Instituto Politécnico de Tomar

Escola Superior de Tecnologia de Tomar

José Luís Picão Santos

Gesture recognition for interaction tasks in immersive reality

Supervisors

Prof. Gabriel Pereira Pires

Prof. Luís Agnelo de Almeida

A Thesis Report submitted to obtain the degree of Master Science in Informatics Engineering

ACKNOWLEDGEMENTS

Gostaria de expressar a minha gratidão aos meus orientadores, Gabriel Pires e Luís Almeida, por toda a orientação, discernimento e apoio ao longo deste projeto.

A todos os que de uma forma ou de outra deram o seu apoio e fizeram com que a realização deste projeto fosse possível.

Por fim, agradeço à minha família pela motivação, confiança e apoio constante dado ao longo do projeto.

Institucionalmente, agradeço os recursos e condições oferecidos pelo laboratório VITA.IPT – Vida Assistida por Ambientes Inteligentes e pelo Instituto Politécnico de Tomar.

ABSTRACT

With continuous advancements in Immersive technology, enhancing human-computer interaction (HCI) with virtual environments has become a relevant topic. Commercial virtual reality (VR) systems are equipped with handheld controllers, which are limited in naturalness and user interactions. Immersive technologies benefit from the sense of presence, immersion, and embodiment to augment user experience.

This project addresses these aspects by developing a computer vision-based dynamic gesture recognition system with full-body pose tracking integrated into a virtual reality experience. Unlike most existing applications that use only body tracking, our system combines both body and hand tracking and includes a gesture recognition module. To demonstrate the effectiveness of the approach, we propose a 3D virtual immersive environment scenario where the user engages in a game of "rock-paper-scissors" against the system, aiming to outperform it.

This project uses the Mediapipe framework as a body pose tracking mechanism to extract the user’s articulation coordinates. The obtained data is processed using a long-short-term memory (LSTM) deep neural network (DNN) to classify dynamic gestures. The game engine Unity 3D is used to represent the avatar and to present the immersive experience to the user.

To validate the developed system, experimental tasks were conducted with eight participants. Each participant had to play the game 5 times. The system was validated quantitatively by measuring the online classification accuracy, and the

subjective sense of realism, presence, involvement and system usability were evaluated using the iGroup Presence Questionnaire (IPQ).

Results show the effective possibility of tracking both body and hands, with potential applications ranging from rehabilitation to sports. Regarding the integration of pose tracking for controlling the avatar in the 3D immersive VR environment, results indicate the experience causes a positive impact on users in terms of realism and presence, as evidenced by a 73.44% score on the IPQ. However, users reported a mismatch in their movements and the avatars during the experience. The performance of the gesture recognition classification model did not match the accuracy achieved during the offline validation and testing phases. This lack of generalisation of the model is attributed to the limited number of training samples and the low variability in participant's gestures, as the dataset included only a single individual.

Overall, the biggest challenge of the project was the integration of the pose data to control the avatar. Nevertheless, the results demonstrated the feasibility of combining computer vision-based gesture interaction and full-body tracking in a VR experience.

Keywords

computer vision; pose tracking; skeleton tracking; gesture classification; immersive reality

Contents

List of Figures

List of Tables

Acronyms

HCI - Human-Computer Interaction

HMI – Human-Machine Interaction

VR – Virtual Reality

HAR - Human Activity Recognition

HMD – Head Mounted Display

DL – Deep Learning

AI – Artificial Intelligence

DNN – Deep Neural Network

ML – Machine Learning

ANN – Artificial Neural Network

RNN – Recurrent Neural Network

LSTM – Long Short-Term Memory

DNN – Deep Neural Network

CNN – Convolutional Neural Networks

API – Application Programming Interface

DLT – Direct Linear Transform

PnP – Perspective-n-Point

ASM – Active Shape Models

YOLO – You Only Look Once

1 Introduction

1.1 Context and Motivation

The technological sector has been experiencing exponential growth in innovations, with increasingly capable hardware, more efficient algorithms, and easier access. In daily life, the interaction with technology, whether for entertainment, work, shopping or communication, has become a routine. The typical means of computer interaction, including mice, keyboards, controllers, touchscreens and joysticks, may not be sufficient to exploit all interaction possibilities. The trend in the evolution of these interaction systems aims to make tasks more natural, consequently adapting the elements of Human-Computer Interaction (HCI) and Human-Machine Interaction (HMI) to this reality in a way that does not hinder technological advancement.

The supremacy of the typical HCI/HMI instruments mentioned in the previous paragraph is due to user habituation and the fact that they are sufficient for performing the most necessary tasks. However, some tasks or areas could be enhanced for which these instruments are not enough or even suitable, such as interactions with 3D environments where the interaction significantly improves the user experience. Traditional controllers offer a straightforward way of interacting with the digital environment but can be less efficient in Virtual Reality (VR) [1].

There is a pressing need to discover new methods for extracting information during interaction processes to enhance HCI/HCM devices. Today, Human Activity Recognition (HAR) is increasingly used as a non-verbal interaction

methodology for VR. Common approaches for HAR tasks include mobile devices [2], data gloves [3], wearable sensors [4] and computer vision systems [5]. However, commercial VR headsets like the Oculus Meta Quest 2 and HTC Vive require two handheld controllers, which are limited in terms of interaction. Hence, developing a gesture-based interaction system would be more entrancing to users [1]. Moreover, many researchers have found that when users are given a virtual body that matches their movements, the immersion level increases. Of all the interaction techniques mentioned above, computer-vision-based ones are the only ones that offer the most freedom and the least constraints since the user does not need to wear other equipment besides the Head-Mounted Display (HMD). Due to the abovementioned limitations, this project aims to explore gesture-based VR interaction using computer vision to augment immersion and intuitiveness.

1.2 Objectives

This project proposes the development of a virtual experience that integrates dynamic gesture recognition as a form of interaction with the 3D environment without requiring any wearable sensor or controller. The project's goals can be specified into three key points:

- Use computer vision techniques to track the movement of the user's body and hands.
- Dynamically classify a set of user gestures using the information obtained from user tracking (referred to in the previous point).

- Develop a virtual immersive experience that presents the user's body motions in the digital realm and takes advantage of gesture recognition to improve the experience.

1.3 Main Contributions and Developments

The main contribution of the project is the development of a system that integrates computer vision-based dynamic gesture recognition and full-body pose tracking within an immersive environment. Unlike most existing applications that use only body tracking, our system combines both body and hand tracking and incorporates a gesture recognition module. Moreover, static gestures are the most common type of gesture recognised in VR. Our solution allows users to watch their movements in the virtual environment and interact with an experience demonstrating a gesture identification system to beat the user in a "rock-paper-scissors" game by predicting the user's gesture.

The project uses Mediapipe framework, a state-of-the-art, pre-trained holistic solution to track the user's body and obtain 3D articulation coordinates. This body and hand motion data is fed into a Long Short-Term Memory (LSTM) that was trained to classify a set of pre-defined gestures related to the game "rock-paper-scissors". For that purpose, a dataset was created to train the LSTM model. To enable real-time detection, a multi-threaded UDP Python Client was developed to track body and hand movements, classify gestures, and send this information to a graphics and rendering application built in Unity.

In Unity 3D, a C# server receives the information from Python. To integrate the information into the virtual environment, three standalone projects were created:

- **Skeleton –** plots the users' articulations from Mediapipe's output in a 3D environment, visually displaying the user's movements.
- **Avatar –** integrates a 3D avatar that mirrors the user's movements to the avatar's with mappings established between Mediapipe's landmarks and the avatar's physical skeleton.
- **VR Integration –** combines the avatar and motion tracking from the previous project, adds the user's point of view via an HMD, and implements the rock-paper-scissors game.

Figure 1 presents the main modules that were developed to integrate body and hand tracking, gesture classification, and VR interaction tasks.

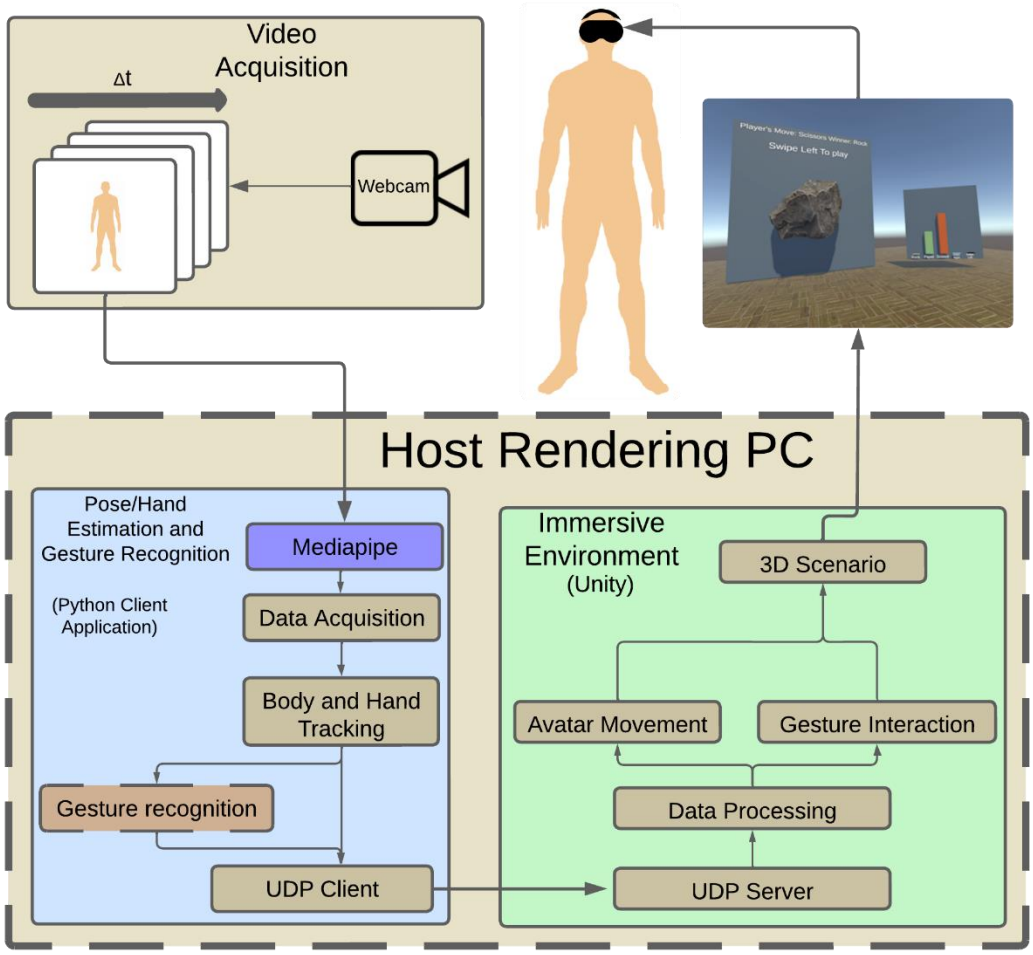


Figure 1 - Immersive Reality System Overview

1.4 Document Structure

This Master report comprises six chapters and one appendix:

- **Chapter 2** introduces the concepts related to technologies and techniques used in development.

- **Chapter 3** presents the State of The Art.

- **Chapter 4** describes all developments carried out and challenges encountered.

- **Chapter 5** presents the experimental results.

- **Chapter 6** contemplates the conclusions about the work developed and future work considerations.

- **Appendix** contains the questionnaires used during the experimental phase.

2 Concepts, Background Methods and Materials

This chapter covers the key concepts necessary to understand both the practical components and theoretical aspects related to the project.

2.1 Introduction

Computer vision, a subset of Artificial Intelligence (AI), is a field that enables computers to acquire, process, and understand visual information such as images and videos. By leveraging AI algorithms, computer vision facilitates the automatic analysis of visual data, enabling computers to 'see' the world around them and perform tasks such as object identification and tracking, as well as human activity recognition. This process can replace physical instruments, such as controllers or wearable devices, used to implement the human-computer interface by creating a more natural form of interaction.

When recognising human interaction with devices or other people, it is essential to define what constitutes a gesture or a movement in order to identify what should be extracted from an individual's actions:

- **Human Body Movement** refers to the change of position or bodily movements in space and time involving the application of varying degrees of force [6].
- **Gesture** is a pose or physical movement of the hands, arm, face or body that is meaningful and informative [7].

According to [8], the hand is the most used body part, as seen in the graph in Figure 2. For this reason, the gesture recognition component of the project focuses only on hand gestures.

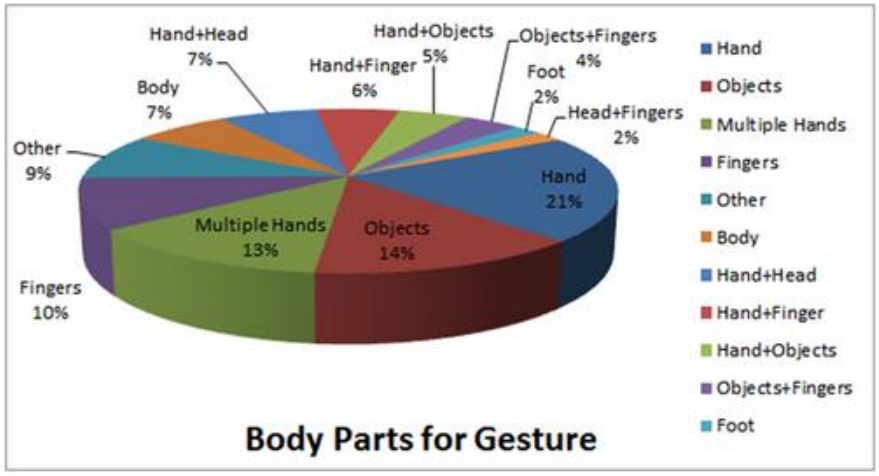


Figure 2 - Percentage of use of Body Parts for Gesturing *[8]*

Computer vision opens the possibility of extracting intention from images or video, a process known as gesture recognition. This area of research has practical applications in diverse fields, including gaming, robotics, medicine, and virtual and augmented reality environments. Gesture recognition includes a subarea called gesture classification, which involves categorising the positions and movements of limbs into predetermined classes. The spatial-temporal dimension of gestures allows their subdivision into static and dynamic gestures [9].

According to [10], deploying a gesture recognition system can be categorised into two categories: the traditional and deep learning approaches. The first involves using a hand-crafted feature extractor for the gesture recognition task.

In contrast, the second one consists of the use of deep learning methods, where the feature extraction task is automated. Traditional recognition methods are fast but have poorer generalisation ability and lower efficiency. The features evaluated in both techniques involve the kinematic visual data (e.g. joint position, body texture, edge, shape).

A gesture recognition system typically comprises a real-time mechanism for capturing information about the user's actions (e.g., camera, wired gloves), an algorithm for extracting features from the captured information to identify the user's intention, and finally, classify those features and present the classification/recognition output to the user or a third-person.

These components will be further discussed in the following sections.

2.2 Classification Models

A classification problem occurs when an object needs to be assigned to a predefined group or class based on several observed attributes related to that object [23]. Classification relies on Machine Learning (ML) one of the pillars of artificial intelligence, enabling learning from datasets, which can be divided into structured and unstructured data. Structured data are scarcer because they involve some form of preprocessing and are required to be labelled. Depending on the type of data, the algorithms used will also differ, as the relationships between the features to be identified in the data may or may not be implicit. In the case of labelled data, machine learning techniques are called supervised, while the others are called unsupervised.

Besides supervised vs unsupervised models, classification models can also be grouped into sequential and non-sequential. Non-sequential models process data such that the learning does not consider any relation between input data. Some examples of non-sequential models include Artificial Neural Networks (ANN), Regression Tree and Support Vector Machine. In contrast, sequential algorithms consider the time factor, making it possible to analyse motion and recognise human activity. Examples of sequential algorithms are Recurrent Artificial Neural Networks (RNN) and Long Short Term Memory (LSTM), which are specialised forms of ANNs designed for processing sequential data.

The next sections will briefly present classification methods related to the project.

2.2.1 Neural Networks

First introduced in 1943 by [11], inspired by the networks of biological neurons, ANNs are the core of DL. The basic unit of an ANN is the perceptron, represented in Figure 3, whose similarity with a real neuron lies in how neurons and perceptrons operate independently yet are interconnected. This connection aims to mimic the network structure of the brain.

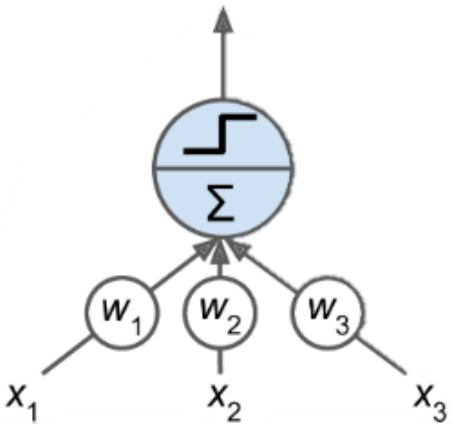


Figure 3 - Representation of an artificial neuron *[12]*

The perceptron receives input values, each input connection has an associated weight. In Figure 3:

- X_1, X_2 and X_3 correspond to the input features.
- W_1, W_2 and W_3 are the weights associated with each input connection.
- $\sum$ represents the weighted sum.

The output of a perceptron is given by $h_W(x) = step(z)$ where:

$$z = X_1 * W_1 + X_2 * W_2 + X_3 * W_3$$

The connection of multiple layers, as shown in Figure 4, allows for executing more complex tasks with additional weights, operations, and non-linearity. However, stacking multiple layers, where the next perceptrons receive as input the value of $h_W(x)$ will output a linear result, no matter how many layers are stacked. For this reason, activation functions play an essential role in NNs, as they guarantee the non-linearity of the output of one layer to the next. This way, the ANN can identify classes whose decision boundaries are non-linear.

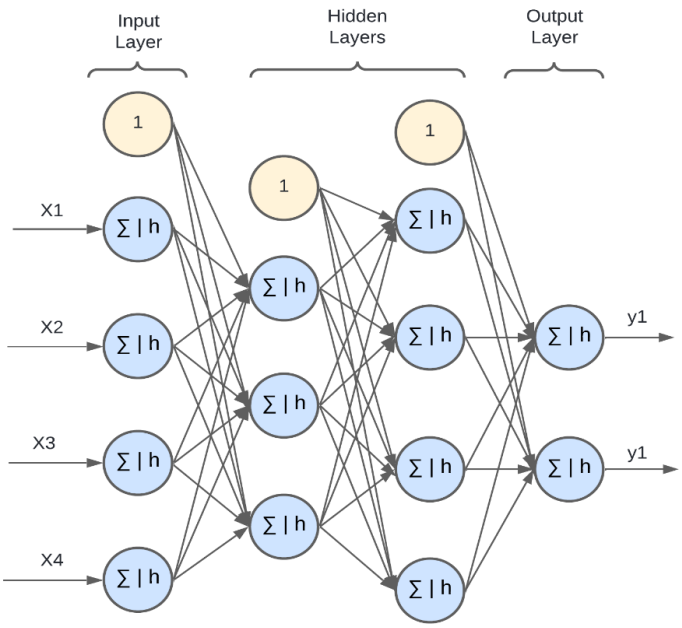


Figure 4 - Multi-layer Neural Network

There are various types of activation functions, such as:

- **Sigmoid**: This function can be used in hidden layers but is more typically used in the output layer of binary classification, as its output value ranges between 0 and 1, as displayed in Figure 5.
- **Rectified Linear Unit (ReLU)**: This activation function is one of the most used. It requires simpler mathematical operations and thus has a more negligible computational cost. As a linear function, the output can range from 0 to +∞, whose graph is visible in Figure 5.

Additionally, in classification tasks, the output layer includes a normalisation function called:

- **Softmax:** Commonly used in the output layer for multi-class classification tasks, converts the raw output scores into probabilities, as

illustrated in Figure 5. It ensures that the sum of the probabilities across all classes equals 1, with each probability ranging between 0 and 1.

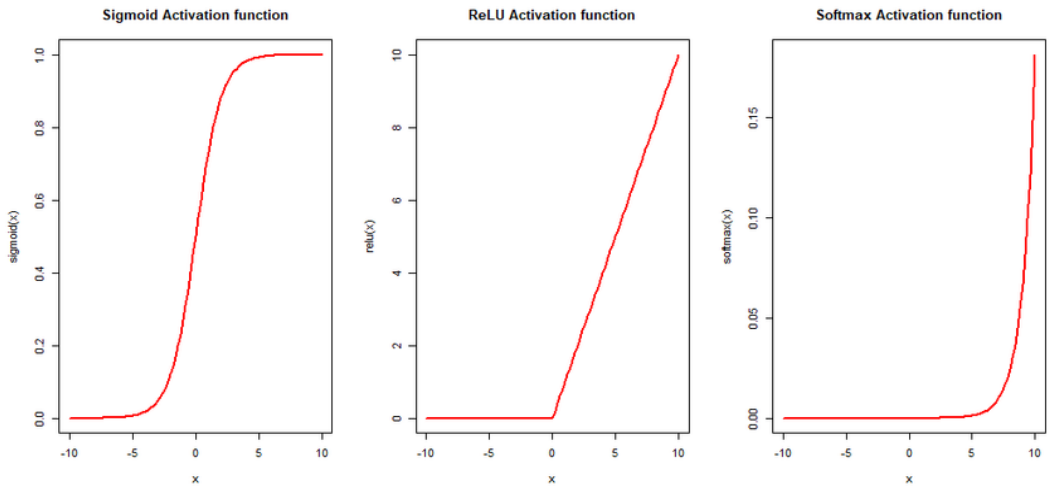


Figure 5 - Sigmoid, ReLU and Softmax Activation Functions graphs *[13]*

When all the neurons in a layer are connected to all the neurons in the next layer, it is called a fully dense layer. Moreover, the bias is a constant offset in the activation of neurons, allowing the model to capture patterns that cannot be represented solely by the input features.

DL is a subset of machine learning. It comprises multiple layers of neurons, including input and output layers, typically with several hidden layers in between. Current applications of DL include Computer Vision, where it is used in image processing to solve problems such as segmentation and detection. Before the advances in Computer Vision, extracting features was a complex task, particularly when there were many classes [14]. DL introduced the concept of end-to-end learning, where the algorithm is given annotated data with multiple classes, and the underlying patterns and most descriptive features of

classes are automatically calculated. It is established that Deep Neural Networks (DNNs) can achieve record-breaking results on challenging datasets [15].

2.2.2 Long Short-Term Memory (LSTM)

An LSTM is an ANN type suitable for sequence prediction tasks capturing long-term dependencies [16]. A network cell is composed of a forget gate, input gate and output gate, as seen in Figure 6. The forget gate controls the information to be forgotten in the cell state, which captures and retains long-term dependencies in sequential data. The input gate is responsible for adding information to the cell state, and the output gate controls the information that will be added as input in the next step. The self-parametrized controlling gates decide whether the information will be kept in the cell. The forget gate (f_t) decides the discarding of information depending on the previous cell's output (h_{t-1}) and input vector (x_t).

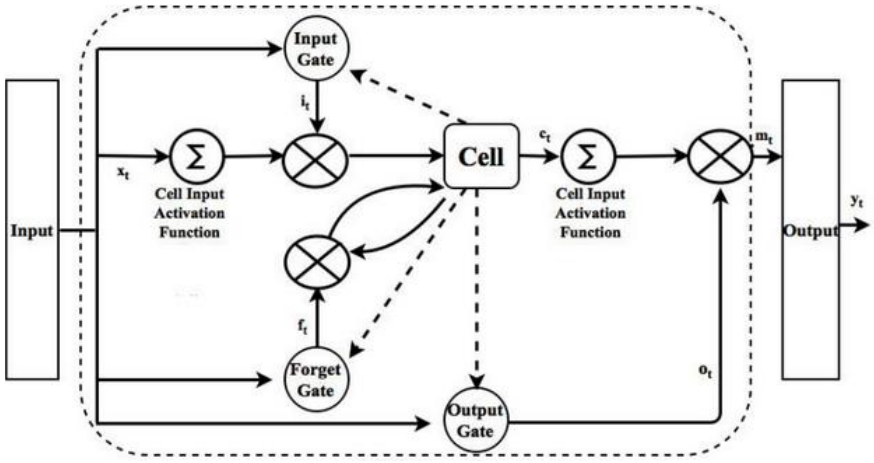


Figure 6 - LSTM network cells *[17]*

The output gate controls whether the current cell state information is visible.

2.2.3 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) emerged from [18], inspired by studies of the brain's visual cortex. These types of ANN distinguish themselves in their performance on complex visual tasks, voice recognition and natural-language processing.

A traditional DNN is suitable for recognition tasks on small images. However, on larger images, it breaks down due to the large number of parameters it requires. For example, a photo with a size of 100 by 100 has 10,000 pixels. This requires a first layer with 10,000 neurons, leading to a total of 10 million connections just on the first layer. CNNs solve this problem using partially connected layers and weight sharing.

The network comprises convolutional layers, where most of the computation occurs. A convolution is a mathematical operation similar to cross-correlation that slides a signal by the other and measures the sum of the pointwise multiplication.

The neurons on the first layer are not connected to every single pixel in the input image but only elements in their receptive fields (see Figure 7). Each neuron is connected to only neurons located in a small rectangle of the next layer. This makes it possible for the network to concentrate on lower-level features on the first layers and make its way up to the higher-level features on the next layers.

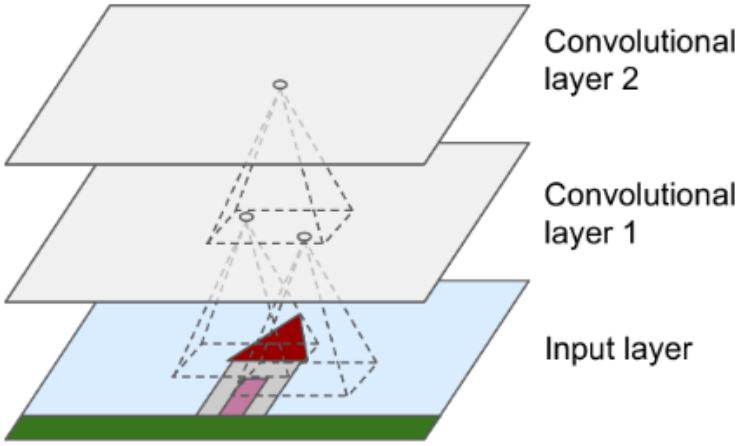


Figure 7 - Convolutional layers receptive field representations *[12]*

A convolutional layer needs a filter (convolutional kernel) to extract a feature and can have multiple filters. Each pixel in the image is traversed, and the inner product operation is performed to obtain a new feature map. There will be an output for each filter if there are multiple filters in a layer. The filter is obtained through training using gradient descent and backpropagation.

As shown in Figure 8, when a filter is applied to a matrix 5 by 5, the result is transformed into a 3 by 3 matrix, where each cell value is obtained by applying the following correlation operation:

$$Result_{\ y,x} = \sum_{i=-n2}^{n2} \sum_{j=-m2}^{m2} Input(y-i, x-j) * Filter(i,j)$$

Where:

- $n2$ is half of the height of the filter.
- $m2$ is half the length of the filter.
- x is the column position of a pixel.
- y is the line position of a pixel.

The output of the convolution in Figure 8 is given y:

$$Result_{1,1} = 9 * 1 + 4 * 2 + 1 * 0 + 1 * 0 + 1 * 1 + 1 * 4 + 1 * 1 + 2 * 0 + 1 * 1 = 24$$

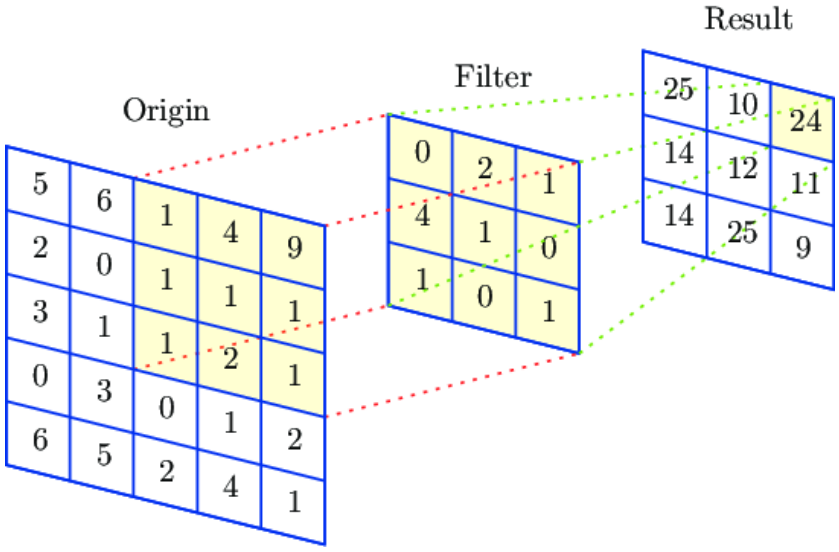


Figure 8 - Convolution operation with a 3 x 3 filter and no bias *[19]*

Because the neurons on a feature map share the same parameters, the parameters are significantly reduced in the model. Once the CNN has learned to search for a feature in a location, it can search for the same feature in any other location. Even with reduced parameters, CNNs require large amounts of RAM, specifically during training. To decrease the computational load, memory usage and parameter numbers, pooling layers are used, and their objective is to subsample the input image. They also introduce a level of tolerance to invariance to small translations.

Similarly to convolutional layers, pooling layers are only connected to neurons in their receptive field, as shown in Figure 9. However, a pooling neuron has no weights. All it does is aggregate the inputs using a max or mean function.

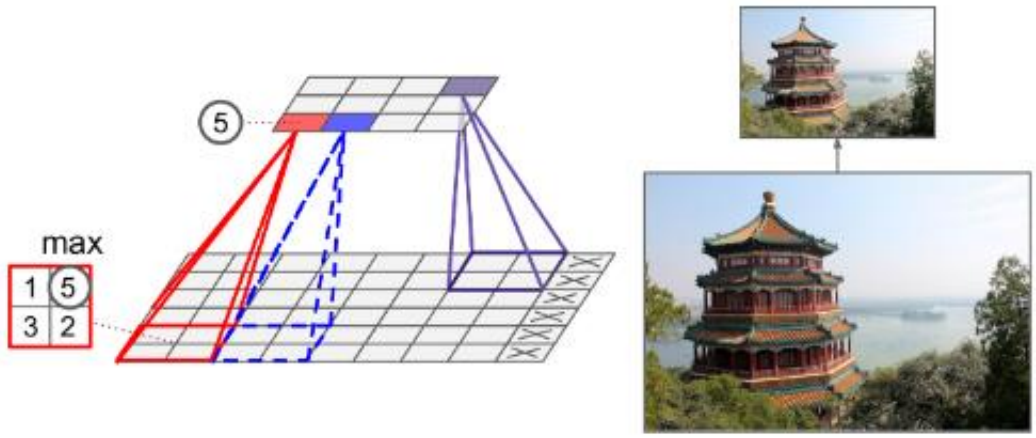


Figure 9 - Max pooling layer *[12]*

CNNs are used in frameworks like Mediapipe for object detection, hand tracking, face detection, and pose estimation tasks. These frameworks will be approached in the following subsections.

2.2.4 Machine Learning Application Programming Interfaces (APIs)

To implement the algorithms mentioned above, there are APIs that serve as an interface between ML models and practical applications. They provide stability since the model can change, but the code does not and validate data passed onto the model.

The most relevant are:

- **Keras** [20] is a Python high-level interface to backend libraries like TensorFlow and others. It facilitates rapid implementation by simplifying building and training DL architectures and making the complex operations of TensorFlow more accessible.

- **TensorFlow** [21], a framework suitable for intricate and higher-level operations. It is built to support mobile and edge devices.
- **Pytorch** [22] is a fully featured framework commonly used in applications that involve computer vision and language processing

In Table 1, a comparison between APIs is made.

	Keras	TensorFlow	Pytorch
API Level	High	High and Low	Low
Architecture	Simple	Not easy to use	Complex
Speed	Slow	Fast	Fast
Written in	Python	C++, CUDA, Python	Lua

Table 1 - ML API comparison

2.3 Machine learning approaches for body tracking

2.3.1 Pose Detection Deep Learning Models

Human body Pose estimation, also called vital point detection, is a computer vision technique that pinpoints key joints of the human body in images or videos. Research on pose estimation began with the emergence of computer vision as a field in the 1960s and 1970s. Traditional computer vision focused

on geometric calculations and feature-based approaches to estimate the pose of objects on human subjects, like the Direct Linear Transform (DLT) algorithm. By determining the camera's pose, it is possible to calculate the positions of persons or objects in the room.

The Perspective-n-Point (PnP) algorithm is similar to DLT. This originated from the camera calibration problem, which has many implications in computer vision, including 3D pose estimation and augmented reality. It consists of estimating the pose of a calibrated camera based on several 3D points in the world and their corresponding projections on the image. As shown in Figure 10, it is inferred from an image of a 3D object.

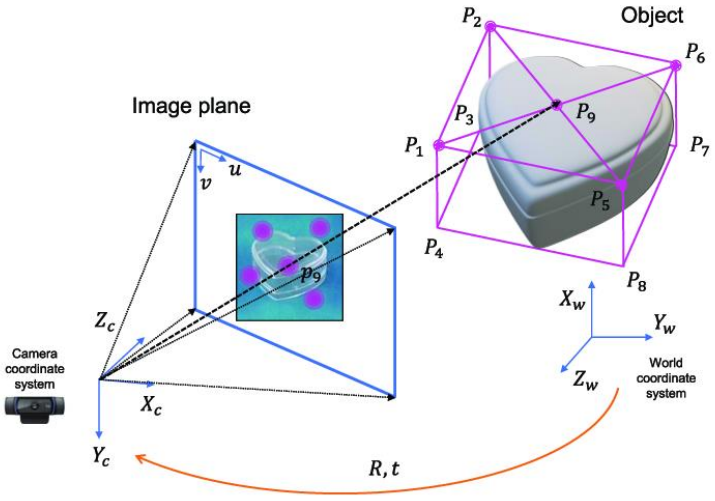


Figure 10 - Perspective-n-Point Diagram *[24]*

A few decades later, model-based approaches appeared, as the vision methods had many limitations. For example, they were incapable of learning from a large amount of data and suffered from limited model representation capability [14].

For example, they were sensitive to noise and outliers and relied heavily on camera parameters. However, in shape alignment and pose estimation tasks, this mechanism alone is not enough. For this reason, PnP is combined with Active Shape Models (ASMs).

ASMs are statistical models that combine shape and appearance information to estimate the pose or shape of an object in an image. First, they generate a suggested shape by looking at the image for the best position for each point, and then they conform the shape to the distribution model.

The next iteration of algorithms was feature-based, which identified and matched distinctive image features to estimate the pose of objects or human subjects. The three most relevant pose detection models are Mediapipe, You Only Look Once (YOLO) and OpenPose.

Mediapipe was chosen as the main algorithm to extract the user's landmarks due to its easy implementation and was the first model used. Due to time restrictions, it was the one integrated, as it had an out-of-the-box solution and detected significantly more landmarks than the other.

2.3.2 Mediapipe

Mediapipe was created by Google in 2010 [25] while improving machine learning and computer technologies. Nowadays, it is an open-source framework for building and developing machine learning pipelines.

Mediapipe enables the customisation of ML pipelines with Mediapipe Studio, a web-based application for customising and evaluating models. Any existing model can be retrained with custom data to create a model specialised for the user.

The framework consists of three main components:

- **Packet:** basic data flow unit.
- **Graph:** processing unit that defines the packet flow between nodes.
- **Nodes:** produces and consumes packets. Each node defines several in and output ports.

Mediapipe has several out-of-the-box solutions, including object, hand, face, pose detection, image generation, and image segmentation.

For this project, it is essential to mention the tasks that include segmentation and classification of body parts.

The Holistic solution joins the different models for hands, pose, and face components. The pipeline is visible in Figure 11. Each is optimised for its purpose, yet the input to one element is not suited to others. For example, the pose estimation model receives a fixed-resolution video frame (256*256) as input. If the hand pose detection models were to accept this frame as input, they would not have the necessary resolution to make an accurate detection, and the same would happen for the other two solutions.

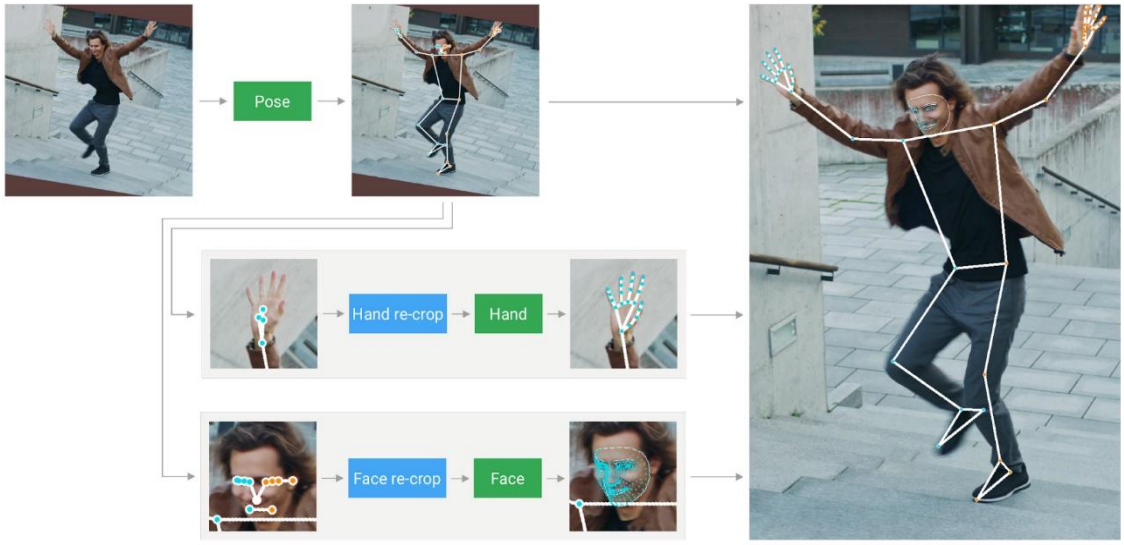


Figure 11 - Mediapipe Holistic Pipeline *[25]*

The output data from Mediapipe is obtained in the order specified in Figure 12. The data from different Mediapipe solutions are separated into different attributes of the returned object. In total, the solution returns 75 landmarks, 33 from pose and 21 for each hand. Face mesh landmarks are not considered in the project. Each landmark has 3 coordinates, correlated with the x, y, and z coordinates from a 3D referential. Pose returns an extra value that refers to the visibility of the landmark contained between the $[0.0, 1.0]$ range.

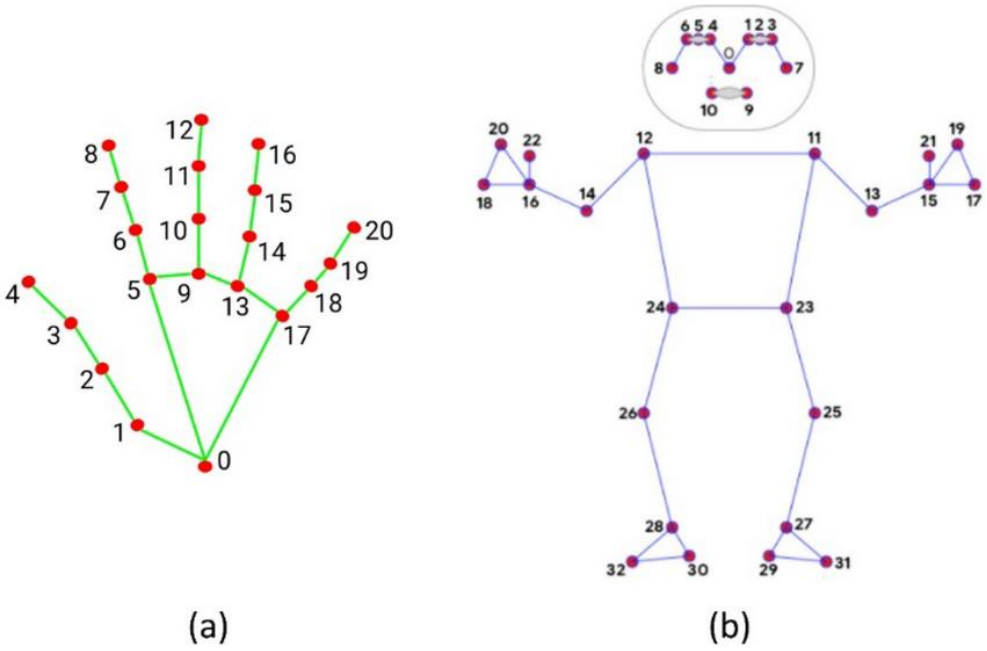


Figure 12 - Mediapipe Holistic Hand (a) and Pose Landmarks (b) *[26]*

2.3.3 OpenPose

OpenPose was the first real-time multi-person system to jointly detect human body, hand, facial and foot key points on single images. Based on the work [27], which proposed a 2D pose estimation in a real-time model. It features real-time 3D single-person key-point detections and 2D multi-person key-point detections. The first layers of the library extract features from an image, and then the extracted features are used as input into two parallel divisions of convolutional network layers.

2.3.4 YOLO

YOLO is a famous algorithm for its object detection characteristics, introduced by [28]. It can output the position and category of a bounding box through the NN.

The algorithm has several out-of-the-box solutions, such as:

- Object detection, classification and segmentation.
- Pose.

2.3.5 Model Comparison

The main features of interest between each model are compared in Table 2. The evaluation of the points described in the table below contemplates the features available on installation of the model, not what is impossible to do. For example, both YOLO and OpenPose can detect a hand's landmarks. However, the model needs to be trained first.

Characteristic	Mediapipe	Yolov8	OpenPose
Human Skeleton Pre-trained Features	Hands, Head, Pose	Pose	Pose
Python API	Yes	Yes	YES
Built-in Cuda Support	No	Yes	Yes
Max. Number of People	1	>1	>1
3D Landmarks	Yes	Yes	No

Table 2 - Pose Detection Algorithm Feature Comparison

2.4 Virtual Reality

VR is an immersive computing technology. It simulates a virtual environment that immerses users to the extent that they feel they are there. When mentioning VR, it is important to address the following concepts [29], [30]:

- **Immersion** refers to the level of fidelity of the physical stimulation provided by the technology to the human system and the consequent reaction and sensitivity of the human system to those motor inputs, assuming consistency between actions and the resulting sensory feedback.
- **Interactivity** is the user's influence when interacting with the virtual environment.
- **Sense of Presence** is the psychological product of technological immersion and refers to the feeling of “being present” in a virtual environment through technological mediation.
- **Embodiment** aims to process the user’s features in an environment as if they were the properties of its own biological body.

To achieve this, it integrates real-time processing of 3D images, sounds, and other forms of sensory input to create a virtual space with which the user can interact. and possess interactivity in response to the user’s actions. The virtual environment is presented to the individual through HMD, which typically consists of two screens for each eye and can be accompanied by audio through earphones or headphones. The user’s field of vision is adapted according to their movements in the real world through tracking with gyroscope sensors and/or using infrared emitters.

2.4.1 Virtual Reality Devices

Virtual reality devices are divided into two categories based on their function: input and output. Output devices are visualisation and audio technologies, including haptic actuators and vibration. HMDs fall into this category and can be further divided into three subcategories: wired HMDs, wireless HMDs, and those that use a smartphone display to present the image. Standalone displays do not require a connection to a computer (as is the case with wireless ones), while dependent displays need to be connected to a processing unit. These devices have accelerometers and gyroscopes to detect user movements, typically used with other tracking systems (e.g., cameras or infrared).

An important aspect common to all VR systems is the input device. This category includes headsets and controllers. The latter usually have buttons, analogue sticks, and position-tracking systems. Base stations are used to monitor the movements made by the user. These are necessary to identify the exact position of each device, mainly relying on optical tracking systems (IR laser beams). In HTC systems, both the HMDs and hand controllers have IR sensors that detect the intersection of the laser beams generated by the base station (lighthouse) and, through proprietary algorithms, determine the exact position of the device.

2.4.2 Videogame Engines

A game engine is a software framework designed to develop video games. Its core functionality may include a rendering engine and a physics engine. Before the existence of video game engines, the development of a game implied the creation of a game engine for each game. The success of this software is due to the possibility of reusing tools and objects needed for its development, reducing the cost of development.

Unity [31] and Unreal [32] are the most popular game engines. Both enable the design and development of fully immersive environments, but they have major differences and features. Unity has a user-friendly interface and extensive support for many platforms, including VR, and relies on C# for scripting. On the other hand, Unreal is renowned for its graphic fidelity, advanced physics, and backing for C++.

2.4.3 Unity

Unity was the chosen game engine due to previous experience with the framework, intuitive flow with C# scripts, and its large asset store and community. For this reason, referring to concepts relative to the framework is essential:

- **Scenes** are the largest unit of organising objects, representing a single level in a game.
- **Game Objects** can represent physical (i.e. ground, environment) and metaphysical things (i.e. avatar controller, scene manager).
- **Components** define the behaviour of a game object.
- **Assets** are resources that make up the project (i.e. sounds, textures)

The game engine incorporates a script-based action system, enabling the assignment of a script as a component to define the behaviour of a game object. These scripts are structured with pre-built methods such as `Start()` and `Update()`. The `Start()` method executes code before scene initialisation, commonly utilised for setting up object properties, while the `Update()` method is employed to run instructions before each frame is processed.

In the virtual environment, game objects all have the transform property in common, which defines position, rotation, and scale. Game objects can be hierarchical. In this case, a game object's position can be relative to its parent or the world. The same hierarchical property is applied when defining a rotation, and both Euler angles or quaternions can define rotation. Euler angles define the orientation of a general basis in 3D linear algebra.

This representation is susceptible to the gimbal lock problem. The three axes allow for three degrees of rotational freedom. However, when two axes are aligned, as illustrated in Figure 13, one degree of rotational freedom is eliminated. In Unity, this can lead to atypical movements in the avatar's limbs. Conversely, quaternions can represent a 3D rotation without encountering the abovementioned issue. They achieve this by utilising four complex numbers to describe the rotation.

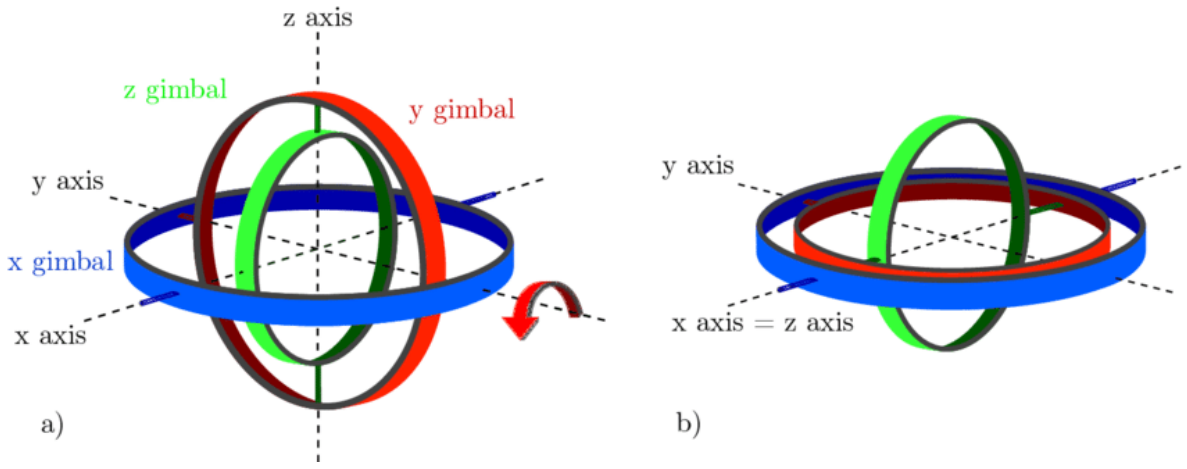


Figure 13 - Gimbal lock visual representations *[33]*

Quaternions provide a convenient mathematical notation for representing special orientations and rotations in a 3D space. A quaternion is represented by four elements:

$$q = q_0 + iq_1 + jq_2 + kq_3$$

- q_0, q_1 and q_2 are real numbers
- i, j and k are imaginary orthogonal unit vectors

A further benefit of quaternions is that the computational effort needed to perform a rotation is less intense as there is no use of computationally intensive trigonometric functions. The use of homogenous matrices to compute composite transformations such as rotations can be simplified using quaternions.

To define a set of movements in the 3D plane, it is important to introduce the concepts of yaw, roll and pitch. These movements are described in Figure 14.

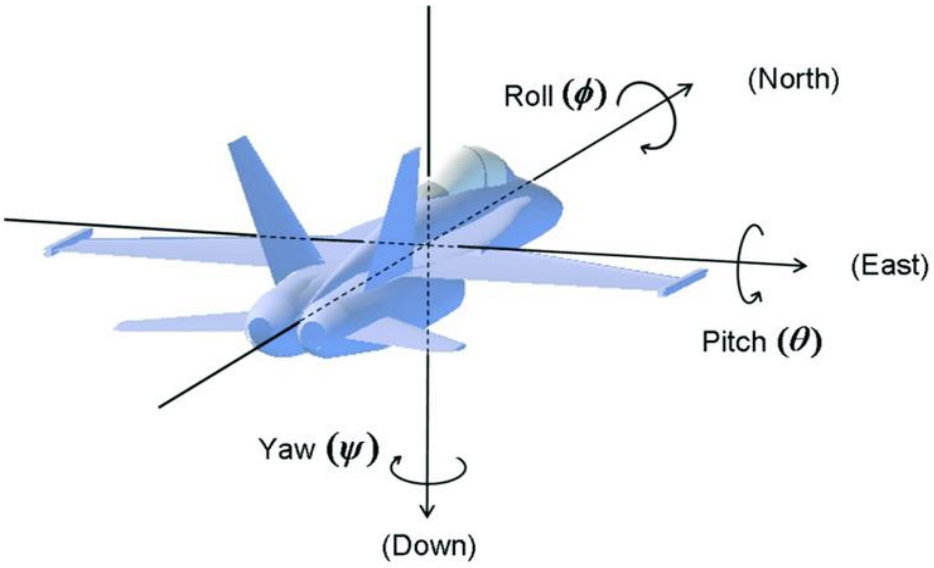


Figure 14 - Movements applied to a 3D Object *[34]*

Avatars in Unity have a humanoid bone structure composed of multiple elements that aim to give the character human-like freedom of movement in animations. This skeleton is composed of HumanBodyBones, observable in Figures *Figure 15* and *Figure 16*.

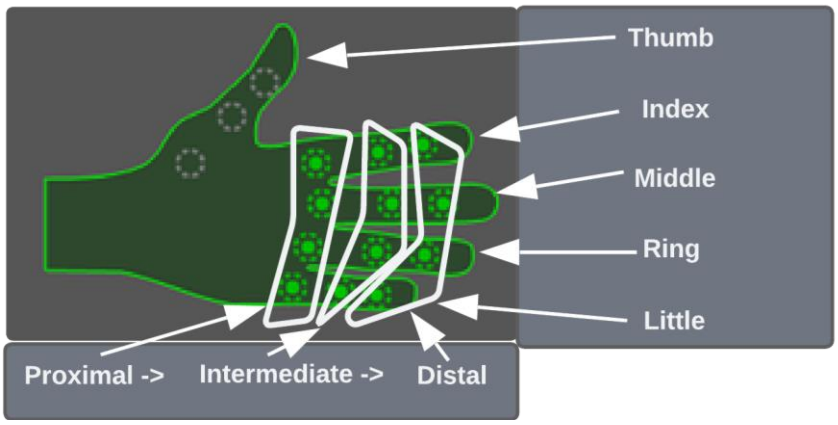


Figure 15 - HumanBodyBones Hand Description *[31]*

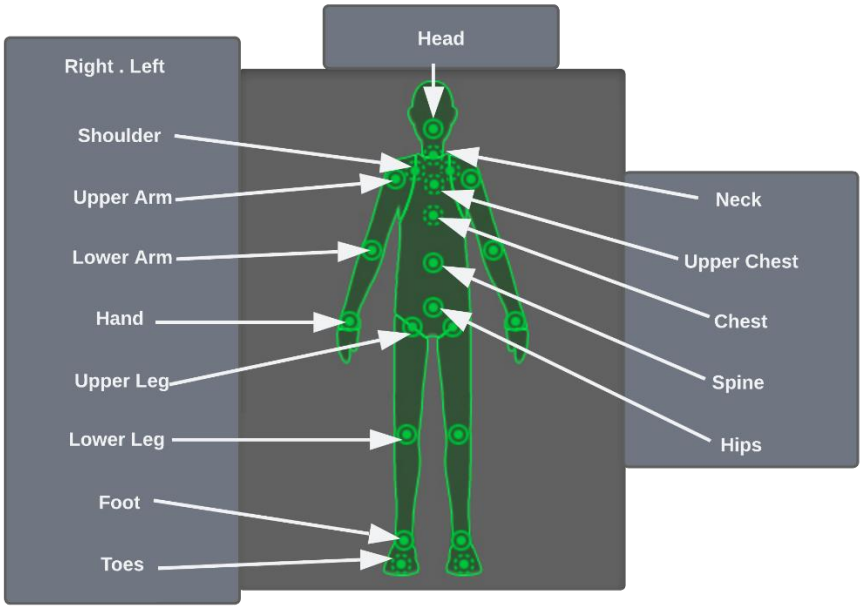


Figure 16 - HumanBodyBones Body Description *[31]*

Unity has the XR package to develop VR, augmented reality, and mixed-reality applications. It facilitates the setup of a project with specific aspects of the type of experience. In the case of VR applications, it sets up the controllers and the respective interaction mechanisms to interact with the environment, as well as a locomotion system and a special user interface (UI). It also provides a system to manage the support for specific devices and manufacturers.

3 Related Work

In the field of immersive reality environments, various studies involving different senses have been conducted to improve the feeling of presence by providing users with outputs of their real-world actions in the immersive environment.

This section briefly presents prior work on hand interaction and gesture recognition in VR.

3.1 Interactions in 3D Virtual Environments

In VR, interactions with the user's hands are supported by hand-tracking interfaces. The aim is to reduce the gap between virtual reality and the actual environment intentions and interactions. Such techniques have been studied [35]. The devices and techniques are essential in capturing the hand's position and orientation in the 3D environment. These input devices can be divided into data gloves, motion sensors and vision-based systems [1].

Data gloves are wearable equipment equipped with inertial sensors that provide data on the hand's physical metrics, such as angular acceleration, yaw, pitch, and hand orientation. In [3], the authors developed the CyberGlove-HT, a set of wearable gloves that wirelessly communicate with a computer to transfer the user's gestures to the virtual world. In experiments conducted by the authors, 100% of the survey participants rated interactions using the CyberGlove-HT as equally or more natural than the traditional controller approach. A wearable device can be equipped with haptic feedback capabilities to augment the experience. Most of these devices are built for the hands and only provide

vibrational feedback [37]. This has been demonstrated in [38] in a simulation of a surgical training VR experience to operate a patient using gloves with haptic feedback. However, the authors in [1] mention that data gloves offer precise hand gesture representation but are associated with the sensor's high costs and fragility.

The tendency to integrate wearable sensors with immersive VR is increasing, which is due to better quality, less costly, easier-to-use devices [39]. Three standard wearable sensors include an accelerometer, magnetometer and gyroscope, which users can conveniently wear [40]. In [4], the authors proposed a skin-like sensor system that can be applied to multiple joints to detect movement based on electrical or mechanical signals. The proposed device was applied on the wrist to detect hand gestures. A previously trained DNN decodes the temporal signals captured by the sensors to generate the corresponding finger motion, visible in Figure 17.

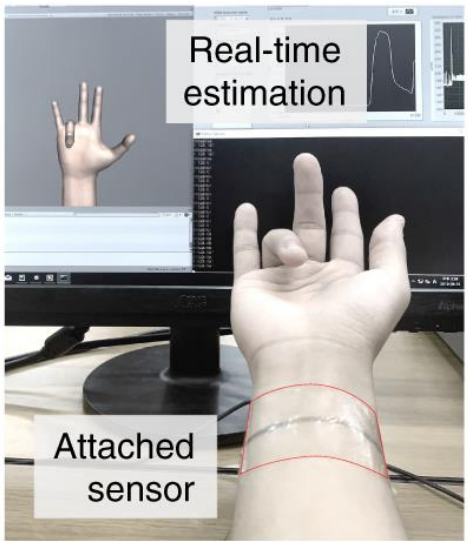


Figure 17 - Finger Movement Real-Time Detection by Wrist Sensor [4]

In recent years, a significant portion of research has focused on vision-based HAR due to the affordability of the systems and ease of data collection compared to previous sensor-based systems [40]. These vision-based systems do not require the user to wear accessories during the interaction, providing a more convenient approach. In [7], the authors state that the challenges of vision-based systems are mainly the changes in illumination from the testing environment and image processing operations, which may be affected by background and performance issues. These systems are mostly coupled with gesture recognition techniques to interact with the VR environment and to infer intentions from human behaviour. Such is the case with [41], who developed a solution for gesture interaction in virtual reality. The experience comprised doing sets of gestures in multiple scenes.

3.2 Gesture Recognition in VR

A vision-based gesture recognition system depends on multiple sequential components, according to [42], namely limb detection, feature extraction and gesture classification.

Limb and feature extraction methods can be implemented, each with its limitations. These techniques can recognise dynamic and/or static gestures. Some of the most relevant traditional methods are linear classifiers [43], which are better at static recognition, Hidden Markov Models [44] and Dynamic Time Warping [42], which are capable of interpreting sequences of data. These

methods have achieved remarkable performance yet require multi-processing and hand-crafted features. For this reason, DL approaches have become increasingly more common in HAR systems [40]. Furthermore, current DL algorithms can do limb detection and feature extraction on their own [25], [28].

The work in [41] reports the development of a gesture recognition system integrated with VR. The authors used OpenPose [27], a DL pose estimation algorithm, to realise the feature extraction task and an ML to handle the recognition. The gestures evaluated involve continuous motion. Thus, the solution is capable of recognising gestures with a temporal component. Their solution results showed the created ML model is capable of effectively recognising gestures, although some gestures performed better than others. The inquiries reported a good usability score of 78.75 out of 100, and the overall interaction score was lower when compared with a controller.

4 Developments

The original idea to sense human pose was to capture data from a 2D webcam to offer flexibility, create a cost-effective solution, and streamline implementation. When dealing with real-world input, it becomes necessary to use an algorithm to gather data on the user's pose, as the camera is limited to two dimensions and obtaining depth data from the image is challenging. Without depth data, there is a risk of incorrectly classifying gestures due to the absence of a dimension our body uses to convey specific intentions.

This chapter presents the developed system's hardware and software architectures. Section 4.1 describes the system architecture. Section 4.2 presents the gesture recognition system, model architecture, the dataset acquisition process and Python Client’s architecture. Section 4.3 describes the integration of the Python Client and Unity, the body data visualisation in the 3D environments, and the integration of the previous modules to create an immersive experience.

4.1 System Architecture

The developed system can be divided into two main modules: Pose/Hand Estimation and Gesture Recognition and Immersive Environment. Each component is isolated from the others, allowing for component modification, whether to fix or improve the service, without affecting the other layers.

As shown in Figure 18, the system pipeline starts with the user's video capture of the user in the real world. The Pose Estimation and Gesture Recognition module on the computer running the solution receives the image data as an array. This value is then passed to Mediapipe as input, where the algorithm detects the user's body landmarks. These values are post-processed to correct the existing offset visible when plotted in 3D. Since the values of the hands and body pose come in separate objects, they are concatenated. The joined data is put into a list containing 30 frames of landmark data corresponding to the temporal data, and the batch is used as input for the LSTM classifier. The classification is put into an object, together with the current frame's key point data and gesture probabilities and serialised into a JSON string. Given the real-time nature of this system, execution speed is crucial UDP communication protocol was employed. To send the data via UDP sockets, it is converted into bytes and sent by the socket client to the server implemented on the Immersive Environment's side (Unity 3D). After receiving the bytes from the message, the resulting JSON string is serialised into an object with an array of key points and a string with the classification. The key points are used to calculate the rotation of the avatar's limbs, and the gesture is applied to the experience to give the user feedback from the virtual environment.

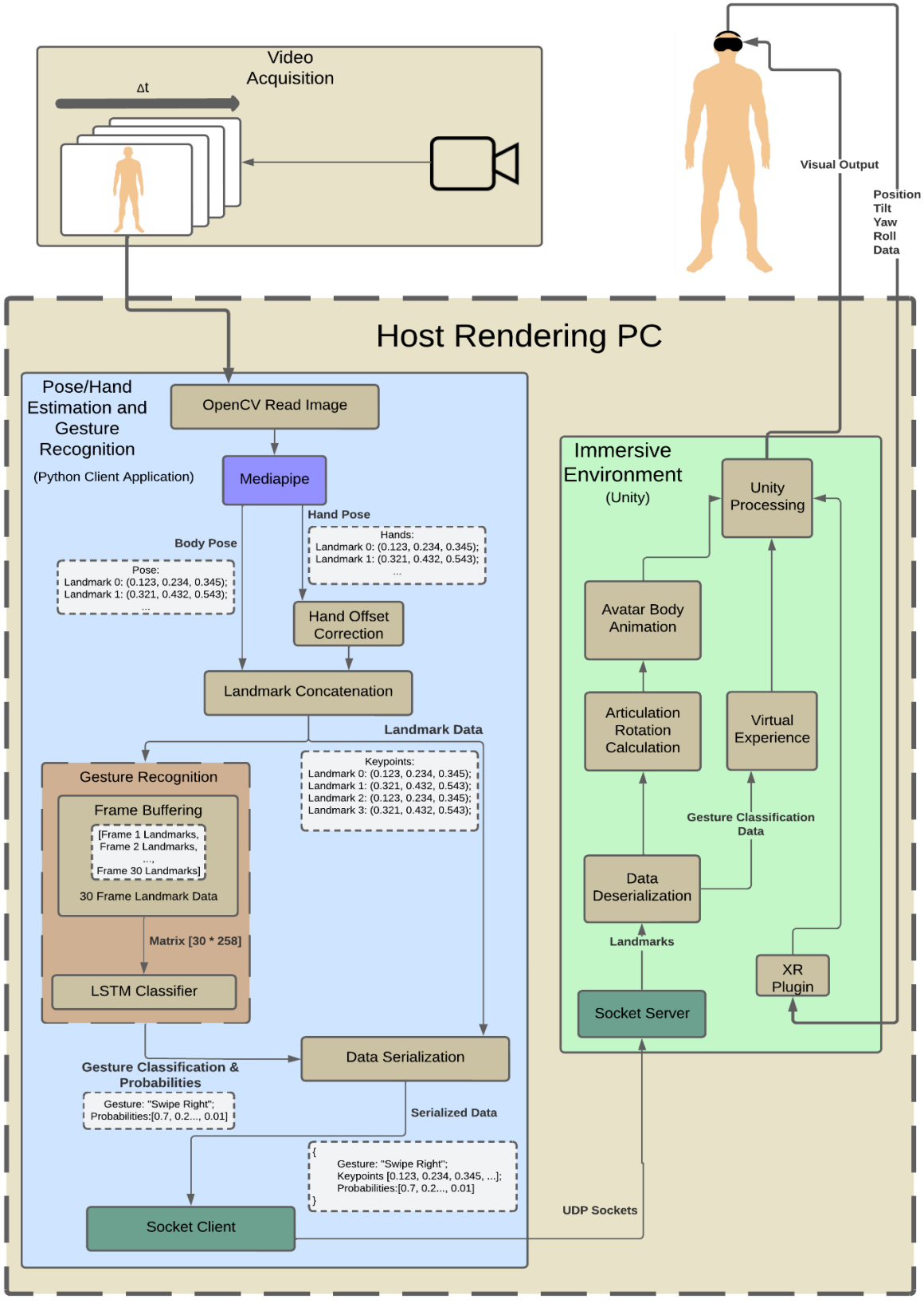


Figure 18 - System Architecture

4.2 Pose/Hand Estimation and Gesture Recognition

The system begins by obtaining the last frame captured by the webcam. The image is then passed as input to Mediapipe, which returns the body's landmarks and left and right hands in separate lists. The pose's body returns an array with 33 elements, each with the coordinates x, y, z and a visibility value, used only as input for the classifier. Each hand outputs 21 elements with the x, y, and z values. Due to the outcomes from different solutions, the hands have an offset when related to the body, visible in Figure 19. However, when plotted in 2D, the problem is not noticeable, such as in the case of Figure 20. This happens due to an offset in the z coordinate of the hands. The main correspondence between the hand and body pose is the wrist, as they are supposed to overlap, as seen in Figure 12. In the body pose, the wrist is represented by landmark 16 and the hand by landmark 0. To solve the issue, the z coordinate of the respective wrist was added to every hand landmark's z coordinate.

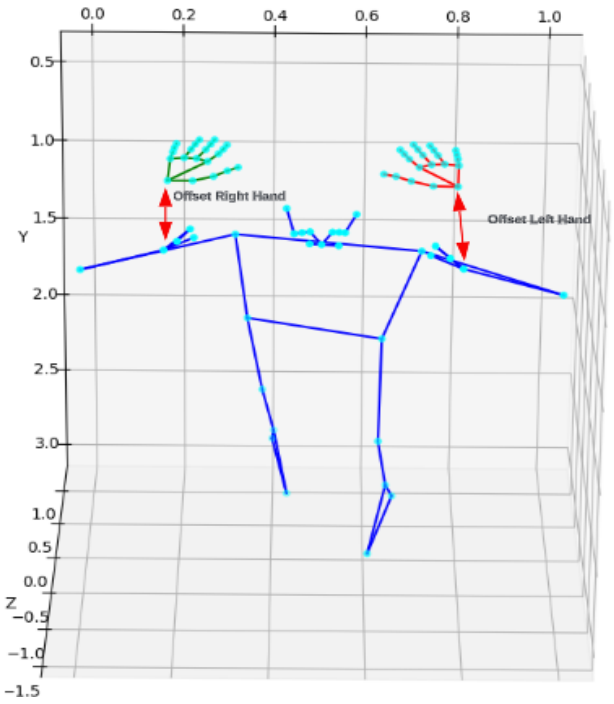


Figure 19 - Hands - Body Offset plot

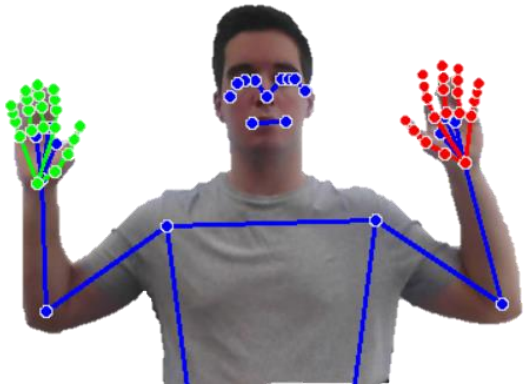


Figure 20 – 2D Mediapipe Body and Hands Pose plot

The lists are then concatenated to the following order [Body, Left Hand, Right Hand]. When concatenated into an array, it is necessary to account for the other solution landmarks to access the right-hand wrist, for example.

$$Right\ Hand\ Wrist\ Landmark\ x = [33 * 3\ + 21 * 3]$$

$$Right\ Hand\ Wrist\ Landmark\ y = [33 * 3\ + 21 * 3\ +\ 1]$$

$$Right\ Hand\ Wrist\ Landmark\ z = [33 * 3\ + 21 * 3 + 2]$$

This data is applied to the LSTM classifier approached in the next section.

4.2.1 Gesture recognition

This section details the training, testing, and implementation of the machine-learning model for gesture classification.

Taking pose and hand estimation landmarks from Mediapipe, the next step is classifying the gestures from these features. A couple of tools were investigated for this purpose, such as Mediapipe Model Maker, which allows the

customisation of provided models with custom data. Due to dependency incompatibilities, the Model Maker was not possible to implement. Even in a Linux environment, the problem persisted. By changing the environment, the problem could be mitigated since Linux has the latest versions available for these solutions. However, it was not certain that Model Maker was able to classify using temporal data, and therefore, we opted to build a classification model using an LSTM.

A dynamic gesture involves breaking down a person's actions into specific spatiotemporal locations, which, in a video, are captured as frames. When adapting these parameters to the inputs of a neural network's LSTM layer, it needs to accept multiple frame data organised by a metric. For this project, the criteria that best fits is the sliding window approach since an evaluation analyses every frame captured, making it difficult for the gesture motion not to be recognised by the classifier. A dynamic window makes determining when one gesture ends and the next begins challenging. An interval of 30 frames was defined to accommodate the cameras that capture a minimum of 30 fps. This means that the gesture motion has a duration of 1 second. If the camera used has a higher capture rate, the gesture duration will be considered lower than it is.

Based on the solution by [45], the model is composed of three LSTM layers, three dense layers, and a Softmax output layer, as seen in Figure 21 and the respective code implementation in the code segment 4.1.

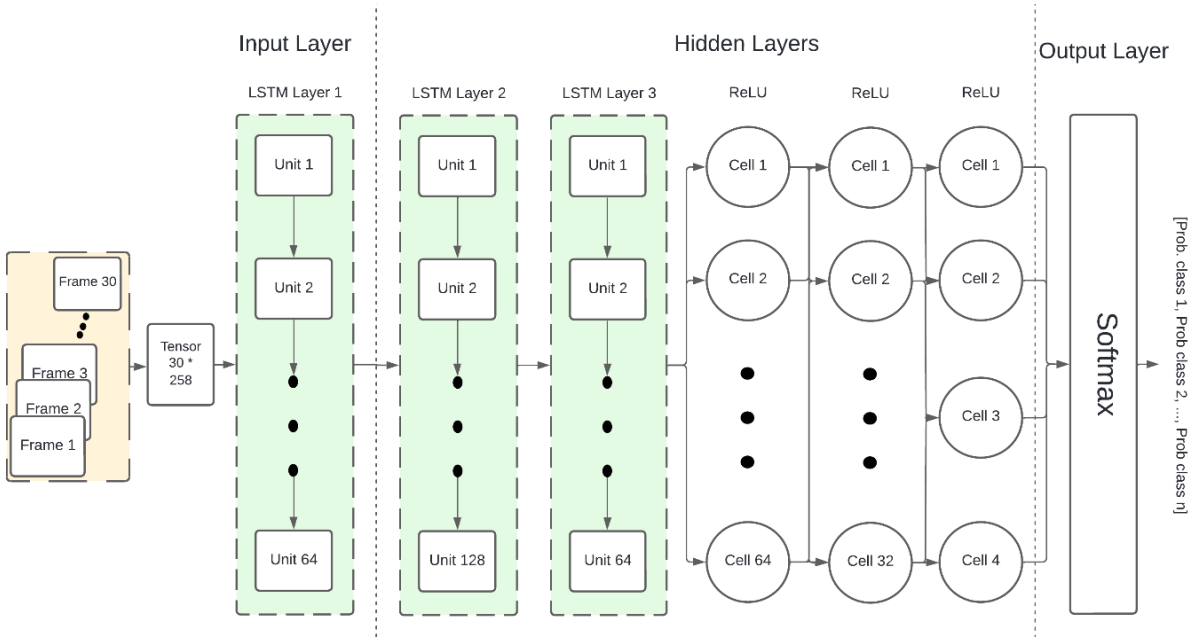


Figure 21 - LSTM Classifier Architecture

```
class LSTMModel(nn.Module):
    def __init__(self, output_size):
        super(LSTMModel, self).__init__()
        self.lstm1 = nn.LSTM(258, 64, batch_first=True)
        self.lstm2 = nn.LSTM(128, 64, batch_first=True)
        self.lstm3 = nn.LSTM(64, 64, batch_first=True)
        self.fc1 = nn.Linear(64, 64)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(64, 32)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(32, output_size)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        out, _ = self.lstm1(x)
        out, _ = self.lstm2(out)
        out, _ = self.lstm3(out)
        out = out[:, -1, :]# take the last time step's output
        out = self.fc1(out)
        out = self.relu1(out)
        out = self.fc2(out)
        out = self.relu2(out)
        out = self.fc3(out)
        out = self.softmax(out)
        return out
```

Code 4.1 – Pytorch Model Definition

By adding the 3 LSTM layers, the time axis is present in the gesture classification. The input LSTM layer takes a tensor of 30 by 258 landmarks as input. The sliding window comprises information on the most recent 30 frames. When a part of the body is not visible, this value comes as 0, although the algorithm infers the joint anyway.

To improve performance, the model needs to have GPU acceleration.

4.2.2 LSTM Training

The first iteration of the model was trained using the gestures “Cool”, “Left Swipe”, “Not Cool”, and “Right Swipe”, as shown in Figures Figure 22 - Figure 25. In each picture of the gesture dataset, it is possible to observe the initial position (a), the intermediate pose (b) and the end of the movement (c).

(a) (b) (c)

Figure 22 - Right Swipe

(a) (b) (c)

Figure 23 - Cool

(a) (b) (c)

Figure 24 - Left Swipe

(a) (b) (c)

Figure 25 - Not Cool

A labelled dataset was created to train the LSTM classifier. Folders organise the dataset, each action containing thirty sequences, each representing a fixed time window of movement capture. For this purpose, the dataset consists of a directory associated with a gesture, containing 30 folders, each with a format ".npy" file. The Numpy file format is used to store a Numpy array on disk. The flowchart in Figure 26 represents the process that creates the dataset structure. First, it checks if the directory has been created in the current workspace. If not created, it creates the folder shown in Figure 27 (a). Then, depending on the pre-determined number of actions to classify, a folder for each is created, as exemplified in Figure 27 (b). To conclude the flow, it creates a folder for each sequence to be captured. In this case, each action folder contains 30 folders, each representing a captured frame during dataset capture, visible in Figure 27 (c).

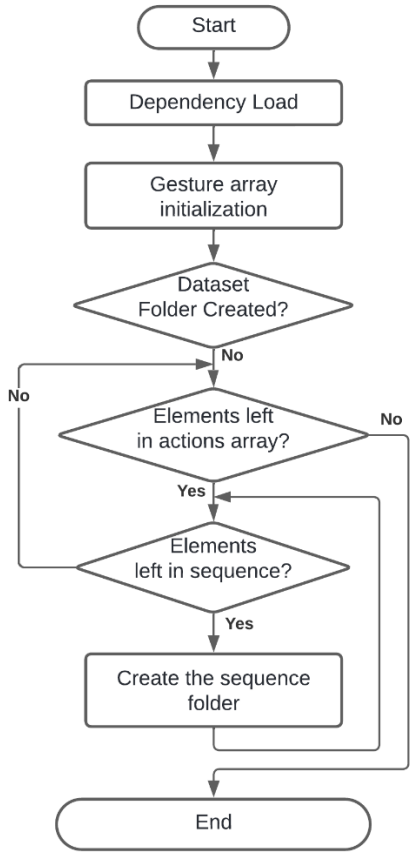
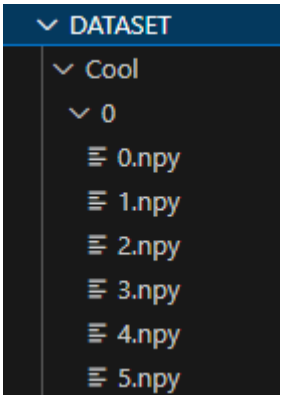


Figure 26 - Dataset Folder Creation Flow

a)

b)



c)

Figure 27 - Dataset Action Sequence Folders

To create the dataset, a male user was asked to perform the previously defined gestures in front of a webcam. For each gesture, 30 frames were captured. The flowchart visible in Figure 28 shows the process of data acquisition. For each action defined and for each sequence, a frame is run through Mediapipe, and the landmarks are extracted into three lists, one for each hand and one corresponding to the body. The data is previously concatenated into a Numpy array and stored in a ".npy" file. This step has the benefit of being lighter than storing video and guarantees that a model is trained with the exact same values when using the model at a different time. The landmark's coordinates could vary if the videos were run through Mediapipe. In code segment 4.2, it is possible to observe the code that handles the image landmark extraction and the concatenation of the lists.

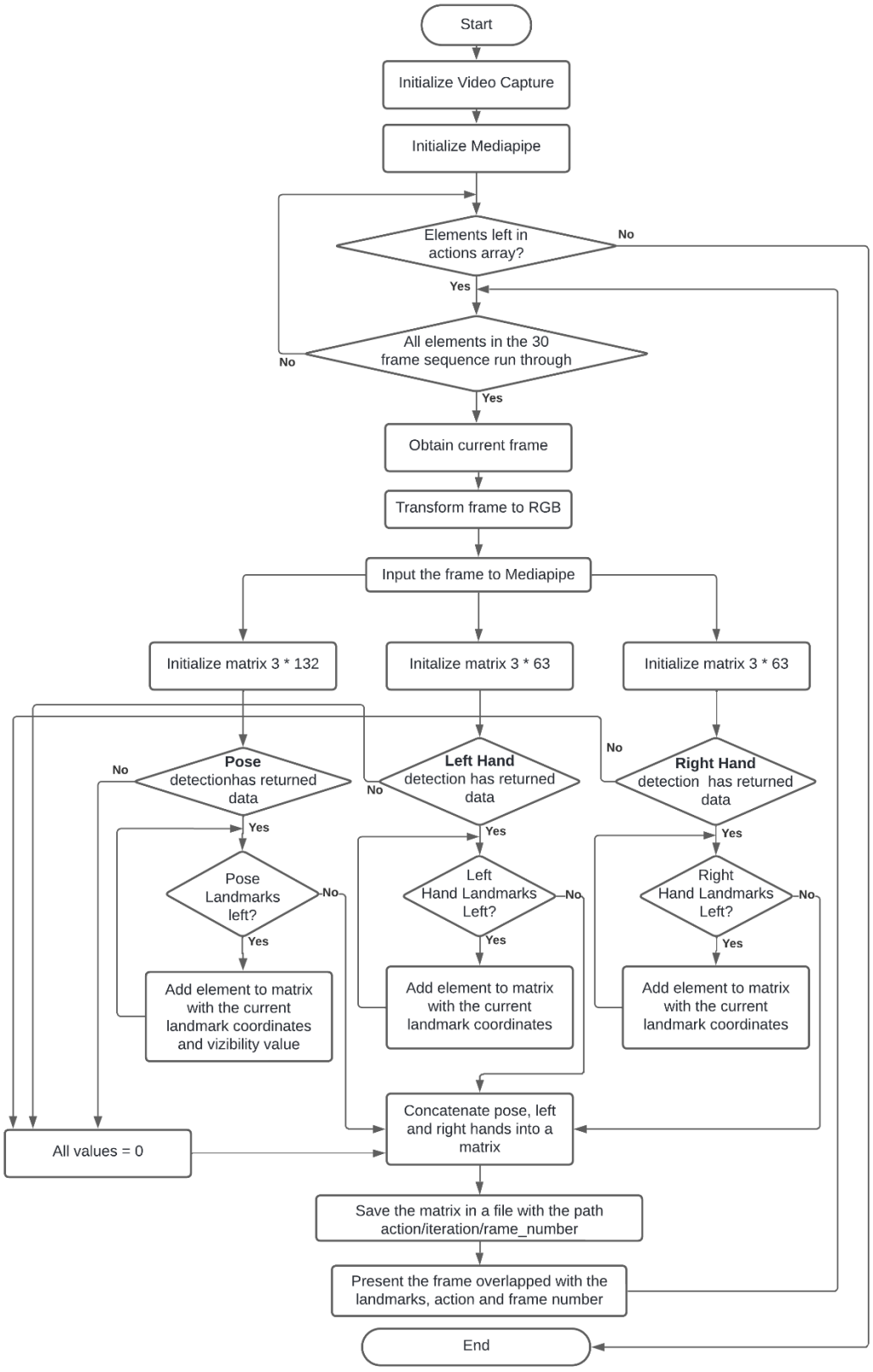


Figure 28 – Dataset Gesture Capture algorithm

```
mp_holistic = mp.solutions.holistic

with mp_holistic.Holistic(min_detection_confidence=0.5,
     min_tracking_confidence=0.5) as holistic:

     image = cv2.cvtColor(image,cv2.COLOR_RGB2BGR)
     results = model.process(image)

     pose = np.array([[res.x, res.y, res.z, res.visibility]
     for res in
     results.pose_landmarks.landmark]).flatten()
          if results.pose_landmarks
          else np.zeros(132)

     lh = np.array([[res.x, res.y, res.z] for res in
     results.left_hand_landmarks.landmark]).flatten()
          if results.left_hand_landmarks
          else np.zeros(21*3)

     rh = np.array([[res.x, res.y, res.z] for res in
     results.right_hand_landmarks.landmark]).flatten()
     if results.right_hand_landmarks
          else  np.zeros(21*3)

     keypoints = np.concatenate([pose,lh,rh])
```

Code 4.2 – Data acquisition code segment.

With a view to creating a more intuitive experience in the immersive environment, a second dataset with different gestures was created. The new set comprises “Rock”, “Paper”, “Scissors”, “Swipe Right”, and “Swipe Left” the two last gestures are the same as defined in the first iteration of the model. To avoid overfitting, more samples were taken, this time from two individuals, one male and a female, with 60 video sequences for each gesture. Because the newly added gestures can be done with both hands without a specific orientation, 30 samples were taken doing the gesture with the right hand, and another 30 were taken using the left hand. The result is 300 video samples, with 30 frames each, all captured from one individual’s movements in a sitting position from a fixed distance from the camera. Due to the addition of the classes, the model’s last

layer was changed from 4 to 5 to correspond to the number of actions in the dataset.

4.2.3 Python Client Software Architecture

This section describes the software architecture of the Python Client module.

The component is implemented in Python with the use of the following libraries:

- OpenCV.
- Threading.
- Socket.
- Pytorch.
- OpenCV.

As mentioned earlier, the system benefits from a fast execution time. For this purpose, multi-threading techniques were used, as well as the use of UDP as the communication protocol. In Figure 29, the flowchart corresponding to the Python client can be observed. Three threads run concurrently: the main, communication, and video stream threads. Running the `VideoCapture` in a separate thread guarantees no bottleneck from the video stream processing, ensuring the main thread reads the last frame. To speed up the process, after the main thread finishes processing a frame, it hands over the information to the communication thread to send the data in parallel.

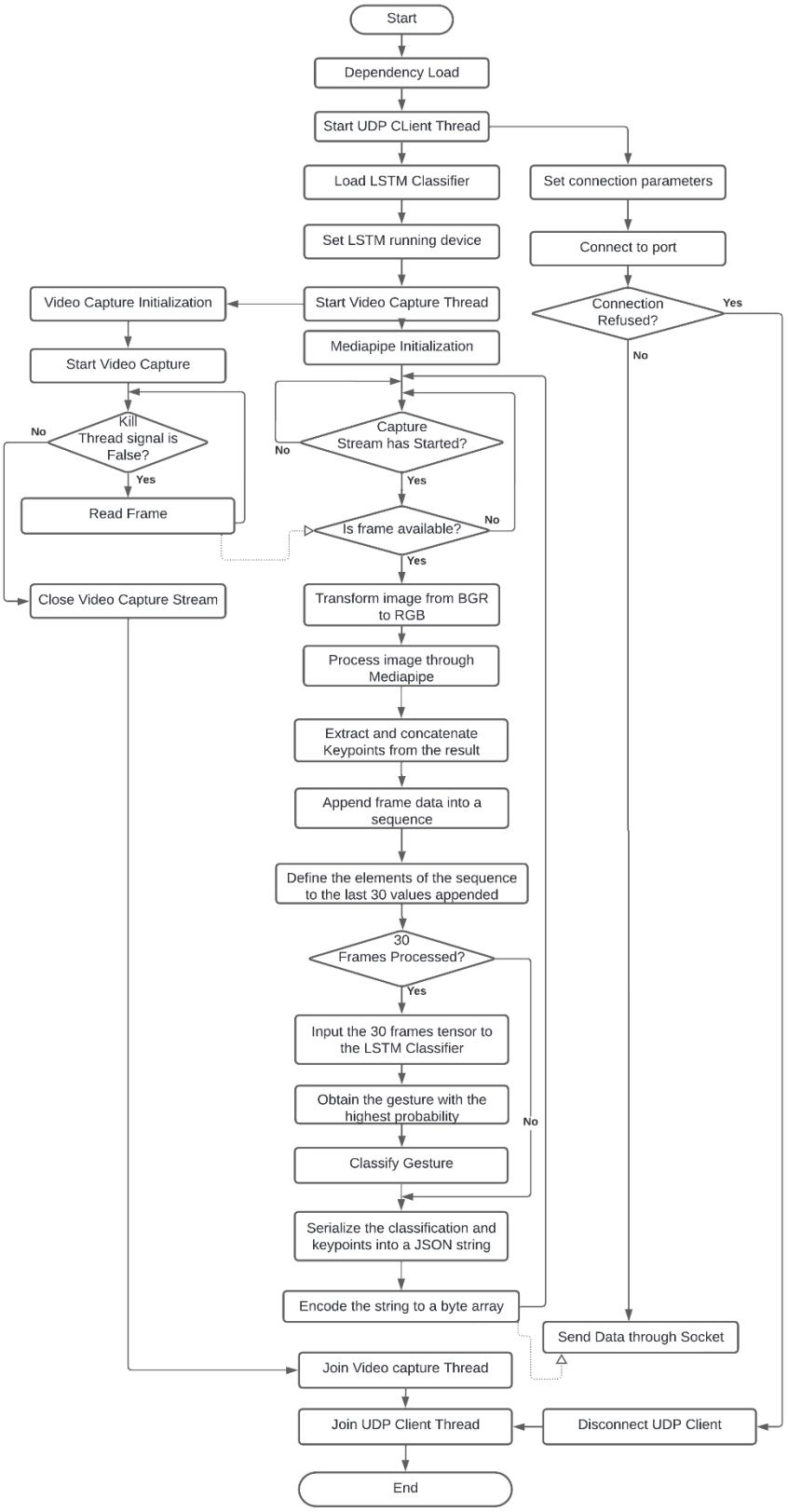


Figure 29 - Python Client Flowchart

Another way to accelerate the process was to use the Graphical Processing Unit (GPU) to handle the classification model. This topic was crucial when choosing a machine learning API from Keras, TensorFlow, and Pytorch. Tensorflow needs to run on WSL so as not to have dependency version mismatch. When doing practical experiments with Mediapipe, Pytorch is the only one that does not have dependency incompatibilities. For these reasons, Pytorch was chosen. The code segment defining the device the model will run is in code segment 4.3.

```
device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')

model = LSTMModel(input_size, len(actions)).to(device)
```

Code 4.3– GPU Device and Model Initialization

4.3 Immersive Environment

This section provides an overview of the primary stages of developing the Virtual Environment and the server-side communication with the Python client. It represents a critical system constituent responsible for user feedback on their actions.

To organise the development of the virtual environment and to check the ground for improvement 3, Unity projects were created:

- Skeleton.
- Avatar.
- VR Integration.

The Skeleton project establishes the interpretation of the data coming from the client and plots it in the virtual environment. Avatar incorporates a 3D avatar into the experience to create a relation between the user's joint movement and the avatar's limb movement. Finally, VR integration joins the previous developments and creates an immersive experience to take advantage of gesture recognition while changing the user's POV to that of the avatar.

All of the projects use the same communication flow. The data incoming from the Python Client is received by a C# socket server running on a thread different than the Unity main thread `Update()`. This has the benefit of not having to wait for the rest of the code from each `Update()` method, hence avoiding information loss. The information flow is seen in Figure 30.

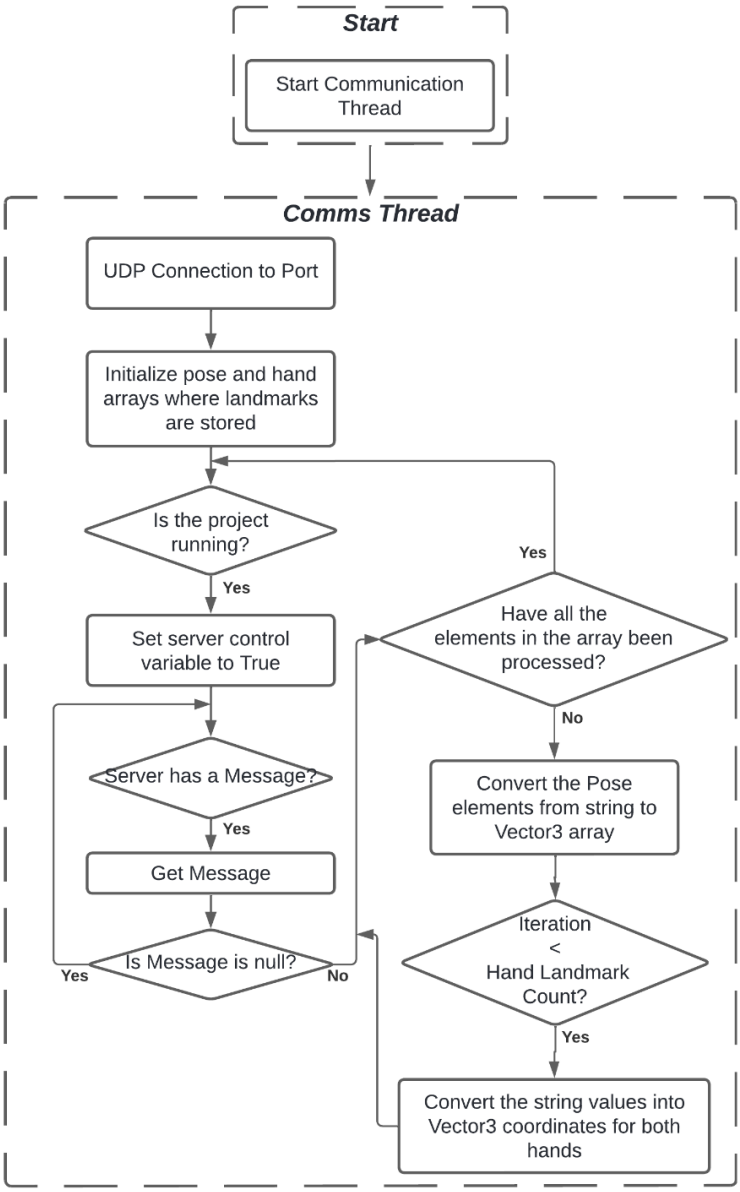


Figure 30 - Unity UDP Server Implementation

4.3.1 Skeleton

The first solution developed aimed to show Mediapipe's output in a 3D environment.

To achieve this, multiple spheres, each corresponding to a joint, were instantiated, and their position in the world was updated in real-time with data from the Python Client. Each joint's position change is made based on the previous position of the spheres and the current coordinates, as seen in 4.4.

```
keypoints = server.keypoints;

for (int i = 0; i < LandmarkCount; i++)
{
      body.instances[i].transform.localPosition =

      Vector3.MoveTowards(body.instances[i].transform.localPosition
            , this.keypoints[i], Time.deltaTime * maxSpeed);
}
```

Code 4.4– GPU Device and Model Initialization

In order to make the skeleton more perceptible, lines were added to draw the connections between joints. The drawing operation is done with the instructions from 4.5.

```
//Pose Lines
//Left Foot

//Define the number of connections this line has
lines[0].positionCount = 3;

//Initiate the line course
lines[0].SetPosition(
      0, instances[(int)Landmark.LEFT_FOOT_INDEX].transform.position);

lines[0].SetPosition(
      1, instances[(int)Landmark.LEFT_ANKLE].transform.position);

lines[0].SetPosition(
      2, instances[(int)Landmark.LEFT_HEEL].transform.position);

//Indicate end and the start of the line connect
lines[0].loop = true;
```

Code 4.5– GPU Device and Model Initialization

The result of the Skeleton solution is visible in Figure 31.

a) b)

Figure 31 – Real World – Unity Landmark Plot

4.3.2 Avatar

This part of the project aims to map a 3D avatar to the user's movements in the real world.

In the initial stage of development, two solutions that filled the requirements for this component were found. The project [46] and [47] moved an avatar using the inputs of Mediapipe. The main difference between them was the use of previously obtained calibration data to move the avatar, which the first did not use. Neither of the projects used hand tracking. Due to the project of [46] being more straightforward, it was used as a base for this stage. The avatar chosen was the standard for projects found [48].

The avatar's motion is calculated using the difference between the last captured movement by the user and the positions of the avatar's body limbs. The calculation starts with determining the orientation of the new movement, using the shoulder's movement as an example. The key points corresponding to the shoulder and elbow are subtracted, and the same occurs for the elbow and wrist joints. The $\overrightarrow{Shoulder\ Elbow}$ vector determines the direction the upper arm is pointing, while the $\overrightarrow{Elbow\ Wrist}$ vector represents the forearm's direction, defining how the shoulder should rotate.

$$\overrightarrow{Shoulder\ Elbow} = [Shoulder\ Landmark] - [Elbow\ Landmark]$$
$$\overrightarrow{Elbow\ Wrist} = [Elbow\ Landmark] - [Wrist\ Landmark]$$

With these vectors, the rotation needed to align the shoulder joint with the shoulder-elbow vector can be defined. The first rotation, as shown in Figure 32 is applied to the shoulder by setting the target of the rotation of the $\overrightarrow{Shoulder\ Elbow}$ vector to the $\overrightarrow{Shoulder\ Elbow}'$ vector, setting the new position of the elbow (Elbow') and wrist (Wrist').

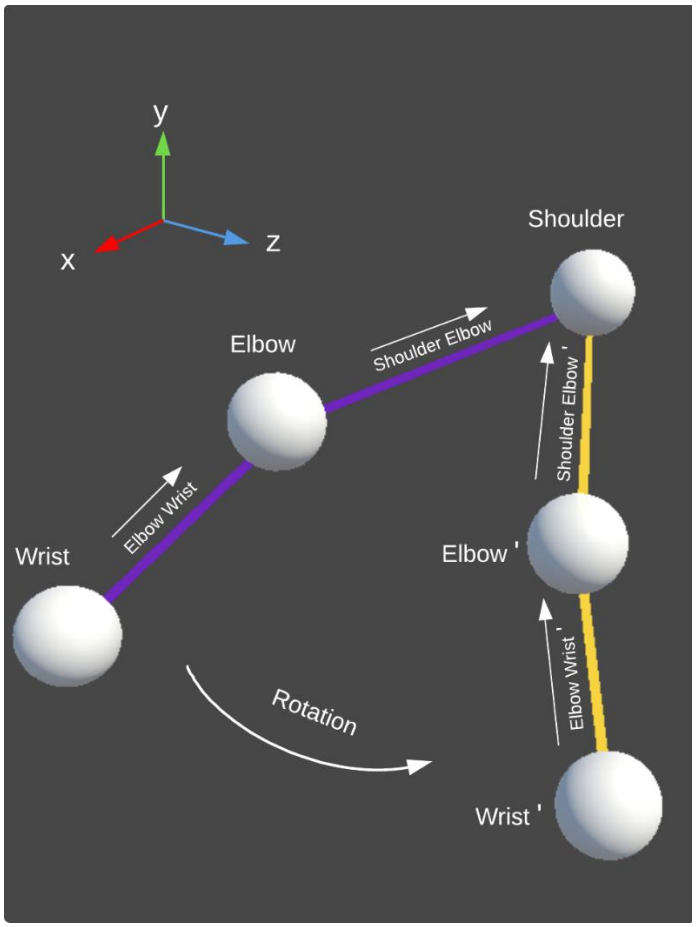


Figure 32 - Shoulder Rotation

This step covers a simple rotation on the shoulder axis, however when making a more complex rotation like a roll. As exemplified in Figure 33 To calculate the direction and how much we need to rotate, the cross-product of the vectors $\overrightarrow{Shoulder\ Elbow}$ and $\overrightarrow{Elbow\ Wrist}$ is calculated. As demonstrated, the plane in yellow is defined by $\overrightarrow{Cross\ Product(\overrightarrow{Shoulder\ Elbow}, \overrightarrow{Elbow\ Wrist})} * \overrightarrow{Shoulder\ Elbow}$, is the initial position and the red plane $(\overrightarrow{Cross\ Product(\overrightarrow{Shoulder\ Elbow'}, \overrightarrow{Elbow\ Wrist'})} * \overrightarrow{Shoulder\ Elbow'})$ represents the target position of the rotation.

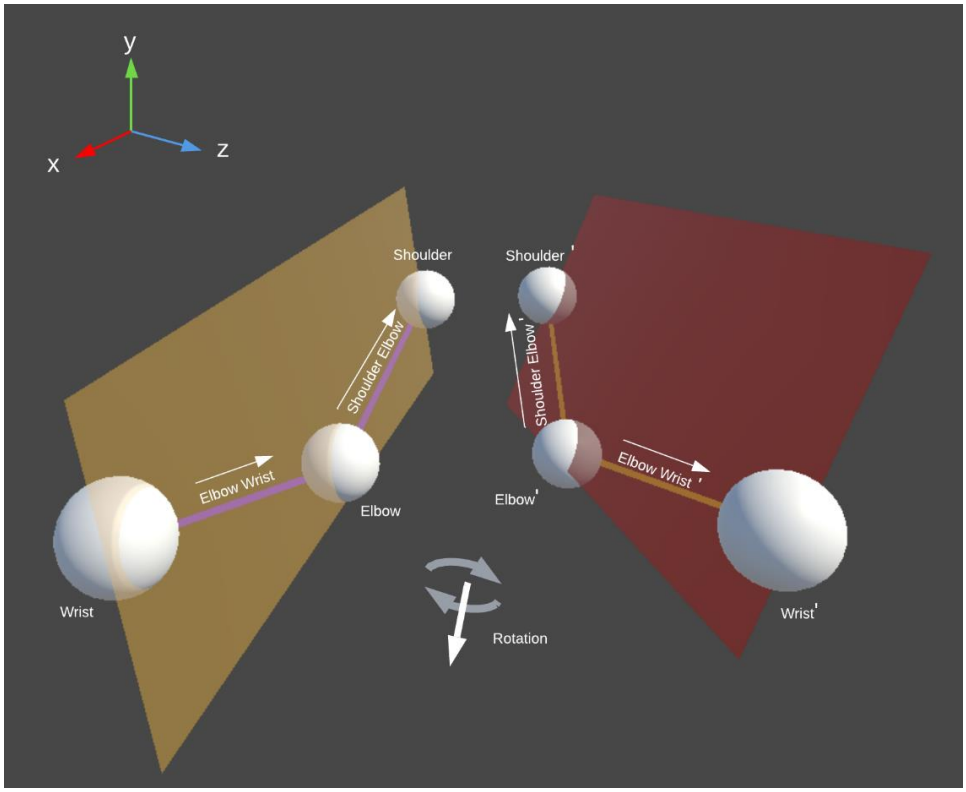


Figure 33 - Arm Roll Rotation

The original project implemented rotations for different limbs in different ways. For example, the wrist, a limb capable of pitch, roll and yaw movements (exemplified in Figure 14), has only an axis of freedom to do a roll. Besides defining a joint's degrees of movement freedom by hand, the solution does not accurately translate the user's movements to the avatar in real-time. Observable in code section 4.6 is the code mapping of a Mediapipe joint, in this case, the shoulder and HumanBodyBones. In 4.7, the code that rotates the shoulder can be seen. It shows the loop that iterates through each articulation, the condition that checks the type of body part to apply the specific transformations and the code approach of the concepts explained above.

```
//Left                                                                  Shoulder
jointList.Add(
    new                                                                   Joint(
        anim.GetBoneTransform(HumanBodyBones.RightUpperArm),
        Side.Right,
        Articulation.Shoulder,
        Landmark.LEFT_ELBOW,        //parent1
        Landmark.LEFT_SHOULDER,     //child1
        Landmark.LEFT_WRIST,        //parent2
        Landmark.LEFT_THUMB));      //child2
```

Code 4.6 – Correspondence between HumanBodyBone and Mediapipe Landmark.

```
foreach (var joint in jointList)
{
      if (joint.bodyPart is Articulation.Shoulder)
    {
        var v_shoulder_elbow = -keypoints[(int)joint.parent1]
      + keypoints[(int)joint.child1];

        var v_elbow_wrist = -keypoints[(int)joint.parent2] +
      keypoints[(int)joint.parent1];

        joint.articulation.Rotate(
            Quaternion.FromToRotation(-
            joint.articulation.right,
                  v_shoulder_elbow).eulerAngles,
                  Space.World
        );

        var norm_x = Vector3.Cross(v_shoulder_elbow,
                        v_elbow_wrist);

        joint.articulation.Rotate(
            Quaternion.FromToRotation(-
            joint.articulation.forward,
      norm_x).eulerAngles,
            Space.World
        );
    }
```

Code 4.7 – Rotation application to shoulder.

Due to the unsatisfactory body tracking and movement translation to the avatar, the second project referenced [47] above was used as a base.

The use of calibration to compute a rotation provides a threshold for movement. Storing the values of the user's joints when making a default pose, for example, a T-pose that matches the avatar's, makes it possible to create a reference for the movement.

The built-in calibration system stores the following information:

- Reference to the avatar bone.
- Initial direction and rotation of:
    - Parent joint (e.g. Shoulder)
    - Child joint (e.g. Elbow)

To have a calibration file, it is necessary to obtain it in a calibration mode. This operation only needs to be completed once the values are stored in memory. In order to obtain the calibration data, the experience has to be running, and it is only possible to do it in the Unity editor. To start the calibration process, the user has to press the "c" key, and the process is shown in Figure 34.

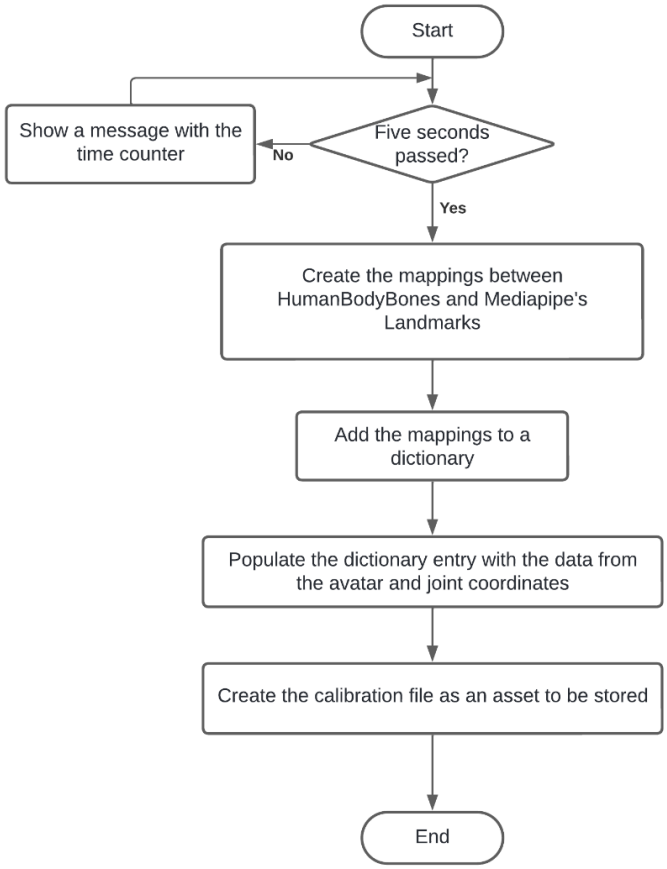


Figure 34 - Calibration Flowchart

In the code snippet 4.8, a calibration entry fetches the positions of the avatar's limbs using the method `animator.GetBoneTransform(parent)` and the landmark coordinates are passed as a parameter. The result is a Unity Script-associated class.

```
private void AddCalibration(HumanBodyBones parent,
      HumanBodyBones child, Transform tParent, Transform
      tchild, Landmark landmark)
{
    parentCalibrationData.Add(Tuple.Create(parent, landmark),
        new CalibrationData(animator.transform,
      animator.GetBoneTransform(parent),
      animator.GetBoneTransform(child),
        tParent, tchild));
}
```

Code 4.8 – AddCalibration method

The result is the file with the fields visible in Figure 35, where information on 42 articulations, rotation, and an inferred joint. In this case, Mediapipe does not provide the coordinates for the middle back. Thus, the avatar is not able to lean forward. To tackle this issue, the points between the shoulders and hips are calculated and used to reference the Avatar's neck and spine, respectively. The other articulations with exceptional rotation cases, like the chest, spine and head, are separated from the other.

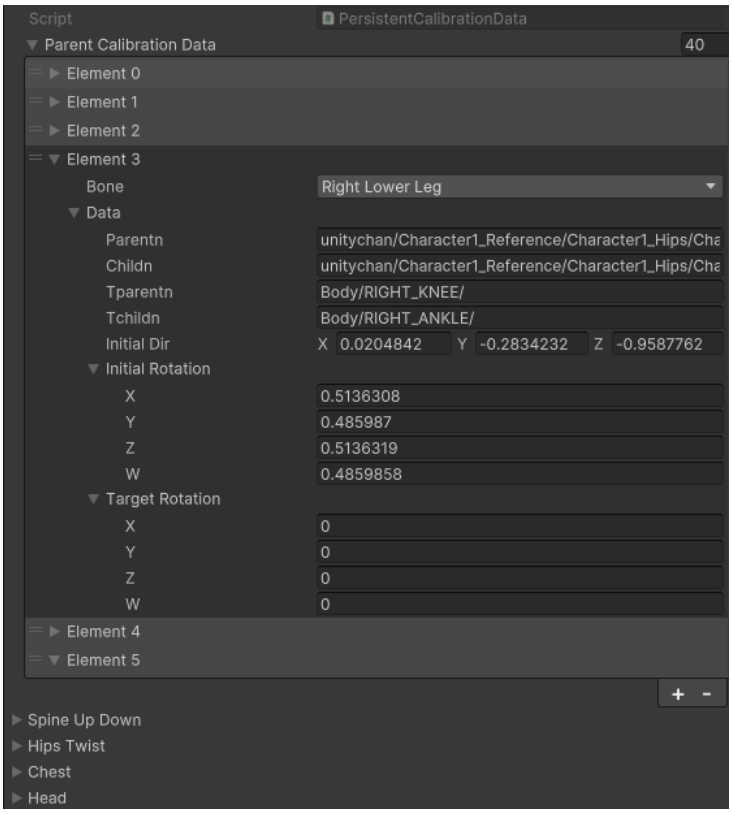


Figure 35 - Calibration Data File

Similar to the previous solution, rotation is defined by the parent subtracting the child's position. One significant change is the inexistence of Euler angles, which, besides eliminating the possibility of a gimbal lock, defines only two operations: the rotation of the limb to the desired position and the roll

movement. Without accounting for the spine and head movement, the code that updates the position of a limb is defined in 4.9. Every iteration calculates the rotation value of the limb, from the calibration direction to the current, multiplying this value by the initial rotation of the limb, thus defining the direction and rotation of the joint.

```
foreach(var i in parentCalibrationData)
{
      Quaternion deltaRotTracked = Quaternion.FromToRotation(
                                 i.Value.initialDir,
                                 i.Value.CurrentDirection);

      i.Value.parent.rotation = deltaRotTracked *
                                 i.Value.initialRotation;
}
```

Code 4.9 – Update Joint Rotation

The segmented articulations are calculated using the operations in code section 4.10. The difference between the initial and current directions defines head rotation. Both hip and spine rotations have a filter applied using `Vector3.Slerp()` to calculate 25% of the interpolation between the initial and current directions to smoothen the movement.

```
//Head Direction
Vector3 hd = head.CurrentDirection;

//Head Rotation
Quaternion headr = Quaternion.FromToRotation(head.initialDir, hd);

//Hip Rotation
Quaternion twist = Quaternion.FromToRotation(
                        hipsTwist.initialDir,
                        Vector3.Slerp(hipsTwist.initialDir,
                                    hipsTwist.CurrentDirection, .25f));
//Spine Rotation
Quaternion updown = Quaternion.FromToRotation(
                        spineUpDown.initialDir,
                        Vector3.Slerp(spineUpDown.initialDir,
                                    spineUpDown.CurrentDirection, .25f));
```

Code 4.10 – Update Joint Rotation

The computation of additional adjustments, shown in code section 4.11 to the rotation values defines the final rotations. The h, s and c values are used to calculate complex compound movements. After the compound values are added to the initial rotations, the `Tick()` section 4.12 method is applied to calculate the spherical interpolation based on the time passed to polish the movement between the initial and target rotation.

```
Quaternion h = updown * updown * updown * twist * twist;
Quaternion s = h * twist * updown;
Quaternion c = s * twist * twist;

float speed = 10f;

hipsTwist.Tick(h * hipsTwist.initialRotation, speed);

spineUpDown.Tick(s * spineUpDown.initialRotation, speed);

chest.Tick(c * chest.initialRotation, speed);

head.Tick(updown * twist * headr * head.initialRotation, speed);
```

Code 4.11 – Update Joint Rotation

```
public void Tick(Quaternion newTarget, float speed)
{
    parent.rotation = newTarget;
    parent.rotation = Quaternion.Lerp(parent.rotation, targetRotation,
                                          Time.deltaTime * speed);
}
```

Code 4.12 – Update Joint Rotation

The entire body is rotated based on the hips to add responsiveness. The hip rotation is interpolated and flattened to lower the value of the entire body rotation. As shown in 4.13.

```
//Interpolation of the hip rotation
Vector3 d = Vector3.Slerp(hipsTwist.initialDir,
                                hipsTwist.CurrentDirection, .25f);
d.y *= 0.5f;

Quaternion deltaRotTracked =Quaternion.FromToRotation(
                                                hipsTwist.initialDir, d);
targetRot = deltaRotTracked * initialRotation;
transform.rotation = Quaternion.Lerp(transform.rotation, targetRot,
Time.deltaTime * speed);
```

Code 4.13 – Update Joint Rotation

The final phase of the project can be seen in Figure 36. It is important to note that the arms and wrists do not bend proportionally to the real-life pose. This limitation was not addressed due to time constraints.

a) b)

Figure 36 - Real World - Avatar motion tracking

4.3.3 VR Integration

This solution integrates the previously described experiences, aiming to create a VR environment where the user’s movements are seamlessly translated into the immersive world, augmented by a gesture prediction layer. To leverage gesture recognition and highlight its advantages, a "rock-paper-scissors" game was implemented. By predicting the user’s intentions, the system aims to outperform the player, adding a clear objective to the experience while demonstrating the effectiveness of the classifier algorithm.

The project was created with the XR plugin with support for Oculus and OpenXR. This allows for the use of multi-manufacturer HMD support. This plugin automatically associates the camera rotation, direction and movement. In this experience, the controllers were discarded.

To enhance immersion, the user is represented by a virtual avatar or skeleton linked to the camera's position. As the user moves, the virtual body mirrors their motions. However, the experience is constrained by the camera's field of view. To address this, the environment's main focus is positioned directly in front of the user, encouraging them to maintain a forward-facing orientation with their arms and hands visible to the camera.

Rock-paper-scissors is a game played with the hands in which two players simultaneously make one of the three shapes: rock, paper or scissors. Each play has three possible outcomes: a win, a draw or a loss. Rock defeats scissors, which defeats Paper, who defeats Rock.

When the [48] avatar is displayed in the HMD, there is a shader problem where the object only appears in one eye. For this reason, the game object's materials were changed.

In each frame, there is an attempt to identify the user's gesture or their intention to perform it, and the system responds accordingly. To begin the game, the player must "Swipe Left". Then, a time counter starts to define the deadline for the user's play, and it registers the gesture classification value in the frame after the timer has passed. The timer is verified for each frame, and if the elapsed time has been incremented further than the stipulated time, it assumes the last gesture classification is the user's intended play. Right after the system captures the play, it uses it to determine the winning move.

In case the movement is not associated with the game (e.g. Swipe Left/Right), the current play is invalid.

A panel featuring a bar chart was designed to visually represent the temporal component of gesture recognition. However, when displayed in real-time, the chart updates rapidly, resulting in jerky bar movements rather than smooth transitions. To solve this issue, the probabilities sent by the Python Client are taken, and a LIFO queue of 30 elements in Unity is created. The elements of this queue are used to show the average probability of each gesture in the 30-frame interval. The most recent classification probabilities are added to the queue in each frame, and the oldest values are dequeued.

These results are used to size the bars representing each gesture's probabilities. Visible in Figure 37 is the end of a round where the user makes the Scissors

movement, and the system plays rock, with the probabilities chart visible on the right side.

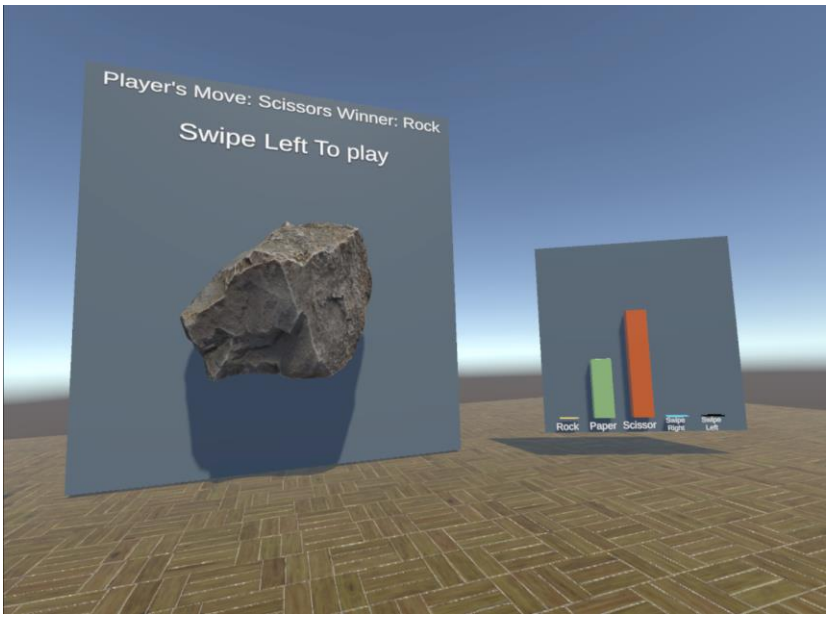


Figure 37 – User's POV in the VR Experience

To add to the experience, when the user makes the "Swipe Right" motion, the avatar's body is hidden, and the skeleton is shown to the user with the camera perspective near the nose landmark.

5 Results

This chapter presents the proposed evaluation methodology for the VR application and the results obtained during development.

5.1 LSTM Classifier Results

The LSTM Classifier was trained with the “Cool”, “Left Swipe”, “Not Cool”, and “Right Swipe” dataset with a training set comprised of 80% (96 samples) of the data, while the validation and test sets comprised 10% (12 samples). Both train and test sets were shuffled. The optimiser was Adam, with a learning rate of 0.001. Each batch had a size of 30 elements. The results obtained from testing the model are visible in the confusion matrix in Figure 38, with a precision of 96% (23 of the 24 samples were correctly classified) and a loss value of 1.1391.

Although the dataset is small, a high precision score with minimal false positives was possible. The inference time was 0.002s.

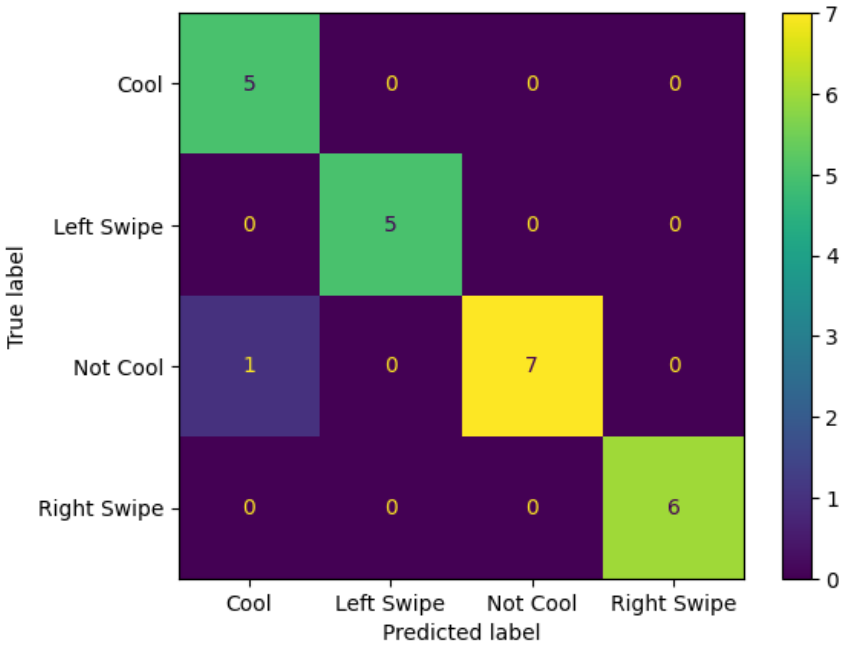


Figure 38 – First Dataset LSTM Classifier Confusion Matrix

On the second iteration of model training, the Rock-Paper-Scissors dataset was split into 80% (240 samples) training, 10% (30 samples) test and another 10% validation set. The training was done with a lower learning rate, achieving lower loss values during the process. This resulted in a loss of 1.0670, a training accuracy of 84.17% and a validation accuracy of 73.33%. The accuracy of the classifier against the test set is 93.33%, with 2 misclassifications in 30, as seen in the confusion matrix in Figure 39.

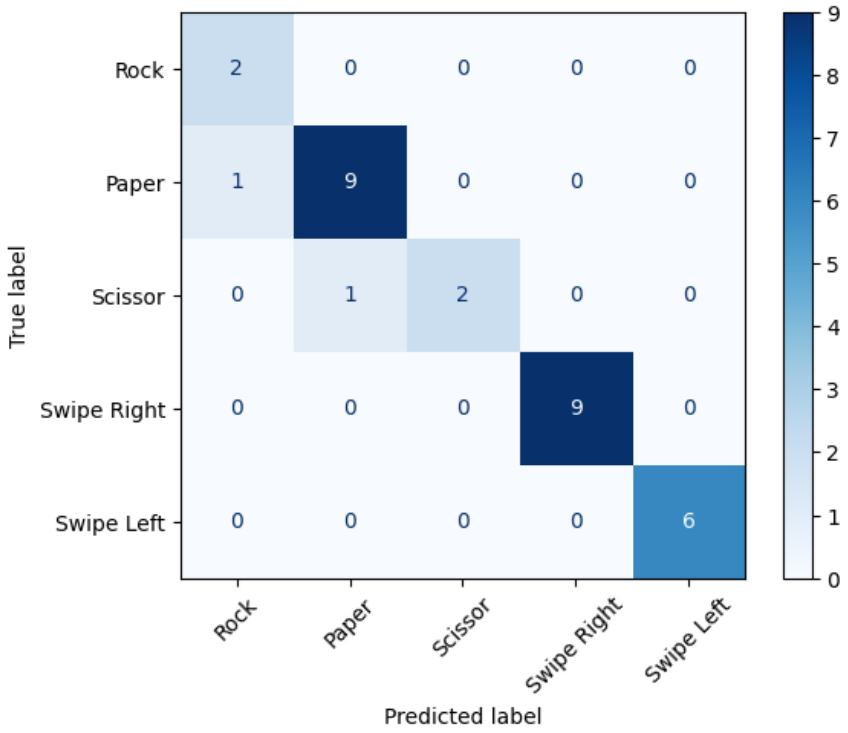


Figure 39 - Second Dataset LSTM Classifier Confusion Matrix

5.2 Interaction Tasks

In this study, the users were asked to play the Rock Paper and Scissors game 5 times against the program and switch bodies. Each play consists of initiating the game with the gesture “Swipe Left” and the play at the end of the counter.

A. Participants

The testing phase was conducted with 8 participants (3 men and 5 women). The participants were between 29 and 63 years old, and participation was voluntary. Four participants had never used virtual reality technology, and 4 had experience with VR equipment. Each participant was asked to take part individually and was informed of the purpose of the experience. All participants

were familiar with the rock-paper-scissors game and were not explained how to perform the gestures. At the end of the experience, they were asked to answer a questionnaire.

B. Materials

The experiments were made with the following equipment:

- Oculus Quest 2 headset.
- Logitech Streamcam (Field of view: 78ª diagonal, Maximum video resolution: 1080p @ 60fps)
- Desktop (CPU – i5 – 12400F, GPU – Nvidia 3070 Ti, 16GB DDR4 RAM)

C. Experience design

The users are asked to perform five plays of the rock, paper, scissors game against the computer. Each play is registered to see if the gesture made by the user was correctly identified. In the end, a form with multiple questions to qualitatively evaluate the sense of presence, involvement and realness, the participants answered the iGroup Presence Questionnaire (IPQ) [49] [50] were employed. Moreover, to appraise system usability, the questions from [51] were used. All questionnaires were written in Portuguese. Specific questions related to the project were added to analyse the user's feeling towards gesture interaction in the experience on a 5-point Likert scale:

- Q1 – "Gesture interaction feels natural".
- Q2 – "Rate the gesture interaction".

5.2.1 Experimental Results

The IPQ results, visible in Figure 40, show the results are all positive, with the highest value being the feeling of presence and the lowest being Involvement, still above the positive threshold (>2.5). The calculations were made using [52].

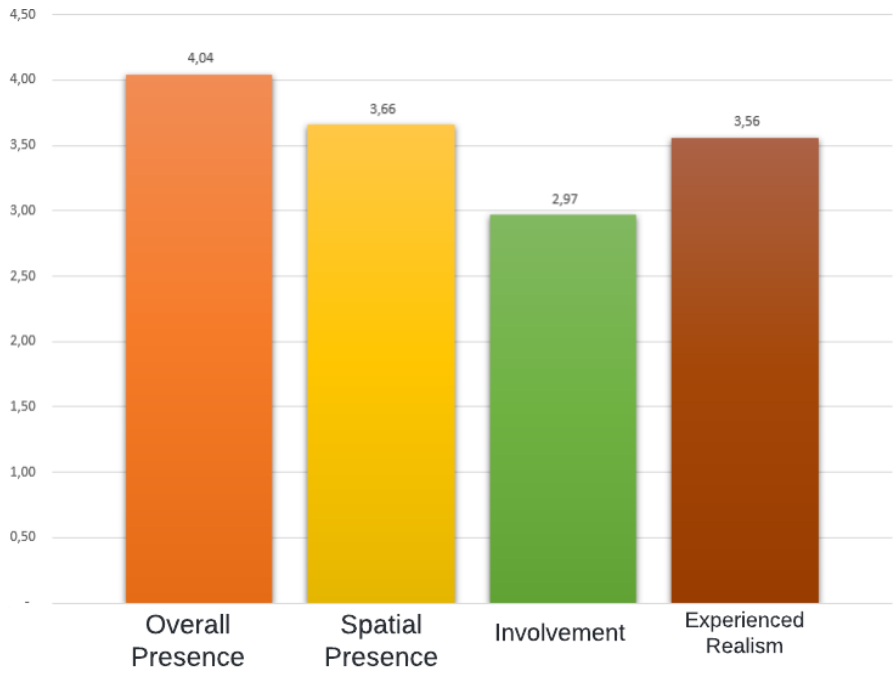


Figure 40 - IPQ results graph

The feedback from the usability test was positive, with a score of 73.44, since everything above 68 is considered usable.

In the gesture-specific questions, the results were also positive, and the results can be seen in Figure 41.

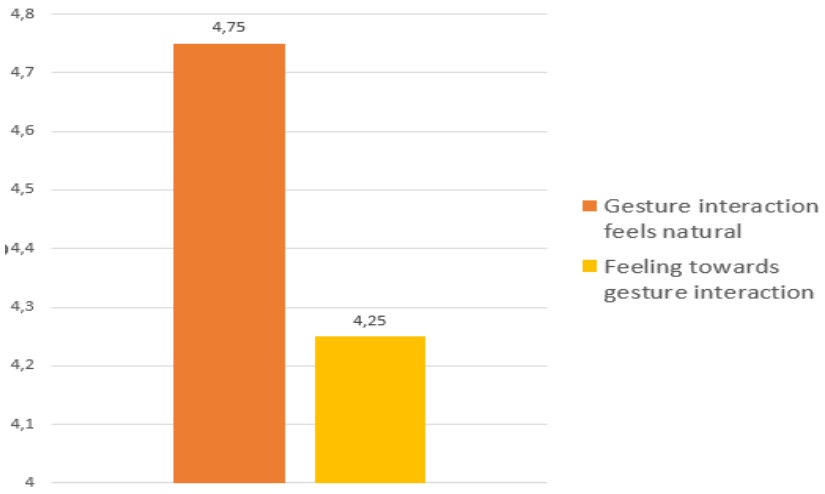


Figure 41 – Gesture-Specific questions Average results

The user's RPS game rounds were registered during the experiment. With 40 samples collected, the classification accuracy was 45% (18 gestures well classified over 40). The confusion matrix, visible in Figure 42, shows the preferred play was "Rock", with 45% (18 plays) of plays. The other two gestures were played equally with a frequency of 27.5% (11 plays) each.

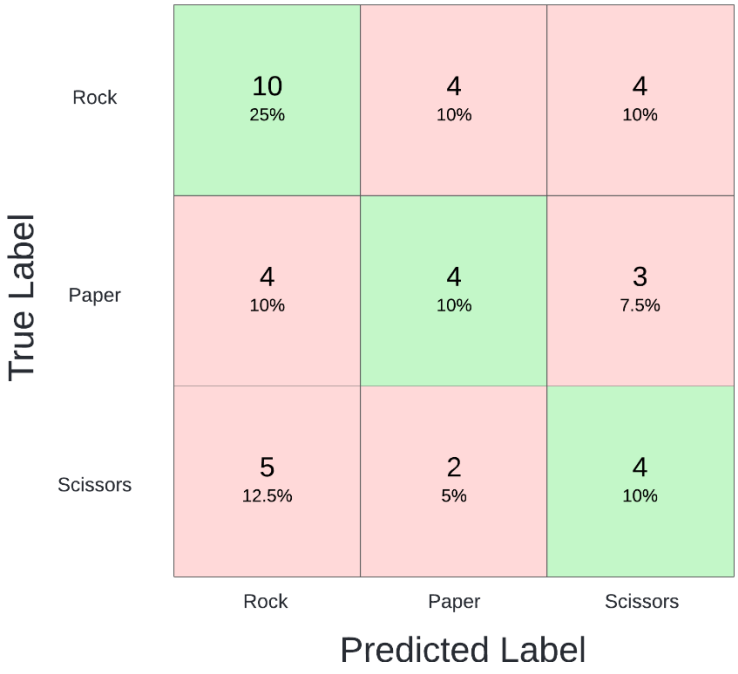


Figure 42 - RPS rounds Confusion Matrix.

5.3 Discussion

The users reported overall positive results in the previous sub-section. Still, the lowest value regarding IPQ questions is related to involvement, which refers to the attention devoted to the virtual environment and the involvement experience. This can be explained due to the simplicity of the experience and the respective interface. As said in 4.3.3, the virtual environment is focused in front of the user. The space is empty when the individual looks around, besides seeing its own body in the environment, it is not enough to provide an interesting and captivating surrounding.

Compared to the offline validating and testing results, the LSTM applied online during the experiment decreased its performance. This may indicate poor model generalisation. The individuals who participated in the experiment were likelier to perform each gesture differently, and the dataset was obtained from one single individual. As stated in Figure *40*, the "Rock" movement has the highest number of False Positives, which can be explained by the failure of Mediapipe to capture the hand's landmarks, leading to the phenomenon captured in Figure 43, where the landmarks are wrongly overlapped in Figure 43 b), and the classifier assumes the gesture is "Rock. can be influenced by lighting conditions [7] or hardware latency.

a) b)

Figure 43 - Hand Landmarks used in gestures identification for a) - "Paper" and b) "Scissors"

The different distances and angles could also explain another explanation for the model's underperformance of the participants when trying the experience. During training, the gestures were made from a fixed position, leading to incorrect classifications on untrained distances or angles.

The participants stated that the 3D avatar did not make consistent gestures with their motions in the real world. Since the avatar's movements are directly related to Mediapipe's detection, as stated in the last paragraph, loss of body detection or misdetection harms the user's experience.

6 Conclusion

The rapid advancements in technology have transformed the capabilities of immersive environments and HCI, opening a path for more immersive and intuitive ways of interaction. While traditional interfaces (e.g. keyboard, mouse) are effective for many tasks, their limitations show in 3D virtual environments, where systems benefit from natural and seamless interactions. This project aimed to develop a system that lacks controllers or physical interfaces in a VR experience. Furthermore, full body tracking is done with a regular webcam, allowing the user to visualise their own body in the virtual environment represented by a 3D avatar.

The project has addressed the issues in HCI stated above by creating a DNN model that receives the pose features extracted from the DL model Mediapipe's Holistic as input. In the previous chapter, the experimental results showed that the developed system has a positive subjective scoring in terms of experienced realism by the metrics of [49] [50] and reached a score of 73.44 regarding system usability. Additionally, the experiment participant's feelings towards gesture were very positive. Despite the positive results, the gesture classification model performed well below the threshold in the validation and testing phases, which is related to the limitations of the training dataset, regarding size and participants variability.

Future work includes upgrading the classification system to perform better on real-world implementations. This requires a dataset with more individuals performing the pre-determined gestures and with more samples for each gesture. Additionally, an extra class representing the "None" gesture to detect

non-classifications needs to be added to improve the classifier's results in terms of false positives and negatives. Regarding the virtual experience, the 3D avatar's movements were not a direct translation of the user's, with many inconsistencies caused by pose detection noise. Filters need to be applied to body tracking to prevent random and inconsistent movements.

7 References

[1] T. Di Qi, F. L. Cibrian, M. Raswan, T. Kay, H. M. Camarillo-Abad and Y. Wen, "Toward Intuitive 3D Interactions in Virtual Reality: A Deep Learning- Based Dual-Hand Gesture Recognition Approach," *Engineering Faculty Articles and Research,* vol. 12, pp. 67438-67452, 5 2024.

[2] H. N. Liang, Y. Shi, F. Lu, J. Yang and K. Papangelis, "VRM controller: An input device for navigation activities in virtual reality environments," vol. 1, pp. 455-460, 12 2016.

[3] B. Yalçınkaya, "DESIGNING 3D SCENARIOS AND INTERACTION TASKS FOR IMMERSIVE ENVIRONMENTS," 2020.

[4] K. K. Kim, I. H. Ha, M. Kim, J. Choi, P. Won, S. Jo and S. H. Ko, A deep-learned skin sensor decoding the epicentral human motions, vol. 11, Nature Publishing Group, 2020, pp. 1-8.

[5] S. S. Rautaray and A. Agrawal, "Interaction with virtual game through hand gesture recognition," pp. 244-247, 2011.

[6] V. Barbanti, "O Movimento Humano," 1988.

[7] B. K. Chakraborty, D. Sarma, M. K. Bhuyan and K. F. MacDorman, "Review of constraints on vision-based gesture recognition for human–computer interaction," vol. 12, no. 1, pp. 3-15, 2 2018.

[8] M. Karam, "A framework for research and design of gesture-based human-computer interactions," 2006.

[9] J. H. F. T. H.-A. W. G.-Z. D. Yang L1, "Gesture interaction in virtual reality," 2019.

[10] M. Zhang, Z. Zhou, T. Wang and W. Zhou, "A Lightweight Network Deployed on ARM Devices for Hand Gesture Recognition," vol. 11, pp. 45493-45503, 2023.

[11] G. P. Zhang, "Neural networks for classification: a survey," *IEEE Transactions on Systems,* vol. 30, no. 4, pp. 451-462, 2020.

[12] W. S. Mcculloch and W. Pitts, "A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY," 1943.

[13] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition, O'Reilly Media, Inc., 2019.

[14] N. Cruz, K. Lobos-Tsunekawa and J. Ruiz-Del-Solar, "Using convolutional neural networks in robots with limited computational resources: Detecting NAO robots while playing soccer," vol. 11175 LNAI, pp. 19-30, 2018.

[15] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan and J. Walsh, "Deep Learning vs. Traditional Computer Vision," *Advances in Intelligent Systems and Computing,* vol. 943, pp. 128-144, 2020.

[16] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM,* vol. 60, no. 6, pp. 84-90, 5 2017.

[17] S. H. a. Schmidhube, "Long short-term memory," *Neural Computation 9(8),* pp. 1735-1780, 1997.

[18] U. Singh, S. Chauhan, A. Krishnamachari and L. Vig, "Ensemble of deep long short term memory networks for labelling origin of replication sequences," 2015.

[19] K. Fukushima, Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, vol. 36, Springer-Verlag, 1980, pp. 193-202.

[20] J. N. Fuhg, A. Karmarkar, T. Kadeethum, H. Yoon and N. Bouklas, Deep convolutional Ritz method: parametric PDE surrogates without labeled data, vol. 44, Springer Science and Business Media B.V., 2023, pp. 1151-1174.

[21] Keras, "Keras," 2024. [Online]. Available: https://keras.io/. [Accessed 9 11 2024].

[22] Google, "TensorFlow," 2024. [Online]. Available: https://www.tensorflow.org/?hl=pt. [Accessed 9 11 2024].

[23] L. Foundation, "Pytorch," 2024. [Online]. Available: https://pytorch.org/. [Accessed 9 11 2024].

[24] M. Byambaa, G. Koutaki and L. Choimaa, "6D Pose Estimation of Transparent Object From Single RGB Image for Robotic Manipulation," *IEEE Access,* vol. 10, pp. 114897-114906, 2022.

[25] Google, "Mediapipe," 5 4 2023. [Online]. Available: https://github.com/google-ai-edge/mediapipe/commits/master/docs/solutions/holistic.md. [Accessed 27 10 2024].

[26] H. J. Kim and S. W. Baek, Implementation of wearable glove for sign language expression based on deep learning, vol. 29, Springer Science and Business Media Deutschland GmbH, 2023, pp. 1147-1163.

[27] G. H. T. S. S.-E. W. Y. S. Zhe Cao, "OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields," 2018.

[28] J. R. a. S. D. a. R. G. a. A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016.

[29] C. J. Bohil, B. Alicea and F. A. Biocca, "Virtual reality in neuroscience research and therapy," 12 2011. [Online]. Available: https://www.researchgate.net/publication/51765068_Virtual_reality_in_neuroscience_research_and_therapy. [Accessed 26 10 2024].

[30] K. Kilteni, R. Groten and M. Slater, "The Sense of Embodiment in virtual reality," vol. 21, no. 4, pp. 373-387, 2012.

[31] Unity, 2024. [Online]. Available: https://unity.com/. [Accessed 29 10 2024].

[32] E. Games, "Unreal," 2024. [Online]. Available: https://www.unrealengine.com. [Accessed 11 11 2024].

[33] J. Zeitlhöfler, "Nominal and observation-based attitude realization for precise orbit determination of the Jason satellites," 06 2019.

[34] A. Civita, S. Fiori and G. Romani, A mobile acquisition system and a method for hips sway fluency assessment, vol. 9, MDPI AG, 2018.

[35] Y. Li, J. Huang, F. Tian, H. A. Wang and G. Z. Dai, "Gesture interaction in virtual reality," *Virtual Reality & Intelligent Hardware,* vol. 1, no. 1, pp. 84-112, 2 2019.

[36] J. LaViola, E. Kruijff, R. McMahan, D. Bowman and I. Poupyrev, 3D User Interfaces: Theory and Practice, 2nd ed. Reading, MA, USA: Addison-Wesley, 2017.

[37] Y. Liu, "Analysis of Interaction Methods in VR Virtual Reality," vol. 39, pp. 395-407, 4 2023.

[38] J. Boutin, J. Kamoonpuri, R. Faieghi, J. Chung, S. de Ribaupierre and R. Eagleson, "Smart haptic gloves for virtual reality surgery simulation: a pilot study on external ventricular drain training," vol. 10, p. 1273631, 2024.

[39] H. Guillen-Sanz, D. Checa, I. Miguel-Alonso and A. Bustillo, "A systematic review of wearable biosensor usage in immersive virtual reality experiences," vol. 28, no. 2, pp. 1-28, 3 2024.

[40] L. Minh Dang, K. Min, H. Wang, M. Jalil Piran, C. Hee Lee and H. Moon, Sensor-based and vision-based human activity recognition: A comprehensive survey, vol. 108, Pergamon, 2020, p. 107561.

[41] C. Huesser, S. Schubiger and A. Çöltekin, "Gesture Interaction in Virtual Reality: A Low-Cost Machine Learning System and a Qualitative Assessment of Effectiveness of Selected Gestures vs. Gaze and Controller Interaction," vol. 12934 LNCS, pp. 151-160, 2021.

[42] S. Riofrio, D. Pozo, J. Rosero and J. Vasquez, "Gesture recognition using dynamic time warping and kinect: A practical approach," Vols. 2017-November, pp. 302-308, 7 2017.

[43] M. Hoffman, P. Varcholik and J. J. LaViola, "Breaking the status quo: Improving 3D gesture recognition with spatially convenient input devices," pp. 59-66, 2010.

[44] L. Kratz, M. Smith and F. J. Lee, "Wiizards: 3D gesture recognition for game play input," pp. 209-212, 2007.

[45] Jaykumaran, "https://github.com/Jaykumaran/HandGestureRecog," 23 8 2023. [Online]. Available: https://github.com/Jaykumaran/HandGestureRecog. [Accessed 2024 08 21].

[46] GanniPiece, "MettU," 10 3 2023. [Online]. Available: https://github.com/GanniPiece/MetU. [Accessed 24 10 2024].

[47] Ganesh, "UnityPythonMediaPipeAvatar," 9 9 2024. [Online]. Available: https://github.com/ganeshsar/UnityPythonMediaPipeAvatar. [Accessed 4 10 2024].

[48] U. Japan, "Unity-Chan Model," 12 6 2020. [Online]. Available: https://assetstore.unity.com/packages/3d/characters/unity-chan-model-18705. [Accessed 24 10 2024].

[49] M. Melo, G. Goncalves, J. Vasconcelos-Raposo and M. Bessa, "How Much Presence is Enough? Qualitative Scales for Interpreting the Igroup Presence Questionnaire Score," vol. 11, pp. 24675-24685, 2023.

[50] T. W. Schubert, "The sense of presence in virtual environments:," vol. 15, no. 2, pp. 69-71, 4 2003.

[51] J. Brooke, "SUS: a retrospective," 2 2013.

[52] J. Vasconcelos-Raposo, M. Bessa, M. Melo, L. Barbosa, R. Rodrigues, C. M. Teixieira, L. Cabral and A. Augusto Sousa, *iGroup Presence Questionnaire – Versão Portuguesa,* 2016.

[53] R. Cipolla and A. Pentland, “Computer Vision for Human-Machine Interaction,” 1998.

[54] V. I. Pavlovic, R. Sharma and T. S. Huang, “Visual Interpretation of Hand Gestures,” 1997.

[55] M. Oudah, A. Al-Naji and J. Chahl, “Hand Gesture Recognition Based on Computer Vision: A Review of Techniques,” 2020.

[56] R. Heeg, “Possibilities and limitations, of unstructured data,” ESOMAR, 20 fevereiro 2023. [Online]. Available: https://researchworld.com/articles/possibilities-and-limitations-of-unstructured-data. [Accessed 20 1 2024].

[57] J. T. H. N. C. M. E. U. M. H. Camillo Lugaresi, “MediaPipe: A Framework for Building Perception Pipelines,” 14 6 2019.

[58] T. B. page, “New research aims to bring odors into virtual worlds,” 9 5 2023. [Online]. Available: https://www.technologyreview.com/2023/05/09/1072731/vr-smell/. [Accessed 21 1 2024].

[59] J. Murphy, “Hand gesture system enhances human-computer interaction,” 18 1 2022. [Online]. Available: https://www.laserfocusworld.com/science-research/article/14223766/hand-gesture-system-enhances-human-computer-interaction. [Accessed 21 1 2024].

[60] A. K. V. G. Shubham Shinde, “YOLO based Human Action Recognition and Localization,” 2018.

[61] D. M. S. N. M. M. D. Poddar, “YOLO-Pose: Enhancing YOLO for Multi Person Pose Estimation Using Object,” 2022.

[62] O. a. P.-R. A. a. B. G. a. C. A. Bahceci, “Supervised Machine Learning Hand Gesture Classification in VR for Immersive Training,” 2022.

[63] M. Gilliam, “Exploring Gesture Recognition in the,” 2019.

[64] C. D. Metcalf, S. V. Notley, P. H. Chappell, J. H. Burridge and V. T. Yule, “Validation and application of a computational model for wrist and hand movements using surface markers,” *IEEE Transactions on Biomedical Engineering,* vol. 55, no. 3, pp. 1199-1210, 3 2008.

[65] T. Kang, M. Chae, E. Seo, M. Kim and J. Kim, “DeepHandsVR: Hand Interface Using Deep Learning in Immersive Virtual Reality,” *Electronics 2020, Vol. 9, Page 1863,* vol. 9, no. 11, p. 1863, 11 2020.

[66] A. Clark and D. Moodley, “A system for a hand gesture-manipulated virtual reality environment,” *ACM International Conference Proceeding Series,* Vols. 26-28-September-2016, 9 2016.

[68] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, You Only Look Once: Unified, Real-Time Object Detection, Vols. 2016-December, IEEE Computer Society, 2015, pp. 779-788.

[69] G. Lan, Y. Wu, F. Hu and Q. Hao, “Vision-Based Human Pose Estimation via Deep Learning: A Survey,” *IEEE Transactions on Human-Machine Systems,* vol. 53, no. 1, pp. 253-268, 8 2023.

[70] E. N. Arcoverde Neto, R. M. Duarte, R. M. Barreto, J. P. Magalhães, C. C. Bastos, T. I. Ren and G. D. Cavalcanti, Enhanced real-time head pose estimation system for mobile device, vol. 21, IOS Press, 2014, pp. 281-293.

[71] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," Vols. 2016-December, pp. 779-788, 6 2015.

[72] M. A. Ozdemir, D. H. Kisa, O. Guren and A. Akan, "Hand gesture classification using time–frequency images and transfer learning based on CNN," *Biomedical Signal Processing and Control*, vol. 77, p. 103787, 8 2022.

[73] D. Han, C. Lee and H. Kang, "Gravity Control-Based Data Augmentation Technique for Improving VR User Activity Recognition," vol. 13, no. 5, p. 845, 5 2021.

8 Appendix

8.1 Questionnaire

Mei Immersive Reality

IPQ Questionary
Por favor indica o QUANTO CONCORDAS OU DISCORDAS com cada uma das seguintes afirmações selecioando apenas UM dos números utilizando a escala de 5 pontos

1 - Discordo totalmente
2 - Discordo
3 - Não concordo nem discordo
4 - Concordo
5 - Concordo totalmente

* Indica uma pergunta obrigatória

1. Estive consciente do mundo real enquanto navegava no ambiente virtual *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

2. O ambiente virtual pareceu-me completamente real *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

3. Tive a sensação de estar a atuar num espaço virtual *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

4. A experiência no ambiente virtual pareceu-me tão real como as minhas vivências do dia-a-dia *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

5. O ambiente virtual pareceu-me tão real como o mundo que conheço O ambiente virtual pareceu-me completamente real *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

6. Não me senti presente no ambiente virtual *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

7. Eu não estava consciente do mundo real que me rodeava *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

8. Eu tive a sensação de "estar" no ambiente virtual *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

9. De alguma forma eu senti que o mundo virtual me envolveu *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

10. Senti-me presente no ambiente virtual *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

11. Durante a experiência continuei a prestar atenção ao local onde estava a ter a experiência *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

12. O ambiente virtual pareceu-me mais realista do que o mundo real *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

13. Senti-me como se estivesse apenas a visualizar imagens *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

14. Senti-me completamente atraído pelo ambiente virtual *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

SUS

Por favor indica o QUANTO CONCORDAS OU DISCORDAS com cada uma das seguintes afirmações selecionando apenas UM dos números utilizando a escala de 5 pontos
1 - Discordo totalmente
2 - Discordo
3 - Não concordo nem discordo
4 - Concordo
5 - Concordo totalmente

15. Acho que gostaria de usar este sistema com frequência. *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

16. Considerei o produto mais complexo do que necessário *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

17. Achei o sistema fácil de utilizar *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

18. Acho que necessitaria do apoio de um técnico para conseguir utilizar este sistema *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

19. Considerei que as várias funcionalidades deste produto estavam bem integradas. *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

20. Achei que este sistema tinha muitas inconsistências *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

21. Suponho que a maioria das pessoas aprenderia a utilizar rapidamente este sistema. *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

22. Considerei o produto muito complicado de utilizar. *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

23. Senti-me muito confiante a utilizar este sistema *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

24. Tive que aprender muito antes de conseguir lidar com este sistema *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

25. Já experimentou outros sistemas de realidade virtual? *

Marcar apenas uma oval.

◯ Sim
◯ Não

26. Já experimentou outros sistemas de realidade virtual? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

27. Género *

Marcar apenas uma oval.

- ◯ Masculino
- ◯ Feminino

Gestos

1 - Discordo totalmente
2 - Discordo
3 - Não concordo nem discordo
4 - Concordo
5 - Concordo totalmente

28. Como avalia a sua interação com gestos? *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

29. A interação é intuitiva *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente