FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Flexible and Interactive Navigation in Synthesized Environments

Vítor Emanuel Fernandes Magalhães

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Master in Informatics and Computing Engineering

Supervisor: Luís Corte-Real

Co-Supervisor: Pedro Carvalho

External Supervisor: Américo Pereira

22 July, 2020

Flexible and Interactive Navigation in Synthesized Environments

Vítor Emanuel Fernandes Magalhães

Master in Informatics and Computing Engineering

Approved by:

22 July, 2020

Resumo

Tradicionalmente, o teclado e o rato são considerados as principais interfaces entre um utilizador e um computador para a interação com ambientes 3D. No entanto, a sua utilização está limitada à superfície em que é usada, como uma mesa. Este tipo de interação é invasivo, pois restringue a liberdade que o utilizador tem para interagir e pode tornar-se repetitivo, portanto, adaptar formas existentes de interação permitirá novas experiências para o utilizador. Além disso, estes ambientes 3D estão cheios de conteúdo interativo, mas este é feito à mão e os ambientes são, normalmente, criados sem qualquer tipo de descrição que facilita a sua criação.

Esta dissertação visa a desenvolver um sistema flexível e não invasivo, capaz de interagir e visualizar uma cena sintetizada, que inclui descrições do seu conteúdo, focando no fornecimento de mecanismos para interação com uma cena com restrições reduzidas. Para este fim, uma solução que abrange Reconhecimento Gestual e Estimação de Poses é apresentada: O sistema captura o gesto do utilizador utilizando uma câmara RGB, que é, de seguida, processado e, após ser identificado corretamente, é enviado para a cena 3D, que gere a ação correspondente, permitindo interação e navegação pelo ambiente. Ao interagir com determinados objetos, os utilizadores conseguem obter informações acerca destes objetos. Além disso, os diferentes ângulos das câmaras interna permitem que os utilizadores vejam diferentes perspectivas da cena, alternando entre as câmaras.

Os principais desafios consistiram em garantir flexibilidade na captura de *input* do utilizador usando uma câmara RGB e na definição e desenvolvimento de um conjunto de mecanismos de interação que pudessem ser perfeitamente incorporados em cenas visuais sintetizadas, assim como fazer a respetiva ligação de modo eficiente, garantindo o menor atraso possível no sistema.

Diferentes técnicas de Reconhecimento Gestual e Estimação Pontual são examinadas, assim como os tipos de câmaras que permitem capturar os movimentos do utilizador. Software variado para sintetizar a cena 3D também é apresentado e a arquitetura do sistema, além de alguns detalhes de implementação, são discutidos, com especial ênfase no primeiro.

Um método de avaliação foi criado para voluntários poderem testar e fornecer comentários. Os resultados dessa avaliação mostraram que os voluntários gostaram da imersão causada durante a navegação no ambiente 3D, tornando-os mais interessados em explorar e interagir com a cena. Apesar de o sistema por vezes misturar alguns gestos, nenhum atraso foi sentido assim que o sistema identificasse o gesto.

Apesar da rara identificação incorreta dos gestos, o sistema desenvolvido oferece uma interação não invasiva com ambientes 3D. Associado às descrições dentro destes ambientes, o sistema estabelece uma forma cativante para utilizadores navegarem e interagirem com uma cena virtual e o seu conteúdo disponível.

Abstract

The keyboard and mouse are considered the main interface between a user and a computer, when interacting with 3D environments. However, their use is limited as they're applied to a surface, such as a table. This type of interaction is considered invasive, due to restricting the user's freedom to interact, and can become monotonous, therefore, adjusting existing ways to interact will increase user experience. Moreover, these 3D environments are filled with interactive content, but this content is hand made and the environments are usually built without any kind of description which can facilitate the creation process.

This dissertation aims to develop a flexible and non-invasive system capable of interaction and visualization of a synthesized scene, which includes descriptions for its content, with focus on providing mechanisms for interacting with a scene with reduced constraints. To achieve this, a solution involving Gesture Recognition and Pose Estimation is presented: The developed system captures the user gesture using an RGB camera, which is then processed and, after correct identification, is sent to the scene, which handles the corresponding action, allowing for interaction and navigation of the environment. By interacting with certain objects, users can obtain information regarding these objects. Furthermore, different camera angles inside the virtual scene allow users to analyse different perspectives, by switching between the virtual cameras.

The main challenges comprised in guaranteeing flexibility in capturing the user input using an RGB camera and the definition and development of a set of interaction mechanisms that can be seamlessly incorporated into synthesised visual scenes, as well as connecting the previous two efficiently, guaranteeing the smallest amount of delay in the system.

Different Gesture Recognition and Pose Estimation techniques are examined, in addition to the types of cameras that allow capturing of the user movements. Different software for the Scene Synthesis is also presented and the architecture of the system, as well as some implementation details, are also discussed, with a specialized focus on the former.

An evaluation methodology was created so that volunteers could test and provide feedback. The results of the evaluation showed that the volunteers were engaged when navigating in the 3D environment, making them more engrossed and attentive to explore and interact with the environment further. Despite the system occasionally intermingling a few gestures, no delays were felt as soon as the system correctly identified the gesture.

In spite of the rare incorrect identification of the gestures, the system offers a non-restricted interaction with 3D environments. Combined with the descriptions of the scene, it establishes an alluring way for users to navigate and interact with a virtual scene and its content.

Keywords: Computer Vision, 3D, Interaction, Game Engines

Areas: CCS - Computing methodologies - Artificial intelligence - Computer vision;

CCS - Human-centered computing - Human-centered computing - Interaction techniques - Gestural input;

Acknowledgements

I would like to thank my supervisors, Professor Luís Côrte-Real, Pedro Carvalho and Américo Pereira, from INESC TEC, for guiding me during the internship preceding this dissertation, and for all the help and support throughout its planning and realization.

My internship colleague, Ricardo Carvalho, with whom I spent many hours discussing the nature of our dissertations, was instrumental in the improvement of the developed system.

I wish to express my deepest gratitude to my parents, Cidália and Aníbal, and my brother Pedro, for their continued support throughout the years, and to my friends and colleagues, in particular to Antero Gandra, Alexandre Martins, Bernardo Leite, Diogo Teixeira, Fábio Azevedo, João Esteves, José Pedro Martins, José Crisanto Magalhães, José Diogo Martins, Mariana Dias, Mariana Neto, Ricardo Lopes, Tiago Santos and Xénia Lazera, which had to endure my nonsense but made sure I was entertained and happy. Last but not least, I'd like to acknowledge Luke Lafayette, whose encouragement and kindness was the foundation for the writing of this document.

To all the mentioned entities, I am very grateful. Thank you.

Vítor Magalhães

'No Cost Too Great"
Pale King, Hollow Knight

Contents

List of Figures

List of Tables

Abbreviations and Symbols

2D	Two Dimensional
3D	Three Dimensional
API	Application Programming Interface
COCO	Common Objects in Context
CNN	Convolutional Neural Network
FPN	Feature Pyramid Network
FSM	Finite State Machine
GAN	Generative Adversarial Networks
GR	Gesture Recognition
HPG	Hand Pose Discriminator
HPG	Hand Pose Estimator
HPG	Hand Pose Generator
HTML	Hypertext Markup Language
JS	JavaScript
JSON	JavaScript Object Notation
N/D	Not Disclosured
PS	Playstation
PWA	Progressive Web App
RANSAC	Random Sample Consensus
RGB	Red Green Blue
RGB-D	Red Green Blue - Depth
SDK	Software Development Kit
SSD	Single Shot Detector
VGG16	Visual Geometry Group 16
VR	Virtual Reality

Chapter 1

Introduction

1.1 Context and Motivation

Computer-generated environments have been developed for many different uses, including designing three dimensional buildings and videogame scenarios, as well as reconstructing monuments, such as the Notre Dame. With new technologies, modelling lets creators build inanimate objects, life-like creatures or even full-scaled dynamic environments, like a football match and a shopping centre, whether they exist in reality or not.

3D environments are important as they permit users to immerse themselves in an assortment of scenarios through navigation and exploration, such as Virtual Museums, which are a concept primarily used to enhance the experience through interactivity, as explored by Geser and Niccolucci [1]. For example, Google has a collection of virtual museums which can be explored by anyone for free [2]. However, the way we generally interact with these environments has been the same for some time, which discourages any new form of interaction or adaptations of already existing means of interaction. In 3D scenes, interaction is mostly based on the movement of the mouse and keyboard or with touchscreens. Not only are these types of interactions becoming repetitive, since the actions aren't usually varied, but they also tend to belong to systems with invasive interactions. This downgrades the experience of the user, due to restricting their freedom. For example, the mouse and keyboard can be considered restrictive since they are limited to a surface's functionality. Juxtaposed with these are systems with non-invasive interactions which are advantageous due to their refinement and advancement of user experience as they minimize intrusions during any interaction. A hands free system is the most notable example, such as the one developed by Horvitz and Shwe [3], which allows for the user to have a more relaxed and stimulated interlinkage, due to not being bound to a surface. To surmise, adapting existing non-invasive mechanisms to improve interactions with virtual scenes will grant users new and more diverse opportunities to connect to the environment, making it more entertaining and engaging to anyone.

Developments on Gesture Recognition and Virtual Reality have helped to cement new ways to be immersed in virtual scenarios and these technologies can be used to enhance interaction with

Figure 1.1: Usage of VR on military training (adapted from Robots.net [4])

scenarios by removing limitations of the keyboard and mouse. For example, since its creation, Virtual Reality allows this type of immersion for military usage, as seen in Figure 1.1. It also helps managing anxiety levels [5] or designing intricate products, such as aircrafts [6], with its most popular application being video games. In another example, Gesture Recognition allows synergy between humans and robots, by making gestures for robots to perform through programmable actions communicated through mirroring and mimicry. Using these tools acts as an asset to the enrichment and betterment of the user's experience and permits the user more fulfilling and immersive interactions. There are also multiple scenarios where users can use these tools, such as security, monitoring and representing information. For example, during a football match, these tools can function to provide information about the players, while administering and monitoring spectators.

1.2 Objectives

The objective of this dissertation is to provide a flexible and non-invasive mechanism to interact with and visualise a 3D virtual scene. To reach this end, the following sub-objectives were achieved:

- Definition and implementation of a module for the automatic recognition of a set of gestures using captured visual information in less restricted scenarios;

- Implementation of mechanisms for associating interaction points with an automatically synthesised visual scene, filled with information;
- Integration of gesture recognition with the visual scene to enable an interactive navigation.

The main challenges consisted in guaranteeing the flexibility of capturing the user's hand gestures using video cameras and defining the mechanisms that allowed the hand pose to be translated into input, in order to interact with the virtual environment. Correctly capturing the user input and translating it into actions was a difficulty, since the action must be correct according to the user's gesture. Moreover, when making a gesture, the user expects the visualization of the actions as soon as possible; ergo, any latency between making a gesture, processing it, sending it to the scene and then executing the corresponding action can decrease the user's experience. Because these actions interact with the scene, the generation of 3D content must be precise in order to maintain immersivity and consistency. The mentioned method of Gesture Recognition and Interpretation worked as an interface between the user and the virtual scene. A setup scene based on a shopping center was created to test all functionalities, which includes basic 3D objects and free 3D models to enrich the environment. The proposed solution allows a greater immersion on exploring a 3D environment and, most importantly, motivate users to experience new ways to connect with the objects in the scene, since the users' interactions aren't restricted to a surface, as is the case of the keyboard and mouse, thus being much more relaxed.

1.3 Dissertation Structure

In addition to the introduction, this document contains six other chapters. Chapter 2 explores the state of the art and presents related works. In Chapter 3, the main problem that is approached in this dissertation is more fundamentally explored and the proposed solution is presented, followed by the architecture of the developed system which is presented alongside all implementation details as well as definitions related to the created system are explained. In Chapter 4, experiments made throughout development and obtained results are discussed, alongside the evaluation methodology for said results. Finally, in Chapter 5, conclusions and future work are presented.

Chapter 2

Literature Review

2.1 Introduction

The usage of non-invasive interactions requires tools that minimize any restriction for the user. These tools should capture user input, process it and then present its result in the scene; ergo, they must be discussed and identified in order to understand what can be used for each objective. This chapter is divided into Input Capture, Gesture Recognition and Interaction Synthesis: each section dwells into tools explored and articles read that fit into the scope of the dissertation.

2.2 Input Capture

This section covers the necessary tools to capture user input. Due to the requirement for reducing the usage of invasive input mechanisms, the usage of cameras is investigated for gesture recognition application, by capturing the input made by the user. After the input is properly captured by a video camera, it must be processed via Gesture Recognition.

There are three categories of cameras that will be discussed due to their capability of capturing user input information: RGB, RGB-D and Stereo Cameras. For each, the type will be described, with their pros and cons discussed with some application examples. Fig. 2.1 presents an example of each type. The first one is a Logitech RGB camera [7], whereas the second one is a RGB-D camera from Microsoft called Kinect [8]. The latter is the Intel Realsense Stereo camera from Intel [9].

Figure 2.1: Example of RGB, RGB-D and Stereo Cameras, respectively

2.2.1 RGB Cameras

An RGB Camera is able to capture colored digital frames and they have no dedicated sensor to perceive depth. These cameras are easily obtainable in the market, since they are cheap, and most laptops have one. In order to obtain the sense of depth, one can identify a marker for the camera to track, by calibrating the camera, or one can use two RGB Cameras and compare the captured frames [10] by calculating triangulation between them. Saxena et al. [11] were able to generate depth maps from still images with a learning approach and then apply those maps to create 3D models. Regazzoni et al. [10] developed a system using six Sony PS Eye RGB Cameras and concluded that, while they are able to capture a large environment, it required a uniform and light color background.

2.2.2 RGB-D Cameras

An RGB-D Camera has a sensor that is able to obtain the color image and depth perception. These cameras aren't as cheap as generic RGB cameras, for example, the Microsoft Kinect Sensor costs around 150 euros whereas RGB cameras can cost as low as 30 euros. Spinello and Arras [12] used a Microsoft Kinect Sensor and, according to them, the sensor has an infrared camera, an infrared projector and a standard RGB camera. To measure the depth, it follows the principle of structured infrared light. This process [13] involves projecting a pattern, such as grids or horizontal bars, and the way they deform when striking the surfaces allows the calculation of depth.

Regazzoni et al. [10]'s system also included two Microsoft Kinect for Windows Cameras and concluded that the setup time is smaller compared to the PS Eye cameras, but Kinect isn't as sensible to light as the PS Eye and the captured information files are larger.

2.2.3 Stereo Cameras

A Stereo Camera has two or more lenses and allows the simulation of human binocular/stereo vision, thus, it is more expensive than the previously discussed cameras. For example, the Intel Realsense costs around 320 euros. O'Riordan et al. [14] identify three types of stereo vision: *Passive*, which captures static objects; *Active*, which shares principles with the prior but allows dynamic motion of the cameras and objects and *Multiple-Baseline*, which uses various cameras to capture several images that are matched together to improve accuracy. Kim et al. [15] used a Bumblebee Stereo for 3D range data acquisition, which is calculated after the two sensors are rectified. They experimented by capturing images of an irregular surface and calculating the range data via stereo system of the camera. The data was used for the neural network they were applying for a multiple plane area detection. Liu and Kehtarnavaz [16] created a system to recognize gestures using stereo images captured by a stereo webcam to achieve robustness under realistic lighting conditions. By comparing the accuracy between a single image and two stereo images, both achieved a similar frame rate, but, when merging both stereo images, the average detection rate improved by almost 60 %.

2.3 Gesture Recognition

Gesture Recognition is described as the process of identifying the gesture performed by a user and usually has the aim of interpret certain commands [17].

Gestures are expressive body motions which include the movement parts of the body such as the fingers, hands, arms, head, face or even the whole body in order to transmit information or interact with an environment. Gesture recognition has a wide range of possible applications, including recognizing sign language, teaching young children to interact with computers and lie detection.

Since gestures are ambiguous, there is a challenge in defining the same meaning for the different gestures and vice-versa, for example, a hand facing forward slightly above the head can represent a greeting ("Hello") or an appreciation ("Thank you").

Gestures can be static or dynamic. As the name indicates, static gestures are simply poses and dynamic gestures have prestroke, stroke and poststroke phases, in order to represent the complete gesture.

To detect and determine these aspects, tracking devices are required. These can be attached to the user, like gloves, or they can be cameras which use computer vision techniques in order to detect and recognize the gestures [18]. This dissertation will focus on the latter as it is less invasive and does not require users to have very specific devices.

Gesture Recognition typically has three different phases: **Hand Detection**, where the hand is recognized, **Tracking**, where the hand is tracked and its movement is processed and **Classification**, where the gesture is classified. Besides these phases, *Hand Pose Estimation* is also discussed.

2.3.1 Hand Detection

Image segmentation is the operation of partitioning an image into several segments, in order to simplify the original image [19]. The objective is to simplify or change the representation of an image into something more meaningful and easier to analyze. Patrick de Sousa [20] analised several methods in order to recognize gestures for Human-Robot interaction. According to him, the most popular type of segmentation is skin-color based. The downside of this method is that the user cannot use gloves and skin colored objects can't appear in background as well as some being susceptible to light variation. Park and Lee [21] developed a system where the hand was detected based on the color of the face, but not all users would like to have their face detected. Cerlinca and Pentiuc made a real time 3D hand detection and assumed the hands were always closer to sensor than the head [22]. Despite being based on head location, one can also assume that the hands will always closer to the sensor in order to navigate and interact with the scene. The use of skin-color and depth information for a real-time 3D hand recognition has also been used by Bergh and Gool [23]. By using a Time of Flight and an RGB camera, they were able to detect the face of the user and the distance between them and the camera. With the calculated distance, the system is able to discard background objects.

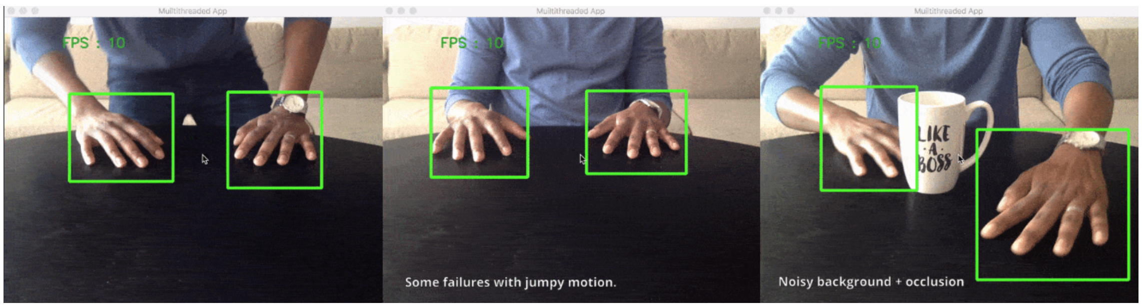

FPS : 10
FPS : 10
FPS : 10
Some failures with jumpy motion.
Noisy background + occlusion
LIKE A BOSS

Figure 2.2: Detector in action (adapted from Victor Dibia's HandTracking [24])

Victor Dibia [24] developed a real-time Hand Detection and Tracking system using Neural Networks, as they are trained to perform on several environments with challenges such as poor lighting, occlusion and noisy backgrounds, as seen in Fig. 2.2. He also published the code as Open Source, available online on Github.

2.3.2 Tracking

After segmenting the image, the next step is to track the hand and to detect the corresponding gesture. This is used so the system knows the location of the hand and to avoid further segmentation costs.

Markov models have been used to identify a number of gestures to control an application [25]. Ramey et al. [26] classified a gesture of a waving hand, by using a finite state machine. A Convolutional Neural Network (CNN) was proposed [27] to detect gestures from different inputs with depth, color or stereo-IR sensors, achieving an 83.8 % accuracy when recognizing gestures. To implement a real-time hand tracking, Chen et al. [28] used a region growing technique with the seed point on the estimated center of the hand based on the previous frame. The estimation of the first seed was obtained by a hand click detection method in order to start the tracking. As mentioned in section 2.3.1, Dibia's system is able to track the hand movement as well as detect it [24].

There are sensor devices that allow tracking, such as the Leap Motion [29], developed by Ultraleap. It is a Hand Tracking device that uses hand and finger motions as input, analogous to a mouse, but requires no hand contact or touching, as seen in figure 2.3. It's a cheap device but users are dependant of the position of the sensor, which limits the area to be tracked. Due to this limitation, it's not a good device to be used on this dissertation.

2.3.3 Classification

The classifier plays a major role in the system: its task is to identify the gesture based on the result of the hand detection and tracking sections. According to Sonkusare et al. [17], the input features can be compared with features of a database. Molchanov et al. [27] perform classification by splitting the captured video feed into clips of a certain length and computing a set of

Figure 2.3: Leap Motion Controller (adapted from Ultraleap [30])

class-conditional probabilities. They also consider calculating the final classification score with a support vector machine, achieving a final accuracy of 83.8% when classifying gestures. A CNN was used by Jiang et Al. [31] to detect ASL (American Sign Language) dataset. To test it, they used five different gestures, obtaining a 96.01% accuracy on recognizing gestures. Patrick de Sousa [20] used a simple Finite State Machine (FSM) for each gesture to be detected. It was designed according to the movement of the proposed gestures. Hidden Markov Models were also used to classify Indian sign language [32], obtaining an accuracy of 89.25 %.

2.3.4 Hand Pose Estimation

Hand pose estimation is the procedure of modeling the human hand as a set of some parts, for example, the fingers, and calculating their positions in the image of a hand [33]. Its application is vast for a wide range of field. For example, the area of robotics can greatly benefit with this technique, since users can control a robotic hand using specific poses that command the machine. In the medical field, this can be extended for a doctor to operate during a surgery or even a student to practice. Hands are usually modeled as a number of joints and, by finding the position of the joints, the real hand pose can be estimated. The most popular number of joints to model hands is 21, although there are some models which use 14 and 16 [33].

According to Doosti, there are two type of networks used in the Hand Pose Estimation problem: **Detection-Based** and **Regression-based** [33]. In the first, for every existing joint, the model produces a probability density map. Assuming the network uses a 21-joints model, each image will produce 21 different probability density maps in the form of heatmaps. To find the exact location of each joint, one only needs to use a *argmax* function on the corresponding heatmap. In comparison, regression-based method attempts to directly estimate the position of each joint.

Assuming the model composed by 21 joints, it should have 3 x 21 neurons in the last layer to predict the 3D coordinates of each joint. Training a regression-based network requires more data and training iterations because of to the high non-linearity. However, producing a 3D probability density function for each joint is a substantial task for the network, therefore, regression-based networks are utilized in tasks of 3D hand pose estimation.

The two main methods for Hand Pose Estimation are **Depth** and **Image** based methods. The first one makes use of depth map images, which was traditionally the main method for estimation. Based on a depth map, Sinha et al. [34] used a regression-based method to find 21 joints in the hand. They sought to learn the location of joints in each finger independently, so, they trained a separate network for each finger to regress three joints on that finger. Despite using a depth map to regress the coordinates, they also used RGB to isolate the hand and to remove all other pixels included in the hand's cropped frame.

Baek et al. were able to use Generative Adversarial Networks (or GAN) to estimate the hand pose by making a one to one relation between depth disparity maps and 3D hand pose models [35]. GAN is a specific CNN that generates new samples based on the previous learned ones. It consists of a discriminator and a generator network which are competing with each other: while the discriminator network is a classifier trained to distinguish real and fake images, the generator, also a convolutional neural network, generates fake images. The authors used a GAN, named Cyclic-GAN, for transferring one image from one domain to another. In this context, one domain is the depth map of the hand and the other is the 3D representation of the hand joints. They used a Hand Pose Generator (HPG) and a Hand Pose Discriminator (HPD) in their model. The HPG generates a hand, based on the 3D representation of the joints. Meanwhile, they used a Hand Pose Estimator (HPE) whose objective is to generate the 3D hand pose, based on the input depth map. Thus, in the training step, HPG, HPD and HPE are optimized to reduce the error of HPE and to increase the consistency of the HPE-HPG combination. In the testing phase, the algorithm refines the 3D model, guided by the HPG, to generate the best 3D model whose corresponding depth map is akin to the input depth map.

The second method for Hand Pose Estimation is **Image Based**, which makes use of RGB images. This is a significantly harder task because the data has its dimension reduced, which loses some of the information, however, RGB cameras are more commonly used than RGB-D cameras. These methods need to isolate the hand (segmentation) first and then pass the cropped image to the network to estimate the pose.

Zimmerman and Brox developed a network, called ColorHandPose3D, which estimates a 3D hand pose from a single RGB image [36], by detecting the associated keypoints from the joints of the fingers, as seen in Fig. 2.4. The system uses the RGB image on the left and creates a 3D hand pose, presented on the right. They used several different deep learning streams in order to make a 3D estimation of hand joints using a single color image. The network uses *PoseNet* [37], a **detection-based** network which produces one probability density function for each hand joint. These predictions are in two dimensions and on the same coordinates of the input image. To obtain a 3D estimation, Zimmerman et al. used a network named *PosePrior* to convert the 2D

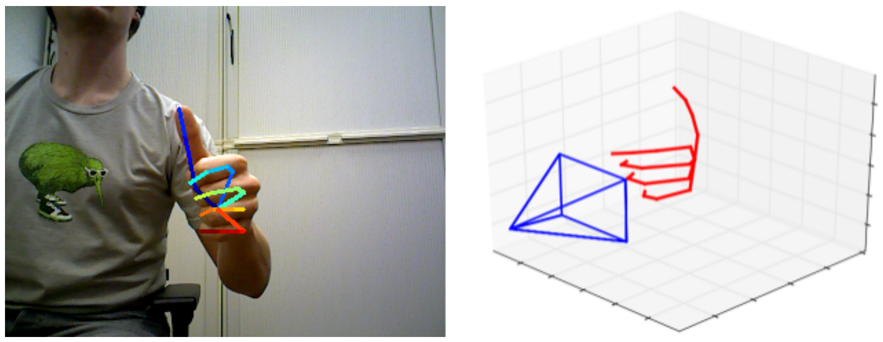


Figure 2.4: 3D hand pose (adapted from ColorHandPose3D [36])

predictions to the 3D hand estimation. *PosePrior* is a **regression-based** network, able of predicting the coordinates of the joints followed by a normalization. In this step, the authors normalize the distances of the joints, considering the farthest distance as 1 and dividing the other distances to that number. Finally, they find a 3D rotation matrix such that a certain rotated keypoints is aligned with the y-axis of the canonical frame.

The biggest issue regarding image-based methods is occlusion of the hand. Simon et. al [38] used a multicamera approach to estimate the hand pose. First, they trained a weak hand pose estimator with a synthesized hand dataset. Afterwards, they put a person in the center of the panoptic and applied the hand pose estimator on all the cameras recording video. The algorithm produces a slightly inaccurate pose estimation for all of these views. It functions in most of the views, but in the ones in which the hand is occluded, it poorly works. In the next part, which the authors called *triangulation* step, they converted their 2D estimation to 3D estimation to evaluate their results. To estimate the correct 3D pose, for each joint, they used the RANSAC [39] algorithm to randomly select 2D views and convert them to 3D. Then they kept the model with which most of the views agree. Finally, in a reverse operation, they projected the 3D view to the pictures and annotate that frame. Instead of manually annotating data, the authors used multiple views to annotate the data for them. Then, with this annotated dataset originated from multiple views, they trained the network again and made it more accurate. By repeating this process of annotation and training, they ended up with a superior and accurate model and dataset of annotated hand poses.

2.4 Interaction Synthesis

This section covers tools that allow the generation of the 3D content from the scene, as well as making the scene interactive by receiving the result of the Gesture Recognition.

A Game Engine is a software-development environment designed for people to build video games. The core functionality typically provided by a game engine includes a rendering engine, a physics engine or collision detection (and collision response), sound, scripting, animation, artificial intelligence, networking, streaming, memory management, threading, localization support, scene graph, and may include video support for cinematics.

Table 2.1: Game Engines Table

Game Engine	**Author(s)**	**Licence**	**GR Integration**
Blend4Web	Triumph [40]	GPLv3	N/D
CryEngine	Crytek [41]	Royalties	No
GameMaker	YoYo Games [42]	Shareware	No
Godot	Linietsky and Manzurr [43]	Free	No
jMonkey Engine	jMonkey Team [44]	BDS3	N/D
Ogre 3D	Ogre Team [45]	MIT	No
Panda 3D	Carnegie Mellon University [46]	Free	No
Three.js	mrdoob [47]	MIT	N/D
Unity	Unity Technologies [48]	Free	Yes
Unreal Engine 4	Epic Games [49]	Free	Yes

For this dissertation, the used game engine should support gesture recognition and virtual reality integration. Table 2.1 sums up the searched engines and conclusions regarding the Gesture Recognition. One can conclude that *Unity* and *Unreal Engine 4* are able to integrate Gesture Recognition, therefore, they're the most capable engines to generate the content of the visual scene as well as integrate the interaction mechanisms. They are also the most popular engines, which come with regular updates and fixes from the developers.

2.5 Discussion

In order to capture the user input, the reached conclusion is that RGB and RGB-D cameras are the best options, due to their inexpensiveness and ease of access to users. This is supported by multiple studies. For instance, Regazzoni et al. [10] presented a study comparing both types of cameras in terms of information captured from the scene. Dibia [24] also showed that this type of cameras can be used to detect and track hands. More recently, Jiang et al. [31] presented a new CNN architecture that was capable of detecting and classifying gestures.

These two systems use CNNs, which appear to be the most interesting technique to use for Gesture Recognition. Hand Pose Estimation also seems to return the best results in terms of accurately identifying a gesture. Finally, the most adequate Game Engines are Unity and Unreal Engine 4, since they allow integration with Gesture Recognition.

Due to the context of the dissertation, many sources are from websites and usually regard authors of Game Engines.

Chapter 3

Proposal and Implemented System

This chapter is divided into three parts: In the first part, the Problem is discussed and a Proposed Approach is presented; the second part details and discusses the Architecture of the Developed System, including a description of its modules along with their connections. Additionally, the selected gesture dataset alongside the corresponding actions are also revealed; the third and final part of this chapter serves to detail the system's implementation. The used tools and definitions are indicated, as well as the modules of the system and descriptions used for the objects inside the 3D scene.

3.1 Problem and Proposed Solution

Three dimensional environments have many uses, such as virtual museums and video games, but the way we interact remains the same and this is usually part of systems with restrictive interactions, which downgrades the user's experience. Furthermore, these environments aren't usually built with the assistance of descriptions to facilitate the process.The main problem consisted on the usage of invasive interaction mechanisms, due to restricting the user's freedom to interact and the lack of adoption of descriptive scenes, as they can easily translate the environment of a scene. These restrictions can also limit the input capture, thus, the automatic connection between the input and the 3D environment is impacted. The consolidation between the non-invasive mechanisms and descriptive scenes allows an easier creation of 3D environments and simpler interactions with it.

The proposed solution aims to uplift the use of non-invasive interaction mechanisms in order to not only reduce user's limitations when interacting with a scene, but also increase the immersiveness of 3D visualization, by enabling interactions between the user and the scene. The solution also includes the creation of interaction mechanisms and their integration with the scene. To demonstrate the usability of such a solution, a system was developed that enables these interactions. A virtual scene, based on a shopping center, was also defined, composed by objects and virtual cameras. The objects correspond to elements present in the surroundings, such as benches, vases and even people. Some of these objects may be interacted with: by selecting an object, a user may obtain information about it or even switch to the perspective of that object, if it contains said functionalities. The virtual cameras permit the user to switch between perspectives inside the scene: this way, they can find new details and quickly navigate between the various cameras.

Users are able to navigate and interact with the scene, gesturing with their hands. These gestures are captured by an RGB video camera and then processed by the system and sent to the scene in the shape of actions, but the traditional way of using the keyboard and mouse is maintained if the users wish to use them.

The system can be summed up in two components, as presented in Fig. 3.1: one that captures and handles user input and another that processes the scene and actions from the user. The first component captures any hand gesture with an RGB video camera, classifies it and then sends the corresponding action to the second component. This one contains a scene with objects and virtual cameras, as well as information about them, and converts the actions from the previous component into an interaction inside the 3D scene.

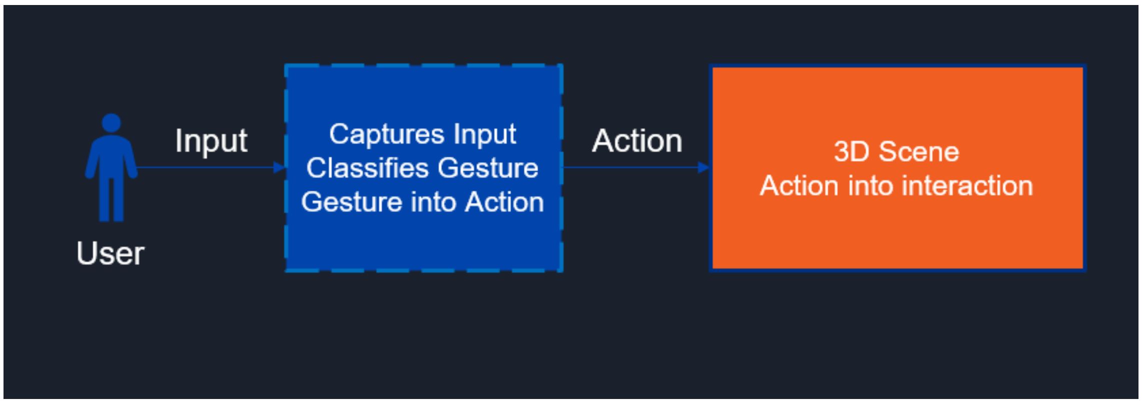


Figure 3.1: Abridged Dissertation Scheme

Initially, tests took place to define the Game Engine to be used as well as an analysis of available projects that could serve as a base for the system. A 3D scene was created throughout development, with interactive objects and different camera angles.

After having a basis for recognizing gestures with an RGB camera and classifying them, the next step was to establish communication with the Game Engine and translate a couple of gestures into interactions. As the communications were settled, all actions inside the scene were developed, including movement, changing between virtual cameras and interacting with objects. Afterwards, objects' descriptions were defined to be read by the system.

Finally, the creation of the complete set of gestures followed shortly after, mapping all possible interactions from the user. Most gestures that were defined took practical and regular poses in consideration, as well as making sure the gestures wouldn't intermingle with each other.

Throughout the dissertation, experiments were made and volunteers were requested to operate the developed methods against the Keyboard and Mouse method, providing feedback.

3.2 System Architecture

The system is divided into several modules, each with their own functionality. A visual representation of the system can be seen in Fig. 3.2. The **Input Capture** Module captures gestures made by the user and translates them into poses, which are used by the **Gesture Classification** Module to be classified. Once it's classified, the **Action Correspondence** Module verifies the gesture and sends an action in the form of a message to the **Scene Processing Module**. This Module converts the message into an action inside the virtual environment, which, in turn, returns the end result of the system in the **Scene Synthesis** Module. The successive use of these previous modules allows the user to experience built-in mechanics to interact with a virtual scene.

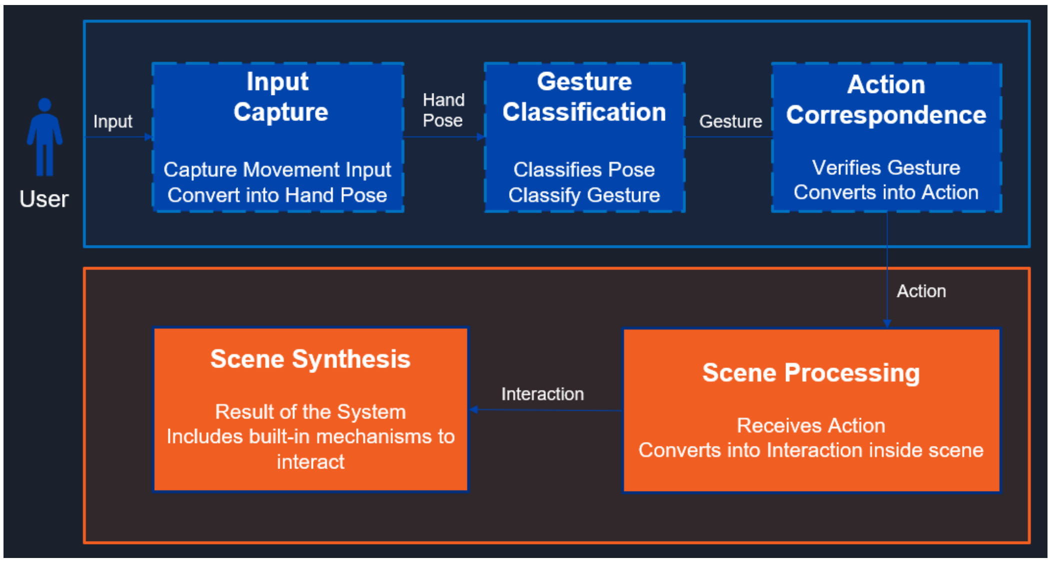


Figure 3.2: Visual Representation of the System

3.2.1 Input Capture

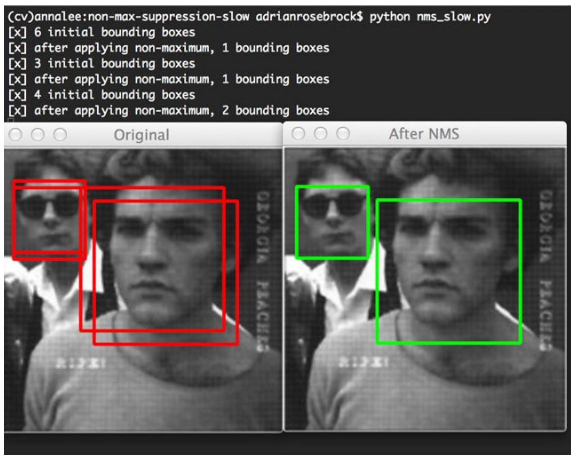
The Input Capture Module is used to capture the user's hand with an RGB camera and transcribe its gesture into the user's hand pose. The module uses two models from the Handpose project [50]: a *Palm Detector* and a *Hand Landmark Detector*.

The ***Palm Detector*** receives an image and outputs bounding boxes of the detected palm. It is based on a Single Shot MultiBox Detector [51], or SSD, a Neural Network capable of detecting objects in images which uses multi-scale convolutional bounding box outputs attached to multiple feature maps at the top of the network, allowing to efficiently model the space of possible box shapes. It eliminates proposal generation and any pixel/feature resampling stages, encapsulating all computation in a single network, while maintaining accuracy. SSD uses **VGG16** [52] as the base network. It's a standard architecture used for high quality image classification and, in this case, to extract feature maps, being located in the early network layers and truncated before any classification layers.

The authors then add an auxiliary structure to the network to produce detections with the following key features:

- Multi-scale feature maps - Convolutional layers are added to the end of the base network. They decrease in size progressively and allow predictions of detections at multiple scales.
- Convolutional predictors for detections - Each added feature layer can produce a fixed-set of predictions using convolutional filters.
- Default boxes and aspect ratios - Default bounding boxes are associated with a feature map cell at the top of the network. The default boxes tile the feature map in a convolutional manner, so that the position of each box relative to its corresponding cell is fixed.

Given the large number of boxes generated during a forward pass of SSD at inference time, it is essential to prune most of the bounding boxes by applying a technique known as *non-maximum suppression* to remove duplicate predictions pointing to the same object. Hosang et al. explain that the algorithm selects high scoring detections and deletes close-by less confident neighbours since they are likely to cover the same object [53], as seen in Fig. 3.3: the result greatly removes duplicated bounding boxes that represent the same object. This also ensures only the most likely predictions are retained by the network, while the noisier ones are removed.



```
(cv)annalee:non-max-suppression-slow adrianrosebrock$ python nms_slow.py
[x] 6 initial bounding boxes
[x] after applying non-maximum, 1 bounding boxes
[x] 3 initial bounding boxes
[x] after applying non-maximum, 1 bounding boxes
[x] 4 initial bounding boxes
[x] after applying non-maximum, 2 bounding boxes
```

Figure 3.3: Non-Maximum Suppression (adapted from PyImageSearch [54])

Fig. 3.4 reveals the architecture of an SSD. It receives an image and the initial layers are based on VGG16, which extracts feature maps, and detects objects using the other deeper layers. After the detections, the *non-maximum suppression* removes duplicates, effectively reducing the number of detections.

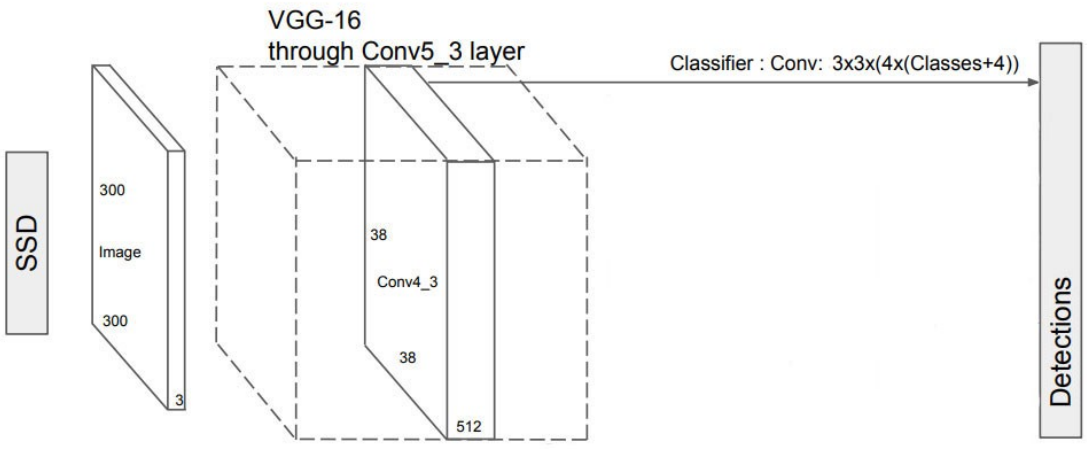


Figure 3.4: SSD Architecture (adapted from Medium [55])

Training an SSD and training a typical detector that uses region proposals differs in the assignment of ground truth information to specific outputs in the fixed set of detector outputs. For an SSD, once the assignment is determined, the loss function and back propagation are applied end-to-end. Training also involves choosing the set of default boxes and scales for detection as well as hard negative mining and data augmentation strategies.

During training, each default box is corresponded to a ground truth detection and the network is trained accordingly. For each ground truth box, a default box that varies location, scale and aspect ratio is selected. First, each ground truth box is matched to a default box with the best jaccard overlap. Then, the default boxes are matched to any ground truth with overlap higher than a threshold (0.5). This simplifies the learning problem, allowing the network to predict high scores for multiple overlapping default boxes rather than just one.

Three other strategies are used to enhance the training. The first one is *Choosing a set of default boxes and scales*, in order to handle different scales and obtain a diverse set of predictions. The second strategy involves *Hard negative mining*: after matching, most default boxes are negative, which introduces a significant imbalance between positive and negative examples; ergo, this technique balances the positive and negative examples, leading to a faster optimization and stable training. The final technique is *Data Augmentation*, which makes the SSD more robust to various object sizes and shapes.

The ***Palm Detector*** model trains the detection of the palm rather than the hand because estimating bounding boxes of rigid objects like palms and fists is significantly simpler than detecting hands with articulated fingers. Palms can be modelled using square bounding boxes (anchors) ignoring other ratios, thus reducing the number of anchors by a factor of 3-5. Besides this, the *non-maximum suppression* algorithm works for smaller objects, which boosts performance when dealing with palms. However, the SSD alone isn't enough to detect small objects (in this case, the palms), with high precision. These objects can only be detected in higher resolution layers, which are the leftmost layers, but these contain low-level features, like edges or color patches, that are less informative for classification. In order to solve this, the model uses a similar approach to the Feature Pyramid Networks [56], or FPN, to provide a bigger scene context awareness even for small objects. FPN provides a Top-Down Pathway, as seen in Fig. 3.5 to construct higher resolution layers from a semantic rich layer, as well as Bottom-Up Pathway, which is what convolutional networks usually use for feature extraction. With Bottom-Up, as the image gets processed, the semantic value for each layer increases and the spatial resolution decreases. While the reconstructed layers are semantically strong, the locations of objects are not precise after all the downsampling and upsampling. Adding lateral connections between reconstructed layers and the corresponding feature maps helps the detector predict the location better; ergo, FPN allows for a better small object detection in lower resolution layers, further boosting the precision. The combination of lateral connections and a Top-Down Pathway greatly increases accuracy. According to the authors of FPN, it improves accuracy by 8 points on the COCO dataset and, for small objects, it improves by 12.9.

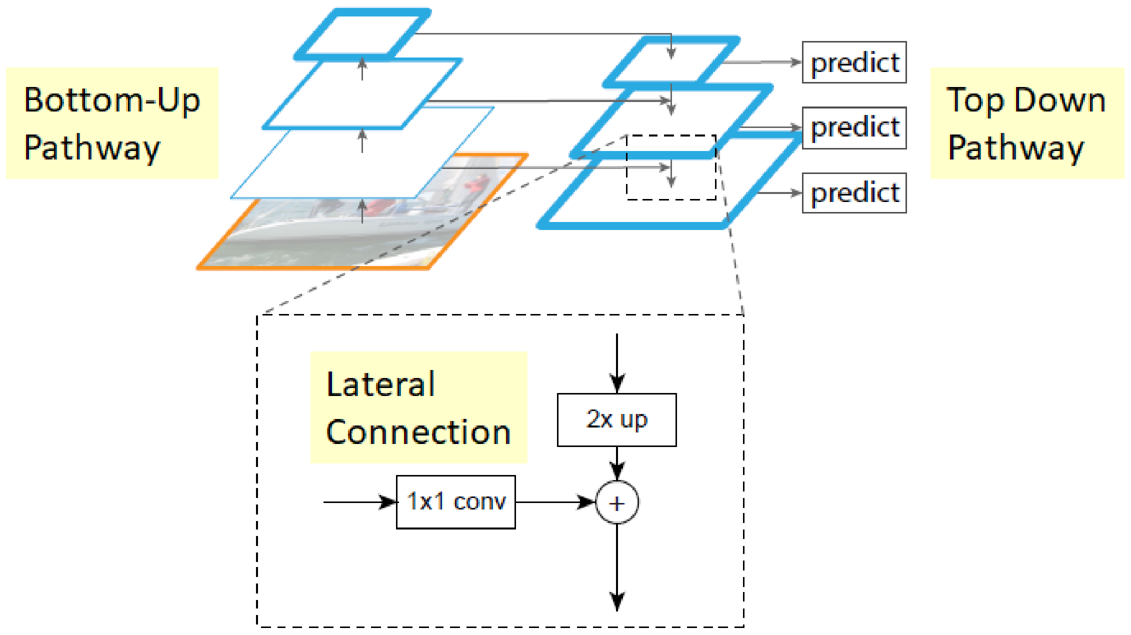


Figure 3.5: Feature Pyramid Networks architecture (adapted from TowardsDataSciene [57])

Lastly, the *Palm Detector* model minimizes focal loss during training to support large amount of anchors resulting from high scale variance. Focal Loss [58] focuses training on a sparse set of hard examples and prevents the vast number of easy negatives from overwhelming the detector during training. According to Bazarevsky and Zhang, the authors of the model, the final model achieves an average precision of 95.7% in palm detection [59].

In short, the ***Palm Detector*** model is based on the SSD, with a feature extractor based on FPN to improve small objects detection. During training, the model minimizes focal loss to reduce the number of easy negatives.

The second model, ***Hand Landmark Detector***, works on the bounding boxes from the previous model and makes all calculations based on this input. It's a CNN regression model that performs precise keypoint localization of 21 3D coordinates through a *Regression-based* method, as discussed in subsection 2.3.4 in chapter 2. These keypoints represent the 3D estimation of the hand's pose, which are then used by the *Gesture Classification* Module. The model learns a consistent internal hand pose representation and is robust to partially visible hands and self-occlusions. To obtain ground truth data, the authors manually annotated around 30 thousand real-world images with 21 3D coordinates [59]. To better cover the possible hand poses and provide additional supervision on the nature of hand geometry, they also rendered a high-quality synthetic hand model over various backgrounds and map it to the corresponding 3D coordinates. In order to improve training, they utilized a mixed training schema which uses both synthetic and real world data, resulting in a significant performance boost, with a mean regression error normalize by palm size of 13.4%, whereas only real-world data resulted in 16.1% and the synthetic data in 25.7%.

3.2.2 Gesture Classification

The Gesture Classification Module is tasked with classifying the gesture the user performs. To do so, it receives the 21 keypoints from the previous module, as seen in Fig. 3.6 and calculates the estimations of the curls and orientations of each finger. It then compares the obtained estimations with the known gestures in the dataset, saving all gestures with a confidence over a threshold.

The module begins by estimating the curls and orientations of each finger, using the received keypoints. Firstly, it must calculate the slopes of each finger. Each finger has a set of four mapped pair of points, in the form of indexes, which represent the connection between the finger's joints. For each pair of points from the set, the system calculates the slopes, using function 3.1, considering P1 and P2 as the two points from the pair:

$$Slope = \frac{\arctan(\frac{(P1.y-P2.y)}{(P1.x-P2.x)}) * 180}{\pi} \tag{3.1}$$

Secondly, the system calculates the curls of the fingers. After obtaining the indexes like before, the system gets the start, middle and end points of the finger. For example, the points of the thumb, in Fig. 3.6, are the points 2, 3 and 4, respectively. These points correspond to the fingers' joints and are obtained by associating the indexes with the keypoints from the previous module.

To verify the curl of the finger, the distances between the three points are calculated (start to middle, start to end and middle to end). Then, it determines the angle between the distances and

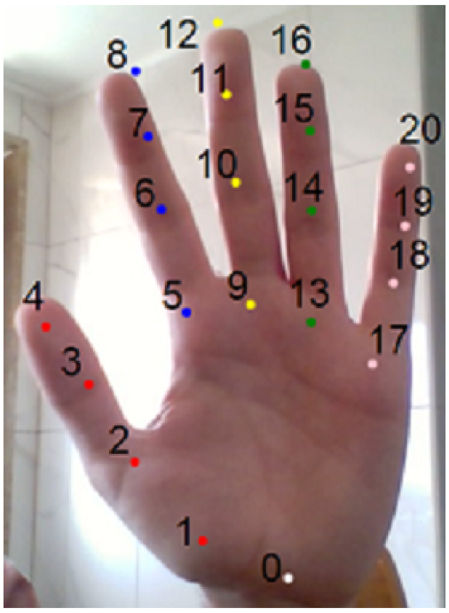


Figure 3.6: 21 detected Keypoints of a gesture

defines whether the finger is curled or not. Fig. 3.7 shows an example of the different detected curls in the Index finger alongside the obtained degrees on each curl.

To distinguish each possible curl, limits were defined which represent the angle between the palm of the hand and the finger. As the finger curls, the aforementioned angle decreases. During performed tests, the finger is half curled up to angle values close to 60º and completely curled below that, while 130º was the minimum angle for the finger to be considered stretched. The curl distinction is made as follows:

- If the angle is higher than the No Curl Limit (130º), it is not curled;
- If the angle is higher than the Half Curl Limit (60º), it is half curled;
- If the previous two don't apply, it is curled.

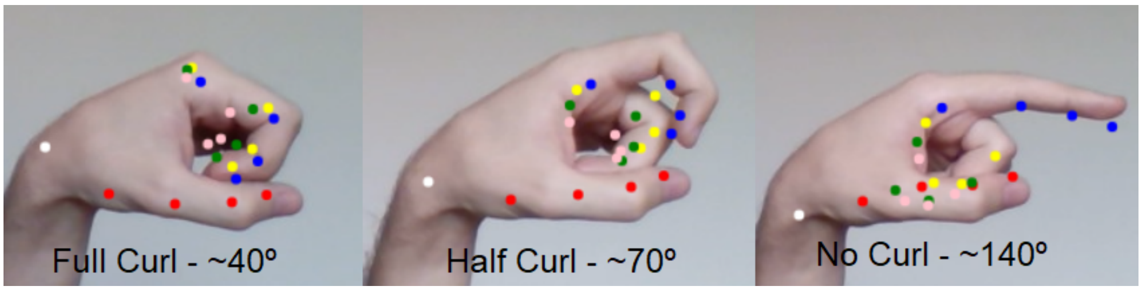


Figure 3.7: Example of Curl Difference on the Index Finger

Thirdly, the system estimates the direction (Vertical, Diagonal or Horizontal) and orientation (Up, Down, Left or Right) of each finger, using the previously calculated slopes of the finger, in addition to the start, middle and end points. Throughout this segment, the system verifies the type of direction and adds a weight to the most likely direction.

After obtaining the longer distance between the points (on both Y and X axis), the system verifies the ratio between said distances and checks the tendency of being Vertical, Diagonal or Horizontal, while associating a weight to it. During tests, a vertical direction was more noticeable with ratio values higher than 1.5. As this value decreases, this direction tends to be diagonal and with even lower values, the direction becomes horizontal. To sum up:

- If the ratio is higher than 1.5, the direction is likely to be Vertical;
- If the ratio is higher than 0.6, the direction is likely to be Diagonal;
- If the previous two don't apply, the direction is likely to be Horizontal.

Furthermore, the system also verifies the longer distance between points (start to middle or middle to end) and obtains the angle between those two points, with the function 3.1, using the result to verify the direction and associating a weight again. Performed tests revealed that when this angle was close to values between 75 and 105, the finger was visually closer to a vertical direction, while a diagonal direction corresponded to values between 25 and 155. If the angle was lower than 25 or higher than 155, then the direction was relatively horizontal. To condense this information:

- If angle $\in [75, 105]$, the direction is likely to be Vertical;
- Else if angle $\in [25, 155]$, the direction is likely to be Diagonal;
- Else, the direction is likely to be Horizontal.

An example can be seen in Fig. 3.8, which indicates the mentioned values associated with the Thumb. The maximum distance between the thumb points in the X axis is 57, while in the Y axis, it's 103. Since the ratio between Y and X is 1.22, the direction is most likely Diagonal. Additionally, all angles between the three points are between 25 and 155, thus, the likelihood of Diagonality increases. Both tests indicate toward the same direction, thus, the Thumb's direction is Diagonal.

After making the same verifications for the previously calculated slopes, the system concludes which is the most probable direction, taking the calculated weights into account, and estimates its orientation: Vertical and Horizontal Directions can either have Up or Down Orientation, whereas Diagonal can have UpLeft, UpRight, DownLeft or DownRight Orientation. To verify the orientation for an estimated Vertical Direction, the system uses the maximum distance between the finger points in the Y axis. The Orientation is Down when the maximum distance corresponds to a distance between two finger points(start to mid, start to end or mid to end) and this distance

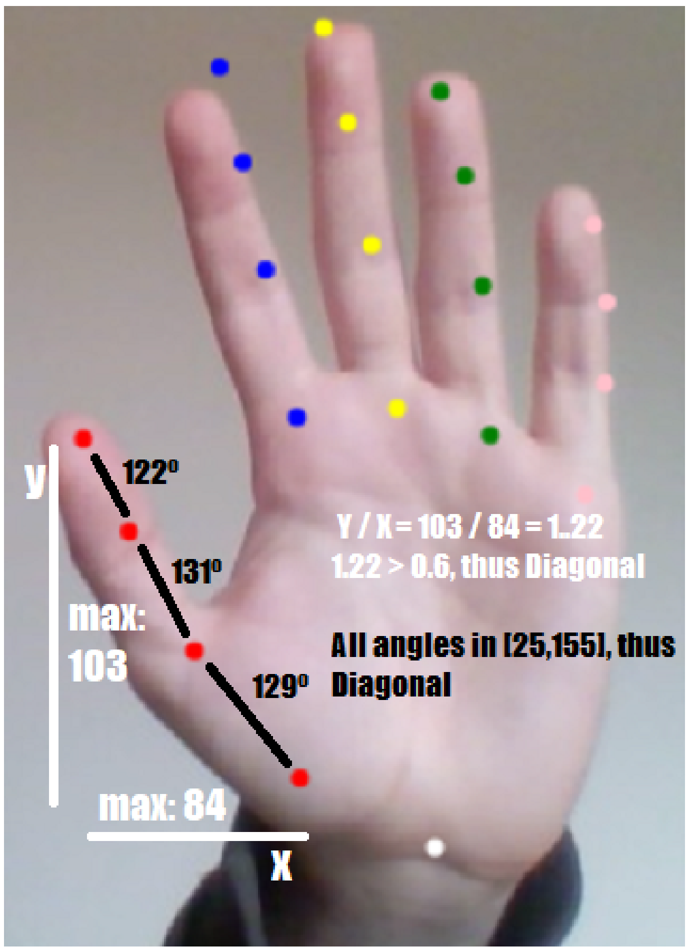


Figure 3.8: Example of Direction calculation in the Thumb

is negative. If the distance is positive, then the Orientation is Up. For an estimated Horizontal Direction, the system uses a similar approach, but towards the X axis: The Orientation is Left when the maximum distance corresponds to a distance between two finger points and this distance is positive. If the distance is negative, then the Orientation is Right. Finally, for an estimated Diagonal Direction, the system starts by estimating the Orientation in the Horizontal and Vertical directions and makes matches between them: if it estimates VerticalUp and HorizontalLeft, then the final orientation will be UpLeft. If it's HorizontalRight, then the final orientation will be Up-Right. If it instead estimates VerticalDown and HorizontalLeft, then it's DownLeft, otherwise it will be DownRight.

Using the example of Fig. 3.8, since the Direction is Diagonal, all that remains is to calculate the Orientation. Vertical wise, by verifying that the Maximum Distance in the Y axis is equal to the distance between the start and end points, with the latter being positive, the system concludes the Orientation is Up. Besides, when verifying that the Maximum Distance in the X axis is also equal to the distance between the start and end points, with the latter positive, the system also concludes the Orientation, Horizontal wise, is Left; ergo, the final Direction is Diagonal, with an

Up Left Orientation.

Figures 3.9 and 3.10 are examples of a Horizontal Right and Vertical Up orientations, respectively. On the first figure, the ratio between Y and X is 0.08, therefore the direction is most likely Horizontal; additionally, all angles not in $[75, 105]$ nor $[25, 155]$, thus further indicating that the direction is Horizontal. On the second figure, the ratio between Y and X is 42, therefore the direction is most likely Vertical; moreover, all angles are in $[75, 105]$, which indicates a Vertical direction.

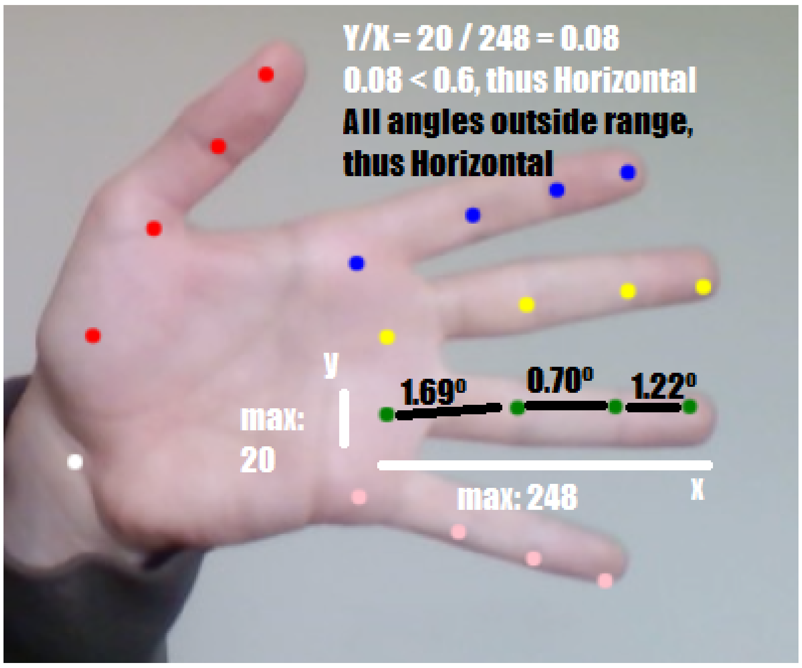


Figure 3.9: Example of Direction calculation in the Ring Finger

The module's second task is to compare the determined estimations with each gesture in the database. Each comparison has an associated confidence and the module saves the gestures whose comparison is higher than the minimum required. The system compares the curl of each detected finger and compares it with the expected curl of the finger of the gesture that is being compared, updating the confidence. Afterwards, it does the same for the orientations, always updating the end confidence.

Using the list of gestures with the minimum confidence, the system concludes that the detected gesture is the one with the highest confidence.

3.2.3 Action Correspondence

This module's function is to check the requirements to send an action to the Scene Processing module by receiving the gesture with the highest confidence that was processed in the previous module.

If the gesture received corresponds to a Raised Hand, then the system ceases all actions, preventing any occurring movement in the virtual scene. To send any other actions, the user must

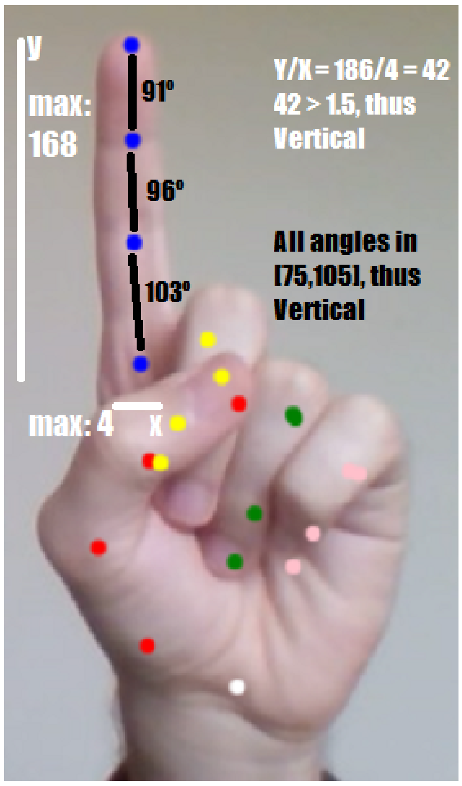


Figure 3.10: Example of Direction calculation in the Index Finger

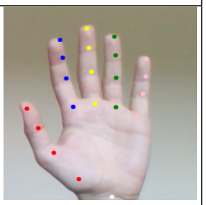
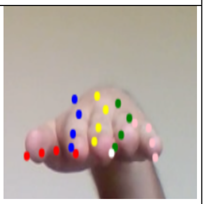
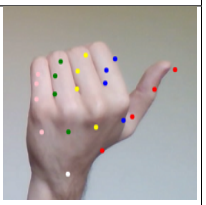
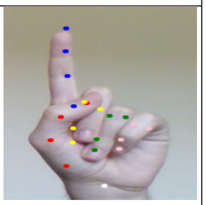
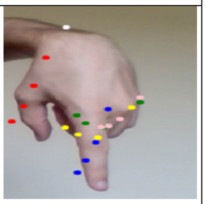
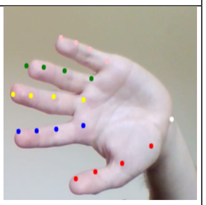
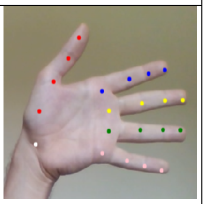
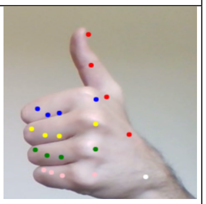
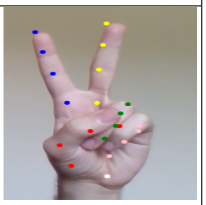
make a Raised Hand gesture followed by the corresponding gesture. For example, if the user wishes to move up in the scene, they must gesticulate a Raised Open Hand followed by a gesture with the Index Finger pointing up, while, the remaining fingers are closed.

In order to have an accurate perception of realistic gestures, some projects were analysed. The authors of these projects developed their own gestures based on studies and tests. Ideum developed a Gesture Markup Language [60] to use on their products. They detail some of the gestures that were defined into their systems and, despite being a markup language, one can extract information from the website to use for this dissertation. In addition, Lee and Sohn worked on interfaces for navigation of 3D maps, using a VR Headset and a Microsoft Kinect [61]. The user could either navigate with gestures that simulate a Bird or Superman. Despite the full-body track, some gestures can derive from their proposal, using only the hands to interact.

The proposed Gesture Dataset is presented in Table 3.1, which also includes the message associated with the gesture and the correspondent action that the virtual scene processes. All these gestures were analysed to match natural and regular poses, as well as making sure that when the user performs a gesture, it won't be mistaken with another.

After verifying the existence of the action corresponding to the gestures, the module sends a message to the Scene Processing module with that action as content.

Table 3.1: Actions and Gestures Table

Message	**Action**	**Gesture**
stop	Stops Any Action	
forward	Moves Forward	
backward	Moves Backward	
up	Moves Up	
down	Moves Down	
left	Rotates Left	
right	Rotates Right	
changeCamera	Changes Camera View	
interact	Interacts with object in front	

3.2.4 Scene Processing

The Scene Processing Module receives a message from the previous module, which is translated into an action inside the virtual scene. Table 3.1 represents the list of messages and correspondent actions when the user is controlling a Camera inside the scene.

Each action represents a specific type of interaction: the first ones are straightforward and represent movement in the scene; *changeCamera* allows the user to change between the existing array of virtual cameras inside the 3D Scene and *Interact* allows interaction with the object in front of the controlled camera. These objects may contain information and/or a camera. If the object contains both, the user will be given a choice between the two options, as well as exiting the interaction, as seen in figure 3.11. To choose, the user must make the gesture *Up* or *Down*, which selects the upper and lower choice, respectively. If the user makes the *Interact* gesture, the prompt will exit.

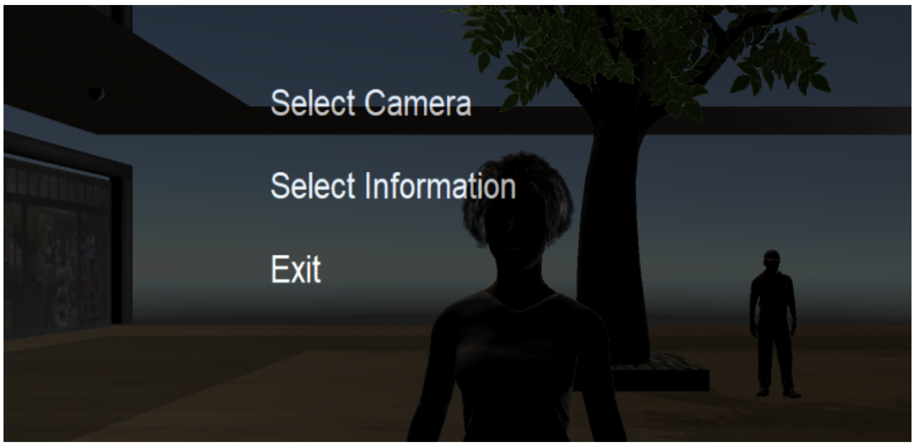


Figure 3.11: User Interface of Decision

If the user chooses *Select Camera*, the user will view the 3D environment from the perspective of the object. During this mode, the user can look to the left and right, as well as up and down, using the *Left, Right, Up and Down* gestures, respectively. By *Interacting*, this mode will end, shifting to the camera scene.

If the user chooses *Select Information*, the system will present information from that object, as seen in Fig. 3.12. If the object contains more information than the ones presented on the screen, the user can use the *Up and Down* gestures to scroll between them, which lists different data, suggested by the Up and Down Arrows. By *Interacting*, this information will exit, going back to the Decision prompt.

3.2.5 Scene Synthesis

This module represents the end result of the system: a synthesized 3D virtual scene, based on a real life environment (in this example, a shopping mall), with built-in mechanics interactions via gestures, to ease the user interface.

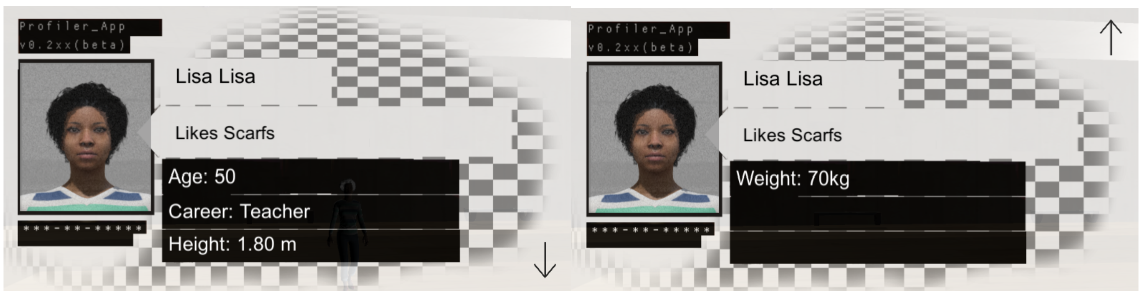


Figure 3.12: Information UI

Initially, some tests were made to identify the best Game Engine to use, between Unity and Unreal Engine 4. Besides this, throughout development, the system went through various changes in order to contain an efficient method of gesture classification and pose estimation, while testing available projects that could further close the gap between gesturing and interacting with a virtual environment.

The chosen Game Engine was Unity, because it's much lighter than Unreal Engine and it facilitates any development that isn't game related, in this case, reading *JSON* files which include the description of objects. Moroever, due to the immense content from the Asset Store, which is community based, a SocketIO asset was found that eased the communication with the Gesture Recognition and Interpretation component.

The created 3D scene was based on a shopping centre, using Unity's geometric objects to create the foundation of the environment and free 3D models to enrich it.

One of the studied projects was **Project Prague** [62], an SDK that allows users to control and interact with technologies through hand gestures, developed by Microsoft. The test involved using the SDK in Unity, which implied detecting and classifying gestures, as well as synthesising the 3D scene in the same application. To run it, one needs a supported depth camera, therefore a Kinect for Windows version 2 was used. A simple interaction was experimented, using one gesture to create 3D objects and another one to delete them. The project has promising functionalities, with the limitation of only supporting three depth cameras, at the time of writing. Another studied project was **Handpose** [63], a Python program capable of detecting and classifying hand poses using deep learning techniques. Its Neural Network is able to detect previously trained poses, by recording them and then training the network. Some work was developed for this project but some difficulties arose for being very sensitive to the position and angle of the hand, leading to wrong predictions. Due to these constraints, the project was later dropped, however some work and concepts were reused for the final system. **Fingerpose** [64] was the last studied project as well as the final major change to the system. It requires to open the browser to execute the Pose Classifier. This quickly resulted in a limitation to the system, therefore, a solution to transform the program into a desktop application was idealized. The first attempt was to convert the project into a *Progressive Web App* [65], or PWA, which is built and enhanced with modern APIs to deliver native-like capabilities, reliability, and installability. However, at its heart, it's still a web app:

it is useful for websites, because it looks exactly like a website installed on the desktop, which may include browser plugins and logged in accounts, but not appropriate for this dissertation's context. The second attempt proved more fruitful: converting the system into a desktop app using *Electron* [66]. Not only it allowed for an improved desktop app, without plugins nor any browser account information, but it also fixed communications with Unity via Socket.IO [67].

3.3 Implementation

This section is utilized to detail the system's implementation. Firstly, the used tools for developments are indicated and explained, followed by system definitions. Afterwards, the system's modules are detailed as well as the descriptions used for the objects inside the 3D scene.

3.3.1 Development Tools

To capture user input, the system was based on the project developed by Github user *andy-potato* [64], a browser based gesture classifier, which makes use of Tensorflow.js Handpose [50] models to detect the hand and return the pose, while using its own pose classifier.

Since the project needed browser usage, the framework *Electron* [66] was a necessary step, because it is able to construct a desktop application based on a web app, thus removing the necessity of the browser. *Node* and *NPM* [68] were required, not only to assemble the desktop application in Electron, but also build any existing changes to the gestures in the local dataset.

To create the 3D environment, Unity [48] was the chosen Game Engine. The thesis required communication between the Pose Classifier and the 3D Environment, thus, Socket IO [67] was utilized on the Pose Classifier side, with the Unity side using an Asset based Socket IO [69] from the store.

3.3.2 Definitions

This section outlines terms that were defined and implemented throughout development, whether in code or theory to apply to the dissertation. These must be defined to fully understand how the system works.

Finger Descriptions

A **Finger** is defined by a value that indicates which finger it represents: 0 is Thumb, 1 is Index, 2 is middle, 3 is Ring and 4 is Pinky. It also holds the mapping of joints based on the 21 keypoints. For example, if the finger is a Thumb, this mapping will include the following pair of points: (0,1), (1,2), (2,3) and (3,4). Fig. 3.6 from the previous chapter contains the indexes corresponding to each point.

The **Curl** of a Finger is defined by a value that indicates if it's Not Curled(0), Half Curled(1) or Fully Curled(2).

The **Direction** of the Finger can be Vertical, Horizontal or Diagonal, with its **Orientation** being Up or Down for the first, Left or Right for the second, and UpRight, UpLeft, DownRight, DownLeft for the latter.

Head Rotation

When developing the perspective of a person inside the 3D scene, the human neck and its limiting rotations were analysed to include a more realistic perception of reality.

The human neck has three degrees of freedom: pitch, roll and yaw, as seen in Fig. 3.13. It has a 130-degree range of movement for pitch motion, a 45-degree range of movement for unilateral roll motion, and an 80-degree range of movement for unilateral yaw motion [70]. Thus, in order to satisfy real life conditions, when in a Person Camera Perspective, the user can only rotate a maximum of 80 degrees, for each side, for yaw motion, and a maximum of 50 degrees, for pitch motion.

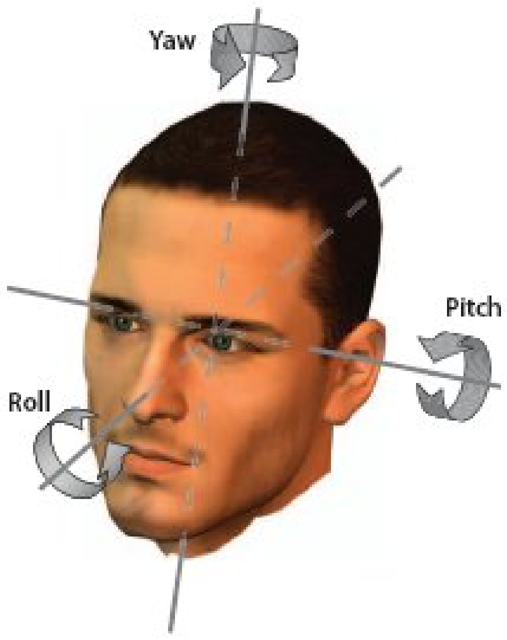


Figure 3.13: Human Head Degrees of Freedom (adapted from Luzardo et al. [71])

3.3.3 Modules

In this subsection, details regarding the main modules are presented, as well as some faced limitations and their solutions.

Input Capture

This module makes use of Tensorflow.js, a Javascript port of the library. The Handpose model, necessary to detect the palm and return the keypoints for classification, is imported via script tags and then loaded with Javascript. To start predictions, a function is called with the frame of the video as an argument, which must be an HTML video element. This imposes a limitation: the function required an argument only obtainable in a browser. To fix this, *Electron* was used to encapsulate the browser into a desktop application.

Gesture Classification

This module makes use of andypotato's library from his project [64], which uses the 21 keypoints predicted by the model, with a minimum confidence of 7.5, to verify if the gesture made by the user exists in the set of predefined gestures.

Action Correspondence

To execute an action, the user must make a combination of two gestures. Thus, this module stores the two most recent gestures made by the user in an array of size 2. It also contains a map which associates the gestures stored in the array with the corresponding action.

Every time the system detects a new gesture, the gesture in the 1st position of the array replaces the element in the 2nd position and the new gesture is stored in the 1st position of the array. After updating the array, the elements in it are merged in a string, with a ":" delimiter in between them. This string serves as a value for the mentioned map: if there's a key with that value, then an action exists and it is sent to the *Scene Processing Module*. For example, if the array contains both "raised_hand" and "hand_back", then they're merged in a "raised_hand:hand_back" string, corresponding to the "backward" action. Whether an action exists or not, the array is cleaned afterwards. The method explored to send messages is defined in the next section.

Messaging through Sockets

In order to establish communication, both Unity and Javascript open a socket on the same port and form a connection. The latter then sends a message with the tag "action". The module *Scene Processing* begins with the reception of the message and processing it.

Scene Processing

Since the gestures are reused between Scene Movement or UI Interactions, after receiving an action, Unity verifies the current context and which action to execute. For example, if the user receives the "Going Up" action, the following options are available:

- Go Up on the scene;
- Look Up if viewing the scene through an object's perspective;

- Select the first option in the presented Decision UI;
- Scroll up if the presented UI is an Information UI.

The system begins by verifying if no Scene Camera is active, which hints to an object perspective situation. If it isn't active, then the prompted action, in this case, Going Up, makes the camera look up in the object perspective. However, if a Camera Scene is active, then the system verifies if any UI display is active and, if so, the correspondent UI action is called. If it's not, then it's a Movement Action, which makes the current Scene Camera go up.

3.3.4 Descriptions Files

Any object inside the virtual 3D scene may include indications that describe information or a camera for perspective. These are files in *JSON* format which are read and inserted prior to booting up the scene. They must also be in a folder whose name is equal to the object's name inside the scene. For example, if an object is called *"Vase"*, then, to present any information about it, a folder called "Vase" must exist, with an "information.json" file inside.

Information files and content

Files that contain information are called "information.json". The existence of the file isn't mandatory and, as such, its content is optional. An example can be seen in Listing 3.1. The following is what the file should contain:

- Name - The Name of the Object;
- Description - A simple Description of the Object;
- Photo - A Picture of the Object. This photo must be named "portrait", thus this attribute's value should default to "portrait".
- List Information - A List that details extra information about the object. Each element contains a Type and a Value.

```
{
    "Name": "Vase of Flowers",
    "Description": "Includes Pink flowers",
    "ListInformation":
        [
            {
                "Type":"Species",
                "Value":"Begonias"
            },
            {
                    "Type":"Origin",
                    "Value":"China"
                }
            ],
        "Photo": "portrait"
}
```

Listing 3.1: Example of JSON file of Information

Camera

Files that contain a camera perspective are called "camera.json". The existence of the file isn't mandatory and, as such, its content is optional. Inside, the item "CameraTransformations" should exist, which represents the list of necessary geometric transformations for the camera to be correctly positioned. Each transformation contains a Type, which must be Position, Rotation or Scale, and a Vector. An example can be seen in Listing 3.2.

```
{
    "CameraTransformations": [
        {
          "Type": "Position",
          "Vector": [ 1, 4, 0 ]
        },
        {
          "Type": "Rotation",
          "Vector": [ 0, 90, 0 ]
        }
    ]
}
```

Listing 3.2: Example of JSON file of Camera

Chapter 4

Experiments and Results

This chapter presents experiments and steps made in order to evaluate the system, before getting to the final version, as well as results from development and encountered limitations. To make sure that the system functions properly, an evaluation was made, using volunteers to test it. For this end, a virtual environment, based on a shopping mall, was created, which includes various objects such as benches, stores and vases, as well as animated people. Figs. 4.1 to 4.2 show the developed scene.

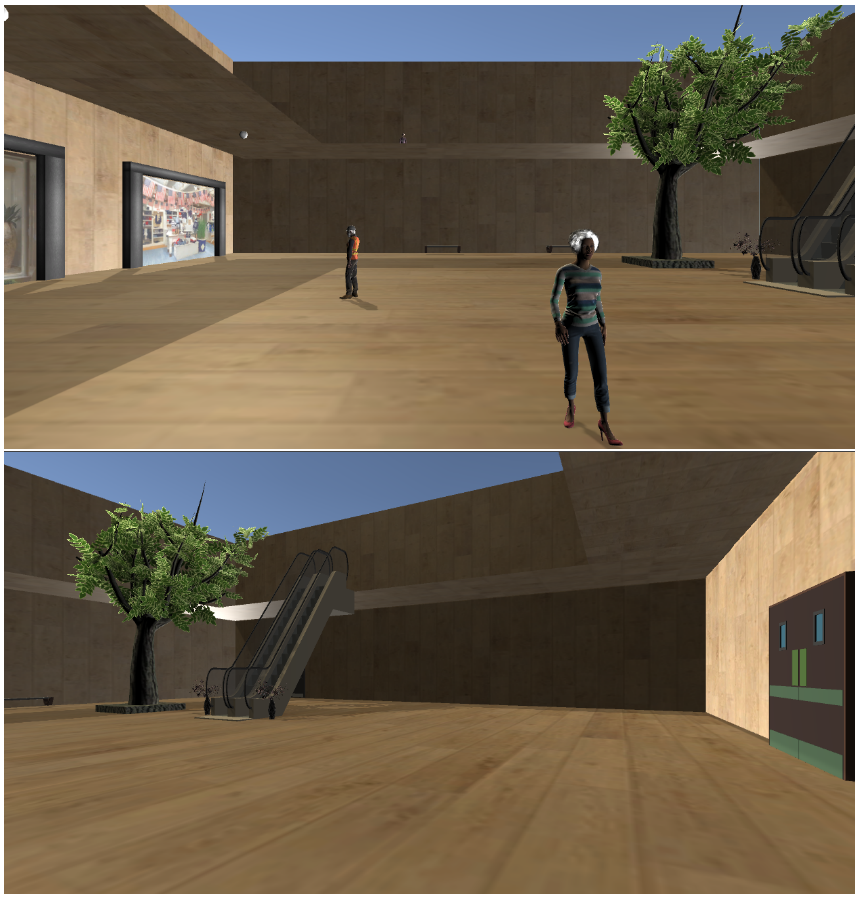


Figure 4.1: Scene visualized through the perspective of the first two cameras

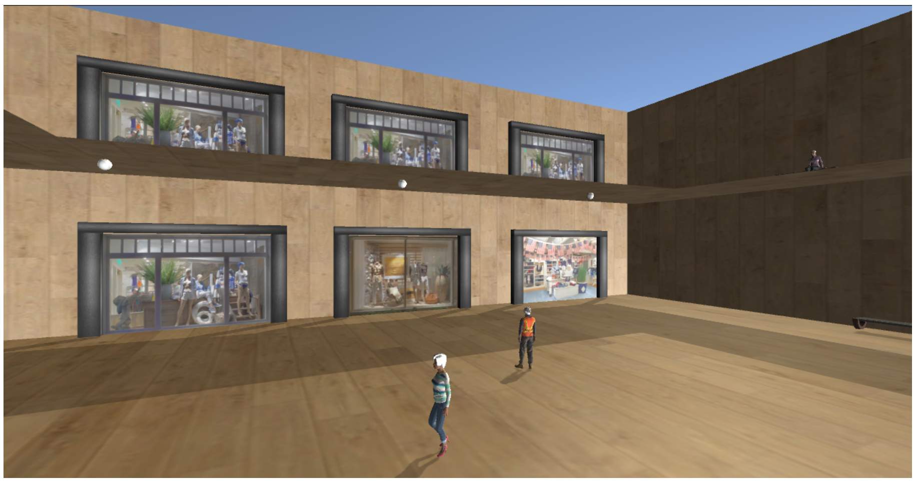


Figure 4.2: Viewing of the stores and some of the animated people

4.1 Experiments

Since Unity and Javascript are different software, communication between them may cause some input lag, resulting in a less immersive experience. A few tests were made to verify any possible bottleneck since the user gesticulates up until an action is presented in the scene. In both tests, the gesture "Raised Hand" had been previously done.

In the first test, the gesture was "Raised Hand Right". It took 2 milliseconds to be detected and verified if it's correct. The next step in the system was to send information to Unity. After 3 milliseconds, the Game Engine received the information and took an extra 2 milliseconds to execute the action, totaling the time between making a Gesture and seeing an action to about 7 milliseconds.

In the second test, the gesture was "Raised Hand Left". The system also needed 2 milliseconds for the Gesture to be detected and verified if it's correct. After 6 milliseconds, Unity received the information and also required 2 milliseconds to start executing, totaling the time to about 10 milliseconds.

This signifies the information sent via socket is what takes longer to execute, with around 2 milliseconds on both Unity and Javascript to perform internal actions. However, the presented times aren't enough to decrease any user experience.

Throughout development, difficulties arose when complex backgrounds were involved while capturing the gestures. If any object or furniture whose color resembles the human skin is present during the usage of the system, they may be incorrectly identified as the user's hand. To overcome this problem, the background should be as clean as possible, i.e, with a reduced number of objects, specially ones with a similar color to the human skin, and without strong lighting.

4.2 Evaluation

In order to test the system with end users, some volunteers were asked to experiment it, by following indications and afterwards filling a form. Users were initially presented to the system with explanations regarding how it works and were shown the gestures as well as how to make them and their result as actions inside the 3D scene.

Afterward, users were required to test the system using the gestures, by following a guide. They began by going forward and backward, followed by looking left and right. Subsequently, then went up, looked left, went down and looked right, in this order. These initial trials were so the users could understand how the gestures work and get used to the system's navigation. Secondly, they were requested to position themselves in front of an object and interact with it. If the object had a camera associated, as well as information, users were given the option to choose between one another. Thirdly, the users were given some minutes to freely roam through the 3D environment.

Finally, they were required to do the same but use the keyboard instead of gestures.

During the tests, users were required to fill a form, which quantified the intuitiveness, responsiveness and tiredness of each gesture, with the answer's value ranging between 1 and 5. An example can be seen in Fig. 4.3.

4.3 Results

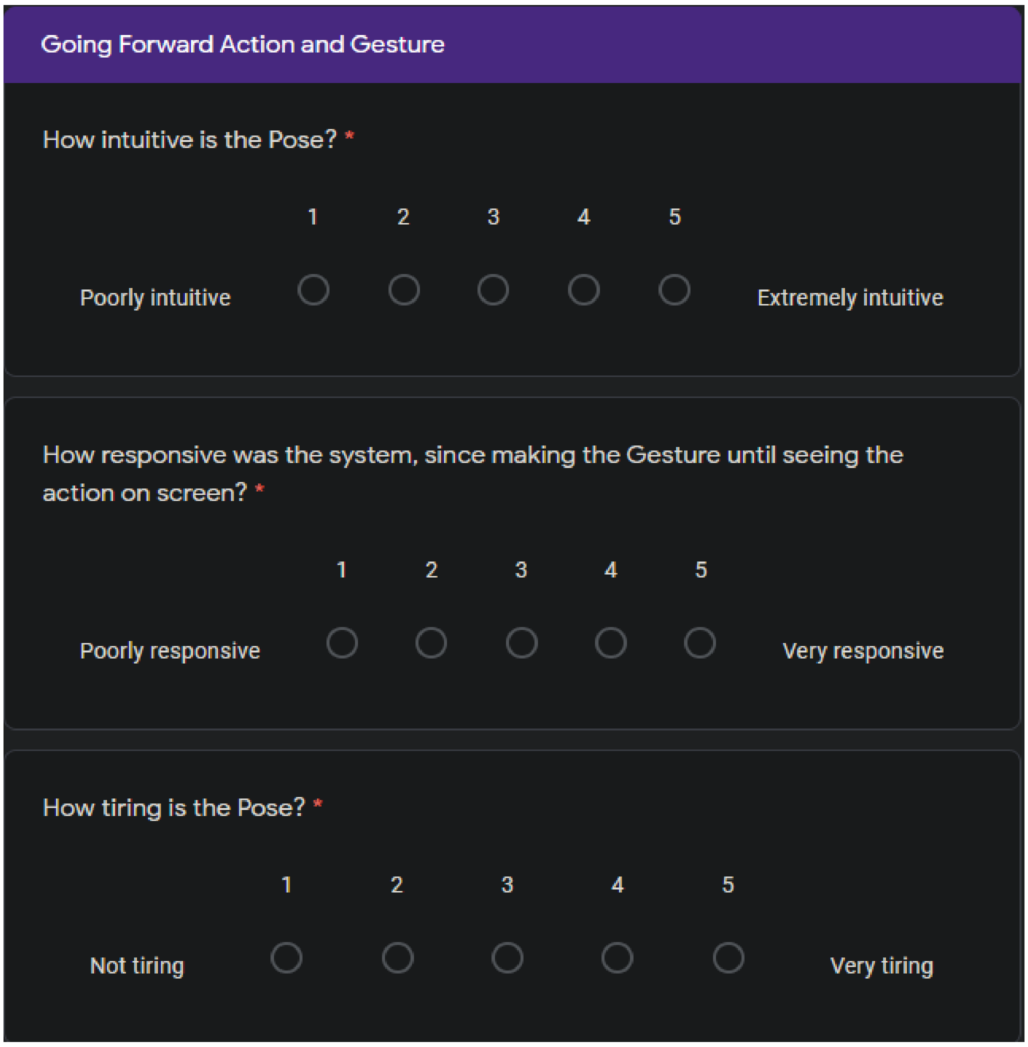
Eleven volunteers, eight of them with ages between 21 and 23, and three with ages between 59 and 61, were asked to test the system, with a form being filled as they followed a guide. For every gesture and action, they were asked about its intuitiveness, responsiveness and tiredness.

The first question ranged between Poorly(1), A Little(2), Relatively(3), Very(4) and Extremely(5) Intuitive. The second question ranged between Poorly(1), A Little(2), Relatively(3), Quite(4) and Very(5) Responsive. The last question ranged between Not(1), A Little(2), Relatively(3), Quite(4) and Very(5) Tiring.

All figures are available on Appendix D.

Going Forward Action and Gesture

As seen in Fig. D.1, for the Forward gesture, 54.5% considered the gesture Extremely Intuitive and 36.4% Very Intuitive, with 9.1% considering Relatively Intuitive. 36.4% believed the system to be Very Responsive, while 45.5% believed to be Significantly Responsive, with 18.2% considering Relatively Responsive. Finally, 63.6% thought the gesture to be Not Tiring, while 18.2% considered a Little Tiring and the remaining 18.2% believed it to be Relatively Tiring.



Going Forward Action and Gesture

How intuitive is the Pose? *

	1	2	3	4	5	
Poorly intuitive	○	○	○	○	○	Extremely intuitive

How responsive was the system, since making the Gesture until seeing the action on screen? *

	1	2	3	4	5	
Poorly responsive	○	○	○	○	○	Very responsive

How tiring is the Pose? *

	1	2	3	4	5	
Not tiring	○	○	○	○	○	Very tiring

Figure 4.3: Form example filled by volunteers

Going Backward Action and Gesture

As seen in Fig. D.2, for the Backward gesture, 45.5% considered the gesture Extremely Intuitive and 36.4% Very Intuitive, with 9.1% considering Relatively Intuitive. 36.4% believed the system to be Very Responsive, while 54.5% believed to be Significantly Responsive, with 9.1% considering Relatively Responsive. Finally, 46.5% thought the gesture to be Not Tiring, while 36.4% considered a Little Tiring. 9.1% believed it to be Relatively Tiring and the remaining 9.1% considered it to be Quite Tiring.

Looking Left Action and Gesture

As seen in Fig. D.3, for the Looking Left gesture, 81.8% considered the gesture Extremely Intuitive and 18.2% Very Intuitive. 72.7% believed the system to be Very Responsive, while 27.3%

believed to be Significantly Responsive. Finally, 81.8% thought the gesture to be Not Tiring, while 9.1% considered a Little Tiring and the remaining 9.1% believed it to be Relatively Tiring.

Looking Right Action and Gesture

As seen in Fig. D.4, for the Looking Right gesture, 81.8% considered the gesture Extremely Intuitive and 18.2% Very Intuitive. 72.7% believed the system to be Very Responsive, while 27.3% believed to be Significantly Responsive. Finally, 81.8% thought the gesture to be Not Tiring, while 9.1% considered a Little Tiring and the remaining 9.1% believed it to be Relatively Tiring.

Going Up Action and Gesture

As seen in Fig. D.5, for the Going Up gesture, 81.8% considered the gesture Extremely Intuitive and 18.2% Very Intuitive. 81.8% believed the system to be Very Responsive, while 18.2% believed to be Significantly Responsive. Finally, 63.6% thought the gesture to be Not Tiring, while the remaining 36.4% considered it a Little Tiring.

Going Down Action and Gesture

As seen in Fig. D.6, for the Going Down gesture, 72.7% considered the gesture Extremely Intuitive and 27.3% Very Intuitive. 45.5% believed the system to be Very Responsive, while another 45.5% believed to be Significantly Responsive, with 9.1% considering Relatively Responsive. Finally, 27.3% thought the gesture to be Not Tiring, while 63.6% considered a Little Tiring and the remaining 9.1% believed it to be Relatively Tiring.

Interacting Action and Gesture

As seen in Fig. D.7, for the Interacting gesture, 54.5% considered the gesture Extremely Intuitive and 45.5% Very Intuitive. 54.5% believed the system to be Very Responsive, while 36.4% believed to be Significantly Responsive, with 9.1% considering Relatively Responsive. Finally, 72.7% thought the gesture to be Not Tiring, while 27.3% considered a Little Tiring.

Free Roam

As seen in Fig. D.8, after freely roaming on the scene, 45.5% considered the mixture of gestures Extremely Intuitive and another 45.5% Very Intuitive, with 9.1% considering Relatively Intuitive. 27.3% believed the system to be Very Responsive, while 54.5% believed to be Significantly Responsive, with 18.2% considering Relatively Responsive. Finally, 36.4% thought the navigation to be Not Tiring, while the remaining 63.6% considered it a Little Tiring.

4.4 Observations

From this analysis, it was observed that the system had trouble detecting the Forward gesture for some volunteers. Furthermore, the Interact Gesture was often mistaken with the Going Up gesture, which lead to some difficulty to interact with objects as the virtual camera kept going up rather than the system presenting the Decision Prompt. Some also commented that the Going Backward gesture was a bit confusing and revealed hard to perform. It was also noticeable that most of the delays that the users experienced were related to the detection of the pose they were gesticulating.

Nonetheless, the volunteers had an overall good experience with the system, while expressing desire to extend the navigation in the 3D scene and curiosity to interact with the objects.

Chapter 5

Conclusions and Future Work

Invasive interaction mechanisms restrain user interactions with three dimensional environments and limit their possibilities. For example, the mouse and keyboard are restricted to a surface, such as a table. Moreover, these environments are complex and creators usually build them without any kind of description which can speed up the process. The usage of non-invasive interaction mechanisms and the adoption of descriptive scenes permit a more accessible interaction with the environment of the scene, since the first doesn't restrict the user's freedom and the latter allows the creation of interaction details from within the scene.

The purpose of this Dissertation was the creation of a system that could enable this; ergo, a methodology was proposed which unified both interaction mechanisms and descriptive scenes. The system is able to recognize and interpret gestures to interact with the scene: to capture these gestures, an RGB video camera was utilized and a Game Engine - Unity - was used to generate the scene's content and its descriptions as well as receive the actions from the gestures and integrate the interactions.

The developed system is able to provide a less restrictive mechanism to interact with and visualise a 3D environment. It is also capable of recognizing a specific set of gestures, such as a vertical raised open hand that represents stopping and a horizontal open hand pointing to the left that represents going left, with captured visual information in less restricted scenarios. These gestures enable an interactive navigation throughout the virtual scene, which contains descriptions regarding many of the existing objects.

The demonstrated detection models require the background to be as simple as possible to avoid fake detections and diminish the user experience, i.e, a white background with good lighting should present the best results, juxtaposed with backgrounds that contain objects whose colour is similar to the skin's and a stronger light. Additionally, the detection models aren't one hundred percent efficient, therefore wrong detections may occur during usage. Moreover, the palm detection is currently limited to one hand at a time and the distance to the camera should be a maximum of 2 meters. It is, however, a system which offers a more relaxed interaction with three dimensional environments, by removing the limits of a surface for a keyboard and mouse combination, providing an improved user experience. Allied with the existence of object descriptions of the scene, it creates a relaxed way for users to navigate and interact with the scene and its content.

Although the proposed system successfully allows an improved user experience when navigating in a virtual environment, further improvements can be proposed. For instance, integration with VR could have a positive impact on the experience. VR is a field that has been in development since the 20th Century, with its main uses being medical and flight simulations, industry design and military training purposes, as seen by Cipresso et al. [72]. It provides a much more immersive experience, with the capability of looking around with the headset while managing scene movements and interactions with VR-compatible controllers. Future work may include the integration of this technology in the system, where users would be capable of either using VR controllers to navigate and interact or fully depend on the developed Gesture Recognition system, with the latter being somewhat less restricted. Besides VR, this work could be adapted to recognize facial expressions or the human voice to include more accessibility options to users with some impairment. Revising some poses could correct inaccurate detections, as seen in the feedback provided by the volunteers, and boost the experience. For example, the Interact Gesture could be changed so that the Index and Pinky Fingers were stretched, while the remaining fingers would be curled. In addition to these levels, further updates of the Handpose model should also allow for an increased detection fidelity, lowering the rate of false detections of the user's hand.

Appendix A

Versions of development tools

The used Tensorflow.js version was 2.0, updated in May 2020. Unity's version was 2019.3.6f1 and should remain the same to avoid possible update conflicts. *Node* version was 12.16.2 and *NPM* was 6.14.4. Finally, the *Electron* version was 80.0.3987.163.

Appendix B

Modules Code

This appendix contains part of the relevant code developed for the system. The first three modules contain code in Javascript while the fourth contains code in C#, due to being developed in Unity. The last section of this appendix contains Javascript and C# code that handles the socket communication.

B.1 Input Capture

```
const model = await handpose.load(maxContinuousChecks = 200);
const predictions = await model.estimateHands(video, true);
```

Listing B.1: Loading and Predicing Model

The first line loads the Handpose Model, which includes both the *Palm* and *Hand Landmark* detectors. The argument *maxContinuousChecks* represents how many frames to go without running the bounding box detector.

On the second line, the model predicts and returns the 21 keypoints, using the current frame in the *frame* argument. The second argument is used to flip the hand keypoints horizontally. These keypoints are used in the following module to classify the gesture.

B.2 Gesture Classification

```
const estimGesture = GE.estimate(predictions[i].landmarks, minConfidence, false);
if (estimGesture.length > 0) {
    // find gesture with highest confidence
    let result = estimGesture.reduce((p, c) => {
        return (p.confidence > c.confidence) ? p : c;
    });
    checkDynamicGestures(result.name);
}
```

Listing B.2: Classifying Gestures

On Line 1, the module uses the 21 keypoints predicted by the model, with a minimum confidence of 7.5, to verify if the gesture made by the user exists in the set of predefined gestures, with the third argument being used for debug purposes only.

The variable *estimGestures* may contain more than one gesture. Thus, on the following line, if the module identified several gestures, it finds the one with the highest confidence and stores that gesture. Its name is then used on the function *checkDynamicGestures*, which is vital for the Action Correspondence module.

B.3 Action Correspondence

```

const gestureActions = {
    "raised_hand:hand_front": "forward",
    "raised_hand:hand_back": "backward",
    "raised_hand:raised_hand_left": "left",
    "raised_hand:raised_hand_right": "right",
    "raised_hand:index_up": "up",
    "raised_hand:index_down": "down",
    "raised_hand:thumbs_up": "changeCamera",
    "raised_hand:victory": "interact",
}
async function checkDynamicGestures(gesture) {
    //stops movement in Unity
    if (gesture == 'raised_hand')
        network.sendAction("stop");
    //saving gesture
    if (!gesture_sm.includes(gesture)) {
        if (gesture_sm[0] == '')
            gesture_sm[0] = gesture;
        else {
            if (gesture_sm[1] != '')
                gesture_sm.shift();
            gesture_sm[1] = gesture;
            analyseDynamicGestures();
        }
    }
}
async function analyseDynamicGestures() {
    let dyn = gesture_sm[0] + ":" + gesture_sm[1];
    let val = gestureActions[dyn];
    if (val !== undefined)
        network.sendAction(val);
    emptyGestureArray();
}
```

Listing B.3: Finding corresponding actions

This module sends an action to the Scene Processing module, which is controlled by the Game Engine Unity, that satisfies the following requirement: the first gesture must be a "raised_hand" followed by the desired gesture. This is seen in the map *gestureActions*, on line 1. There's also an array *gesture_sm* which stores the last two gestures made by the user.

The function *checkDynamicGestures* receives the gesture's name and stops any movement in Unity if it's a "raised_hand". Afterwards, it adds the gesture to the array if it isn't there (to avoid repetitions) in the correct position. With the filled array, the module calls *analyseDynamicGestures*. This function creates a string composed by the gestures in the array and delimited by a ":" character and, if it exists in *gestureActions*, then the module sends the message to the Scene Processing module. In the end, the array is emptied.

B.4 Scene Processing

```
public void VerifyAction(string action){
  if (!IsSceneCameraActive())
    VerifyPersonActions(action, PersonCamera.GetComponent<PersonCameraScript>());
  CameraScript script = SceneCameras[currentIndex].GetComponent<CameraScript>();
  if (DecisionDisplay.activeSelf || InformationDisplay.activeSelf)
    UIActions(action, script);
  else
    MovementActions(action, script);

private void VerifyPersonActions(string action, PersonCameraScript script){
  switch(action){
    case "left":
      script.GoLookLeft();
      break;
    case "right":
      script.GoLookRight();
      break;
    case "up":
      script.GoLookUp();
      break;
    case "down":
      script.GoLookDown();
      break;
    case "stop":
      script.Stop();
      break;
    case "interact":
      script.GoInteract();
      break;
    default:
      break;
  }
}
```

```

private void UIActions(string action, CameraScript script){
  switch(action){
    case "interact":
      if (InformationDisplay.activeSelf) {
        InformationDisplay.GetComponent<InformationDisplay>().ClearValues();
        InformationDisplay.SetActive(false);
      }
      else if (DecisionDisplay.activeSelf){
        script.decision = 3;
      }
      break;
    case "down": //Decide Info (Lower choice)
      if (DecisionDisplay.activeSelf)
        script.decision = 1;
      else if (InformationDisplay.activeSelf) {
        InformationDisplay.GetComponent<InformationDisplay>().UpdateInfoIndex(1);
      }

      break;
    case "up": // Decide Camera (Upper choice)
      if (DecisionDisplay.activeSelf)
        script.decision = 2;
      else if (InformationDisplay.activeSelf) {
        InformationDisplay.GetComponent<InformationDisplay>().UpdateInfoIndex(-1)
            ;
      }
      break;
  }
}
private void MovementActions(string action, CameraScript script){
  switch(action){
    case "forward":
      script.GoForward();
      break;
    case "backward":
      script.GoBackward();
      break;
    case "left":
      script.GoLookLeft();
      break;
    case "right":
      script.GoLookRight();
      break;
    case "up":
      script.GoUp();
      break;
    case "down":
      script.GoDown();
      break;
    case "stop":
      script.Stop();
      break;
    case "interact":
      script.GoInteract();
      break;
    case "changeCamera":
      if (IsSceneCameraActive())
        ChangeCameras();
      break;
    default:
      break;
    }
  }
}
```

Listing B.4: Verifying actions in Unity

This code is inside the PlayerControllerScript. Function *VerifyAction* is called after receiving an action (see section B.5) and checks the conditions to execute an action: if no Scene Camera is active, meaning the user is viewing the 3D environment through the perspective of an object, then it should execute the associated action, on function *VerifyPersonActions*. However, if any of the UI is active, then the user is limited to UI-only actions, on function *UIActions*. Finally, if these restrictions aren't presented, then the module calls *MovementActions* and proceeds to verify the corresponding action.

B.5 Socket Messages

```
1  var io = require('socket.io')(process.env.PORT || 52300)
2  let currentSocket;
3
4  io.on('connection', function(socket){
5      console.log("Connection made");
6      currentSocket = socket;
7  });
8  const sendAction= (action) => {
9      currentSocket.emit("action", {name: action});
10 };
11
12 exports.sendAction = sendAction;
```

Listing B.5: Socket Javascript Side

To send any message from the Javascript side to the Unity side, *Socket.io* is used which opens a port, on line 1, and creates the connection on line 4. Any message is then sent using the function *sendAction*, with "action" as the tag.

```
private void SetupEvents(){
  On("open", (E) =>{
      Debug.Log("Connection made to the server");
  });

  On("action", (E) =>{
    string action = E.data["name"].ToString().Replace("\"", "");
    PlayerController.GetComponent<PlayerControllerScript>().VerifyAction(action);
  });
}
```

Listing B.6: Socket Unity Side

The module Scene Processing, in section B.4, sets up the events in *SetupEvents* to open the connection and receive an action. For the latter, the Module calls the function *VerifyAction* from the PlayerController Script.

Appendix C

3D Scene content

When building the virtual environment, several assets were used to enrich it. This appendix presents the links to all used assets.

C.1 3D Models

Escalator
Lights
Character Models and animations
Vase
Bench
Door
Exit Door

C.2 Textures

Floor Texture
Store Textures
Store Wall Texture

Appendix D

Volunteer Test Results

The following is the results of the forms filled by the volunteers presented in charts.

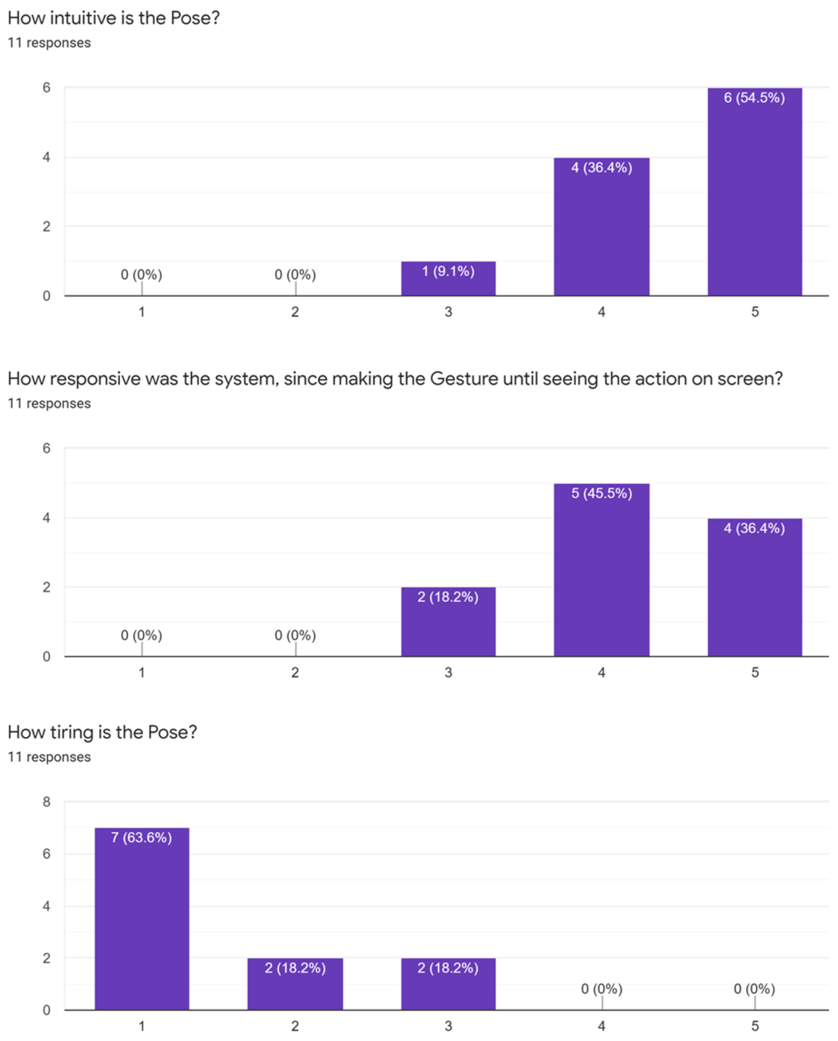


Figure D.1: Results of the Going Forwards question

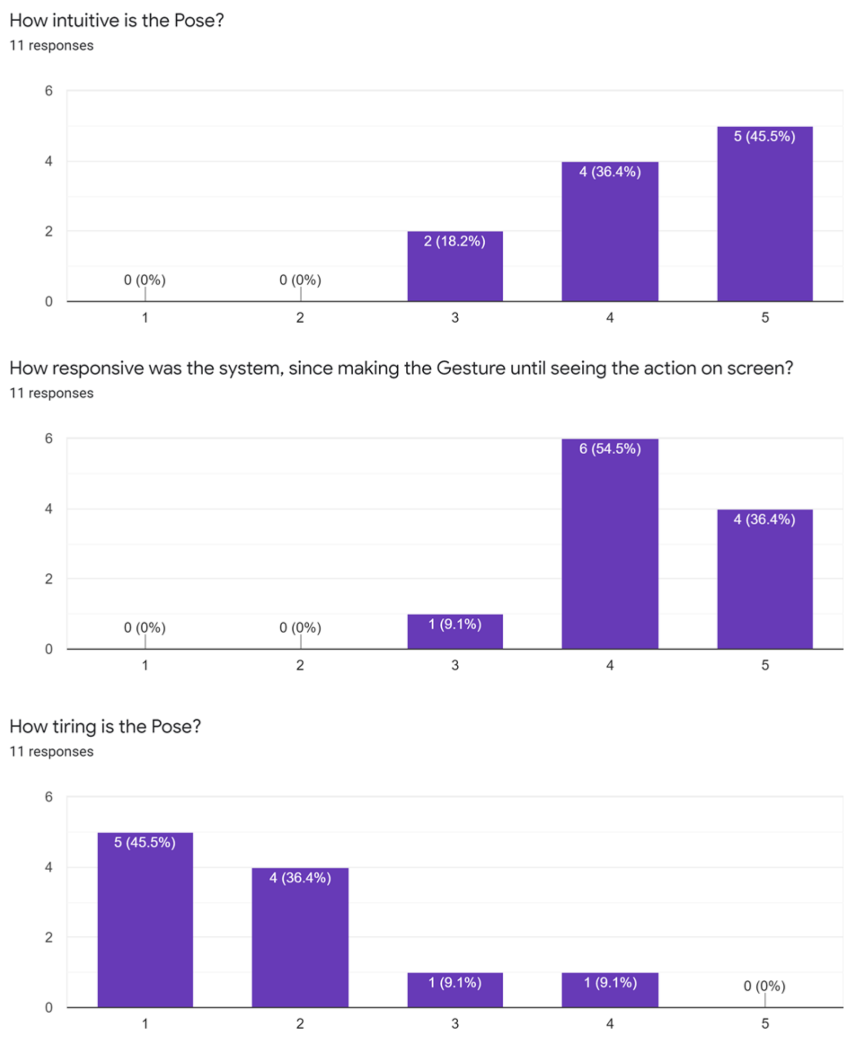


Figure D.2: Results of the Going Backward question

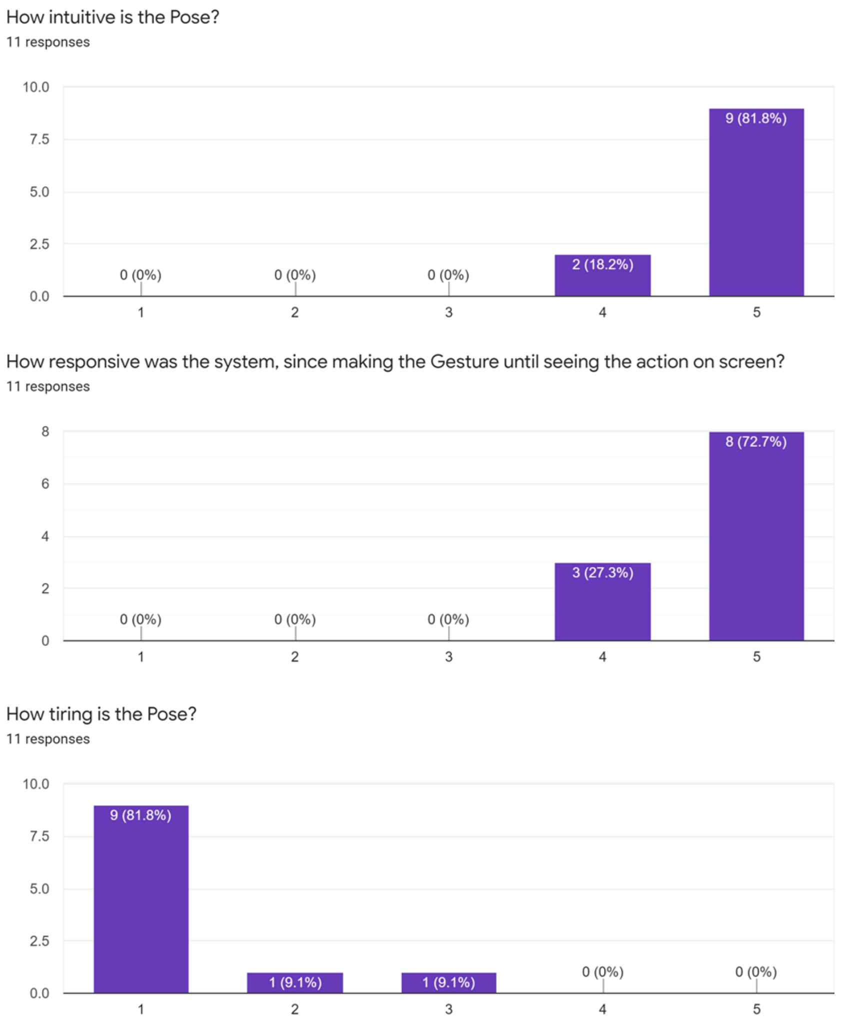


Figure D.3: Results of the Looking Right question

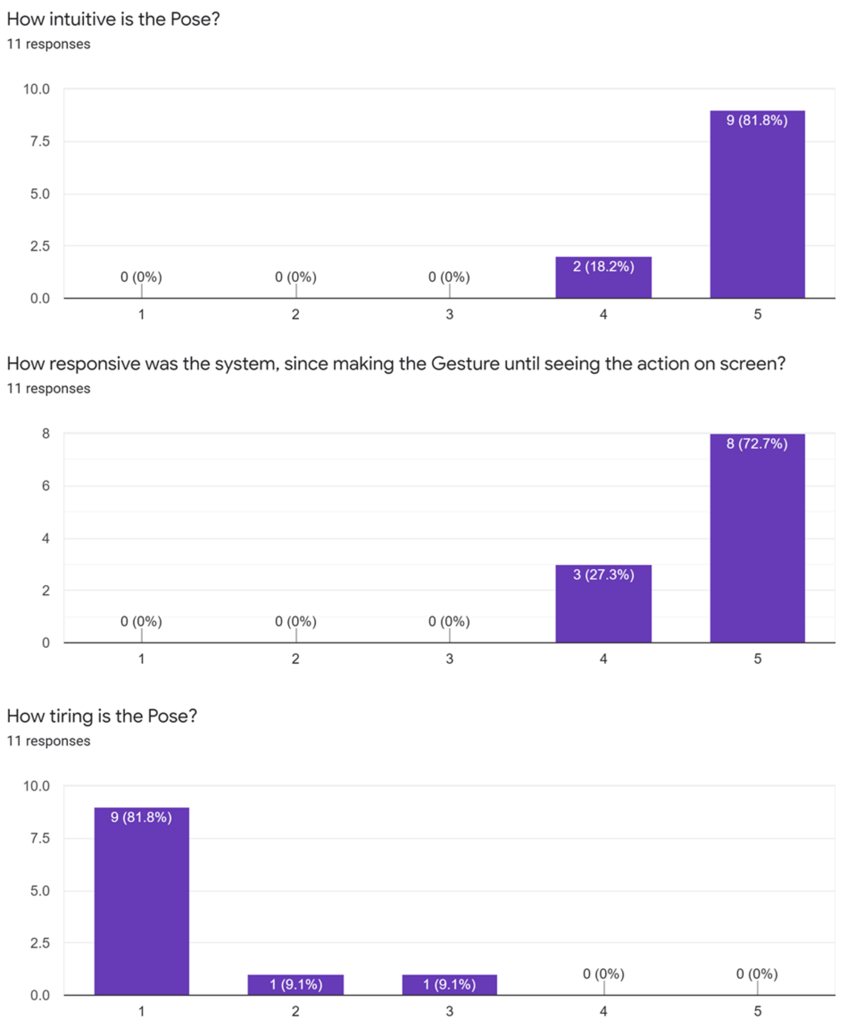


Figure D.4: Results of the Looking Left question

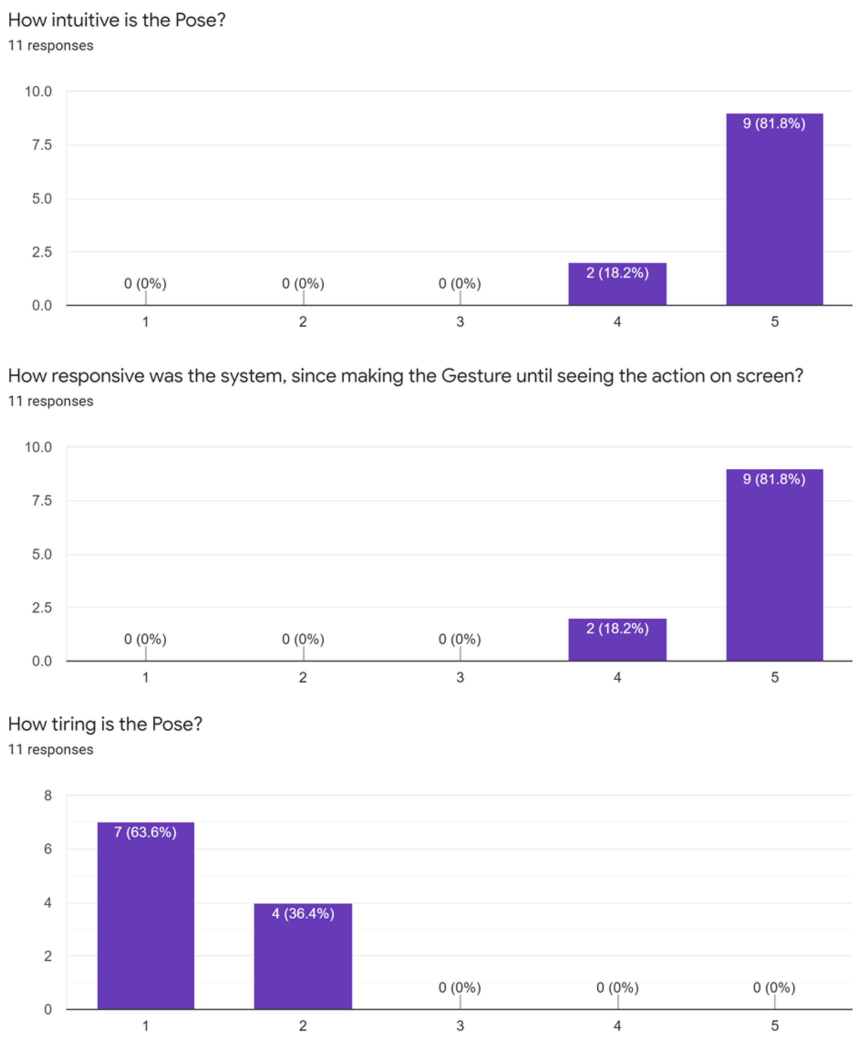


Figure D.5: Results of the Going Up question

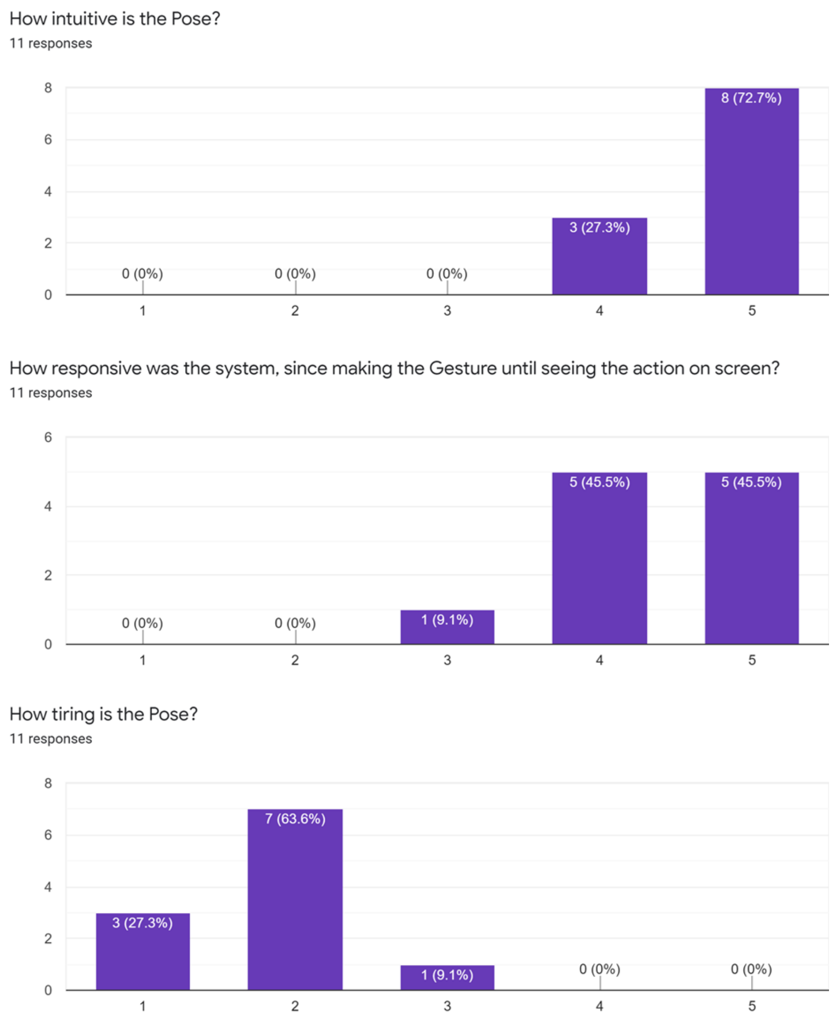


Figure D.6: Results of the Going Down question

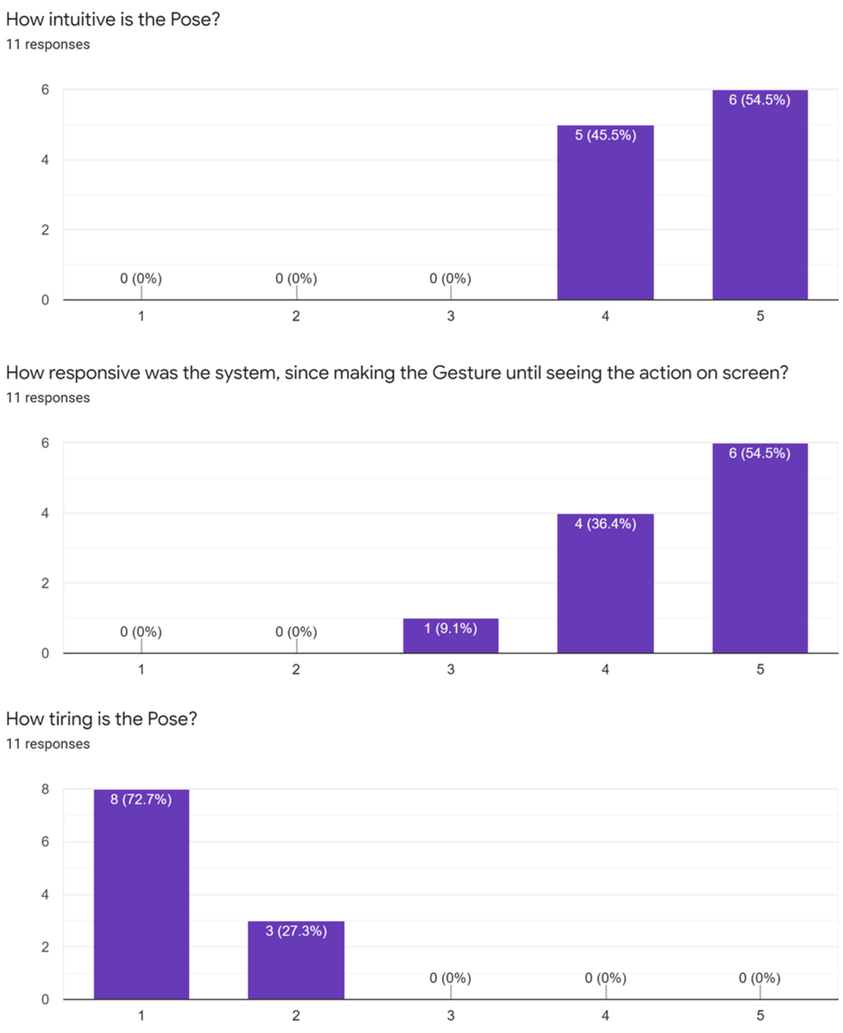


Figure D.7: Results of the Interaction question

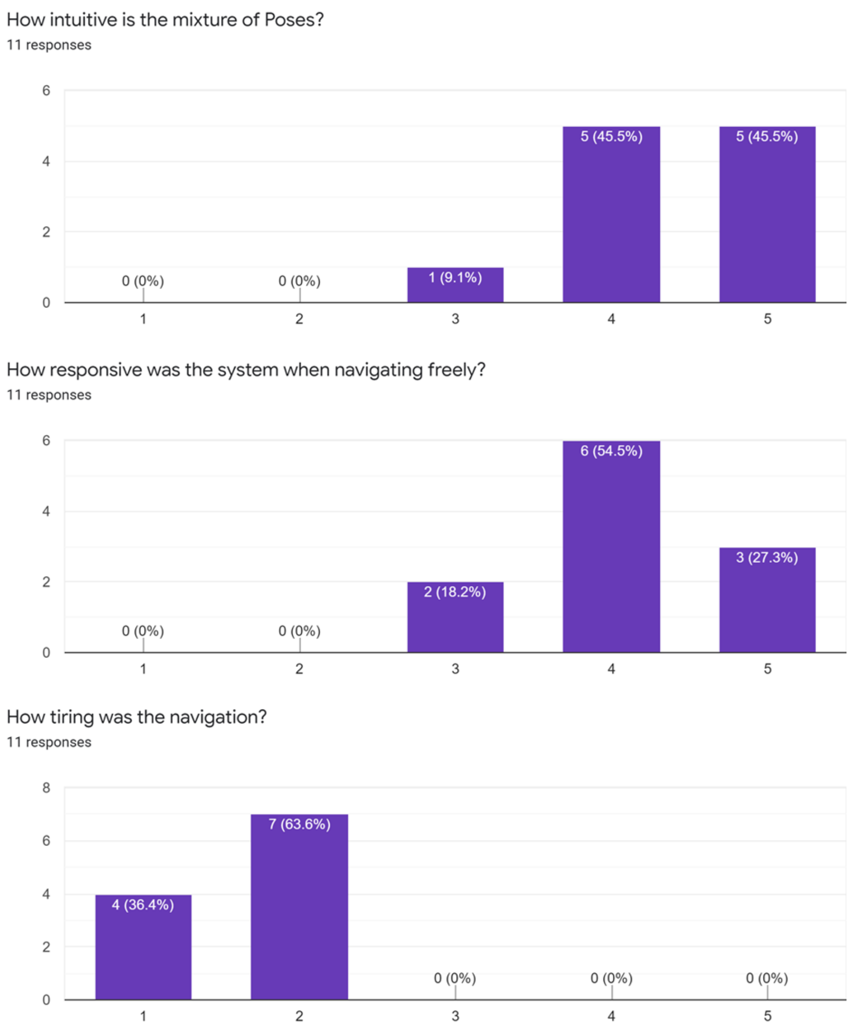


Figure D.8: Results of the Free Roam question

References

[1] G. Geser and F. Niccolucci, "Virtual museums, digital reference collections and e-science environments," *Uncommon Culture*, vol. 3, no. 5/6, pp. 12–37, May 2013. [Online]. Available: https://journals.uic.edu/ojs/index.php/UC/article/view/4714

[2] Google, "Collections - Google Arts & Culture," last accessed 25 June 2020. [Online]. Available: https://artsandculture.google.com/partner?hl=en

[3] E. Horvitz and M. Shwe, "Handsfree decision support: Toward a non-invasive human-computer interface*," *Proceedings / the ... Annual Symposium on Computer Application [sic] in Medical Care. Symposium on Computer Applications in Medical Care*, 01 1995.

[4] Robots.net, "Top 10 VR Applications in Today's," last accessed 13 January 2020. [Online]. Available: https://robots.net/vr-ar/virtual-reality/virtual-reality-applications/

[5] J. L. Maples-Keller, B. E. Bunnell, S.-J. Kim, and B. O. Rothbaum, "The use of virtual reality technology in the treatment of anxiety and other psychiatric disorders," *Harvard review of psychiatry*, vol. 25, no. 3, pp. 103–113, 2017, 28475502[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/28475502

[6] L. Rentzos, C. Vourtsis, D. Mavrikios, and G. Chryssolouris, "Using vr for complex product design," in *Virtual, Augmented and Mixed Reality. Applications of Virtual and Augmented Reality*, R. Shumaker and S. Lackey, Eds. Cham: Springer International Publishing, 2014, pp. 455–464.

[7] Logitech, "Logitech HD Pro Webcam C920," last accessed 29 June 2020. [Online]. Available: https://www.logitech.com/en-us/product/hd-pro-webcam-c920

[8] Microsoft, "Kinect," last accessed 29 June 2020. [Online]. Available: https://developer.microsoft.com/en-us/windows/kinect/

[9] Intel, "Intel Realsense," last accessed 29 June 2020. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html?wapkw=realsense/

[10] D. Regazzoni, G. De Vecchi, and C. Rizzi, "Rgb cams vs rgb-d sensors: Low cost motion capture technologies performances and limitations," *Journal of Manufacturing Systems*, vol. 33, 08 2014.

[11] A. Saxena, M. Sun, and A. Ng, "Make3d: Depth perception from a single still image." vol. 3, 01 2008, pp. 1571–1576.

[12] L. Spinello and K. O. Arras, "People detection in rgb-d data." in *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[13] R. Szeliski, *Computer Vision: Algorithms and Applications*, 1st ed. Berlin, Heidelberg: Springer-Verlag, 2010.

[14] A. O' Riordan, T. Newe, D. Toal, and G. Dooly, "Stereo vision sensing: Review of existing systems," 12 2018.

[15] J.-H. Kim, Z. Teng, and D. Kang, "Real-time multiple plane area detection using a self-organizing map," *Optical Engineering*, vol. 51, pp. 7207–, 01 2012.

[16] K. Liu and N. Kehtarnavaz, "Real-time robust vision-based hand gesture recognition using stereo images," *Journal of Real-Time Image Processing*, vol. 11, 02 2012.

[17] J. S. Sonkusare, N. B. Chopade, R. Sor, and S. L. Tade, "A review on hand gesture recognition system," in *2015 International Conference on Computing Communication Control and Automation*, Feb 2015, pp. 790–794.

[18] S. Mitra and T. Acharya, "Gesture recognition: A survey," *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 37, no. 3, pp. 311–324, 2007.

[19] G. Stockman and L. G. Shapiro, *Computer Vision*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[20] P. de Sousa, "Gesture Recognition for Human-Robot Interaction for Service Robots," 2017.

[21] C.-B. Park and S.-W. Lee, "Real-time 3d pointing gesture recognition for mobile robots with cascade hmm and particle filter," *Image and Vision Computing*, vol. 29, no. 1, pp. 51 – 63, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0262885610001149

[22] T. J. Cerlinca and S. G. Pentiuc, "Robust 3d hand detection for gestures recognition," in *IDC*, 2011.

[23] M. Van den Bergh and L. Van Gool, "Combining rgb and tof cameras for real-time 3d hand gesture interaction," in *2011 IEEE Workshop on Applications of Computer Vision (WACV)*, Jan 2011, pp. 66–72.

[24] D. Victor, "Handtrack: A library for prototyping real-time hand trackinginterfaces using convolutional neural networks," *GitHub repository*, 2017. [Online]. Available: https://github.com/victordibia/handtracking/tree/master/docs/handtrack.pdf

[25] C. Yang, Yujeong Jang, J. Beh, D. Han, and H. Ko, "Gesture recognition using depth-based hand tracking for contactless controller application," in *2012 IEEE International Conference on Consumer Electronics (ICCE)*, Jan 2012, pp. 297–298.

[26] A. Ramey, V. González-Pacheco, and M. A. Salichs, "Integration of a low-cost rgb-d sensor in a social robot for gesture recognition," in *Proceedings of the 6th International Conference on Human-robot Interaction*, ser. HRI '11. New York, NY, USA: ACM, 2011, pp. 229–230. [Online]. Available: http://doi.acm.org/10.1145/1957656.1957745

[27] P. Molchanov, X. Yang, S. Gupta, K. Kim, S. Tyree, and J. Kautz, "Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 4207–4215.

[28] C. Chen, Y. Chen, P. Lee, Y. Tsai, and S. Lei, "Real-time hand tracking on depth images," in *2011 Visual Communications and Image Processing (VCIP)*, Nov 2011, pp. 1–4.

[29] Utraleap, "LeapMotion," 2014, last accessed 16 December 2019. [Online]. Available: https://www.leapmotion.com/

[30] ——, "Leap Motion Controller," 2013, last accessed 13 January 2020. [Online]. Available: https://www.ultraleap.com/product/leap-motion-controller/

[31] D. Jiang, G. Li, Y. Sun, J. Kong, and B. Tao, "Gesture recognition based on skeletonization algorithm and cnn with asl database," *Multimedia Tools and Applications*, vol. 78, no. 21, pp. 29 953–29 970, 2019. [Online]. Available: https://doi.org/10.1007/s11042-018-6748-0

[32] A. Ghotkar, P. Vidap, and K. Deo, "Dynamic hand gesture recognition using hidden markov model by microsoft kinect sensor," *International Journal of Computer Applications*, vol. 150, pp. 5–9, 09 2016.

[33] B. Doosti, "Hand pose estimation: A survey," 03 2019.

[34] A. Sinha, C. Choi, and K. Ramani, "Deephand: Robust hand pose estimation by completing a matrix imputed with deep features," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4150–4158.

[35] S. Baek, K. I. Kim, and T.-K. Kim, "Augmented skeleton space transfer for depth-based hand pose estimation," 2018.

[36] C. Zimmermann and T. Brox, "Learning to estimate 3d hand pose from single rgb images," in *IEEE International Conference on Computer Vision (ICCV)*, 2017, https://arxiv.org/abs/1705.01389. [Online]. Available: https://lmb.informatik.uni-freiburg.de/projects/hand3d/

[37] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," 2015.

[38] T. Simon, H. Joo, I. Matthews, and Y. Sheikh, "Hand keypoint detection in single images using multiview bootstrapping," 2017.

[39] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, p. 381–395, Jun. 1981. [Online]. Available: https://doi.org/10.1145/358669.358692

[40] Triumph, "Blend4Web," 2014, last accessed 15 January 2020. [Online]. Available: https://www.blend4web.com/en/

[41] Crytek, "CryEngine," 2011, last accessed 17 December 2019. [Online]. Available: https://cryengine.com/

[42] Y. Games, "Game Maker," 1999, last accessed 15 January 2020. [Online]. Available: https://www.yoyogames.com/gamemaker

[43] J. Linietsky and A. Manzur, "Godot Engine," 2014, last accessed 17 December 2019. [Online]. Available: https://godotengine.com/

[44] jMonkeyEngine, "jMonkeyEngine," 2003, last accessed 15 January 2020. [Online]. Available: https://jmonkeyengine.org/

[45] O. Team, “Ogre3D,” 2005, last accessed 15 January 2020. [Online]. Available: https://www.ogre3d.org/

[46] C. M. University, “Panda3D,” 2002, last accessed 15 January 2020. [Online]. Available: https://www.panda3d.org/

[47] mrdoob, “ThreeJs,” 2010, last accessed 15 January 2020. [Online]. Available: https://threejs.org/

[48] UnityTechnologies, “Unity Game Engine,” 2005, last accessed 17 December 2019. [Online]. Available: https://unity.com/

[49] EpicGames, “Unreal Engine,” 1998, last accessed 17 December 2019. [Online]. Available: https://www.unrealengine.com

[50] Tensorflow’s Handpose, “Handpose Github,” 2019, last accessed 29 April 2020. [Online]. Available: https://github.com/tensorflow/tfjs-models/tree/master/handpose

[51] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” *Lecture Notes in Computer Science*, p. 21–37, 2016. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46448-0_2

[52] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014.

[53] J. Hosang, R. Benenson, and B. Schiele, “Learning non-maximum suppression,” 2017.

[54] Adrian Rosebrock, “Non-Maximum Suppression for Object Detection,” 2014, last accessed 31 May 2020. [Online]. Available: https://www.pyimagesearch.com/2014/11/17/non-maximum-suppression-object-detection-python/

[55] Jonathan Hui, “SSD object detection: Single Shot Multi-Box Detector for real-time processing,” 2018, last accessed 31 May 2020. [Online]. Available: https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06

[56] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection,” 2016.

[57] Sik-Ho Tsang, “Review: FPN — Feature Pyramid Network (Object Detection),” 2019, last accessed 06 June 2020. [Online]. Available: https://towardsdatascience.com/review-fpn-feature-pyramid-network-object-detection-262fc7482610

[58] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection,” 2017.

[59] Bazarevsky and Zhang, “On-Device, Real-Time Hand Tracking with MediaPipe,” 2019, last accessed 20 May 2020. [Online]. Available: https://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html

[60] Ideum, “Gesture Markup Language,” 2020, last accessed 16 May 2020. [Online]. Available: http://gestureml.org/doku.php/gestures/motion/gesture_index

[61] Y. S. Lee and B.-S. Sohn, "Immersive gesture interfaces for navigation of 3d maps in hmd-based mobile virtual environments," *Mobile Information Systems*, vol. 2018, p. 2585797, May 2018. [Online]. Available: https://doi.org/10.1155/2018/2585797

[62] Microsoft, "Project Prague," 2017, last accessed 04 March 2020. [Online]. Available: https://www.microsoft.com/en-us/research/project/gesture/

[63] V. Meunier, "HandPose," 2019, last accessed 28 March 2020. [Online]. Available: https://github.com/MrEliptik/HandPose

[64] Andreas Schallwig, "Fingerpose," 2014, last accessed 31 May 2020. [Online]. Available: https://github.com/andypotato/fingerpose

[65] P. L. Sam Richard, "What are Progressive Web Apps?" 2019, 20 accessed 21 April 2020. [Online]. Available: https://web.dev/what-are-pwas/

[66] Github, "Electron (software framework)," 2013, last accessed 20 April 2020. [Online]. Available: https://www.electronjs.org/

[67] SocketIO, "Socket IO," 2020, last accessed 20 April 2020. [Online]. Available: https://socket.io/index.html

[68] npm, Inc., "Node Package Manager," 2013, last accessed 11 April 2020. [Online]. Available: https://www.npmjs.com/

[69] Fabio Panettieri, "Socket.IO for Unity," 2020, last accessed 20 April 2020. [Online]. Available: https://assetstore.unity.com/packages/tools/network/socket-io-for-unity-21721

[70] I. Toshima, H. Uematsu, and T. Hirahara, "A steerable dummy head that tracks three-dimensional head movement: Telehead," *Acoustical Science and Technology*, vol. 24, pp. 327–329, 09 2003.

[71] M. Luzardo, M. Karppa, J. Laaksonen, and T. Jantunen, "Head pose estimation for sign language video," vol. 7944, 06 2013, pp. 349–360.

[72] P. Cipresso, I. A. C. Giglioli, M. A. Raya, and G. Riva, "The past, present, and future of virtual and augmented reality research: A network and cluster analysis of the literature," *Frontiers in psychology*, vol. 9, pp. 2086–2086, Nov 2018, 30459681[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/30459681