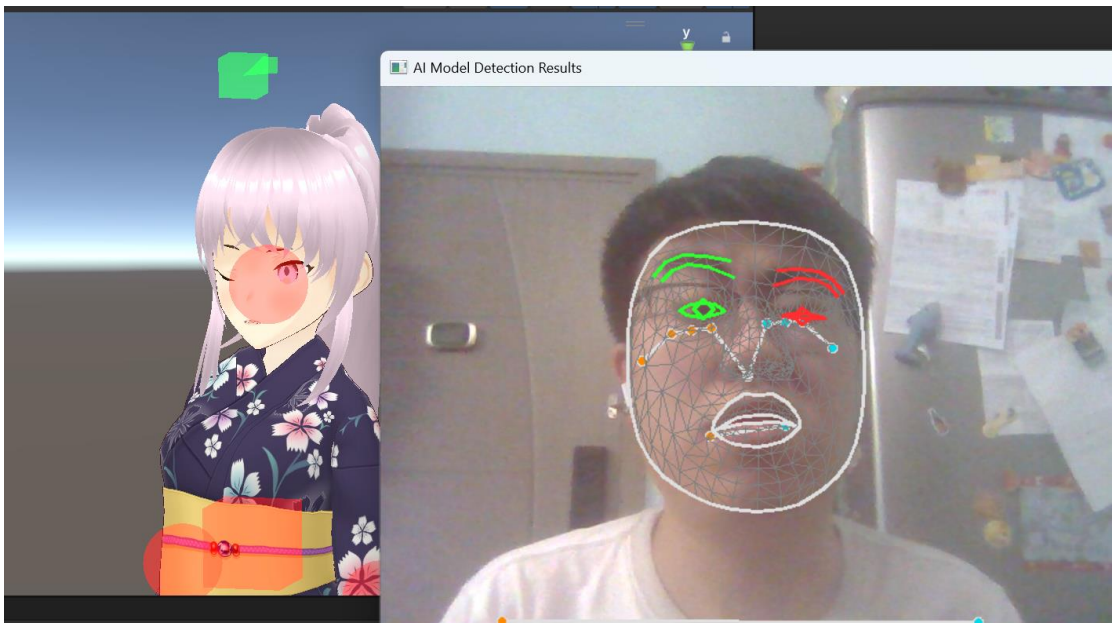


Figure 43: Image of EAR and MAR applied on virtual character.

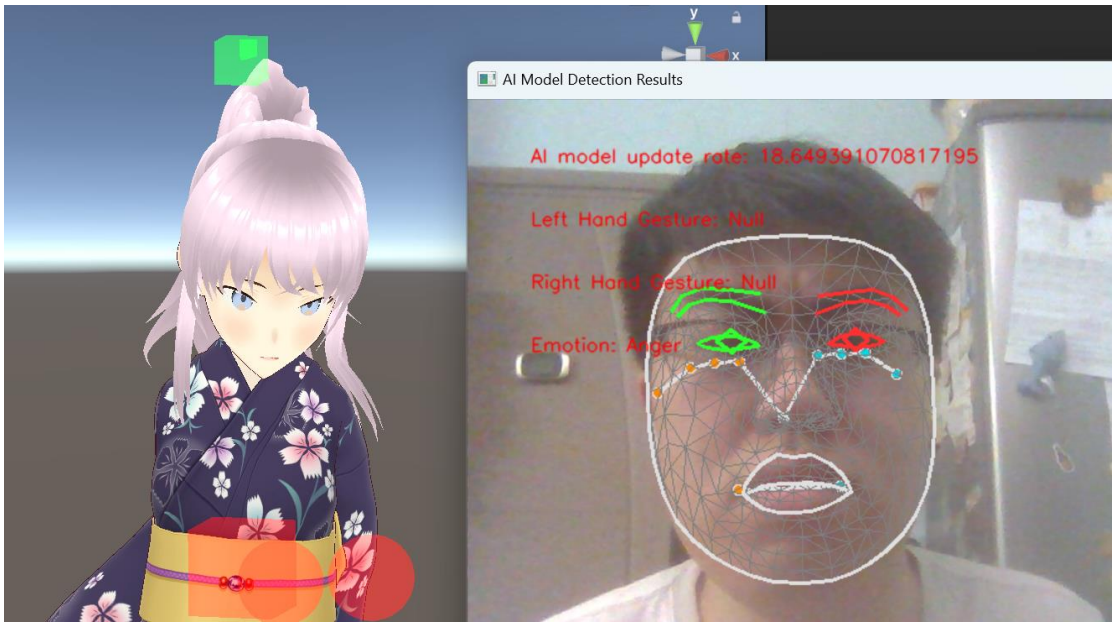


Figure 44: Image of emotion “Anger” applied on virtual character.

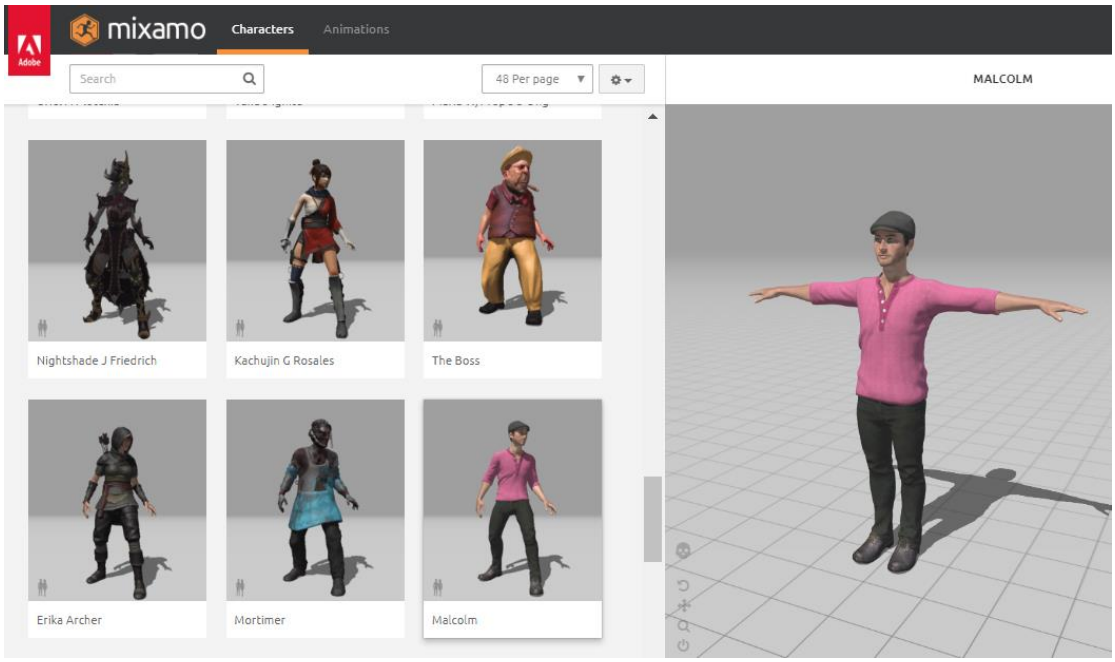


Figure 45: Mixamo.

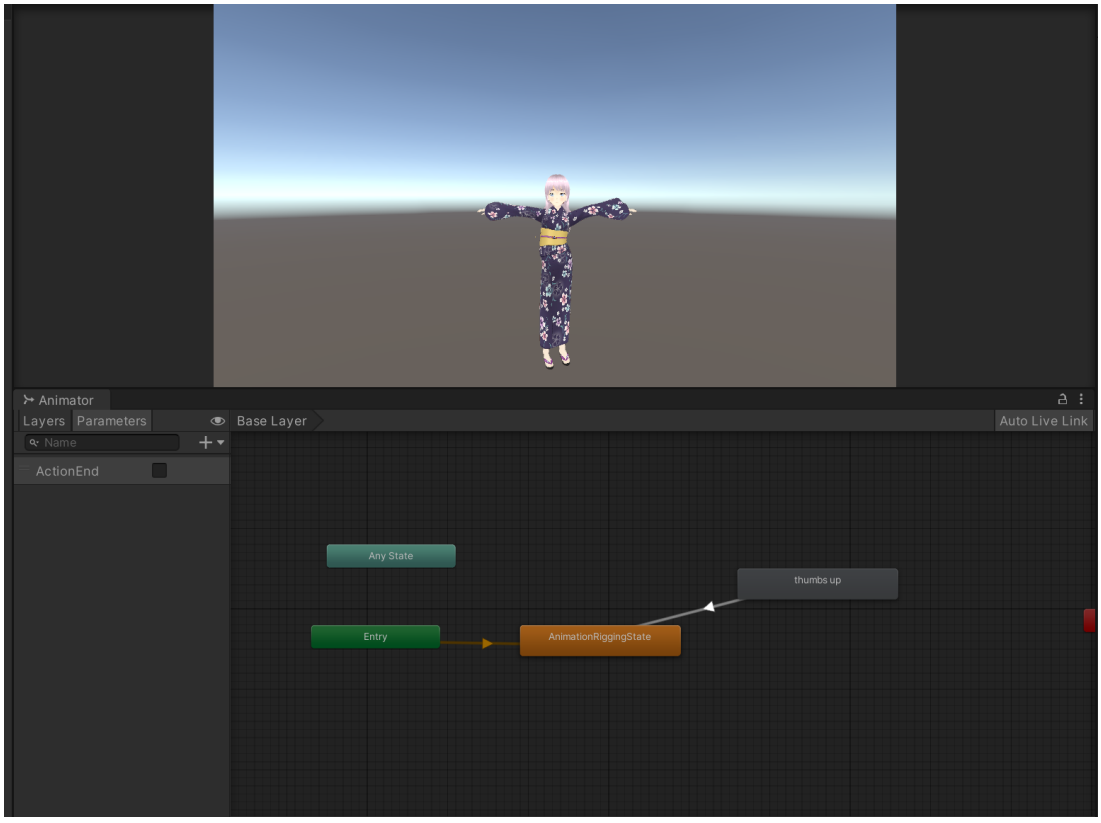


Figure 46: Example of Animator.

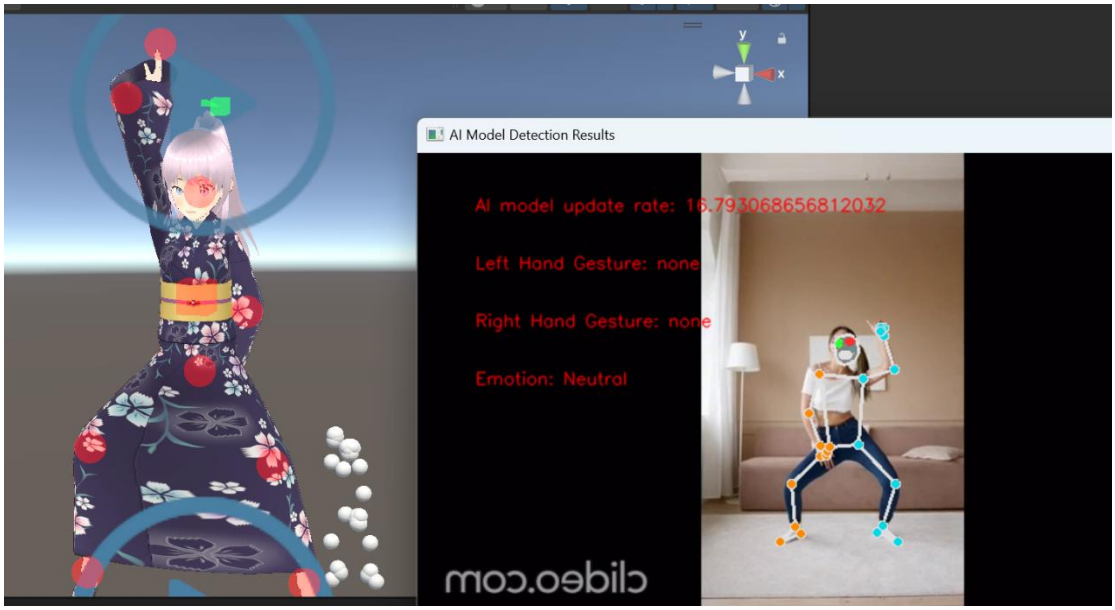


Figure 47: Example of full body control of virtual character in 3D.

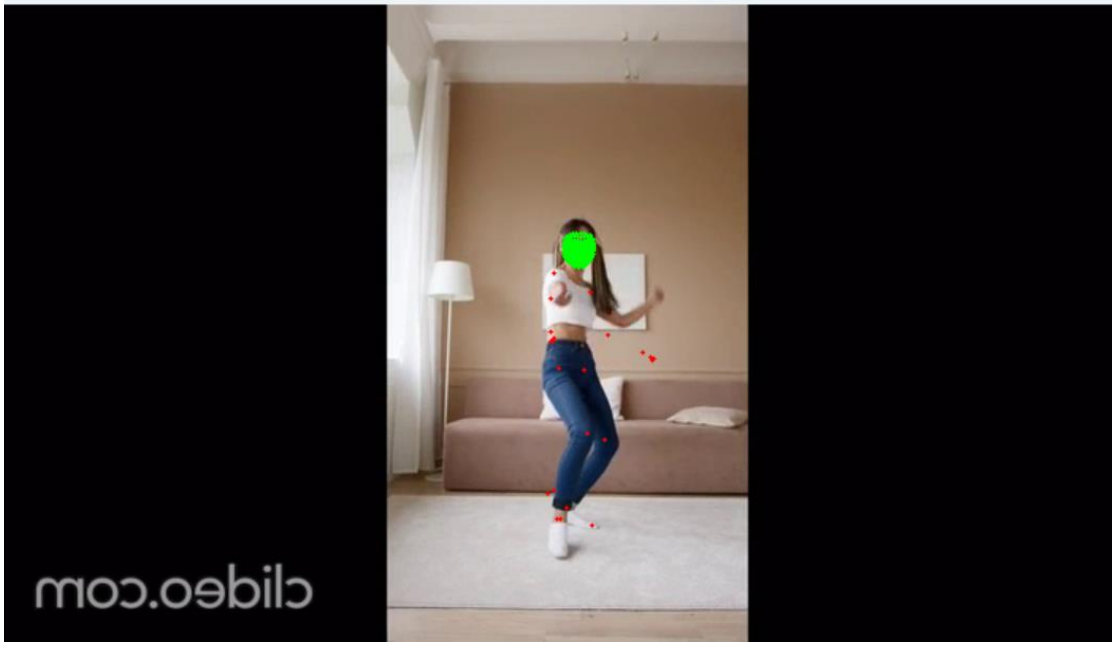


Figure 48: Example of landmark data concentrating at an area due to user too far from camera.

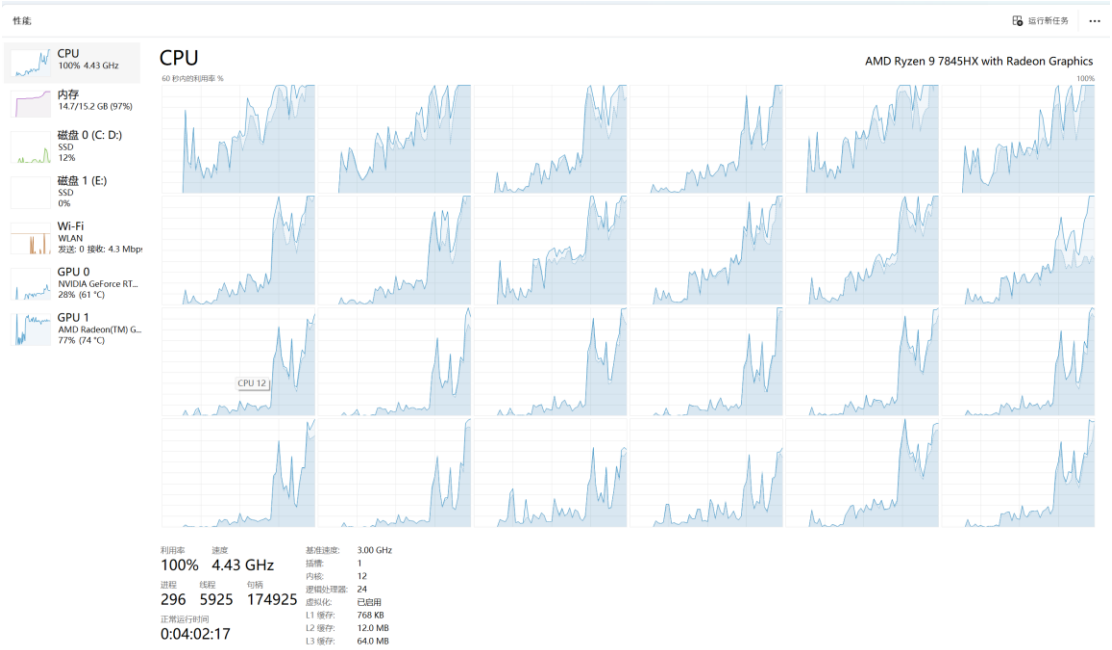


Figure 49: Workload of CPU during running of application.

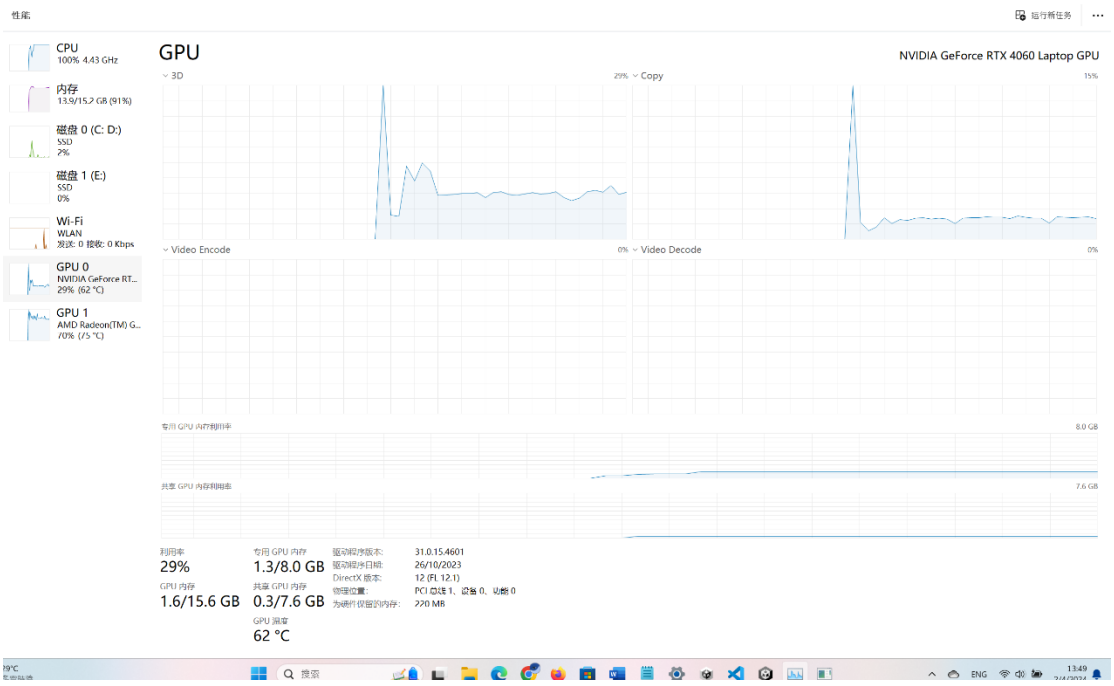


Figure 50: Workload on GPU during running of application.