INSTITUTO POLITÉCNICO
DE VIANA DO CASTELO

ESTG

INSTITUTO POLITÉCNICO
DE VIANA DO CASTELO

SYSTEM FOR CAPTURING AND DISPLAYING PORTUGUESE SIGN LANGUAGE ANIMATIONS IN A 3D ENVIRONMENT

SISTEMA DE CAPTURA E PROCESSAMENTO DE ANIMAÇÕES DE LÍNGUA GESTUAL PORTUGUESA EM CONTEXTO 3D

Masters Thesis

Bruno Filipe Matos Ribeiro

Escola Superior de Tecnológica e Gestão

System for Capturing and Processing Portuguese Sign Language Animations in a 3D Environment | Bruno Filipe Matos Ribeiro

2024

INSTITUTO POLITÉCNICO
DE VIANA DO CASTELO

Bruno Filipe Matos Ribeiro

System for Capturing and Displaying Portuguese Sign Language Animations in a 3D Environment

Sistema de Captura e Processamento de Animações de Língua Gestual Portuguesa em Contexto 3D

Mestrado em Engenharia Informática

Trabalho efetuado sob a orientação do
Professor Doutor Luís Miguel Cabrita Romero
Professor Doutor Pedro Miguel Teixeira Faria

February 2024

This work was carried out under the project entitled “IVLinG: Intérprete Virtual de Língua Gestual”, which was funded by the European Regional Development Fund (ERDF), through the program Compete 2020, with the project id POCI-01-0247-FEDER-068605.

Cofinanciado por:

UNIÃO EUROPEIA
Fundo Europeu
de Desenvolvimento Regional

ACKNOWLEDGMENTS

First of all, I am extremely grateful to my advisors Professor Luís Romero and Professor Pedro Faria for the invaluable support, guidance and feedback they have given me for the past two years. I am deeply thankful for the opportunity they provided me to work on this project, which led to the publication of two articles. I would also like to thank all the professors from my Bachelor's and Master's degrees for everything they taught me during this long journey.

I would like to give a special thanks to my co-workers and friends, Beatriz Miranda, David Verde, Duarte Dias, Tânia Silva and Vasco Alves, for their unwavering support and the cherished memories we shared. Additionally, I am grateful to all my classmates and friends whose encouragement and companionship played a significant role throughout this journey.

A special thanks also goes to Ângelo Costa and Vânia Ferreira for the essential insight they provided regarding Portuguese Sign Language.

Lastly, I want to express my deepest gratitude to my family, particularly my parents and sister, for their constant support and belief in me.

I would also like to extend my gratitude to all those who were a part of this journey, even if they have not been mentioned previously.

ABSTRACT

Communication relies on a system of linguistic structure and rules that creates a common understanding between people. However, several issues, including language variation and cultural differences, can impact this process. A particular challenging context is the interaction between hearing and deaf individuals, since their methods of communication differ greatly. In Portugal, Portuguese Sign Language (LGP) is one of the three official languages, and it is spoken by around 110,000 people. This number is equivalent to around 1% of the population, so it is evident how difficult communication is for deaf people in Portugal. There are already several technologies that allow the translation between a wide range of spoken languages, but this is not yet the case between spoken and sign languages.

This research project, which arises within the scope of the IVLinG project – *Intérprete Virtual de Língua Gestual* (Virtual Interpreter of Sign Language), aims to capture LGP gestures using Motion Capture (MoCap) devices and develop a mobile application to display those animations in a virtual environment, through a 3D Avatar. This part of the system is ready to receive a list of glosses, for example, after a Portuguese sentence is translated. The glosses received are used to determine what animations the Avatar shall animate. The first step was to conduct research on implementations on the same subject, specifically those that previously developed applications using LGP or other sign languages. Then, some LGP signs were captured as animations and an application was developed to receive the translator's glosses and animate the Avatar accordingly. The developed solution was tested to check if the translations could be done quickly and if the animations were understandable and correct according to LGP specialists. Tests carried

out regarding the speed of the solution showed that, although the Avatar module takes a few seconds to load, after this step the animations are ready very quickly. Tests regarding the quality of the animations showed that MoCap can help improve the fluidity and naturalness of the animations, when comparing to manual animation, albeit several changes and refinements to the raw data are required.

The work described in this document serves as a basis for a virtual translator from Portuguese to LGP and can significantly help the deaf community on their daily communication with the hearing community. Future work aims to improve the number of body and facial animations, using MoCap to assist in this process, and to improve the quality of these animations, by performing further testing to them and editing accordingly.

Keywords: *sign language, translation, avatar, virtual interpreter, motion capture, glosses, unity*

RESUMO

A comunicação baseia-se num sistema de estrutura linguística e regras que criam um entendimento comum entre as pessoas. No entanto, vários fatores, incluindo variações linguísticas e diferenças culturais, podem afetar este processo. Um contexto particularmente desafiante é a interação entre indivíduos ouvintes e surdos, uma vez que os seus métodos de comunicação são muito diferentes. Em Portugal, a Língua Gestual Portuguesa (LGP) é uma das três línguas oficiais e é falada por cerca de 110.000 pessoas. Este número equivale a cerca de 1% da população, pelo que se torna evidente a dificuldade na comunicação para as pessoas surdas em Portugal. Existem já várias tecnologias que permitem a tradução entre várias línguas faladas, mas o mesmo ainda não acontece entre línguas faladas e línguas gestuais.

Este projeto de investigação, que surge no âmbito do projeto IVLinG - Intérprete Virtual de Língua Gestual, tem como objetivo a captura de gestos em LGP através de dispositivos de captura de movimento (MoCap) e o desenvolvimento de uma aplicação móvel que permita apresentar essas animações num ambiente virtual, através de um Avatar 3D. Esta parte do sistema está pronta para receber uma lista de glosas, por exemplo, depois de uma frase ser traduzida de Português. As glosas recebidas são usadas para determinar quais as animações que o Avatar deve animar. O primeiro passo foi a pesquisa de implementações sobre o mesmo tema, nomeadamente as que já tinham desenvolvido aplicações usando LGP ou outras línguas gestuais. De seguida, foram capturadas algumas animações que representam gestos em LGP e foi desenvolvida uma aplicação para receber as glosas do tradutor e animar o Avatar de acordo com a tradução. A solução desenvolvida foi testada para verificar se as traduções podiam ser

feitas rapidamente e se as animações eram compreensíveis e corretas de acordo com professores de LGP. Os testes efetuados relativamente à rapidez da solução mostraram que, embora o módulo do Avatar demore alguns segundos a carregar, após este passo as animações ficam prontas rapidamente. Os testes relativos à qualidade das animações mostraram que MoCap pode ajudar a melhorar a fluidez e naturalidade das animações, quando comparado com animação manual, embora sejam necessárias várias alterações e refinamentos às animações adquiridas desta forma.

O trabalho descrito neste documento serve de base para um tradutor virtual de português para LGP e pode ajudar significativamente a comunidade surda na sua comunicação quotidiana com a comunidade ouvinte. O trabalho futuro tem como objetivo melhorar o número de animações corporais e faciais, utilizando MoCap para auxiliar neste processo, e melhorar a qualidade destas animações, realizando mais testes às mesmas e refinando-as de acordo com os resultados.

Palavras-Chave: *língua gestual, tradução, avatar, intérprete virtual, captura de movimento, glosa, unity*

CONTENTS

LIST OF FIGURES

LIST OF FIGURES

LIST OF TABLES

GLOSSARY, ABBREVIATIONS and ACRONYMS

API Application Programming Interface

ArSL Arabic Sign Language

ASL American Sign Language

ATLAS Automatic Translation into sign LAnguageS

Blendshapes Prevalent approach to realistic facial animation, by defining how the mesh wraps around the face

BSL British Sign Language

Cache Hardware or software component that stores data temporarily to speed up future access to that data

HTTP Hypertext Transfer Protocol

IEEE Institute of Electrical and Electronics Engineers

ISL Indian Sign Language

JSON JavaScript Object Notation

LGP Portuguese Sign Language

LIS Italian Sign Language

LP Portuguese Language

LSE Spanish Sign Language

LSR Romanian Sign Language

MoCap Motion Capture, usually through the use of cameras and/or sensors in key points of the body

MSL Mexican Sign Language

OL Oral Language

PSL Pakistan Sign Language

RAM Random Access Memory

SEE Signed Exact English

SL Sign Language

URP Universal Render Pipeline

WebGL Web Graphics Library

XMPP Extensible Messaging and Presence Protocol

1. Introduction

Regarding Sign Languages (SL), William Stokoe [1] explained that one of the first questions presented by those unacquainted with the subject is whether there is a universal SL. Although humans universally make use of facial expressions, arms and other parts of the body to signal, this is not the case when these signs are structured to create complex systems of words and sentences. These linguistic systems, known as sign languages, exist at least since the 4th century [2] and are used by deaf communities. These communities tend to be very small, due to the small percentage of the population that is deaf and have been repressed throughout most of history. This meant that each community created its own sign language. Only in 1771 was the first school for the deaf created. After that, more schools were built, and country-wide sign languages were defined to end the regional sign languages. Nowadays, sign languages differ from country to country, and in some bigger countries, there are still multiple sign languages and/or dialects.

1.1. Context

In Portugal, Portuguese Sign Language (LGP – Língua Gestual Portuguesa) is one of the three official national languages, along with Portuguese Language (LP – Língua Portuguesa) and Mirandese, and it is used by the deaf community. There are around 30,000 people in this community which are deaf [3]. When considering the surrounding community, such as family, friends, teachers, interpreters, among others, there are around 110,000 LGP users in total. According to Censos 2021 [4], this represents around 1% of the total Portuguese population. Most deaf people also have some or a lot of difficulty in reading and interpreting written Portuguese, since it is not their mother tongue.

Given these factors, it is easy to understand that most deaf people have trouble communicating with the rest of society, especially since most of the time the general population does not know how to speak and interpret LGP. However, with the recent advances in technology, it may be possible to create a solution to mitigate this issue.

It is already common practice to use smartphones when translating between two spoken languages, why not use them to translate between spoken languages and sign languages?

In this context, the “IVLinG: Intérprete Virtual de Língua Gestual” project was created, with the goal of developing a digital platform with bidirectional virtual interpretation of LGP. The LP to LGP component, the one that translates text or audio in Portuguese to LGP, was done by Polytechnic Institute of Viana do Castelo (IPVC). The work described in this dissertation is part of that component and contributed to the final version of the project, which was completed in June 2023.

1.2. Main Objectives

Through human interpreters of sign language, there is already some communication between deaf and non-deaf people. These interpreters are usually related to the deaf community, for example family members of a deaf person. However, in their daily lives, deaf individuals often lack constant access to interpreters and may require assistance in communication. In order to solve this translation problem, an accessible virtual interpreter is needed, which can be used at any time. Although several attempts have been made in this direction, with numerous sign languages, there is still a lack of market-ready applications that the deaf community can use.

The main goal of the solution described in this dissertation is the development of an application that shows signs in LGP, through an animating 3D Avatar, to mitigate the troubles in communication between the deaf and non-deaf people. This application receives a sequence of glosses, which are the written form of signs, and animates the Avatar with the corresponding animations, in the correct LGP structure. The acquisition of sign animations will be semiautomatic, since the method used consists in a combination of motion capture systems (MoCap) with some manual refining, using a 3D modelling and animation software. The application must be developed for both mobile and browser versions.

As such, the solution should seek to answer the following questions:

1. Is it possible to create a mobile virtual interpreter that deaf people can use in their everyday lives?

2. Can the translations be ready quickly, so the user does not need to wait a long time?
3. In case the user does not understand a sign, can the virtual interpreter repeat the translation and/or slow it down to better understand it?
4. Can MoCap be used to improve the time it takes to acquire signs animations?

In order to try to answer these questions, some objectives were established for the project, such as:

1. To capture the movements of various signs using MoCap systems;
2. To create a database that stores the acquired animations;
3. To develop an application with a 3D Avatar that can be animated with those animations;
4. The establishment of connections between the user, the translation, the developed application and the database, in order to animate the Avatar in real time.

1.3. Research Methodology

During the development of the work described in this document, the Design Science Research (DSR) methodology was used, which focuses on the creation and validation of novel solutions (artifacts) to relevant problems, using processes such as: designing and evaluating the developed artifacts, producing scientific contributions and presenting the achieved results [5]. In [6], the author studies several published works on DSR, identifies what they have in common and proposes a process for conducting DSR in information systems. The proposed system consists in 6 steps, Problem Identification and Motivation, Definition of the Objectives for a Solution, Design and Development, Demonstration, Evaluation, and Communication:

- **Problem Identification and Motivation**: Describes the problem, developing a literature review and establishing the importance of the solution. This is explained in 1.1. Context sub-section and 3. Literature Review section;
- **Definition of the Objectives for a Solution**: Determines the objectives for the solution. This encompasses determining what is

possible and achievable. It is also required to have knowledge of existing solutions of the problem. The objectives may be qualitative, which describe how the artifacts will achieve solutions to the problem, or quantitative, which demonstrate how the proposed solution is better than others, using metrics to compare them. The objectives are described in 1.2. Main Objectives sub-section;

- **Design and Development**: Designs an artifact architecture and then creates it. These activities are shown in 4. System Architecture and 5. System Implementation sections;
- **Demonstration**: Shows that the developed artifact can in fact solve the identified problems. This demonstration can be done through experimentation, simulation, conduction of use-cases, or others. The demonstration is available in 5.3.3. Example of Using the Avatar sub-section and 6. Tests and Results section;
- **Evaluation**: Verifies and measures the developed artifact behaviour when solving the problem, in order to determine if the artifacts have the expected results, taking into consideration the proposed objectives. The evaluation is described in 6. Tests and Results section;
- **Communication**: Shares the problem, the solution that was developed, and its results, through means such as publishing articles in conferences or journals. This activity is presented in the appendices of this document.

1.4. Structure Overview

Having outlined the objectives and research methodology, this document was divided into 8 chapters.

- In chapter **1 – Introduction**, a brief introduction to sign languages is given in order to contextualise the topic, followed by the project's objectives and specific questions. Next, the methodology used to develop this project is presented..
- In chapter **2 – Theoretical Background**, an explanation is given on various relevant concepts regarding the main technologies used in the development of this project, such as motion capture and the engine

used to develop the application, as well as the fundamentals of LGP and its grammar.

- In chapter **3 – Literature Review**, the focus is on showing previously developed work, mainly related to the presentation of SLs without the need for a human interpreter, i.e. through virtual avatars, robots or videos.
- In chapter **4 – System Architecture**, it is given a description of the system's architecture and its components. For one part of the system, two architectures are viable options, so both will be designed and then it will be decided which one will be used.
- In chapter **5 – System Implementation**, all of the developed components are presented. It begins by determining and explaining how the animations are acquired. This is followed by the development of a 3D avatar, which can animate with the acquired animations. The creation of support databases is also implemented, followed by the development of the two architectures mentioned in the previous chapter. One of these architectures is selected and implemented with the rest of the system. Finally, a demonstration of the final solution is presented.
- In chapter **6 – Tests and Results**, an analysis is made on the results obtained from the implemented system. The focus of these tests is the quality of several aspects of the system.
- In chapter **7 – Conclusions and Future Work**, a summary of the work carried out is presented, discussing the results achieved and the contributions made, suggesting possible future improvements of this work.
- In chapter **8 – References**, a list of references, used in the development of the work described in this document, is presented.
- In chapter **9 – Appendices**, documents that complement or support the development of this work are presented, such as articles published at international conferences.

2. Theoretical Background

In this section, some theoretical background is given, to better understand characteristics of LGP and how it differs from other languages. Then, an explanation of some software that will be used in the project is introduced, specifically motion capture, its various types and the differences between them.

2.1. Portuguese Sign Language

Portuguese Sign Language, commonly known as LGP, which are the initials of its name in Portuguese, is one of the three official languages of Portugal, being the one used by the deaf-mute community. LGP was originated in 1823 when “Real Instituto de Surdos-Mudos e Cegos” was founded [7], becoming the first school for deaf-mute in Portugal. The Swedish Per Aron Borg had established a school for deaf people in Sweden, so the Portuguese King D. João VI invited him to create this institute. Even today, when looking at the alphabets of the two languages, LGP and Swedish Sign Language, they reveal their common origin [8].

According to the Portuguese Deaf Association [3], there are around 30.000 deaf people that use LGP to communicate. The number of LGP users is estimated to be around 110.000 when taking into consideration the surrounding community, such as relatives, friends, teachers, and other users. As any spoken language, sign languages are very complex, since there are several different signs, and the meaning of the sign changes depending on the position and movement of parts of the body, as well as context. Sign Languages have their own grammar and structure and do not follow the structure of an oral language. In fact, sign languages are independent from the oral languages spoken in their respective countries [9], so they differ from country to country. This means that British Sign Language (BSL) is different from American Sign Language (ASL) even though both countries speak English, and the same is true for LGP and Brazilian Sign Language (LiBraS).

A sign has a set of unique and important characteristics regarding the configuration, position and movement of the hands, as well as body and facial

movements and expressions. This set of characteristics defines the sign, and by changing one or some of them it is possible to make a different sign [10], or a gesture that does not exist in LGP, so it has no meaning. As of the writing of this document, there is a website, named SpreadTheSign, that is developing an LGP dictionary, with videos of native LGP users performing each sign, and can be accessed in [11].

As already mentioned, sign languages have their own structure and set of grammatical rules and, when compared to LP, they are very different. This means that a deaf person who uses LGP sees LP as a second language and may not be able to communicate easily by reading/writing.

2.1.1. Characteristics of Portuguese Sign Language

The characteristics of LGP signs are similar to other sign languages, and consist of 5 *cheremes* [7] [12] [13], which are:

- **Hand configuration**: corresponds to the shape the dominant hand or both hands make during the execution of the sign. In LGP there are 76 possible configurations, mostly based on the alphabet and numbers;
- **Movement**: corresponds to the movement and direction of the hands and fingers. Some signs do not have movement, and other signs have multiple motions, called compound signs, made up of several simple signs. Additionally, a sign might have a different meaning if the movement is fast, slow, short, long, smooth or tense;
- **Non-manual expressions**: these include facial expressions, such as movement of the mouth and brows, and movement of the shoulders, head and body. They are important because they convey the emotion of the sign. Facial expressions portray the sentence as a question, exclamation, negative, affirmation, admiration, fear, among others. Some signs only differ from another based on the facial expression. Furthermore, body language gives context to the sentence, for example when pointing to referents, like subjects and objects;
- **Location**: this defines where the hands should execute the sign and is divided into two ways. The first is when the hands execute within the signing space, an imaginary area in front of the torso and head, without

any contact with the body. The second is when there is contact with other parts of the body, like the face, neck, ears, chest, among others;

- **Orientation**: corresponds to the rotation of the hand when the sign is executed. The reverse orientation might suggest the opposite sign.

2.1.2. Gloss Notation

Gloss notation is the written representation of words or sentences in LGP. It is comprised of a sequence of terms called glosses, and each gloss represents a sign. There are a few rules for writing gloss notation for LGP that are important to know:

- Using only uppercase letters, for example the verb “want” in Portuguese is “querer”, and the gloss is “QUERER”;
- To separate glosses, a “**+**” (plus sign) must be used;
- To determine a short pause, a “***/***” (forward slash) must be used, for example when the original sentence has a comma;
- To determine a long pause, “***//***” (two forward slashes) must be used, for example when there is a period or change of sentence;
- When spelling words, a “-” (hyphen) must be used in between each letter, as well as when using cardinal numbers;
- Names, ordinal numbers and numbers that represent quantities should be spelled out;
- Verbs should be written in the infinitive.

An example of a translation from Portuguese to LGP is “A Joana tem marido” translates to “J-O-A-N-A+MARIDO+TER”. Following the rules mentioned above, the name “Joana” was spelled out using hyphens, the verb “tem” was changed to the infinitive “ter”, and plus signs were added between each gloss.

2.1.3. Grammar and Structure of LGP

Regarding the grammar and structure of LGP, there are not many resources available, so the characteristics mentioned below are mainly derived from the book [13], as well as from meetings with interpreters and LGP users. The book is considered one of the only published works that focuses on LGP grammar, being cited by many works such as [14], [15] and [16] but, at the time this project was developed, it was no longer being printed or sold.

2.1.3.1. Sentence Structure

The structure of a LGP sentence is OSV (Object-Subject-Verb). This differs from Portuguese, where SVO (Subject-Verb-Object) is the correct form. There are some people that argue that LGP uses SOV (Subject-Object-Verb), as seen in [14] and [16], but in most cases both SOV and OSV are understandable by LGP users. When the sentence is a question, exclamation or negative, most of the time it is shown with non-manual expressions, usually facial expressions. However, there are cases where the negative is determined by adding to sign for "NO" at the end, and a question is marked by using questions words (e.g.: "when", "who", "how many").

2.1.3.2. Genders

In LGP, most of the time, there is no grammatical gender, unlike Portuguese which has two, masculine and feminine. When there is a need to distinguish between genders, the sign that means "woman" is added before the sign, like a prefix, to define it as feminine. In the case of masculine, users omit the prefix which determines the sign as masculine by default. This means that in some cases where a noun has two genders, the difference between them is the prefix "woman". However, there are situations in which there are independent signs for each gender.

2.1.3.3. Names

Regarding names of people, most LGP users have their "nome gestual" (name sign), which is a sign they choose to identify themselves with. Since each person chooses their own name sign, there is no connection between it and their written name, which makes translation impossible. However, in cases where a person does not have a name sign, or when talking about someone without knowing their name sign, their written name is spelled out instead, and as far as translation is concerned this is also possible.

2.1.3.4. Iconicity

Sign languages have some signs that try to copy real world objects, making them perceptible even by people that do not speak any sign language. These signs are called "iconic". There are also referential signs, for example when pointing at someone or something, and are harder to portray in a virtual world. Finally, there are arbitrary signs which do not have any relation to the real object.

2.1.3.5. Verbs

Verbs in LGP are not conjugated, instead they have only one sign for each verb. The gloss of a verb is usually the infinitive of that verb in Portuguese. To define when the action of the verb was set in LGP, depending on the situation users use facial expressions, adverbs of time (e.g. yesterday, tomorrow) or even through the use of signs like “past”, “was” and “soon”, which mark the past negative, past positive and future, respectively.

2.1.3.6. Lack of Sign

In the case of a word in Portuguese that does not have a corresponding sign in LGP, a synonym should be used instead. In cases where no synonyms are found, the word should be spelled out in LGP. It is worth noting that LGP users, like any other language, do not know every sign/word, which is the case with some medical terms that have a corresponding sign, yet the general deaf population does not know it.

2.2. Software

This section provides brief explanations of the software used in our project. Unity [17] and Maya [18] were chosen due to our familiarity with these tools and their alignment with the project requirements. Additionally, the FBX format, a type of file that can contain 3D objects and animations, was chosen for the animations because of its flexibility, adaptability, and compatibility with Unity and Maya.

2.2.1. Unity

Unity [17] is a powerful and versatile game engine that is widely used by developers to create interactive 2D and 3D games for various platforms. It offers an extensive set of features and tools that make the entire development process easier, both for games and other types of applications. One of Unity's key strengths is its cross-platform compatibility, which is essential for developers nowadays, due to the wide range of devices and operating systems available.

One key point of games is animation, and Unity is well-regarded for its capabilities in this topic due to several reasons, such as blend trees and state machines, which allow the creation of complex and sophisticated animation systems with smooth transitions. Some features like retargeting, which allows the

use of one animation for several characters, and the support of blendshapes, which will be explained later, for facial animation are also great available tools in this program.

Another useful feature for both games and applications that Unity provides is the creation and use of asset bundles. Asset bundles allow developers to package and load assets at runtime and can be compressed to reduce file size, especially when they are hosted remotely. In an application with hundreds of animations, or other types of assets, this hosting remotely option can help tremendously in performance.

Regarding the state machines mentioned above, they help turning complex animation behaviours into more simple components, which are states and transitions. States represent a specific behaviour or mode that an object can be in. For example, in a simple game character, states could include "Idle", "Walking", "Running" or "Jumping". Whereas transitions define the conditions under which the system should switch from one state to another. After a state machine is associated to a 3D object, it manages the current state of that object and handles the transitions between states.

2.2.2. Autodesk Maya

Autodesk Maya [18], or just Maya, is a powerful 3D computer graphics software, being one of the industry-standard in the film, television, and video game industries. It provides a wide range of tools and features for modelling, rendering, simulation, and the one that stands out the most, animation. Regarding the latter, tools such as keyframe animation and procedural animation techniques enable artists to craft intricate motion sequences. It also supports a wide range of import and export options, allowing users to interchange data with various software and file formats, such as FBX, OBJ, DAE, and many other. The most widely used 3D animation file format is FBX for several reasons. In addition to being one of the few that supports skeleton animation, mesh and texture, it is also a great option for higher-quality animations. Due to being the industry standard, it is supported by most 3D programs and engines.

The creation and animation of human-like models, known as Avatars, is possible in Maya. These Avatars consist of a mesh and a rig. The mesh is a set of hundreds or thousands of polygons that make the skin and clothes, so the

Avatar looks like a human. The rig is what makes the mesh to move properly, by having an array of bones and joints. The joints rotate to move the bones which are associated with the mesh. In Figure 1, the mesh of the Avatar can be seen on the left, and a wireframe version of it can be seen on the right. On the left side the bones and joints are also selected with a green colour.

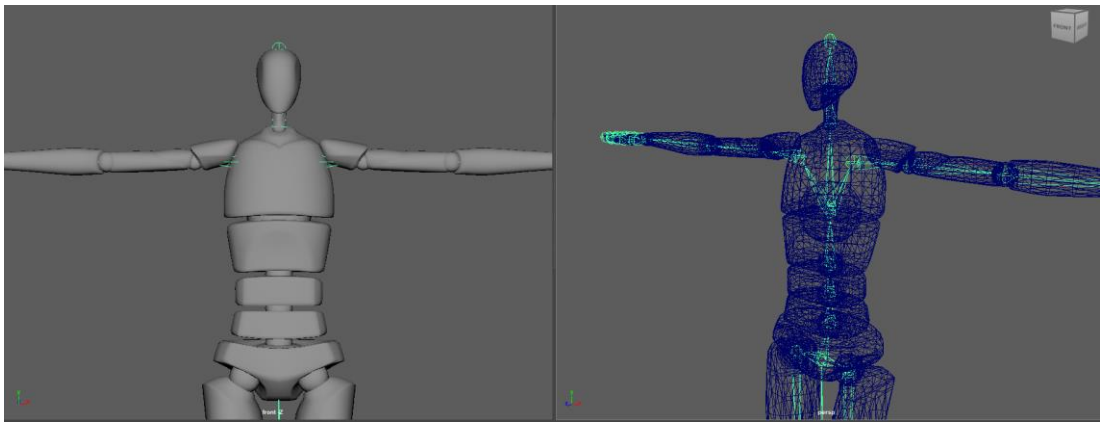

Figure 1 The Mesh of an Avatar on the Left / The Joints and Bones on the Right.

2.3. Motion Capture

Motion capture (MoCap) is the process of recording the movements of real-world objects or people and converting them into digital data [19]. Then, using this data, virtual characters or objects can be animated in a simulated environment. It is used in a variety of industries, including movies, video games, animation, and sports. Over the years, several technologies have been developed to capture motion. These systems can be divided into six types, which are: optoelectronic, electromagnetic, image processing based, acoustic measurement-based, exoskeleton systems or inertial sensors-based [20] and the ones that are more relevant to Avatar motion capture are explained in this chapter.

2.3.1. Optoelectronic Measurement Systems

The Optoelectronic Measurement Systems (OMS) are regarded in literature as the gold standard in motion capture, because they are more accurate and precise than other systems [21]. An OMS is able to detect light and estimate the 3D position of a marker via time-of-flight triangulation. However, they have some disadvantages. Since OMS use fixed cameras and are sensitive to light, they can only be used in restricted indoor areas. These systems require an expensive

laboratory, and accuracy depends on the position of the cameras and markers in relation to each other. The number of cameras and field of view of each camera also influence the results, and by adding more or better cameras, the costs increase even more. The systems are also extremely sensitive to small changes in the set-up, even, for example, by the accidental shifting of one camera.

OMS can be divided into two categories, passive marker and active marker. Passive systems reflect light back to the sensor, while active systems use sensors that contain a source of light. Although active marker systems tend to have more robust measurements, they also require more cables and batteries than passive marker systems, making the movement more restricted.

An example of a passive OMS is the Vicon V-16 Vantage [22] cameras. This system can be seen in Figure 2, where two actors are wearing a full body suit with markers in key positions of the suit. Multiple cameras can be seen in the picture and all of them are calibrated to work together and get really accurate motion capture. As mentioned, changing any of these cameras decalibrates the system, requiring a full recalibration to work properly.

Figure 2 Example of an OMS – Vicon [23].

2.3.2. Electromagnetic Measurement Systems

Electromagnetic Measurement Systems (EMS) use electromagnetic waves (radio waves) that travel from the transponders to the base stations [24]. These systems are less accurate than the worst performing OMS [21] but line-of-sight

between the transponders and the station is not required, unlike OMS which requires it between the sensor and the cameras. The limitations of EMS are the sensitivity towards ferromagnetic material, which decrease accuracy, and the quality and noise of the data vary according to the distance between transponders and stations.

2.3.3. Image Processing Systems

Image Processing Systems (IMS) are vision-based tracking systems, that use optical cameras and computer vision algorithms to track motion, making it a markerless type of MoCap. Although they are more accurate than EMS and have a wider range than OMS, there are some disadvantages to this method, as it is not easy to make real-time image recognition. To be effective, cameras need to have both good quality and be high-speed, which can get expensive and requires more processing power, hence the difficulty to perform real-time detection.

The Kinect [25] is a famous example of a type of IMS system and although it was primarily a game-focused product, many developers noticed it was a cheap motion capture option, such as the one described in [26] and seen in Figure 3 and Figure 4. In the right conditions, it can capture movement with some precision, even though some clean-up is always required.

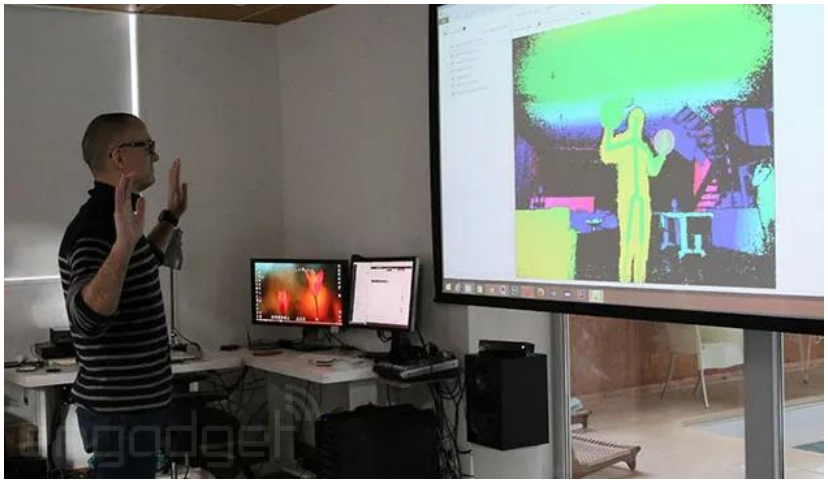

Figure 3 Example of using the Kinect System.

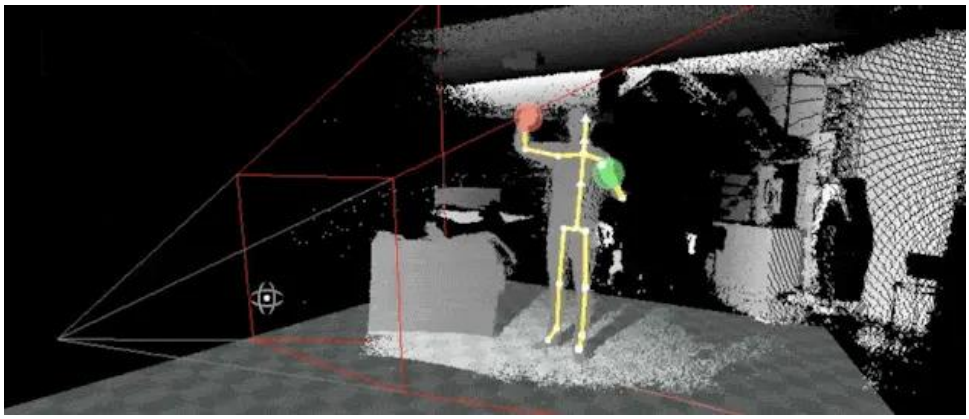

Figure 4 How the Kinect System detects MoCap.

2.3.4. Inertial Measurement Systems

Inertial Measurement Units (IMUs) are electronic devices that consist in several sensors, such as accelerometers, gyroscopes, and magnetometers [27]. The accelerometers can determine the gravitational acceleration, while the gyroscopes determine the rotational value, and when combining the data from the sensors, it is possible to determine the orientation of the device. Magnetometers are used in some IMUs to track the magnetic-north and determine the heading of the device. IMUs are tiny components, which make them lightweight and portable, in addition to being low-cost [20]. Furthermore, the installation and calibration processes are not very time-consuming, making it an easy to implement system. They can be attached to different parts of the body to track human motion into a rigid-body model of a human, and are able to record very fast movement, unlike some other methods. One issue with IMUs is the poor accuracy in some parts, especially when used near iron objects, or other materials that cause magnetic interference [28]. Nevertheless, these usually offer great MoCap data for the price.

Two examples of IMUs are the Rokoko Smartsuit Pro II [29] and the Perception Neuron Studio [30]. These systems are similar in both price and behaviour. The main difference between the two, seen in Figure 5, is the way the user wears the system. The Rokoko consists in a full-body suit that the user needs to wear, and the Neuron consists in strips that the user wears in specific parts of the body. The number of sensors in both suits are almost identical, although the performance differs in certain aspects, mostly due to software and how it interprets the data.

Figure 5 Rokoko Smartsuit on the Left / Neuron Studio on the Right.

2.3.5. Motion Capture for Sign Acquisition

There are several types of motion capture, each with their advantages and disadvantages. However, for the purpose of acquiring sign movements to be represented in an Avatar, three of the types should be considered, the OMS, IMS and IMUs. EMS are not as accurate as OMS, and not as cheap as IMS or IMUs, with some other drawbacks regarding the set-up required to have the best accuracy. Furthermore, the advantages they have over other systems are not relevant to the sign language acquisition.

Table 1 sums up the comparison between each of the three most appropriate systems for capturing sign language, and the data in the table was gathered from [19] and [21]. Although the OMS has the highest accuracy and sampling rates, it comes with big disadvantages. It is the most expensive by far out of the options, has limited range and portability, and has time-consuming set-up with frequent calibrations needed. IMS is the cheapest option, as well as being portable and with a decent range. The drawbacks from this solution are the somewhat hard calibrations and low accuracy compared to the other systems. IMUs offer a

comfort option that is neither expensive nor has bad accuracy. In fact, the price is considerably cheaper than OMS and only slightly more expensive than IMS, while bringing good accuracy when in the right conditions. The sampling rate is good, it is an easy to setup system which makes it the most portable one. The main disadvantage is that even small magnetic interferences can disrupt the data, so capturing motion near metal objects can be highly problematic.

Table 1 Comparison of Motion Capture Systems.

	Optoelectronic	Image Processing	Inertial Measurement Units
Accuracy	Very High	Low	High
Range	Short	Average	Long
Cost	Expensive	Cheapest	Cheap
Portability	Limited	Yes	Yes
Sampling Rate	Up to 250hz	15-30hz	60-120hz
Set up	Time-consuming	Requires checkerboard calibration	Easy

2.3.6. Face Capture

Facial MoCap is more complicated than body MoCap, as the expressions cannot be determined by bones and joints like the body. Instead, blendshapes are used to achieve this problem. Blendshapes consist in pre-defined facial poses or shapes, each representing an expression, e.g., smiling, raising eyebrow, frowning, among others [31]. Then, it is possible to linearly combine the blendshapes, by calculating the weight of each of them, to create any facial expression on an avatar [32].

The industry-standard for facial blendshapes nowadays is the Apple ARKit, containing 52 blendshapes in total [33], as seen in Figure 6. Out of these, 14 are related to the eyes, 27 are for moving the jaw and mouth, 10 are for the eyebrows, cheeks, and nose, and one is for the tongue.

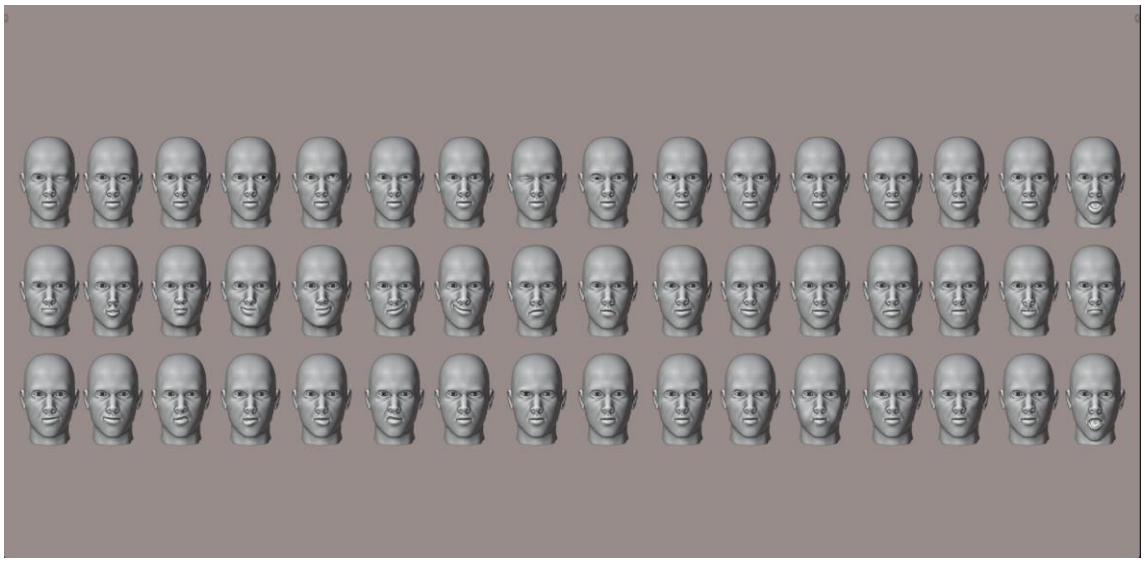

Figure 6 ARKits' 52 Blendshapes.

3. Literature Review

In this chapter, a systematic literature review is made to identify studies which focus on translating from OL (Oral Language) to SL and that explain how the signs are captured and displayed to the user. The selected studies are then explained and analysed individually, and finally a comparison between them is made to understand the methods used and what was achieved.

3.1. Systematic Review

There are over 300 sign languages in the world [34], so it is difficult to create a universal translation system from and to SLs. Furthermore, modern systems for translating from a SL to an OL involve sign recognition using deep learning techniques [35], while translating from an OL to a SL involves displaying the signs to the user without having him read the original sentence, by using virtual avatars, pre-recorded videos, physical human-like robots, or other similar solutions. This means that both translations require completely different technologies and subsequently, most studies that have attempted to develop solutions to the SL translation problem only focus on one SL and are unidirectional, meaning they either go from SL to OL or vice-versa.

The Preferred Reporting Items for Systematic Reviews and Meta-analyses (PRISMA) methodology was used to review the literature on available studies that translate OL into SL [36]. To find research papers, a combination of the query terms “sign language translation” with “avatar” or “virtual interpreter” were used on two databases: Scopus and Institute of Electrical and Electronics Engineers (IEEE). A second combination of words was also used on both databases, and those were “sign language” with “avatar” or “virtual interpreter” with “mocap” or “motion capture” or “mo-cap”. The inclusion criteria were fitting the combinations of keywords mentioned above and being published after the year 2000, since most technologies developed before that year are either too outdated, deprecated or not supported at all. A total of 668 articles from Scopus and 225 from IEEE were found, and out of those, 159 were duplicates, making a total of 734 articles that required screening.

These articles were screened by title and abstract, with the exclusion criteria being wrong language, not written in either English or Portuguese, wrong setting due the translation being from sign language to text or audio, instead of OL to SL, wrong setting due to the focus being only the translation from SL to gloss and disregarding the display of the glosses, not relevant to the research question, and literature reviews. This resulted in 52 articles being screened by full text and evaluated to either be eligible or excluded for specific reasons. These reasons were not obtaining the full text, the outcome evaluated was not relevant to our research, little to no information on the avatar component, and not providing original data. This resulted in 18 articles being screened, albeit some of them were related to the same project, which means that there are a total of 12 unique studies out of the selected articles. Figure 7 illustrates this entire process.

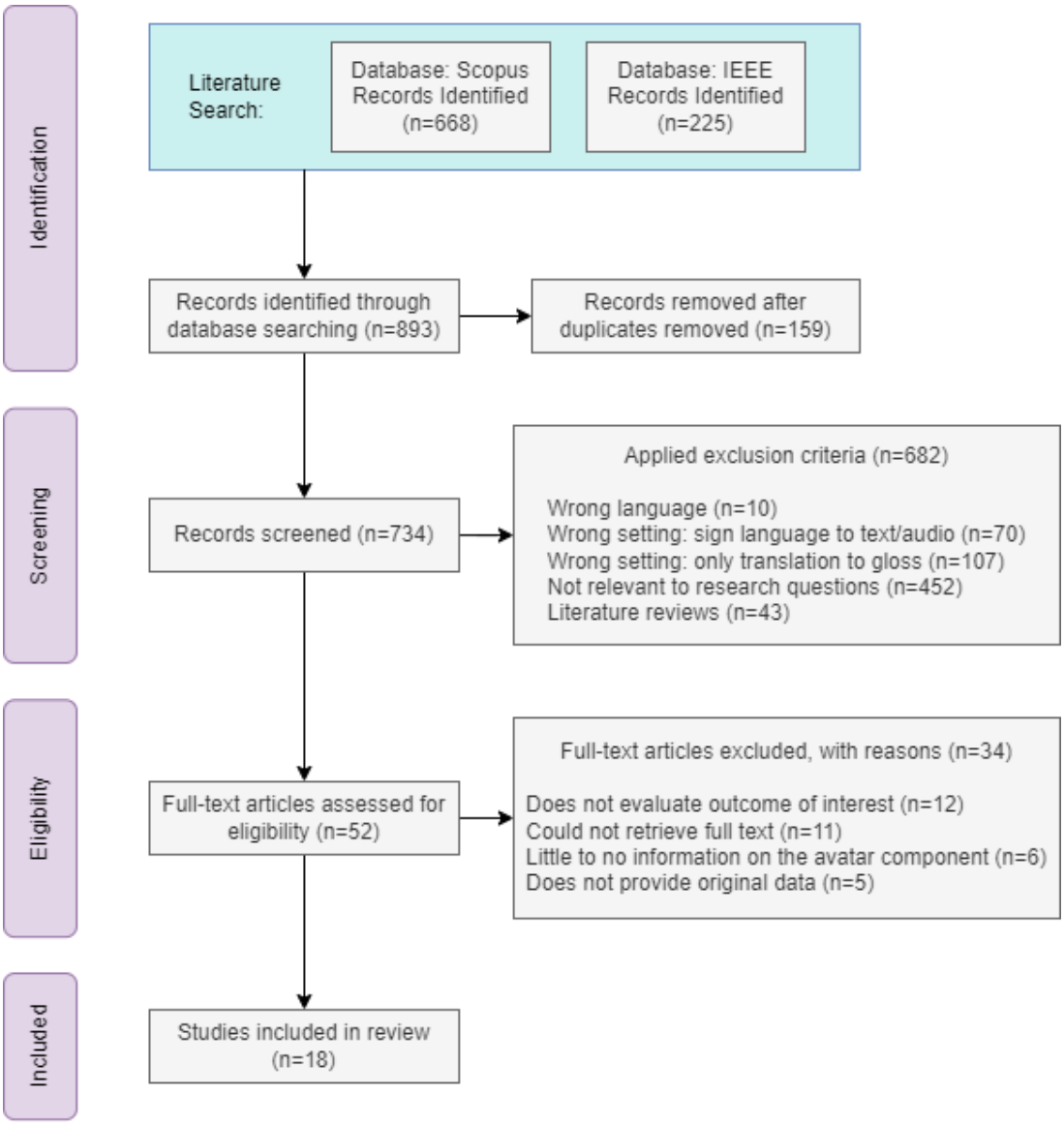

Figure 7 Search Process for Study Selection.

3.2. Related Studies

The twelve resulting studies will be discussed and were selected due to their similarities to the current study. They all develop systems that translate an OL to a SL and explain the avatar component. A detailed explanation and analysis of these studies is required before comparing them. The ones that focus on LGP are presented first, followed by the ones that focus on other SLs.

3.2.1. PE2LGP

The first version of the “PE2LGP: from European Portuguese to LGP” was developed by Inês Almeida [15] [37] [38]. Its main goal was to implement a system that translates text to LGP and consists in three components: translation into glosses, mapping the glosses to the animations, and producing the signs on a 3D Avatar. The 57 possible hand configurations for LGP were implemented manually using Blender [39], a free 3D animation software similar to Maya. Furthermore, some words like “rabbit”, “rabbit house” and “little rabbit” were also manually animated, as well as the glosses required in the sentence “John eats soup”. The glosses on the sentence were combined using the Non-Linear Action (NLA) feature, in Blender, to animate the complete sentence. The words and sentence were shown to deaf users as a preliminary test to verify if they understood them, and they did. The work of Ruben Santos [40] continued on the previous work by exporting the prototype to Unity, thus allowing the export of the application to multiple platforms, something which Blender cannot do, and by creating both a manual and an automatic system of acquiring animations. The former, seen on the left and middle of Figure 8, allows people without prior 3D animation experience to create new signs animations, through an interface with simple commands, that the user can use to make his own gloss. The latter, seen on the right of Figure 8, is more automatic, since it uses the Kinect sensor, the detect body positions, and since Kinect does not capture data from the hands and fingers, when the user is recording a sign, there is a menu with the configuration of the hands used in LGP and the user can select which one is used for that gloss. The system was evaluated by six people, and they showed that the automatic system created animations much faster, while the manual proved to create better animations when shown to deaf users. Nonetheless, on their tests,

the animations from both systems were determined inadequate in the LGP context. A further improvement to this system was demonstrated in [41] with a new Animator tool. The goal is to improve the module that allows the user to manually create sign animations. The recorded times of creating new signs are between 10 and 60 minutes, and they were created by two engineers with basic LGP training that worked in the project. The transitions are made using the Unity's state machine, unlike the NLA in the first prototype. Although the results are better than the previous implementation, there are still plenty of signs and sentences that the testers did not understand.

Figure 8 PE2LGP Manual (Left and Middle) / Automatic (Right) Tools for Creating Animations [40].

3.2.2. Virtual Sign

Another solution for translating Portuguese to LGP is Virtual Sign [42] [43] [44]. The goal of the project is to translate from LP to LGP and vice versa, as well as implementing a serious game for learning LGP. Both the LP to LGP module and the game share the same Avatar. They started by creating a model using Blender. To help acquiring the animations, they developed an application, which they called VSSO [45], where the user can manipulate the arms, body and head, using the mouse and sliders to rotate the joints of the Avatar. Although the facial expressions and hand configurations for a sign can be selected using a menu, there are only six facial expressions available: neutral, happy, sad, surprised, inquisitive, and angry. In LGP the facial expressions are much more complex than just these six, for example, the sign for "work" requires the user to be exhaling through the mouth, similar to what people do when tired, the sign can be seen on the SpreadTheSign [11] website, searching for "work" ("trabalhar" in Portuguese), and selecting LGP. This means that the available facial expressions are very limited, compared to the range of LGP. After creation, the animations are stored in a database to be downloaded when the user requests a translation. The LP to LGP module was implemented using Unity and exported to Web Graphics Library (WebGL), to be used in most browsers, and the user can input a text to get the

translation. The transitions were made using Unity's state machine. Additionally, a PowerPoint Add-On was implemented where the user can select "translate the text" from a slide to LGP. Though it can be used by anyone, it was developed thinking about teachers that give classes to deaf students. The project is expanding from LGP to other SLs, which are: British, Cypriot, Slovenian, German and Greek. An example of the Avatar and the game developed can be seen in Figure 9.

Figure 9 The Avatar [42] on the Left / The Avatar In-Game [46] on the Right.

3.2.3. WebSign

WebSign was one of the pioneer projects that aimed at translating Arabic text to Arabic Sign Language (ArSL) [47] [48] [49]. In the first implementation of the project, they developed a new XML-based descriptive language to write signs. That is, a description of the sign was written in this language in a way that the Avatar they created could be animated. An interface to help the acquisition of the animations was created, where the user could select a joint and use sliders to change the rotation. A keyframe system was implemented, where the user could go to several points of the animation and rotate the joints there. This is essentially a simpler version of an animation software like Blender or Maya. Later in the project they started developing an Arab gloss notation, since some words can have different meanings and, consequently, different signs. Gloss notation allows more context to be given to a sign. However, not much is known about how the Avatar was changed after this implementation and the development of the project seems to have come to a halt as of the writing of this document. The visualization of the Avatar was implemented using Adobe Flash Player, which was blocked in

2021, and there is no public demonstration of the Avatar animation, just one of the animations being created, which can be seen in Figure 10.

Figure 10 Demonstration of Creating a New Animation in SML.

3.2.4. Robot-based Arabic Sign Language Translating System

A more recent approach to ArSL translation is the one demonstrated in [50], which uses the Pepper robot [51] to display the signs. The robot's functionalities and movements can be controlled with a Python package. It uses a speech recognition module to detect Arabic and translate it into signs. However, the process of creating the signs requires taking the measurements of the robot's joints manually, in degrees, and then converting into radians. This resulted in only 39 animations being programmed, which is far from the full spectrum of the ArSL. No mention is made to the transitions between signs. To help users in case they do not understand a sign, an image of each sign is displayed on the screen of the robot. The robot signing in ArSL and the images of the signs can be seen in Figure 11.

Figure 11 Pepper Robot Signing in ArSL [50].

3.2.5. ATLAS Project - Italian Sign Language

The Automatic Translation into sign LAnguageS (ATLAS) project for Italian Sign Language (LIS) [52] implemented a solution to employ virtual signers, or Avatars, in scenarios like web pages, mobile devices and television broadcasting. It uses a AEWLIS representation structure, which takes the Gloss notation and improves it. While analysing the sentence, not only it determines the glosses, but also if the sentence is declarative, imperative or interrogative, if the sign is singular or plural, and the context of the sentence to help distinguish between some specific signs. The production of the animations using motion capture techniques and human interpreters was attempted, yet it proved to be ineffective due to magnetic interference. Therefore, all animations were produced by expert animators, which took longer but had less imperfections than the motion capture data. Facial expressions were also animated by the experts and implemented in the system. The animation engine used for the Avatar was the Enthusiasm system [53], which has blending techniques to help with transition, and supports animations of skeletal bones and joints, blendshapes and a mixture of both. When the system is initialized, a new instance is opened, and the scene and animations are loaded. The only way to test the functionalities of this player is through the command-line. The preliminary evaluations from professional interpreters determined this system as promising, even if it is considered “a bit clumsy”.

Figure 12 Example of Two Signs from the ATLAS Project [52].

3.2.6. Mexican Sign Language Speech-to-Sign Language System

In [54], a translator from Mexican speech to Mexican Sign Language (MSL) is presented, where an automatic speech recogniser detects the text, translates it to MSL, and displayed the signs using an Avatar. A sign dictionary of 70 words

was created using Kinect animations. However, since Kinect does not capture hand or facial data, an algorithm that predicts the finger movements was used to capture the hand configurations. In some cases, the sign required more complex finger positions than the algorithm could predict, so manual refining was necessary. The facial expressions were defined as future work in the conclusions of the article. The engine used to run the Avatar was DAZ Studio 4 [55] and the default Avatar, called *Genesis*, was chosen. There is no information regarding the transitions between animations in the paper, but when searching in the DAZ Studio documentation there does not seem to be any way of creating transitions with this program. The Avatar module was integrated within a Graphical User Interface, namely, the GUIDE toolbox of MatLab [56], so that users can translate words and sentences and see the Avatar. From the tests carried out, both the students and the MSL teachers understood the signs and sentences with over 95% accuracy.

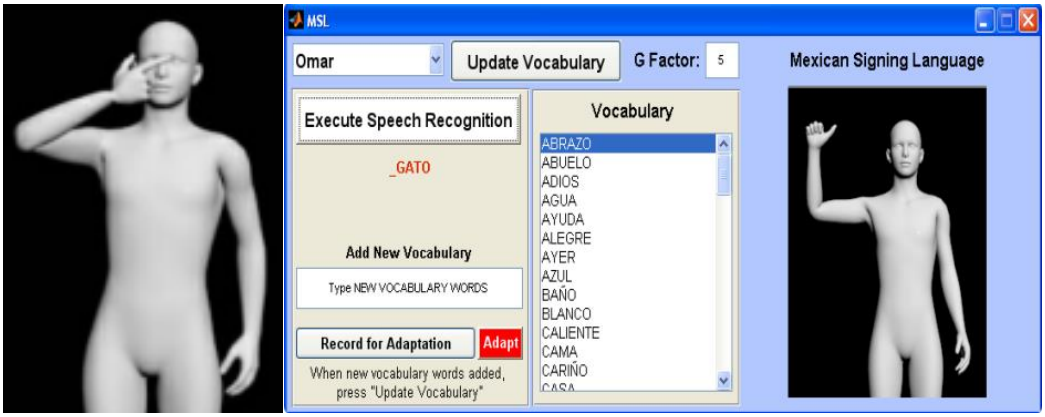

Figure 13 Avatar Signing in MSL on the Left / The MatLab Application on the Right [54].

3.2.7. Spanish Sign Language

One approach to translating Spanish to Spanish Sign Language (LSE) is demonstrated in [57]. The main goal of this project is to translate voice and PowerPoint presentations to LSE through an Avatar, as well as creating a chat for deaf students and teachers to communicate. The body animations are captured using an Optitrack [58] system of 11 infrared cameras and a suit with 34 reflective balls. The CyberGloves [59], with 18 sensors, were used to capture the hands and fingers. After that, the body and hands animations were synchronized using a 3D animation software. To integrate and animate the Avatar, a system based in Cal3D [60] was developed, and to create the transitions between animations, they determined the ideal point of interpolation

for the start and finish of all animations. With these points, they were able to determine when one animation can transition to another without losing the meaning of the signs. Although facial animations were created, they were not the facial animations of the signs, they were only some expressions to make the Avatar look more human. This means that facial expressions of the signs are still needed to complete their meaning. The results of the evaluation showed that the deaf students considered the Avatar useful, but it needs improvements such as: being able to configure the speed of the animation, extending the vocabulary, adding facial expressions, and refining some signs which were not clear enough.

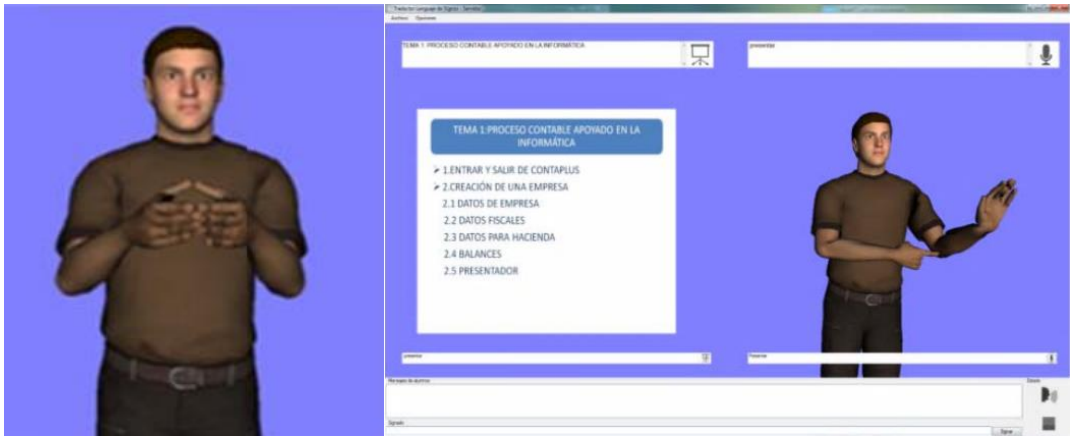

Figure 14 Example of the Avatar and Application for LSE [57].

3.2.8. Virtual Indian Sign Language Interpreter

One approach to translating into Indian Sign Language (ISL) is proposed in [61]. The system is gloss-based and has 1500 signs that were manually animated using Blender. To help animating the mouth movements they used the LipSyncPro plugin [62], which is one step to implementing facial expressions, even though other parts of the face are required too. The engine used for the avatar was Unity, and although they did not specify if they created transitions, Unity creates them by default when using the state machine. When testing, the feedback they got was to add non-manual features, specifically the head and whole-body movements to look more realistic.

3.2.9. Speech to Indian Sign Language System

Another implementation for ISL proposes a solution for bidirectional translation [63]. To capture all the basic ISL signs, the Kinect system was used in a well-lit room with minimum background disturbances. Maya was used to create the avatar, and Blender to map the animations to the avatar and export

the animations to FBX files. All the animations were imported to Unity, which means they did not store the animations externally, and consequently made a larger program. Facial expressions were not integrated, and hand motion was animated manually, since Kinect cannot capture it. From the tests they performed, the application received an average of 7.6 in usability by the evaluators, and most signs were understood by the users, with a few exceptions that were rectified and fed back to the system.

3.2.10. Video Solution for Romanian Sign Language

A different approach to this solution is shown in an implementation for Romanian Sign Language (LSR) [64], where the use of videos instead of an avatar is used. The problem with using videos is that there is no possible way to implement transitions, so in an attempt to solve this issue, they also captured some full sentences. When the translator receives the input, it tries to find the expression in the database. If the expression exists, the video is shown to the user, and if not, the translator tries to find each word individually. If a word is not found in the database, it is spelled out. This means that in the situations where the input is found as a whole sentence in the database, the output is even better than avatars, but when it is not found, there are no transitions, and the translation is slower than the one in an avatar. Since the videos are recorded by professionals, the signs are always correct, including facial expressions, which seems to be the biggest problem with avatars. The application was intended for mobile use, so it was developed in Android Studio [65]. There was no formal testing done to evaluate the implementation, the author only showed this to some deaf children which enjoyed the application and mentioned that it was easy and fun. An example of the application is in Figure 15.

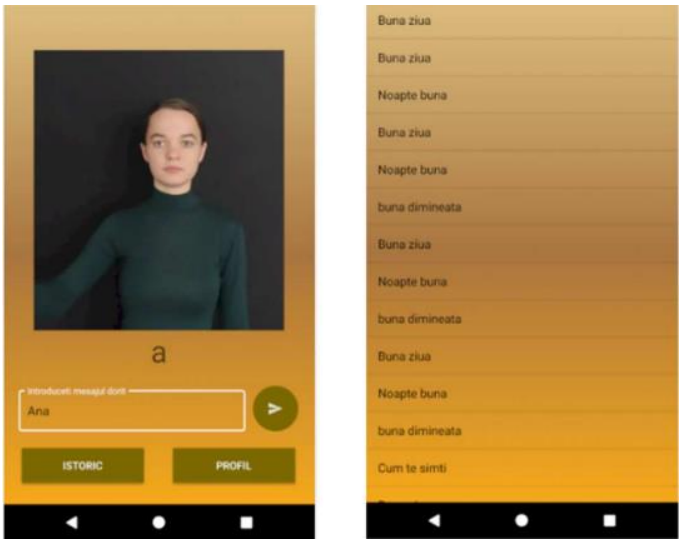

Figure 15 Main Activities from the Solution for LSR [64].

3.2.11. Holographic Sign Language

A recent solution proposed for ASL and Signed Exact English (SEE) makes use of the Microsoft HoloLens [66] to display a holographic avatar, through Mixed Reality [67]. The acquisition of the animations was done with some of the best technologies available. For the body, hands and face, three different systems were used, the OptiTrack, the StrechSense [68] and the FaceWare [69], respectively. The MoCap data was cleaned using Maya and exported to Unity. In Unity, the state machine was used, along with animation controllers to determine which animations are playing. However, the state machine does not look optimized, as it is very confusing from the picture they provided (middle of Figure 16). Although there are no tests available yet for this project, they explained that for future work they intend to conduct a formal study with 12 ASL users to evaluate the quality of the animations and the system.

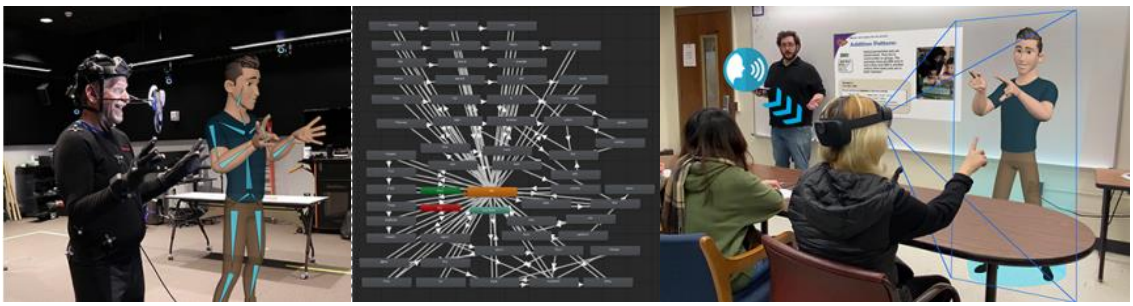

Figure 16 Capturing the Animations (Left) / State Machine (Middle) / Mixed Reality Classroom (Right) [67].

3.2.12. Sign4PSL

The Sign4PSL [70] project's main goal is to develop a lightweight multi-platform application that translated English to Pakistan Sign Language (PSL).

The dictionary was filled by an expert in both PSL and HamNoSys [71]. The latter is an alphabetic system describing signs on a mostly phonetic level, contrary to Gloss notation which associates a name to a sign. After the expert writes the signs in HamNoSys, they are converted to SiGML [72], which is another writing system for SLs, and stored in a database. The Avatar used was developed by another project, named JASigning [73], to facilitate the implementation process. The problem with using notation systems that describe the sign is the fact that the animations look more robotic, as the Avatar is following a script. Furthermore, the HamNoSys notation does not support non-manual features like facial expression and there is no way to control transitions since the Avatar is external, which is also the reason why the evaluation made in this paper was focused only on the translations. Finally, writing in HamNoSys is not an easy task, there are not many people who can do it and, even for those who can, it takes a long time to record each animation.

3.3. Key Concepts of the Related Studies

The related studies were evaluated in eight selected key concepts related to applications that translate to SLs. These concepts are:

- **Sign Languages** - The development of a system to translate text to SL is very complex, and consequently, when doing the systematic review, most projects have focused on only one SL. It is not essential for a project to support multiple SLs, but it is a sign that the project is constantly evolving;
- **Type of Display** - There are several ways of displaying the SLs to the user, the most common being a 3D avatar, and other being through robots, videos, or images. Each have different advantages and disadvantages which should be taken into consideration when developing a system like this;
- **Method of Acquiring Body Movements** - One of the three components of every sign is the movement of the body. The way the body animations are captured may determine if the animations are fluid or robotic. Although MoCap has some drawbacks, it is usually associated with more realistic animations. As for manual animating the

signs, depending on the experience of the animator, they can look almost as fluid as MoCap, or they can look more robotic. Finally, when using external avatars that use notations like HamNoSys, the result is much more robotic, since the avatar is limited to the movements of the notation. Even though robotic animations can be understood, they might make the application less favourable and desirable;

- **Method of Acquiring Hand Movements** - The second component of every sign is the hand and finger movements. For most SLs there are a predetermined number of hand configurations that all signs make use of. These can be acquired manually or using MoCap and, like the body, can help the sign to look more fluid or more robotic. In most cases, a sign uses only one configuration of the hand, so some projects have created SL hand configurations and associated one with each sign. However, there are examples where a sign uses more than one configuration of the hand, changing from one to another in the middle of signing. This is the problem that arises when assigning one configuration to each sign, one that does not exist when using MoCap to record the hand movement of each sign;
- **Facial Expressions Implementation** - The third and final component of each sign is the non-manual features, specifically the facial expressions, which for non-speakers of SL might seem like a less relevant factor, but it is still very important. Facial expressions are more difficult to animate manually and to integrate in an avatar, since they use blendshapes instead of bones and joints. For these reasons, most projects do not integrate them, ignoring them or putting them down as future work. Nevertheless, in recent years they have been more evident than in older projects, which is an indication of evolution in this respect;
- **Engine** - The engine is an evaluation more focused on avatars and videos, since the engine used to implement the virtual interpreter helps to understand the project's capabilities. An engine that can export to multiple operating systems gives a project the flexibility to reach more users, while an engine that focuses only on animation and cannot export to a browser or mobile phone, might not be viable to be used

by deaf communities. Moreover, engines that are deprecated might not give the necessary support to the development of the application, either through customer support or documentation. These are important considerations when choosing an engine to create the application. This concept is ignored in the studies that have used robots, as they usually have specific software, and external avatars in which the study does not use the engine;

- **Transitions** - Adding transitions between signs can give realism to any application that displays SL. In videos this is not possible unless the complete sentence is recorded, but in avatars and robots it can be implemented or not, depending on the engine to a certain extent;
- **Results from the Evaluations** - The results presented in the studies may help to understand if the solution was viable and changes that it needs. The results that were relevant to this review were the ones regarding the understanding of the SL performed to users of the language.

3.4. Discussion

Using the key concepts mentioned above, the selected studies from the systematic review were evaluated and the results were split into two tables (Table 2 and Table 3), since they could not be condensed into a single table. After analysing the results, it can be determined that although these systems have advanced over the years, they also lack key features such as achieving virtual interpreter signing that closely resembles the natural fluency of a human interpreter and availability to multiple platforms. The former is influenced by how the body and hands animations are gathered, if and how facial expressions are implemented, and whether transitions are available. The latter pertains more to the engine where the virtual interpreter is implemented.

Table 2 Results of the Key Concepts 1-4.

Study	SLs	Display	Acquisition of Body	Acquisition of Hands
PE2LGP [15] [37] [38]	LGP	Avatar	Manual and Kinect	Manual, One Configuration per Sign

Virtual Sign [42] [43] [44]	LGP (currently adding 5 more)	Avatar	Manual	Manual, One Configuration per Sign
WebSign [47] [48] [49]	ArSL	Avatar	Manual	No
[50]	ArSL	Robot	Physically Measuring Robot	Physically Measuring Robot
ATLAS [52]	LIS	Avatar	Manual	Manual
[54]	MSL	Avatar	Kinect	Prediction Algorithm
[57]	LSE	Avatar	Optitrack	CyberGloves
[61]	ISL	Avatar	Manual	Manual
[63]	ISL	Avatar	Kinect	Manual
[64]	LSR	Video	Video (Professional)	Video (Professional)
[67]	ASL	Avatar (Mixed Reality)	Optitrack	StretchSense
Sign4PSL [70]	PSL	External Avatar	Professional in HamNoSys	Professional in HamNoSys

Table 3 Results of the Key Concepts 5-8.

Study	**Facial Expressions**	**Engine**	**Transitions**	**Results**	**Animations Captured**
PE2LGP [15] [37] [38]	No	Unity	Yes	Bad	60
Virtual Sign [42] [43] [44]	Only 6 Available	Unity	Yes	-	2400
WebSign [47] [48] [49]	No	Adobe Flash Player	-	-	-
[50]	No	-	-	-	39
ATLAS [52]	Manual	Enthusiasm	Yes	Good but clumsy	-
[54]	No	DAZ Studio/ MatLab	No	Good	70
[57]	Generic Ones	Cal3D	Yes	Good, Needs Improvements	2000
[61]	LipSyncPro (for mouth)	Unity	Yes	Needs Improvements	1500
[63]	No	Unity	Yes	Above Average	“Basic Signs”
[64]	Video (Professional)	Android Studio	Partly	-	“Limited number”
[67]	FaceWare	Unity	Yes	-	62 Sentences

Sign4PSL [70]	No	External	Yes	-	400

Most of the studies used a 3D avatar since these can now be implemented in almost any platform and can be easier to animate and create interpolations between animations to produce transitions. A lot of studies created the animations for the body and/or hands manually, mostly due to the cost of MoCap systems, but also due to the post-processing needed when capturing using these systems. The manual acquisition is a solid first step, but it creates clumsy and robotic animations. MoCap creates more realistic and human-like animations, has a faster acquisition rate per sign, but requires refining of the data, due to magnetic interference or drifting. Regarding facial expressions, it is probably the most important feature that has been implemented in the fewest number of studies, and in a lot of studies that did not implement it, the feedback received asked for that integration. The engine used can help in all of the other concepts, by having compatibility with different animation data and with blendshapes for facial animation, the support for transitions and by having multiple ways to export. Lastly, out of all the studies analysed in this literature review, only one was starting to implement more than one SL, which shows how hard it is to get good results in these implementations.

4. System Architecture

In this chapter, the developed system architecture is explained, as well as all the components that integrate it. Firstly, the overall system architecture is illustrated, emphasising two alternative routes for the implementation process that differ in the way the Avatar is shown to the user. The optimal route for the system will be evaluated and determined in the implementation chapter after some early prototypes have been made. Secondly, the workflow of the whole system is presented for each route, to clarify how the system behaves and executes after receiving a message from the user. Furthermore, the overall architecture highlights the connections to be established between our solution, the translator Application Programming Interface (API) and the user.

Following the overall system architecture, the focus shifts to the "Gloss-to-Avatar" module. This module is the core element of the system, as it plays a central role in receiving the text-based glosses from the translator API, establishing the correct connections to the databases so that the right animations are downloaded, and displaying these animations on the Avatar. For the purpose of implementing this, it was necessary to determine a method to store the animations, which allowed the creation of this module's architecture. Both topics, as well as related components, are explained in this chapter.

4.1. Background

The work presented in this document is part of a bigger project named IVLinG: "Virtual Interpreter of Sign Language"[1]. Its main goal is to provide a bidirectional translator system between LP and LGP, that can be used on a browser or smartphone, focusing on a hospital environment. This project was developed by First Solutions, IPVC and CCG/ZGDV Institute. Therefore, the project is structured in different sub-components to provide translations between Portuguese and LGP and vice-versa, as seen in Figure 17. The segment that translates from LP to LGP was developed by IPVC, and itself is divided into two

[1] IVLinG: Virtual Sign Language Interpreter, Project website: https://tech.ipvc.pt/projeto.php?id_projeto=184

sub-components. The first one translates from Portuguese into written words that represent a sign in LGP, named glosses. The second one receives these glosses and animates an Avatar corresponding to the translation, for the user to see. The reverse translation was implemented by CCG/ZGDV Institute and focuses on using a smartphone or computer camera to capture the user's signs. Through image recognition techniques, the system determines the corresponding glosses for the signs made by the user and attempts to generate the sentence in Portuguese that correspond to the signs. Finally, First Solutions developed a mobile application and a website that integrates both translation modules. The work described in this document relates to the second component of the LP to LGP module, which receives glosses and animates an Avatar accordingly. The IVLinG application and their servers will be mentioned, in this project, as the *IVLinG System* since some specific API connections are required for the communication between our system and the rest of the IVLinG System.

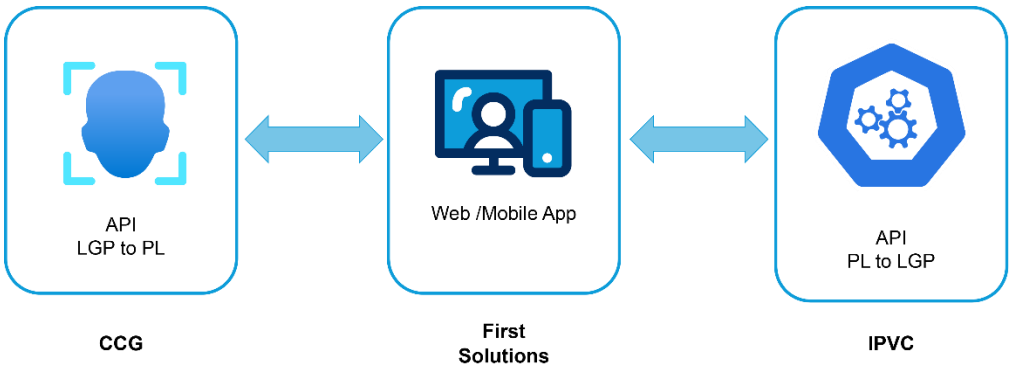

Figure 17 IVLinG Basic Structure.

4.2. Overall System Architecture

The overall system architecture of the LP to LGP system encompasses several components, regarding the connection to the user and to the translator API, the Avatar and other parts of the system. In this section, a discussion on how the Avatar is animated will not be made, nor how the animations are stored and downloaded, as this is a separate discussion made in chapter 4.3.

As seen in Figure 18, the basic framework of this architecture consists of implementing a system that receives audio or text inputs from the speaking user. In the case of audio, an extra step of converting to text is necessary with the use of a third-party speech-to-text system. After having the sentence in text, the translator API translates it into LGP glosses. With these glosses, which must be

stored in a database, the Avatar must be animated and presented to the user. However, the behaviour and workflow of the whole system changes depending on this last step, showing the Avatar to the user, since this could be achieved through various ways such as via livestream, video, or a web application with an integrated Avatar.

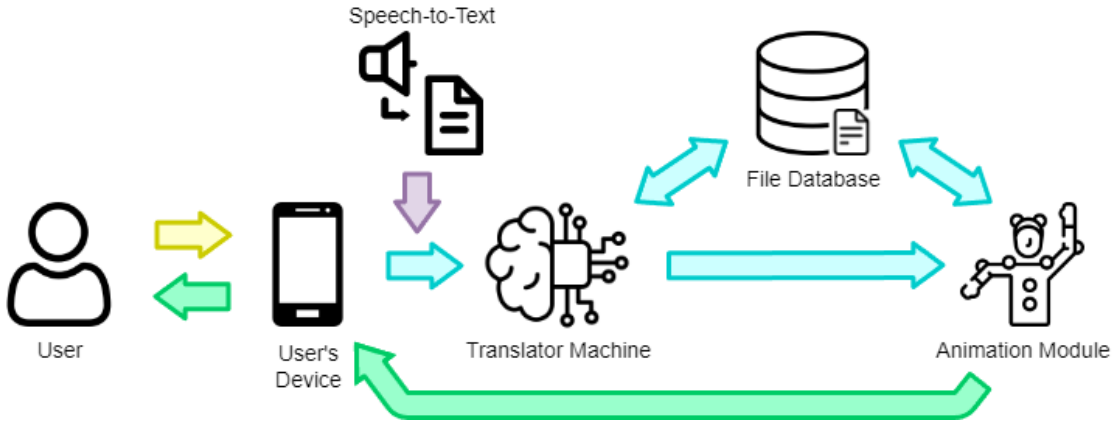

Figure 18 Basic Framework of the Architecture.

At first, the livestream option was thought of, with the intent of starting an instance of the Avatar on a server whenever two users initiated an online video call, through the application developed in the IVLinG System. This instance of the Avatar would be broadcast via livestream to users, creating a three-way video call. The speaking user would then write or speak to record audios of the message he would like to send to the translator API, which would be responsible for the communication with the speech-to-text service, in case of audio files, and the translation. The translation in glosses would be sent to the server with the Avatar, which would be animated while livestreaming to the users.

Since it was not known whether live streaming would be practical to implement, a first attempt was made to connect the IVLinG System to the server using this route. The OpenVidu platform [74] was chosen for this task due to its “Easy to use” and “Easy to deploy” components, as well as multiplatform compatibility. However, even with the help of this platform, creating a livestream channel to the Avatar proved to be extremely complicated. After numerous attempts at integrating this software with our system, a communication between our server and the users could not achieve. The errors found could not be corrected using the free version of OpenVidu and, since there was no access to the paid version, the option of using live Avatar broadcasts was temporarily

discarded and should only be used as a last resort. This failure is mostly due to limitations of the free version of OpenVidu and would most likely be possible to implement with further testing and the paid version of the software.

Now the selection was narrowed to two alternatives, the video system that consists of recording a video of the Avatar and sending it to users, and the direct system which consists of integrating the Avatar in the user device with a web application. Before the implementation, it is not possible to anticipate which of these routes would be better, so architectures for both of them were created and developed. After implementing these routes, both were tested to determine which was best suited.

4.2.1. Video System Architecture

When creating this system based on recording the Avatar, the challenge was to create an architecture that allows the Avatar to be recorded while animating and then sends these videos to the user. Unity allows for the recording of the screen with a package called Unity Recorder. This feature allows developers to determine the names of the video and location to store them dynamically, which is essential to create multiple videos without them overwriting each other.

In this architecture, the communication with the user's device is made through an Extensible Messaging and Presence Protocol (XMPP) connection. After the user sends an input, the application sends it to the XMPP server located in the IVLinG System. The system then sends a message to our XMPP Client with the input, and the Client sends the message to be translated. Knowing this, the process of designing the architecture can start.

As depicted in Figure 19, the process starts when the user inputs a text or audio, and it is sent to the system through the XMPP Server. The XMPP client checks for the call ID and associates a Unity instance, which is running in the server. The association consists in the client sending a message to an available Unity instance with the ID and the user input. Unity, at this stage, is responsible for sending the input to the translator API through websockets. The translator firstly detects whether the input is text or audio, and if it is audio, this input is sent to an external speech-to-text converter, so that the message is converted into text. The text message is then translated using a hybrid approach of neural networks and rule-based translation [75], [76] and sent back to the Unity System.

Following this, the animations are downloaded from a database, and the Avatar is set to animate according to the order of glosses made by the translator API. When everything is ready, a trigger activates both the recording of the video, and the animation of the Avatar. However, the recorder needs to have a variable with how much time it will record before starting the recording. To calculate this, a variable was created to sum the length of each animation and use it to program the time of the recording. With this, the recording stops precisely after the Avatar has animated all the glosses. The recorder then needs to render the video and saves it to a predetermined folder with the name of the call ID. After this, Unity pings the XMPP Client indicating that the recording is done, and the video's name. The client then sends the video to the XMPP Server by encoding it to Base64, which is redirected to the user's device, allowing the user to see the animation.

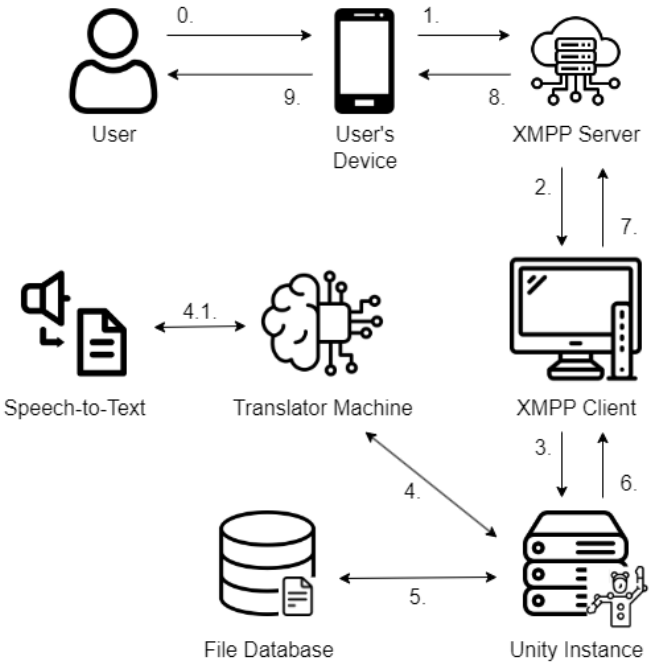

Figure 19 Video System Architecture.

4.2.2. Direct System Architecture

When creating the system ready to be used in the user's device, it becomes easier to display the Avatar to the user, but it becomes harder to communicate through websockets. A lot of changes had to be made to the implementation and architecture. Firstly, the communication between the user and the Avatar does not go through the XMPP server, as this is integrated into the application installed on the device. After the user inputs a message, Unity immediately has access to

it, before translating, so translator API was changed to receive Hypertext Transfer Protocol (HTTP) Requests instead of websockets. These changes remove the need for using both XMPP and websockets to communicate between systems, at the cost of running a 3D application on the user device.

As seen in Figure 20, it starts when the user inputs a text or audio and instead of sending it to the XMPP server, it is sent to the Unity instance running on the device. Unity then sends the input as it is to the translator API via HTTP Request. The translator API is responsible for analysing the input and converting into text when the input is audio. After that, the translation process happens, in which the text is converted to glosses. Then, a response is sent to Unity with the translated glosses, as well as the animations that need to be download. The reason it needs to send both the glosses and the animations is due to the way these animations are stored as explained later in chapter 5.2.4. Storing the Animations. After getting the response from the translator API, Unity requests the animation database to download the required animations. When everything is downloaded, the Avatar starts animating on the device for the user to see.

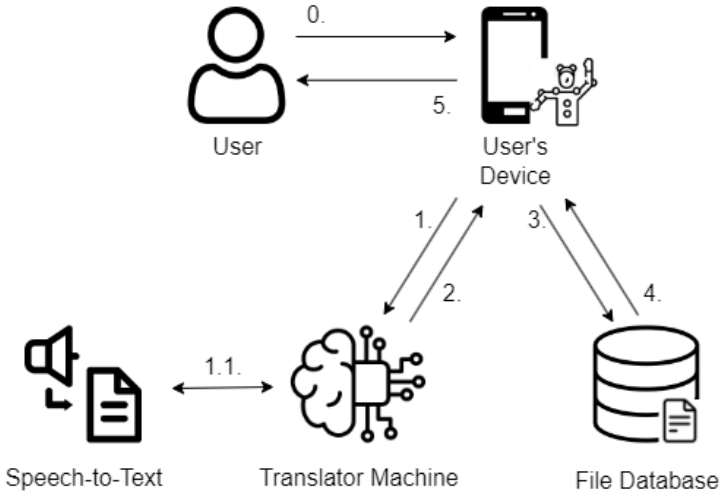

Figure 20 Direct System Architecture.

4.2.3. Systems Comparison

Each of the two architectures mentioned above have advantages and disadvantages. Since a decision will need to be made regarding which architecture will be used in the final prototype, it is important to compare them in advance and take note of the problems that each one may have, so that attempts to resolve them can be performed during implementation. The focus of this

comparison is the features and/or parts of the system that must be developed, since these are some of the project's objectives.

In Table 4 the comparison made can be seen, where an X means the system can implement that or is the best at that parameter, an O means the feature is possible with some limitations, and a blank space means the feature cannot be implemented.

Table 4 Comparison between the Video and the Direct systems.

Parameters	**Video System**	**Direct System**
Which requires less processing power from the user's device?	X	
Which is faster?		X
Which requires less bandwidth for downloads?		X
Can you change the Avatar animating speed?	O	X
Can you add subtitles?		X
Can you repeat an animation?	X	O
Is it possible to create an offline version?	O	O

At first glance, the Direct System seems to be the better system. Although it requires more processing power due to running on the user's device, it compensates for it, by being faster and using less bandwidth. This is the case because it only downloads animations, while the Video System downloads video files. Since the Video System runs on a server, it requires less processing power, at least on the user's device, but it becomes more difficult to make changes such as the speed of the animation or the addition of subtitles, whereas the Direct System can do that without much effort. It is worth noting that video can be sped up easily, although slowing down a video with a low frame rate will make it less enjoyable to watch. In terms of replay, the video in the first system, can be replayed infinitely, as long as the user keeps it on their device. However, the Direct System can only repeat the last message received, and if the user needs to repeat an older message, the animations need to be downloaded again. When

trying to create an offline version of the system, both have different limitations. The Video System cannot have an offline Avatar and, although videos can be downloaded and stored for later use, this is probably not useful for the user. This is because our application aims to help deaf users see LGP without the need of reading Portuguese, and having videos of LGP signs does not help them in this regard. In the second system, there could be an option, albeit far from perfect, in which the user could download a portion, or all, of the animations to their phone. However, there would be no translation and since LP conjugates verbs and LGP does not, most verbs would end up being spelled instead of animating the correct sign. Another option would be to download the translator too, but an application with all the animations and the translator would use up several gigabytes on the user's device, which is impractical for a mobile device nowadays.

4.3. Gloss-to-Avatar Architecture

The Gloss-to-Avatar is one of the most important components of this system, since without showing the animations to the user, the translation process is irrelevant. The architecture explained in this section regards the process of receiving glosses and animating the Avatar with the corresponding animations and can be implemented with either the Direct System or the Video System explained in the previous chapter.

In this part of the system, the main goals are to implement a system with a 3D Avatar with human characteristics that can be animated using pre-recorded files of LGP signs. The animations will need to be stored outside the application to reduce its size. This necessity applies to both the Direct and Video systems, albeit for distinct reasons. The Direct System runs on the user's device, and having all animations on the device takes up a lot of storage. Conversely, the Video System will have a dedicated server, with multiple instances of the application, and by keeping the application size smaller, more instances can be opened at any given time without upgrading Random Access Memory (RAM).

4.3.1. Avatar Animation

Unity, being a Game Engine, has features that facilitate the creation of this component. It allows the creation of a 3D environment with characters, such as an Avatar, that can be animated with ease. The applications developed in this

engine can be exported in several different formats, for Windows, Linux, Android, iOS, browser, and more, which is important since both mobile and browser versions are needed for the Direct Implementation, and a server version (either Windows or Linux) for the Video Implementation. It can also download assets while the application is running, through the use of Asset Bundles, an essential step as the animations must be stored externally, while still being accessible by the application for the Avatar to animate. To do this, the Asset Bundles can be filled with FBX files. Regarding the animation of the Avatar, Unity has the ideal tool called the State Machine, which allows for a single character, in this case the Avatar, to be animated dynamically. The character can have one or multiple states and each state is associated with an animation. The states can be linked together so the Avatar can move from one animation to another, with built in transitions to make for a smoother transition. As previously mentioned, it also has a video recorder feature that can be used for the Video Implementation. For these reasons, Unity was chosen to be used in the implementation of this application. However, there are some issues that cannot be resolved within Unity.

4.3.1.1. Storing and Accessing the Animations

The Asset Bundles that contain animations are stored in a database/file system located in a server. This file system is only responsible for storing the Asset Bundles, and with HTTP Requests, the requested files are downloaded. However, one of the issues of using Unity is that it does not have a way to know what animations are stored in an Asset Bundle without downloading it. In fact, the Asset Bundles can only be downloaded if the name of the bundle is requested, and Unity does not have that information either. To solve this problem, a middle step is required. This consists in a database, which will be called the “mapping database”, with a single table with four columns, which are the id, the name of the gloss, the name of the Asset Bundle that has the animation, and the name of the animation. It is important to note that the name of the animation is not always the same as the name of the gloss, which is why they are split up in the database. This is due to words that are synonyms in LP but have the same sign in LGP. In this case, the animation and animation name are the same, but the gloss name differs.

With the mapping database, Unity only needs to send the necessary glosses and gets in return all the information regarding animation names, glosses and Asset Bundles. Unity can then use that information to download the correct Asset Bundles from the file system.

4.3.1.2. State Machine

Another part of the architecture is the way the Avatar will animate the glosses according to the translation order. After downloading the Asset Bundles, Unity needs to animate the Avatar with the animations, and this is where the State Machine comes in handy. Since this is usually used in games to define different movements of a character, such as walking, running, idle, attacking, among others, the states are usually limited. This can be used for the Avatar by matching each state to an animation. When Unity has animations from the Asset Bundles, the state machine is organized so that the Avatar performs the animations according to the list of glosses from the translation. The state machine also allows the creation of transitions between states, to create smoother graphics and more realistic animation.

4.3.2. Designing the Workflow

After studying how Unity works and what it is capable of, everything could be brought together to design the workflow of this component. Figure 21 shows that after receiving the glosses from the translator API, Unity is responsible for searching for the animations. First, in the mapping database to get the Asset Bundles necessary, and then in the file system where the animations are stored in Asset Bundles. After downloading the Asset Bundles, Unity starts by obtaining the correct animations and storing them in a list, which is the queue used to execute the two states. When all animations are stored locally, the Avatar starts animating with the two states that follow the queue. In the case of the Video System, there is a trigger to start recording at the same time as the Avatar. At the end of the animation, the recording is stopped, rendered and exported. The XMPP would then send the video to the user. In the case of the Direct System, these steps are ignored, the user just sees the Avatar animating.

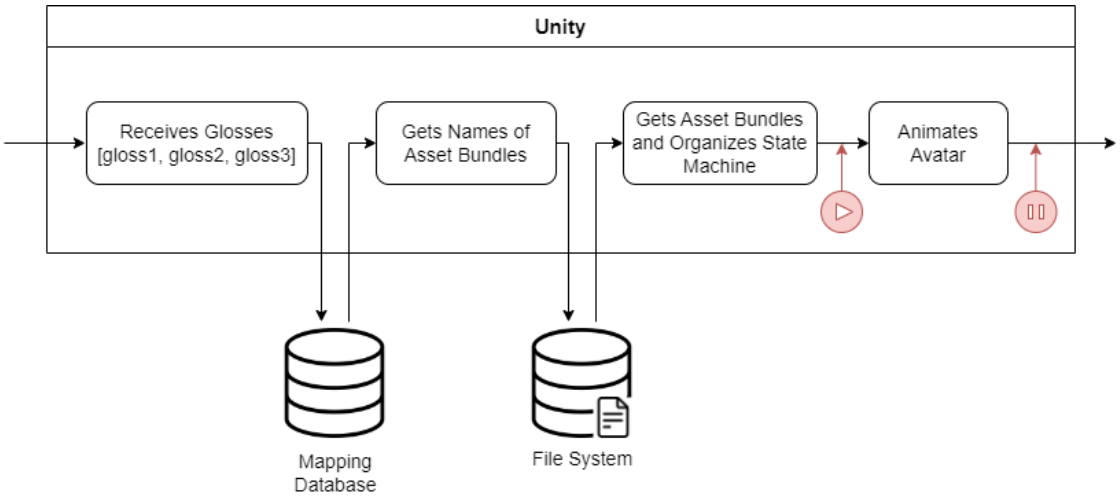

Figure 21 Gloss-to-Avatar Architecture.

4.4. Conclusion

This section has given an overview of the architecture of the LP to LGP system and one of its components, the Gloss-to-Avatar, offering more details on the aspects most pertinent to the work presented here, starting by showing what the basic framework of the system is. Then, the two possible routes for the system, the Video and Direct implementations, were explained with their respective architectures, followed by an initial comparison between them. It was also determined that Unity will be the engine to be used, since the architecture of this module highly varies on the capabilities of the engine, and what external sources will be needed. Regarding these external sources, two databases are necessary with this architecture, one being the mapping database that contains information about the Asset Bundles and their animations, and a file system that stores the Asset Bundles. A brief description of the state machine and why it is essential in this architecture was also given, before providing the full Gloss-to-Avatar architecture.

5. System Implementation

In this chapter, the complete implementation of the system is presented and explained in detail, which includes every component, the communications between components, and the Video System and the Direct System, which were presented in the architecture.

Firstly, the technologies used for the development of every component are presented, starting with a brief description of the hardware and software specifications, and then by evaluating and comparing two MoCap Systems and selecting the set-up that will be used for recording animations.

Following that, the development of the LP to LGP module is displayed, with focus on how the animations are acquired through the MoCap Systems. Then, how to get a 3D Avatar with a skeleton similar to the MoCap animations, and how to set it up using Unity to animate those movements. After that, it is explained how to store the animations to have them be downloaded in real-time when using the application. Furthermore, since facial expressions are a key-point in SLs, and LGP is no different, facial animations that were captured with the MoCap systems are implemented in the Avatar.

After having the Avatar ready to be animated, the implementation of the Video and Direct options is presented since, as mentioned in the architecture, they both need to be developed and tested before choosing which is the ideal route. Finally, an example of the final Avatar is shown.

5.1. Technologies and Software Used

This section delves into the technologies and software crucial for developing the system, as they define its capabilities. Here, all components are described and a comparison of two MoCap systems is made to determine the optimal choice for the final system.

5.1.1. Hardware and Software Specifications

The Avatar component was developed using Unity Engine, version 2020.3.38f1 for Windows 11 Education. This engine was used due to the wide

range of available tools, documentation, and other resources. The mapping database was created with the support of a LAMP (Linux, Apache, MySQL, PHP) server with PhpMyAdmin. Regarding the hardware, the machine that was used to develop these components has the Intel 10900K (10 cores, 20 threads, 3.70 Ghz) processor, the NVIDIA GeForce RTX 3090 24GB graphics card, 32GB of DDR4 RAM, and 3TB of storage (1TB SSD + 2TB HDD). Concerning the body MoCap, the Rokoko Smartsuit Pro II and Perception Neuron Studio were tested with the software Rokoko Studio and Axis Studio, respectively. The face MoCap tested were the Rokoko Face Capture Plugin for Rokoko Studio and the Face Capture Plugin for iClone 8. The animations were refined using the Autodesk Maya 3D animation software.

5.1.2. Selection of MoCap System

Regarding the animation of LGP signs, a manual approach, where an animator uses some 3D animation software and, one by one goes through the huge list of glosses, is a very time-consuming process. Furthermore, the animations tend to have a mechanical look when done this way, since humans have subtle movements when performing any kind of sign that are hard to replicate manually. To speed up this process and get a more genuine look to the animations, MoCap systems were taken into consideration. More specifically the IMUs, since they offer a cheap system that is capable of having a good level of accuracy, range and portability. As discussed in Chapter 2.3.5. OMS are too expensive and require huge laboratories, and the project did not have enough funding for that. IMS were disregarded due to their lower accuracy, despite being a more economical option. This limitation in the accuracy becomes significant when dealing with a large number of signs, each requiring extensive manual refinement, leading to the selection of IMUs instead. However, even among IMUs-based systems, there are a lot of different options, with different budgets, types of sensors, number of sensors, among others.

In this project, two IMU-based MoCap systems were tested and compared to see which fits best for our implementation, the Rokoko Smartsuit Pro II with Face Capture, valued at 5000€, and the Perception Neuron Studio with gloves and the iClone 8 Live Face Plugins, valued at 10000€. These systems are similar in the sense that both use IMUs to determine key positions of the body, however the

calibration process, the accuracy over long capturing sessions and facial capture process are all different. Several test sessions were made with each system in two separate locations before choosing the final set-up, and the results and conclusions of these sessions are described in this chapter.

It is worth noting that both systems need a device that runs iOS to capture facial animation. The need for iOS is because the systems use the 52 ARKit facial blendshapes, created by Apple and exclusive to their devices.

5.1.2.1. Rokoko Smartsuit Pro II

The first system tested was the Rokoko Smartsuit Pro II with the Face Capture Plugin included, and for the rest of the document this whole system will be mentioned as the Rokoko System. The Rokoko System, as seen in Figure 22, is composed of a full-body suits and smartgloves with IMU sensors. The body suit contains 17 sensors, one for the head, one on each shoulder, two on each arm, four on the torso, two on each leg and one on each foot. The sensors are all connected with wires inside the fabric of the suit, and even though they can be swapped to suits of other sizes, if people of different heights use the system, the other sizes need to be purchased separately. The smartgloves contain seven sensors, one on the wrist, one on the back of the hand, and one on each finger. The facial capture (Figure 23) uses the 52 ARKit blendshapes which, as mentioned, require a mobile device with iOS to be captured.

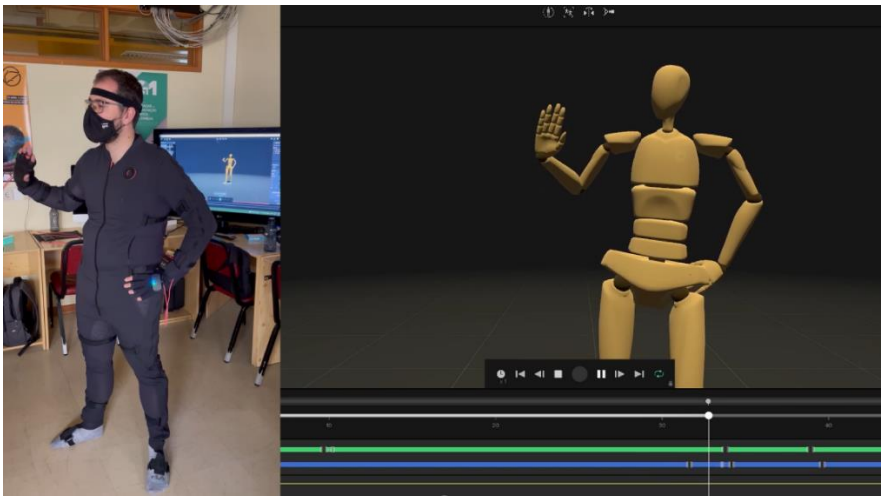

Figure 22 Example of Using the Rokoko Smartsuit Pro II.

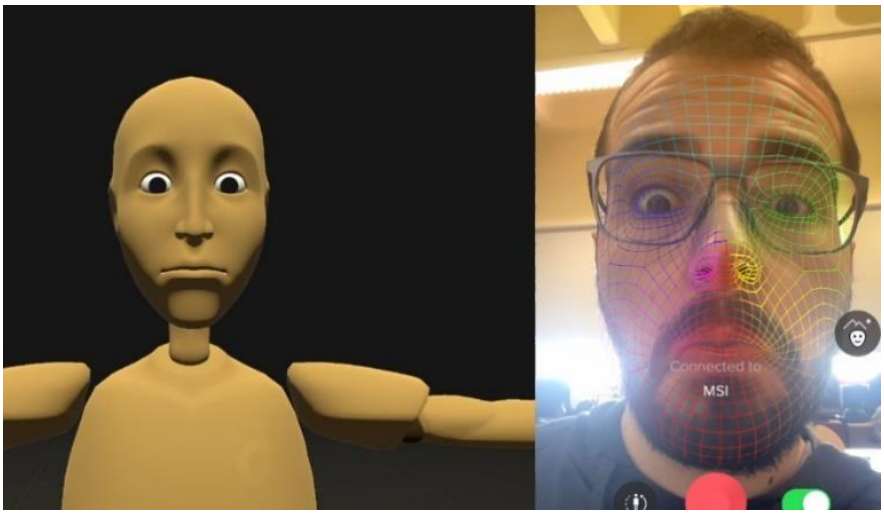

Figure 23 Example of using Rokoko Facial Capture.

The capture of the suit and gloves are controlled using Rokoko Studio, a free software developed by Rokoko, and the face is done with the optional yearly paid plugin for the same software. The iOS device only needs to have the Rokoko application installed and be connected to the same network as the computer running Rokoko Studio to connect it. After that, Rokoko Studio is the one software for all the MoCap.

The calibration is done for all three systems with one button, which the user can click on either on the computer or on the iOS application. A countdown from three to zero is initiated, and after that the user must stay still in a predefined position, as seen in Figure 24. Although the process is very simple, sometimes when calibrating it was noticed that the calibration was not done correctly and it required a recalibration, and at times even more than once.

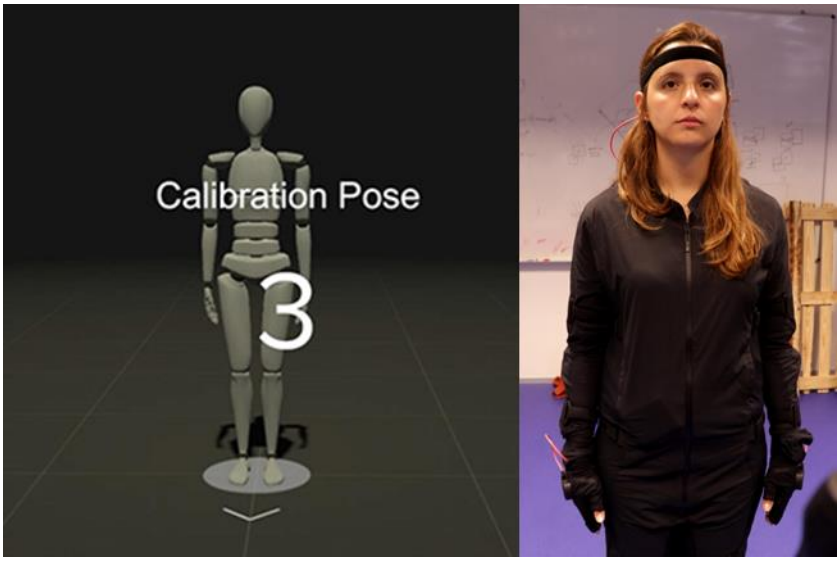

Figure 24 Rokoko Calibration Pose.

Drifting is also a problem using this system, which is when the MoCap data does not reflect the IMU real-life position. The most frequent occurrences are when one or more joints slowly get out of position during a capturing session. This is unnoticeable is capturing for a few seconds, but after several minutes there is a clear difference, and needs to be recalibrated. In rare occasions, recalibrating did not solve the problem and the system needed to be restarted or the session was ended. When capturing over 30 LGP glosses in one session, the drifting influences results more times than not, and post-session manual refining is required to fix the animations. Rokoko as of the time of the writing of this document is attempting to reduce this problem, for example by launching their Sensor Fusion 2.0 algorithm (free in Rokoko Studio) in February 2023 [77] and the announcement of Coil Pro in June 2023, which is still in pre-order as of October 2023. However, these were not released when our tests were made, so the accuracy of them is uncertain, and this document assumes the accuracy of the Rokoko System previous to the release of both the Sensor Fusion 2.0 and Coil Pro.

Another problem with the Rokoko System, which is common in IMU MoCap, is magnetic interference, which is somewhat related to the drift problem. Magnetic interference causes sensors to have a lower performance, hence giving wrong data and accentuating or enabling the drifting problem. This problem is present when the sensors are near ferromagnetic objects, i.e., objects that contain iron, nickel, cobalt, some alloys, among others, and although there are some software mitigations to this problem, it was still very common in MoCap, and in the Rokoko System. The previously mentioned Coil Pro is a hardware solution that will try to remove the problem with both drifting and magnetic interference, but as mentioned, it is still not available.

Rokoko Studio has a feature called actors, which are customizable models that try to mimic the real people using the suit. When you create an actor, you give them a name, usually the initials or name of the person using the suit, and sizes of several parts of the body, including height, length of the arms, legs, among others. This is used so the model represented in the model fits body proportions of the person. Even though during the testing, the LGP interpreter was measured and those values were set to the actor, as seen in Figure 25, the hands are clipping through one another. This is likely a problem related to

magnetic interference, rather than the model having longer arms than the interpreter.

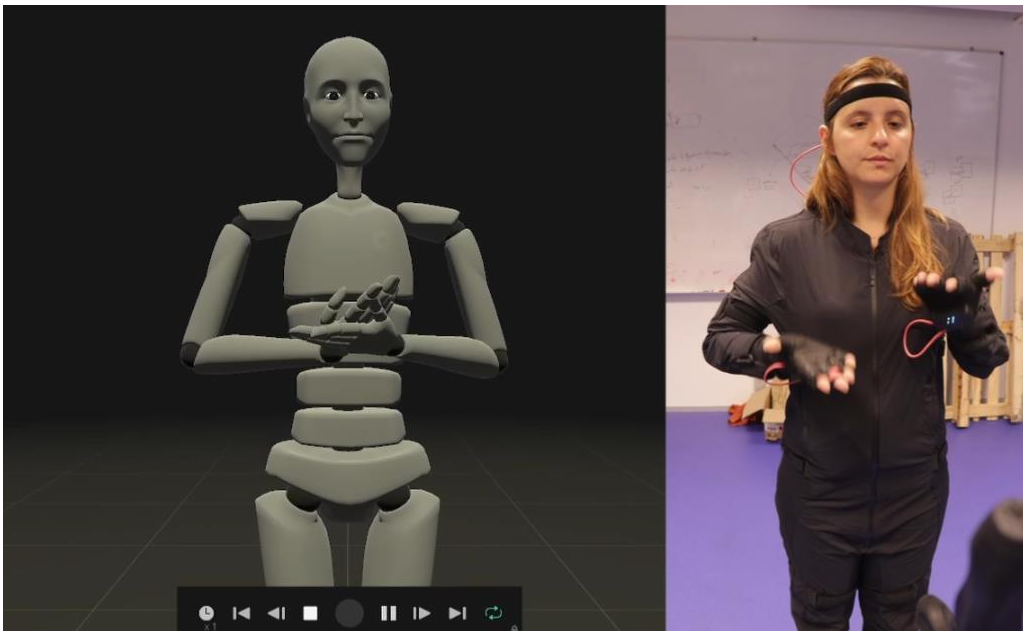

Figure 25 Example of Magnetic Interference on Rokoko System.

Rokoko Studio allows the export of MoCap animations to several types of files, including FBX which Unity is compatible with, and the mesh can be either included or excluded in the export. The face is exported separate to the body to make it easier to apply to other models, both in FBX files. The option to export multiple files at once is also available, and by not including the mesh it is possible to streamline the process of converting tens of sign animations to Avatar-ready FBX files, excluding manual refining.

5.1.2.2. Perception Neuron Studio

The second system tested was the Perception Neuron (PN) Studio with the Glove Kit and the iClone 8 Live Face Plugins, and for the rest of the document this whole system will be mentioned as the PN System. This system differs from the Rokoko System in several ways and the most apparent one is that it does not use a full suit, but rather several strips are secured in predefined key positions of the body which have compartments to hold the IMUs. These strips have advantages and disadvantages compared to the suit. On the one hand, they provide a one-size-fits-all MoCap option, which the Rokoko suit lacks, but on the other hand, in long sessions the strips can physically drift when the user is moving a lot, affecting accuracy. The PN Studio suit has 15 sensors, one on the head, one on each shoulder, two on each arm, two on the torso, two on each leg and

one on each foot. The difference from the Rokoko is only two less sensors and it did not seem to affect performance, at least when using them for sign language acquisition which was the case in our testing. Regarding the gloves, they have six sensors, one on the wrist and one on each finger. In comparison to the Rokoko System, the back of the hand does not have a sensor, but this also did not affect performance. On the contrary, the gloves of the PN System in our tests had better accuracy for LGP, due to the location of the finger sensors being closer to the nails of the users, while the Rokoko has the sensors on the middle of the finger. This small difference was enough to make the results of the PN glove much better in several signs. The facial capture uses the 52 ARKit blendshapes like the Rokoko System, and also requires a mobile device with iOS to be captured.

Axis Studio, a software developed by Perception Neuron, is a paid software, with a free license when buying the PN Studio suit. The face is acquired using a Live Face Plugin for iClone 8, a software developed by Reallusion, with the Live Face App for iOS. Similar to the Rokoko System, there are new technologies and updates to the PN System as well, one of them being facial capture inside the Axis Studio software through the new Axis Face iOS application. Whilst capturing facial animations using the PN System would likely be much easier and practical than using the iClone plugin, at the time that our tests were run, this technology did not exist as it was released in September 2023. Consequently, this system was not as simple to implement as the Rokoko.

To capture the face using the PN System, Axis Studio had to establish a connection with an iClone 8 that had the Motion Live Plugin. The Motion Live plugin is free and is required to activate the paid plugin for face capture, Live Face. By connecting the two programs, the iClone can be responsible for recording both the body and the face capture. However, when recording the body using iClone and trying to export the animations, the full mesh is saved in the exported file. This is not the case with Rokoko System or when recording body animation in Axis Studio. The solutions to this problem are either using files with all the information of the mesh or recording the face separately to the body, in iClone and Axis Studio respectively.

The calibration process in the PN System is more complex than Rokoko. The user has the option to perform up to six predefined poses, each having a bigger

focus on different sensors. This creates a more reliable calibration without taking much longer to do, and it rarely needed a recalibration unlike the Rokoko System.

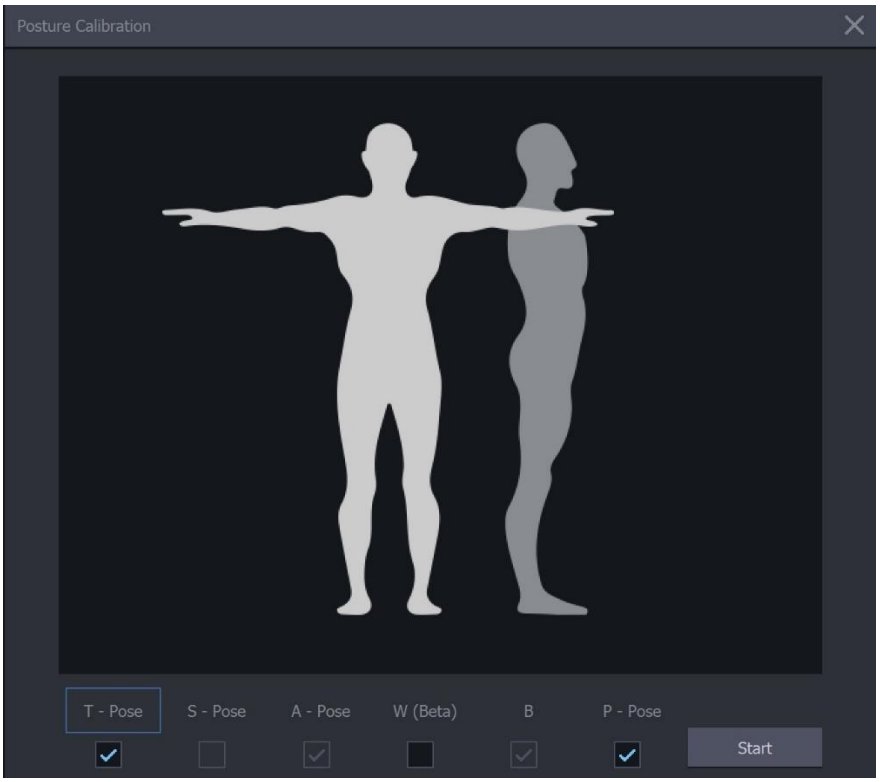

Figure 26 One of the Six Available Calibration Poses of the PN System.

Regarding magnetic interference, it is a big problem in most, if not all, IMU-based MoCap systems, and the PN System has this issue as well. Nonetheless, the PN System demonstrated better data compared to Rokoko System in both testing locations. This might be due to software, to the PN System having two transceivers that come with the suit, or a mixture of both. This also seems to affect the digital drifting problem, which was less prevalent in the PN System. That is, over long recording sessions, there were physical drifting problems in the PN System, as mentioned previously, due to the strips moving when the interpreter was doing multiple signs, but the drifting problems that Rokoko System had were due to poor performance of the sensors and magnetic interference, and this type of drifting was less commonly experienced in the PN System.

Much like Rokoko Studio, the PN System has a feature to change the size of the model according to the user and storing them, which behaved much like the actors in the previous system. Additionally, there is a feature unique to this system, which is the option to capture only the upper body, disabling the sensors on the legs and feet. This option might not seem relevant for most MoCap users,

but for our case it is helpful. In LGP, only the torso, arms and head are required to perform signs, and disabling the capturing of the legs meant less time when setting up the system, smaller sized files and less drift to worry about, since less sensors were being captured.

As mentioned, the export of the body MoCap on Axis Studio is similar to the Rokoko, having multiple file options, including FBX. However, when recording in the iClone application, the body and facial animations can only be saved in one file, which means the mesh always needs to be exported as well, whereas in Rokoko only the facial mesh is required, since they are stored in different files. It is possible to export them separately using the PN System but the process is rather complicated, since the body needs to be exported on Axis Studio and the face in iClone.

5.1.2.3. Final Set-up

After testing both systems, Rokoko an PN, the flaws of each of them became apparent. On the one hand, the Rokoko System had less accuracy in both the suit and the gloves, which was possibly due to the magnetic interference in the two locations we had access to, more drift in the suit, a simpler but less reliable calibration setup, and could only be used by people that could fit in the suit, or other suits needed to be purchased. On the other hand, the PN System face capture was external, making it harder to set-up and calibrate, since two programs were open at once, and iClone could not export the animation without the mesh, which Rokoko did.

This means that Rokoko had the better face capture, and PN had the best body capture. This reason and the fact that two interpreters, with very different heights, used the MoCap systems, led to the realisation that a hybrid solution would be the best approach. The PN System already needed two programs to be used when recording, but now the face recording could be switched from iClone to Rokoko, i.e., the final decision was to use the PN suit with the Rokoko face capture and manually calibrate both to work together.

5.2. Gloss-to-Avatar Module

The implementation of the Gloss-to-Avatar module is a focal point of the system, as it takes a sequence of glosses and animates an Avatar so the user

can understand a sentence without reading it or needing a human interpreter. The development of this module is explained in this chapter, and it was established and tested before implementing with the IVLinG System, to ensure that all parts of the Avatar and other related processes were working prior to deciding on the Video or Direct approach.

Firstly, it is described the implementation of a simple Avatar using a model provided by Rokoko Studio, simple animations created using the Rokoko System and a UI with buttons to define what gloss to be animated. The goal of this first implementation was to understand how to integrate external animations to the Avatar through Unity's state machine. Then, more animations were captured using the set-up explained in the previous chapter, so that the database of animations was bigger. After that, it is explained how these animations need to be stored, so the two databases mentioned in the architecture were created and filled with the MoCap data. After that, Unity was adapted to download those animations in real time and set the state machine to animate according to the glosses asked. While doing this, it was also decided that an upgrade to how the state machine worked was required for it to be more efficient and be able to animate an infinite number of glosses. Up until this point, the state machine was only functional for body and hand animation, which use bones to animate, and the next step was to integrate facial animations, which use blendshapes to animate. Finally, it is described how the communication to the translator API is set, so that glosses are received dynamically instead of using pre-set buttons.

5.2.1. Preparing the Avatar

The first step of this module is setting up an Avatar, albeit a simple one, to test how animating in the state machine works. Some simple animations were captured using the Rokoko System without the face, since this is a more difficult step to implement. The animations do not need to have a meaning, since they will not go into the databases, these are simply to test the Avatar. The Rokoko System allows the export of animations with and without mesh, and the size of the files can be reduced a lot when adopting the latter. Knowing this, only the mesh of the Rokoko default Avatar was exported from Rokoko Studio and imported into Unity as seen in Figure 27. Then, some animations were exported from Rokoko Studio without mesh, which means that the FBX files only contain

the animation of the bones. Doing this, it is possible to have one model, with mesh, that can be animated by all animations that do not have a mesh. To do this, the character needs to have a state machine, in Unity, and this state machine needs different states, each associated to an animation. Then, as long as the animations have the same skeleton as the model, Unity can connect the model to the animations with a simple step. This step consists of naming the animations with "*name_of_the_avatar@name_of_the_animation"*, for example, if the Avatar is named "Kevin" and the gloss is "HELLO", the animation must be named "Kevin@hello". This allows Unity to make the connection of the skeletons automatically, even when the animations do not have mesh. It is also possible to do this with files that contain a mesh, but those files are bigger and since this project intends to download animations in real time, the goal was to create the smallest animation file possible. Now, by creating some buttons that function as triggers to animate each state, it is possible to program the Avatar to animate different states dynamically, i.e., when clicking the button named "animation 1", it animates State 1 and goes back to idle, the same for animation 2 and the rest of the buttons.

In Figure 28 the state machine is demonstrated with several states created, each having a different animation. The first state, State0, contains the idle animations. When the application starts, the Avatar is set as the default to that state. After sorting animations to the other states, the link to the other states can be triggered, for example, with a button press. If the link from State0 to State1 is triggered, the Avatar starts animating State1. If the link from State1 to State2 is also triggered, it animates that state after finishing the previous one. This goes on until the list of animations ends and it goes back to State0.

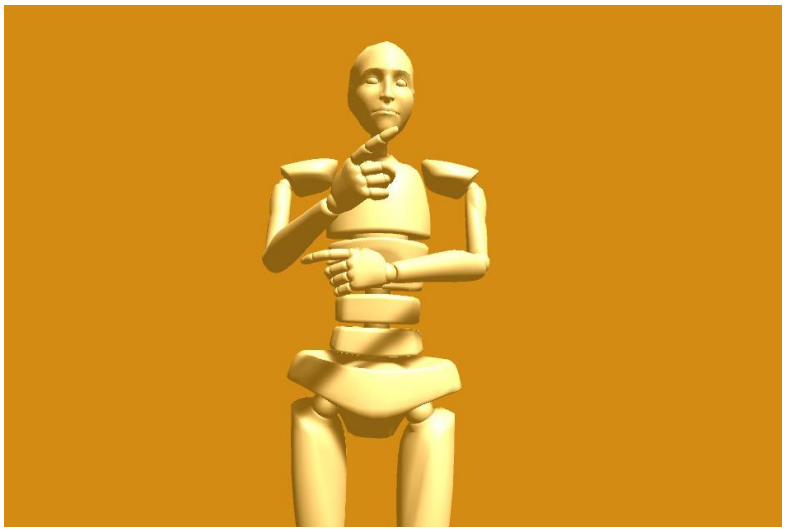

Figure 27 Example of the Rokoko Avatar animating.

Figure 28 Example of a State Machine with Multiple States Created and Linked.

5.2.2. Acquiring Animations using MoCap Systems

The next step is to acquire animations and fill the database with glosses. To get the most realistic animations, two LGP users, one teacher, native user of LGP that can read and write Portuguese, and one LGP interpreter, helped in the process of translating several words and sentences from LP to LGP, and then capturing the movements for them.

The first process was determining what glosses animations should be captured first. As mentioned, some sentences were translated but they were not chosen at random. Since the IVLinG project focuses primarily on translating between these two languages in medical settings and everyday needs, some dialogues that are common in these environments were written in LP and translated into LGP by the teacher and the interpreter. Some examples of these dialogues include a simulation of a doctor's appointment and some interactions

that may occur when asking for directions or while shopping. The dialogues consisted in around 60 sentences with over 100 different glosses. After acquiring all the glosses from these dialogues, it was concluded that the number glosses was not enough and decided to follow a list of the 600 most used glosses in LGP, according to [12]. This helped in capturing some common words that were not used in our dialogues. In total, 612 glosses were captured, which albeit far from the full spectrum of LGP, it covers most basic conversations, and sets a good starting point to test the system.

The second process was the acquisition of the determined 612 glosses by using a combination of the PN suit and Rokoko face capture. Out of the locations we had access to, the one with the least magnetic interference was chosen, even though Axis Studio showed several times that some sensors were having poor performance, as seen in Figure 29. Despite that, there were multiple sessions where magnetic interference was not a big problem. There were some cases where a single sensor misbehaved for a long time during a session, and it went unnoticed until after the capture had ended. An example of this is shown in Figure 30 where the sensor for one of the fingers was either not connected properly or malfunctioning, due to interference, and around 20 signs which were captured in this session had to be fixed due to this issue. A physical drift problem is also seen in Figure 31 where the shoulder sensor in contact with the interpreter's clothes started to drag down and makes it look like the shoulder was dislocated. In an ideal scenario, a re-recording of those animations would be made, but the time the interpreters had to capture was limited, and redoing animations would be spending time that could be spent recording new animations.

Figure 29 Example of Magnetic Interference detected in Axis Studio.

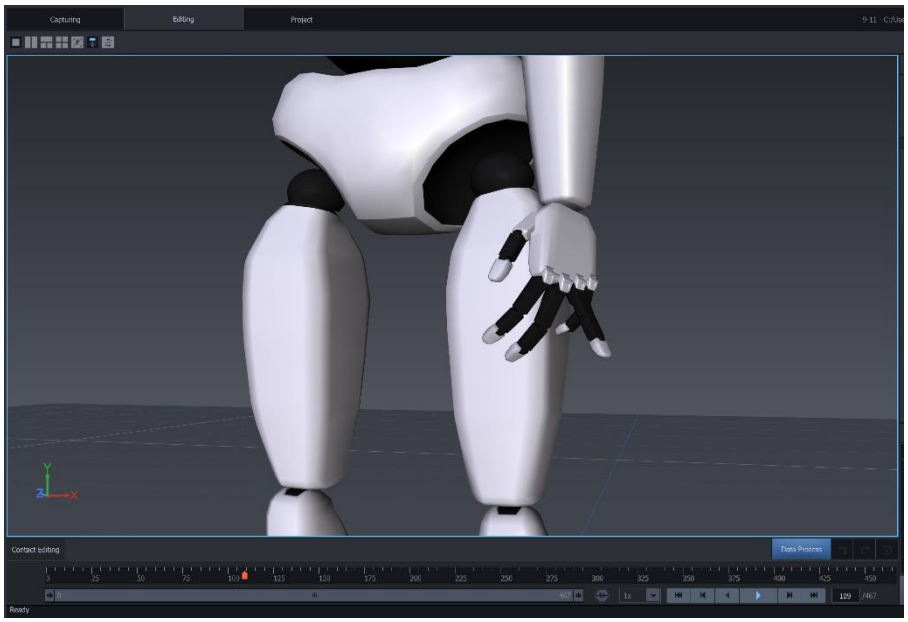

Figure 30 Magnetic Interference Causing Wrong Data of a Finger Sensor.

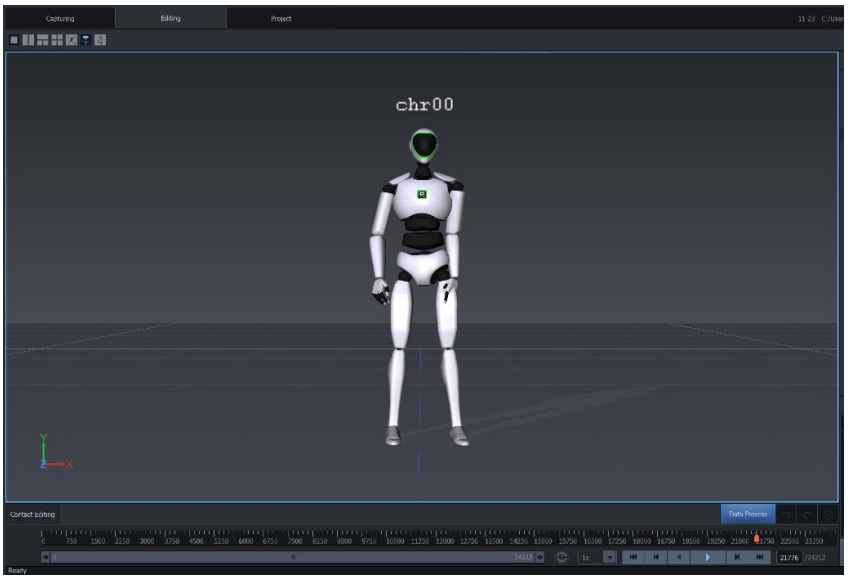

Figure 31 Drift/Physical Dislocation of the Shoulder Sensor.

Acquiring the animations is the first step and exporting them is the second. Since Axis provides several ways of exporting, choosing the best for this case is required. The animations will need to be edited in a 3D software and be used in Unity, and binary FBX files are ideal for both these situations.

Regarding facial MoCap, the animations were recorded in Rokoko Studio, using the iOS application like mentioned previously, at the same time as the body MoCap was captured and both animations were synchronized in the editing phase. However, while in the process of acquiring the animations, the license for the Rokoko Face Plugin expired, and we could not get funding to re-purchase it. This means that facial expressions were only gathered for 60 signs out of the 612, and the remaining only have body animations. However, the module for facial expressions was still developed, and it will be discussed later. Regarding the acquisition process, magnetic interference is not a problem with facial MoCap, as it is an image-based capture, so everything was recorded without the need of refinement. The animations were exported to binary FBX, just like the body ones, in order to have all animations in the same format. In the face FBX files, the mesh is required because of the blendshapes, and this process will be explained in more detail when the integration with Unity is explained.

5.2.3. Editing Animations

The raw body data from MoCap is not usually ready to be used in games or animations, and it is the standard to refine the animations. Although the goal of

this work is not to edit animations, the most obvious issues of drift and magnetic interference were fixed using a 3D animation software, Maya. Out of these two issues, magnetic interference is easier to fix since it can be detected which sensors malfunctioned and edit those sensors for the entire recording. That is, if the LeftArm sensor was 30 degrees rotated wrongly in the X-axis, using Maya's Graph Editor (Figure 32) it is possible to single out that axis and move it 30 degrees to the correct position for the entire animation. Drifting is harder to fix because usually one or more sensors deviate more and more from their initial position over the recording, and the Graph Editor cannot fix it as easily. To solve it, an animator needs to change the sensor several times during the animation, for example, every 50 frames, the sensor needs to be fixed.

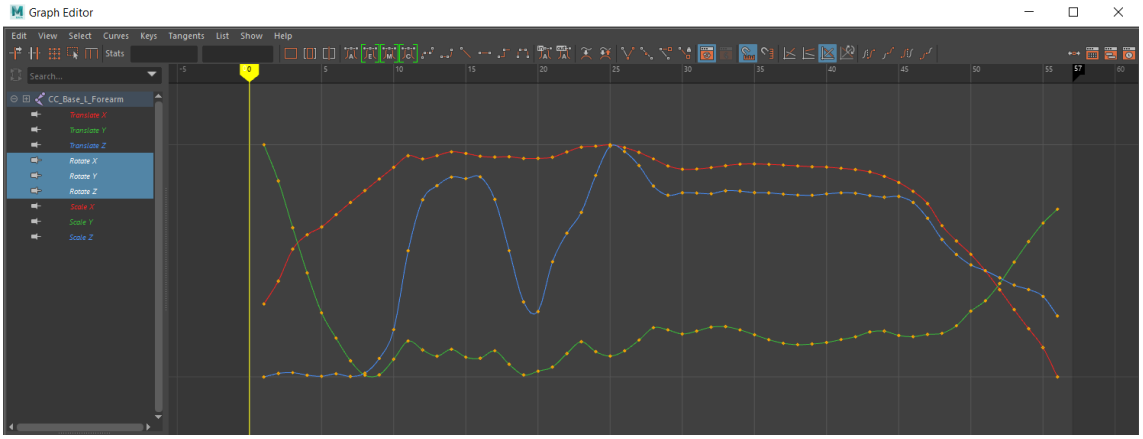

Figure 32 Maya's Graph Editor with the Rotation X, Y and Z of One Joint Selected.

Furthermore, the MoCap session was recorded in one take, to be easier to edit the interference using the method described above, and the animations need to be split into separate files. Maya has the option to export selected frames into binary FBX files, so the animations were split this way. Although the face capture did not need refinements, the step of splitting the animations in Maya was also done for those files.

5.2.4. Storing the Animations

After capturing and editing the animations, they then need to be stored in databases just like defined in the architecture. The mapping database was created with the support of a LAMP server with PhpMyAdmin. Following the architecture, a table was created in the database with four columns, the id, animation name, name (of the gloss), and asset bundle. To facilitate the process

of filling this database, the process of creating the asset bundles was automated, and it is explained later in this chapter.

As for the file system that stores the asset bundles, a folder was created in the same server as the translator API, and the asset bundles can be accessed and downloaded by using the URL to the folder, followed by the name of the asset bundle.

Regarding real-time download of assets, Unity allows it with one condition, they need to be inside an asset bundle, which are rather simple to implement. At first, the asset bundles were created manually and so it was decided that 50 animations were put into each asset bundle, to speed up the manual process. After some initial testing, the application frequently downloaded several asset bundles, for example, if the user needed four glosses and they were all in different asset bundles, four asset bundles would be downloaded. While using only body animations, this is not an issue, but facial animations, which have blendshapes, are much bigger and require more bandwidth to download, and requiring multiple asset bundles with 50 face animations each became too time and resource consuming.

To solve this, it was decided that one asset bundle would be created per animation and automating the process of creating asset bundles was needed. A dedicated application in Unity was developed, called AssetBundleCreator, with a simple screen that tells the user how to create the bundles. The user first needs to import the animations to the correct folder and click “Set AssetBundle Names”. This button is programmed to take each animation and assign it to one asset bundle, which is named after the animation, e.g., “HelloAssetBundles” stores the “HELLO” gloss. After that, the user clicks “Build AssetBundles”, which creates all the asset bundles and stores them in the preset folder. While creating the asset bundles, Unity is also programmed to write a SQL script with the data of the animations for the columns in the mapping database. As previously mentioned, a mapping database will be the intermediate between Unity and the asset bundles, by corresponding the gloss name to the asset bundle that contains the animation. This is useful even when using one asset bundle for an animation, because of synonyms. During the capture of the animations, it was noticed that several synonym verbs in LP have only one sign in LGP. For example, the verb “ter”, which in English means either “to have” or “to possess”, and the verb

“haver”, which means either “to have” or “to be”, are both the same sign in LGP. Storing the same animation twice in the database is redundant, and so two entries in the mapping database were input, one for “ter” and one for “haver”, both with the animation “Haver” and the asset bundle “HaverAssetBundle”. This way, when the translator API translates either of these verbs, the animation “Haver” will be downloaded and used. If, for example, “ter” did not have its own entry on the database, it would be spelled out, even though the animation for it was in the database with a different name. The same is true for some synonym nouns, adverbs and adjectives, so for the ones that were detected, extra entries were created in the database.

The SQL script automatically generated is ready to be inserted in the mapping database to input the data of the animations and bundles. However, the data in this database needs a script in order to be accessed from an external source. A php file was created and stored in the server (available in Appendix 9.2.1. Script to Get the Asset Bundles names), and it allows a HTTP GET Request to access the information in the database. The file simply connects to the database and makes a query asking for the animations, and asset bundles that match the name of the gloss. It returns a JavaScript Object Notation (JSON) with the asset bundles name, as well as both the animation name and gloss name. In Unity, the arrays of animations and glosses are compared to determine if there are synonym words, since if there are no synonyms then both arrays are equal. If the two arrays are different, a temporary IDictionary was created in Unity to store the synonyms, which are required to sort the avatar states according to the translations and when subtitles are implemented.

5.2.5. Animating the Avatar

The next step on this implementation, is to replace the Rokoko Avatar with a better one and animate it using the animations acquired. Firstly, an Avatar that looks more human is required, since the default Rokoko one was just a temporary and easy option to implement. One character from Character Creator 4 was used since they are designed to have the skeleton and joints that are compatible with our MoCap System. The new Avatar can be seen in Figure 33.

Figure 33 New Avatar in Character Creator 4.

Regarding the animation of the Avatar, in the first implementation, described in 5.2.1. , only a few states were created and to each was assigned an animation manually. This is useful in games where the character usually has a predetermined number of states, which correspond to idle, running, attacking, or others. However, in the case of the Avatar, there is no way of knowing how many glosses a translation will have. Some might have only one word, and some might have a lot. When considering that names are spelled out, as well as words that are not in the gloss dictionary, a sentence where multiple words are spelled out can require over 20 states. Unity does not seem to allow the creation of states while the application is running and so one solution would be to create 100 states since it would be unlikely that anyone would require more than those. There are two problems with this solution however, first is the rare occasion that someone needs more than 100 states, the animation would stop, and the second is creating and configuring 100 states manually is inefficient. Another solution that is harder to implement, but with much higher scalability, is to create three states only, one of them being the idle animation. The other two states play all the other animations, by swapping animations while the opposite state is animating. Looking at Figure 34, the flowchart of what happens can be seen. Assuming a sentence of four glosses, it would be as follows:

1. Downloads the Asset Bundles that contain the four animations.
2. Get the animations from the bundles and store them in a list, which works like a queue of animations waiting to be animated.

3. The first animation is set to State 1, and the second animation to State 2. The Avatar changes from the Idle State to State 1, animating the first animation.
4. After animating the first animation, the Avatar changes to State 2 to animate the second animation. While the second one is animating, State 1 is changed to the third animation.
5. After animating the second animation, the Avatar changes to State 1 to animate the third animation. While the third one is animating, State 2 is changed to the fourth animation.
6. After animating the third animation, the Avatar changes to State 2 to animate the fourth animation.
7. Since no more animations are left in the queue, the Avatar returns to the Idle State after finishing the fourth and final animation.

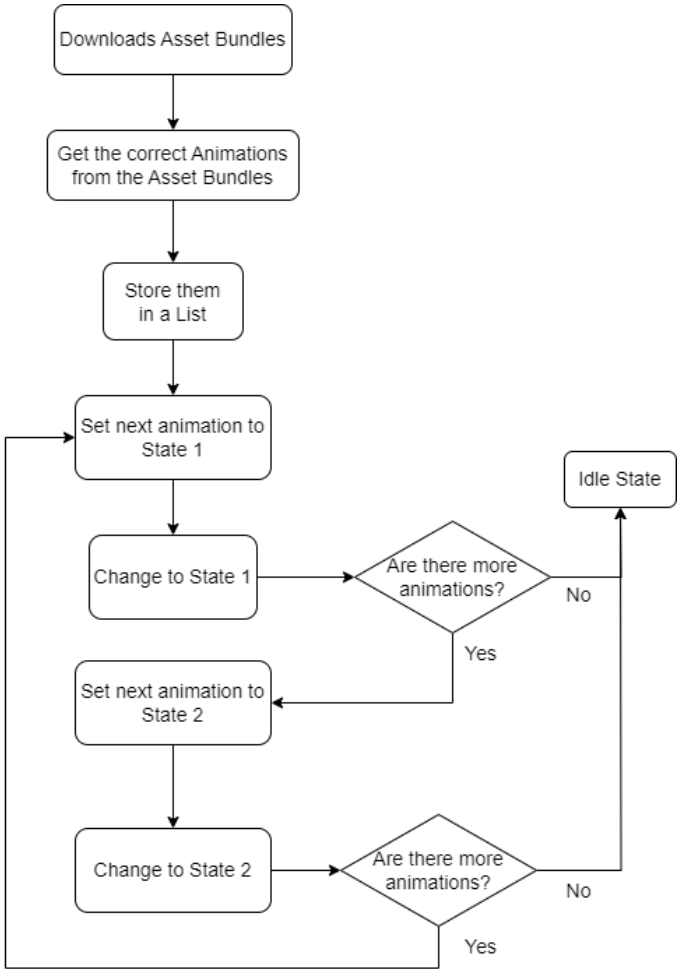

Figure 34 State Machine Flowchart.

With this method, only two states are necessary, other than the Idle, and can go infinitely as long as there are animations in the queue. Since one state

switches animations while the other is playing, it creates a fluid animation with no interruptions.

A better state machine is key to show the user the animations without errors, and connecting to the databases is another essential part too. Unity has a library called UnityWebRequest that helps the implementation of HTTP Requests. A GET request, to the script described in the previous chapter, is possible through a few lines of code. The response is in JSON, so it needs to be parsed in order to get the necessary information, in our case the asset bundles, animation names and gloss names. These three parameters were stored in arrays. The asset bundles array was used in a function that downloaded all of them from the file system. The gloss names and animation names sometimes are different, as explained, so they are both necessary to assure that the order of the glosses in the animation is following the order of the glosses in the translation, and that the asset bundles can extract and associate the correct animation to the corresponding state. The gloss names are also important for the subtitles, for example, when the user writes the verb “ter”, which is a synonym of “haver” as mentioned previously, the animation being animated by the Avatar is “HAVER”, but the subtitle should follow the translation given by the API which is “TER”.

5.2.6. Adding Facial Expressions

Unlike the body and gloves animations, the face needs to have the mesh in the FBX file for it to work, as mentioned previously. This is because the blendshapes are associated to a mesh and cannot be saved in a file without it. Although the Avatar and the Rokoko System have the same blendshapes, they have different meshes, meaning if a state machine is created for the Avatar’s face and the Rokoko facial animations are imported into it, the face would not animate. However, there is a way to solve this issue since the values of the blendshapes are just numbers, and with a script, a 3D model with blendshapes can copy the ones from another model, as long as they share the same blendshape structure. In this case, the standard 52 ARKit blendshapes are used in both the Rokoko Face Capture and the Avatar that was integrated in Unity. Knowing this, in Unity the default head from Rokoko was imported, Figure 35, and a state machine was created for this head just like it was done for the Avatar’s body. Even though the Avatar’s animations did not have mesh and the facial animations they are

needed, the state machine works the same way, as long as the mesh matches in both the model and the FBX with the blendshapes. This means that by simply associating a state in the head's state machine with an animation recorded in Rokoko, the head is animated accordingly. That is, if the name of the animations follows the norm. This is the same as the body, where the animation for the gloss "HELLO" needs to be named "Kevin@hello" to be animated by the Avatar Kevin. Following the same rule, the head is called "RokokoFace", so the animation for "hello" is "RokokoFace@hello".

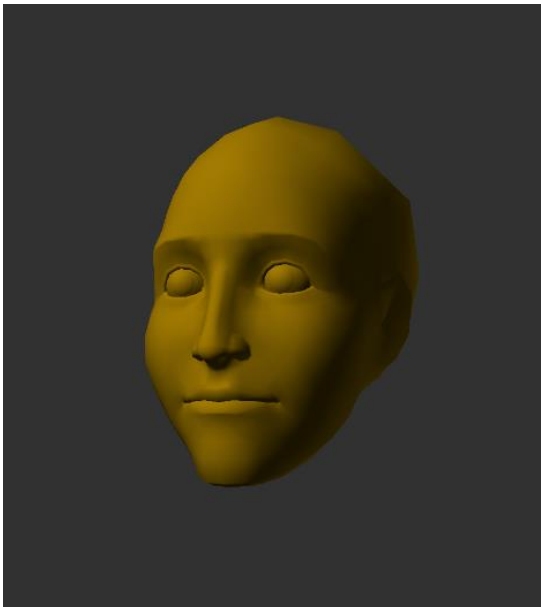

Figure 35 Rokoko's Default Head Imported in Unity.

To store the facial animations, the AssetBundleCreator program was used, with a small change so the asset bundles names had "face" at the beginning of their names. This means that a gloss, for example "HELLO", has the body animation "Kevin@hello" in the asset bundle "HelloAssetBundle", and the face animation "RokokoFace@hello" in the asset bundle "FaceHelloAssetBundle". Doing this, additional entries in the mapping database are not required for the face animations. When Unity receives a list of asset bundles, it is programmed to download both the body and face asset bundles. Then, it detects which asset bundles have the name that starts with "face", and associates those animations in the correct state machine, the one in the Rokoko head.

Although the facial animations are downloaded and animated, they are still in a floating head, which is not ideal for the user to see. In the beginning of the chapter, it was explained that blendshapes are just values that can be copied,

and that is the next step. The Avatar has a face with the 52 blendshapes, and by using a script, Unity can copy the value of each blendshape from the floating head and setting it to the corresponding blendshape on the Avatar in real-time, by using the functions “GetBlendShapeWeight” and “SetBlendShapeWeight”, making it look like the Avatar is animating using the downloaded animations, when in reality the floating head is using those animations and the Avatar’s head is just copying it. Figure 36 shows the Avatar animating at the same time as the head.

Figure 36 Avatar Animating by Copying the Blendshapes of the Rokoko Face.

Despite creating the module and acquiring facial animations of some LGP glosses, we had some problems with the license for the Rokoko Face Plugin, since it expired, as mentioned previously. This meant that this part of the module could not be finished. For the tests that will be run in the following chapter, the face feature is turned off since for most of the glosses tested, there was not a corresponding facial animation recorded.

5.2.7. Connecting to the Translator API

The connection to the translator API is the next important step in the implementation. Clicking button with predetermined glosses is not the goal of the project, the users want to input LP. Before implementing with the rest of the IVLinG System, a textbox was created in Unity so that text could be inputted in Portuguese and sent to the translator API. The answer from the API contains the translated glosses, which determine the Avatar animation.

Firstly, the buttons with predefined text were removed, and replaced with a textbox and a button “Send” which sends the message, by triggering the HTTP

Request used for the mapping database (Figure 37). However, this request was changed to be sent to a server instead of the mapping database. The server implementation is not described here as it is not part of this work, but this server is responsible for communicating to the translator API, receiving the gloss list, and immediately send a request to the mapping database asking for the animation names, gloss names and asset bundles. The response to Unity has all this information, as well as the translation made, all inside a JSON. The information is parsed, similarly to the previous request, and the process of downloading and extracting the asset bundles remains the same.

Figure 37 Avatar with Input TextBox and Send Button.

5.3. Integration to the Complete System

The next step, after having the Avatar animating with external animations, is to implement it with the rest of the IVLinG System. In the architecture, it was not possible to determine whether a video approach would be better or worse than a direct approach in this case, so both were implemented and tested.

5.3.1. Developing the Video Recording Module

The Video approach is the least resource consuming for the user's device out of the two options, however the time that it takes for the user to receive the translation is the key factor in this solution. After searching for ways to record videos using Unity, the Unity Recorder was the only way to implement this without resorting to an outside program. This is a legacy plug-in to Unity, meaning it used to be a part of Unity but now needs to be downloaded separately, that allows developers to record their applications or games. In this case, a recording of the Avatar animating and a render a video that can be sent to the user are needed. Downloading and installing the package is simple using Unity Package Manager.

Unity Recorder allows developers to change multiple variables of the video, like aspect ratio, width and height of the video, frame rate, duration of video, among others. For testing purposes, the resolution was set to 680x480 with 30 frames per second to make an understandable video that has a relatively small size. The location to store the video is set to a folder that our server can access when it needs to send the video to the user. The Recorder needs to have a specific duration of the video before it starts. A small function was created that counts the total number of frames of each animation in a translation. If the sentence has 4 glosses, each with 20 frames, the function will return 80 frames. This was used to determine how long the full animation will take and the video could stop after that, since the Avatar would have stopped by then.

After the video is recorded and exported to the predetermined location, a ping is sent to our server with the name of the video. The server then accesses this location, converts the video to Base64 and sends it via XMPP to the application on the user device. The application converts the Base64 back to video and the user is able to see it.

There are some extra implementations mentioned in the architecture, for example, in some cases, the user might want to have subtitles turned on, for example, if he does not understand a sign but by reading the gloss, he is able to get it. For this module, the user has no control over the subtitles, they are either on or off by default, that is, they can either be integrated in Unity before putting the Avatar in the server and all users will have subtitles on, or they are not integrated and the subtitles are not activated for any users. Regarding the repetition of sentences, if the user wants for some reason the see the translation again, the video is the best option for that since the user can repeat all previous translations, as long as they are stored in either the RAM or cache. When it comes to changing the speed of the animation, it is another parameter that cannot be set by the user. There is a possibility that in the IVLinG application, where the videos are shown to the user, to implement a way to speed the video for 1x, 1.25x, 1.5x or 2x, without loss of quality. However, when going for slower speeds, like 0.5x, the frame rate which is set to 30 fps, will go down to 15 fps, making the video look poor-quality.

5.3.2. Developing the Direct Module

The Direct approach is much simpler to develop, since the goal is to export the Unity application in a way that can be implemented in the IVLinG System, which consists in an application developed using Ionic. This framework is compatible with WebGL and Unity has an easy export to it, so the challenge is to convert the technologies that are not compatible with WebGL into compatible ones, and to create an application that does not consume too many resources on the user's device.

WebGL is web based, meaning HTTP Requests need to follow more strict "rules" than a simple Unity application. We were using self-signed certificates for the server that connected Unity to the translator API and the mapping database, and it created several errors. This could only be fixed by getting a signed certificate for the server, and after getting one the problem was fixed.

Another compatibility issue is with a render pipeline. Although we have not mentioned yet, a render pipeline can make a really big difference on the appearance of a 3D environment, but the better the pipeline, the more processing power it requires. When running on a server this should not be a problem, since we know what the specifications of the machines are. However, when using WebGL and running on users' devices, those devices can be any range from low end to high end, so we must limit the resources used to accommodate most users. Furthermore, WebGL does not support the most recent render pipelines, since it is web based and is not optimized for that.

The Universal Render Pipeline (URP) was chosen as it brings some great improvements, albeit not the best, while being rather small in size so that it can be played in a mobile device. Other reasons for this choice were its compatibility with WebGL and that it is supported by Unity. The comparisons between no pipeline and URP are significant and can be seen in Figure 38. The differences in the rendering are clear considering they are the same character, with the same skin tone. When rendering without a pipeline, the skin tone gets darker than intended, the shadows on the face and hair are way sharper, giving it an older look. With the URP the shadows are much softer, the skin tone and hair are as intended, giving it a more friendly and natural look.

Figure 38 No Render Pipeline on the Left / URP on the Right.

Regarding the extra features, since the WebGL version runs on the user device, it is possible to implement them with relative ease. The subtitles can be implemented in a way to follow the States of the Avatar. When the asset bundles are downloaded and a list of animations is created, a list of subtitles can also be created and when the state in the state machines changes, so does the subtitle. The opacity of this text can be set to 100 or 0 to make the on and off states, respectively. In Unity, the overall animation speed of the Avatar can be set in the state machine and, consequently, via code. With some button presses, the whole animation can go from 0.5x or slower to 2x or faster without loss of frame rate like the Video module. The repetition of animations can also be implemented, but only works towards the last sentence. When a new sentence is received, the cache from the previous sentence is cleared, meaning the animations are no longer available. This means that the Direct module has two extra features which are better, the change of speed and the subtitles, while the Video module has one better feature, which is the repetition.

Now that both solutions are implemented, they are ready to be tested and compared in the next chapter to decide which is more adequate.

5.3.3. Example of Using the Avatar

The examples given here intend to demonstrate what was achieved in this chapter. The examples in Figure 39 and Figure 40 show the Avatar animating the signs for “Good” and “Who” in LGP, respectively, with subtitles enabled on the top left corner.

Figure 39 Avatar Doing the Sign for "Good" (Bom).

Figure 40 Avatar Doing the Sign for "Who" (Quem).

Although in this example the input for these animations was given via XMPP, this version is adapted for both the Video and Direct modules. It features the render pipeline URP, and has functions that can enable the subtitles, change the speed and repeat the animations, even though at the moment in the Video version there is no way to access those functions.

5.4. Conclusion

In this chapter, the developed implementations, as well as various components related to the project were presented in detail. Firstly, we determined which MoCap system would be used, deciding on the Perception Neuron Studio for body and hands, with the Rokoko Face Plugin for the facial expressions. Secondly, we developed a first prototype of an Avatar that animates with simple animations, to test if the system could be implemented and if Unity was the right engine. Then, we captured 612 animations, but only a fraction of those had facial animations due to the expiration of the Rokoko Face Plugin license. After that, we divided the animations into asset bundles to reduce the size of the files. A file system to store those animations was created and a mapping database that serves as a look-up table for Unity to know where the animations are, especially for synonym signs. The state machine was then optimized to only have two states, besides the idle, therefore accepting an endless number of glosses without an issue. After that, the facial expressions were implemented by importing the Rokoko head into Unity, which has the same mesh as the acquired animations, and associating a state machine to it. By doing this, we were able to copy the blendshapes' values to the blendshapes on the face of the Avatar, so it animates accordingly. Following this, we connected the application to the translator API to get the gloss sequence from the translation. Then, the Video and Direct modules that were designed in the architecture were implemented. While doing this, a render pipeline, URP, was added to both versions, and depending on the version, it is possible to repeat the animations, add subtitles and speed up or slow down the animation. Finally, some examples of the final version of the Avatar were shown.

6. Tests and Results

In this chapter, the prototype developed is tested and the results are presented, with the aim of assessing whether the system is a viable solution for the deaf community to use. The first part of this evaluation is to test the performance of the Video and Direct modules, by making several queries using both solutions and recording the time it takes for the user to receive the result. One of the goals of this project is to create a practical solution for deaf users, in which they can get a translation without waiting too much time, so this test is crucial. A discussion of the tests, as well as a full comparison to both solutions is then made, and one is chosen to be used in the final prototype. The second part of the tests is to evaluate the performance of the Avatar, more specifically the animations acquired. The solution is only viable if deaf users can understand the Avatar, so some animations were shown to professional LGP teachers, and they evaluated the Avatar like they would an LGP student. The results of the tests are then discussed, allowing for verification of whether the Avatar with MoCap animations is a good approach for a portable virtual interpreter of sign languages.

6.1. Video or Direct Modules

The two modules, Video and Direct, determine how the application is integrated in the user's device, each with their own advantages. On the one hand, the Direct module with a WebGL approach produces faster downloading times by only requiring asset bundles, and the Video approach produces lower downloading times because videos need to be encrypted and sent between the servers and the application. On the other hand, the Video's main advantage is the less processing required on the user's device, while loading a 3D Avatar through WebGL can be demanding process. The goal of these tests is to determine whether it is better to load the Avatar on the user's device, or whether waiting for the videos is faster. To do this, it is necessary to analyse different scenarios for both solutions. The Direct needs to be tested in both high-end and low-end devices, while the Video needs to be compared to Direct in both fast and

slow internet connections. This enables an understanding of both the best and worst case scenarios, helping to decide which version is preferable.

6.1.1. WebGL Loading: High-End and Low-End Devices

To test the Direct module, two devices were used, having very different processing power, since the aims of the tests are to see how long the Avatar takes to load in the application and to check that the Avatar works smoothly on both high-end and low-end devices. The application downloads the entire WebGL module every time the user starts a videocall, which means the download speed influences how the application behaves. To test how long it takes to load the Avatar, both devices were tested, in a fast and a slow internet connection, by recording the time elapsed from the moment the user opens the video-call until the moment the Avatar is ready to be animated. To have more reliable results, four videocalls were started in each device, and in each internet connection. This step is only a part of the Direct module, making it an essential factor when comparing the two solutions. To test whether Avatar runs smoothly, the tests check that the frames per second are consistently high on both devices. Regarding the devices, one is the Xiaomi Mi 11 Lite, valued at 250€, and the other is the iPhone 12, valued at 930€. The reasons for these devices to be chosen were because they were released within 6 months apart, they have a considerable difference in terms of both price and performance, and they have different operating systems.

According to the previous description, the tests were carried out in four different scenarios: slow internet using Xiaomi, slow internet using iPhone, fast internet using Xiaomi and fast internet using iPhone. In each of these scenarios, four videocalls were started and the average time was calculated. In Figure 41 the comparison of these results is presented, and it shows that although internet connection affects performance, the difference is not very significant. The times compared in both devices are also similar in both fast and slow internet connections, with the low-end device having slightly higher times. During these tests, it was checked that the Avatar ran smoothly on both devices. Once the animations were downloaded, the Avatar animated them on both devices without any visual differences, so any formal testing to this part did not seem required.

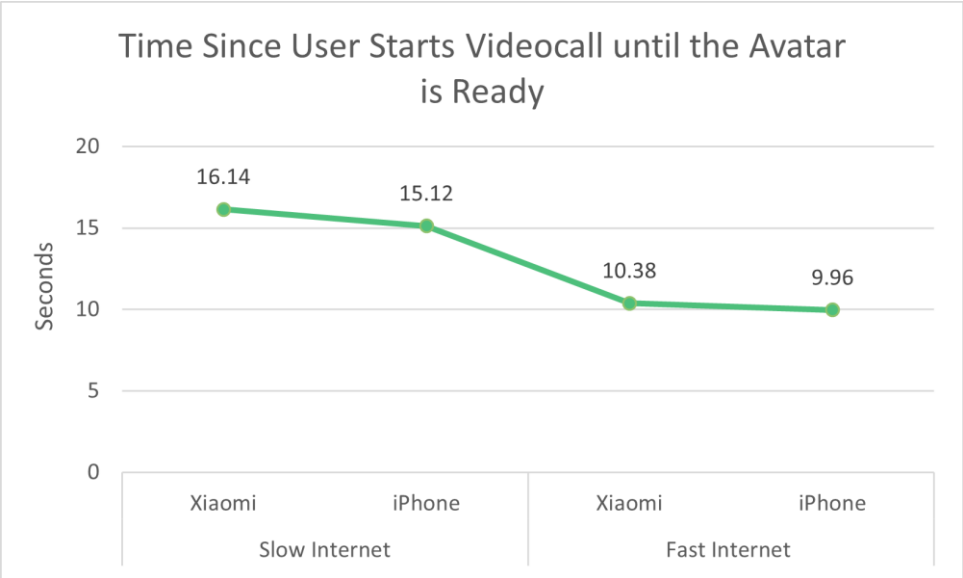

Figure 41 WebGL Loading: High-End and Low-End Devices.

Furthermore, a less formal test was also implemented using a more recent smartphone, the Samsung Galaxy S22 Ultra, in order to verify if it made any difference. The results were practically the same as the High-End device, the iPhone, making them not very relevant to the study.

6.1.2. Video or Direct: Fast and Slow Internet Connections

The previous test checked the time it took for the WebGL module to load, but now it is necessary to target the time it takes since the user inputs a sentence until the animation starts, and this test is related to both modules. To test the impact of internet connections on each of the solutions, two types of connections were used, one fast, which consisted in a 5G wireless connection with only one device connected to the network, and one slow connection, which consisted in a wireless connection with over 20 devices connected to the network. The former should be similar to a user with mobile data using the application in a public area with 5G availability, while the latter should simulate a public internet connection like the ones available in hospitals. To see the changes in longer and smaller translations, we performed the tests in five different sentences, ranging from four to eight signs. The same five sentences were used in both scenarios and for both solutions, and they were selected from the dialogues of tourism and shopping, mentioned in Chapter 5.2.2. since they have signs that are used on a daily basis. The times calculated are measured in seconds and start when the user clicks “Send” message, asking for a translation. The timer is stopped when the

translation is ready to be seen. i.e., when the Avatar starts animating in the Direct version and when the video is ready to be seen in the Video version.

The results of the first scenario, fast internet connection, can be seen in Figure 42. As we can see, the average time for a sentence to be translated and starts animating in the Direct version is 1.10 seconds, with the slowest being 1.42 seconds and the fastest being 0.94 seconds. Whereas in the Video version the average is 21.53 seconds, with the slowest being 26.46 seconds and the fastest being 15.77 seconds.

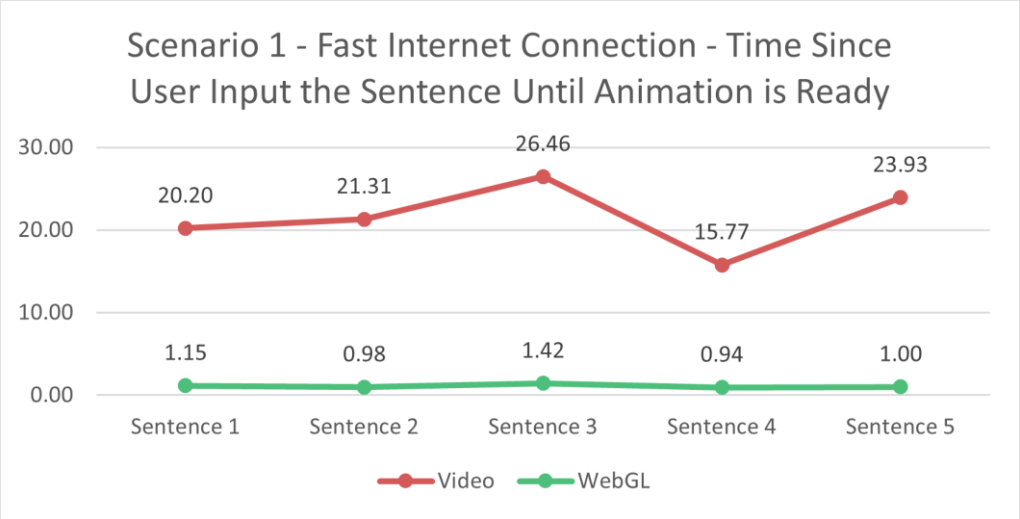

Figure 42 Video or Direct: Scenario 1 - Fast Internet Connection.

The results for the second scenario, slow internet connection, can be seen in Figure 43. For this scenario the times are higher, as expected since both versions are downloading at slower speeds. The Direct version averages 3.96 seconds per sentence, with the fastest translations being 3.34 seconds and the slowest 4.86 seconds. As for the Video version, the average is at 52.02 seconds, the fastest time of 42.23 seconds and the slowest of 62.26 seconds.

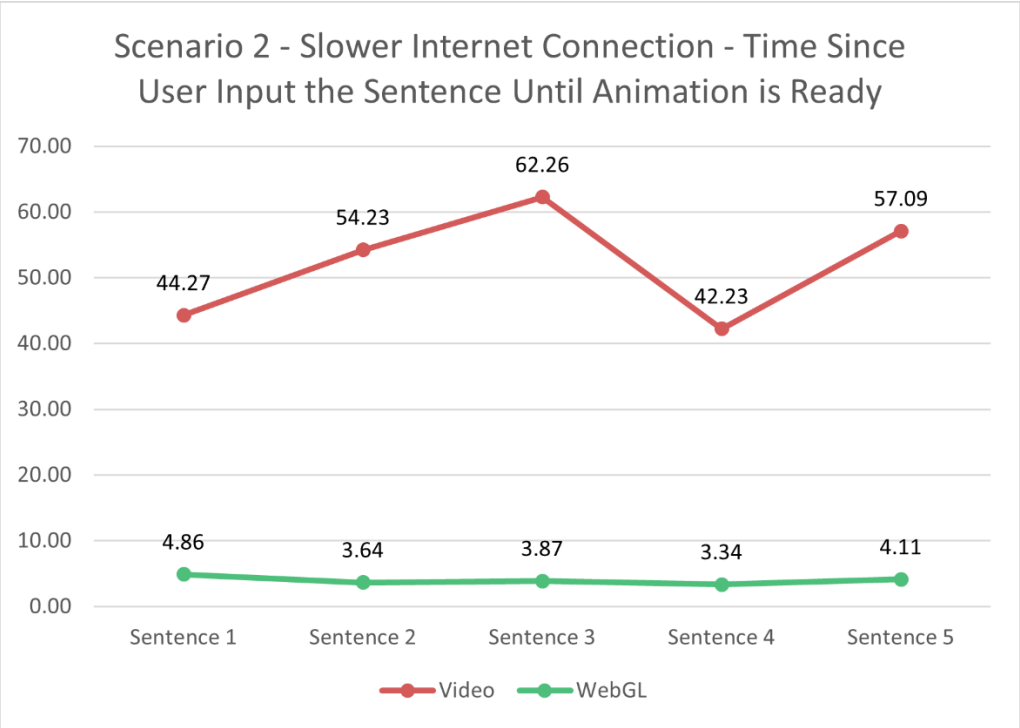

Figure 43 Video or Direct: Scenario 2 - Slow Internet Connection.

6.1.3. Choosing Between Video or Direct

After performing the necessary tests to both solutions, a comparison needs to be made to select which is the better. To summarize the results gathered previously, we created a chart with the best- and worst-case scenarios for each solution, which can be seen in Figure 44.

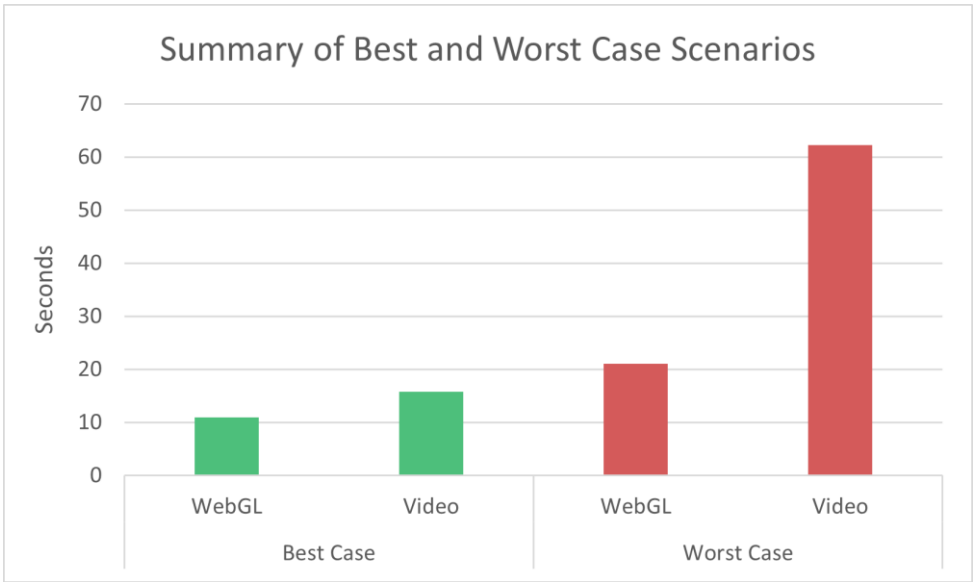

Figure 44 Summary of the Best and Worst Case Scenarios.

For the Video module, the scenarios were determined by selecting the fastest and slowest, whereas for the Direct module, a sum was made of the fastest

loading time of the Avatar and the fastest time it took for an animation to be ready, for the best-case scenario, and equally the slowest Avatar loading time was summed to the slowest time for the animations to be ready to determine the worst-case. According to the data, the Direct is clearly the faster solution in both scenarios, with a wider difference in the worst-case times, due to amount of bandwidth required when downloading long videos. The Direct module times take into account the time for the Avatar to load, which in a day-to-day use only happens once, i.e., the times in the table describe the first translation of a videocall, while the subsequent translations do not need to load the Avatar, which reduces the times to a fraction of the first translation. After the Avatar is downloaded, the user can perform an endless number of translations before turning off the videocall, and the waiting time for each translation is equivalent to the downloading of the animations, which can be seen in green in Figure 42 and Figure 43. Therefore, the best- and worst-case scenarios for the Direct module seen in Figure 44 only relate to the first translation of a videocall, while the following have even faster times, which further demonstrates that this is the faster solution. For this reason, and since the performance did not seem to be worse in the devices we tested the system in, we determined that the Direct module should be the option selected for the final application.

6.2. Animation Testing

The most important factor of our system is having animations that LGP users understand. In the literature review we noticed that most tests only focused on “understood” or “did not understand” the sign, but this is not enough to determine if the sign is fully correct, as context sometimes helps in understanding. To fully analyse the Avatar, we thought the best way was by evaluating it in the same way that a student of LGP would be. To do this, we started by asking a teacher of LGP how they evaluate their students, and the answer was that the five *cheremes* are analysed for each sign, by either being right or wrong. This is especially useful for the Avatar, since we get feedback of exactly what is wrong with a sign, and what needs to be changed for it to be perfect.

To develop the test using this method, we randomly selected 32 signs from our database, recorded the Avatar animating them, and uploaded them to a

questionnaire in Google Forms. Out of those signs, 21 use only the dominant hand and 11 use both hands. As mentioned in Chapter 5.2.6. the facial expressions were turned off for these tests, since we did not have the animations for every sign. This means that we were not able to evaluate the *chereme* for non-manual expressions, since most of the answers would be that it was missing.

To analyse the remaining four *cheremes* on each sign, we designed the questionnaire to have questions for one or both hands, depending on the sign, and at the end an opportunity to leave suggestions for future improvements. The questionnaire contained numerous repetitive questions, differing only in the signs they addressed. As a result, an excerpt of it, written in Portuguese, is available in the Appendix 9.3. Extract From the Questionnaire.

The testing stage took two weeks to complete, in this period we asked teachers and interpreters of LGP to answer the questionnaire. Due to how long the questionnaire took to answer, it had a goal of 5 to 10 participants, which was achieved with a total of 6 participants. The raw results were stored, and the study of those results will be exposed in the following sections.

6.2.1. Participants

Since the questionnaire was designed to be evaluated like a student of LGP, ordinary LGP users cannot fully grasp how the *cheremes* are supposed to be, therefore they do not know how to evaluate signs this way. Originally, we decided that interpreters and teachers would be a good target, however, after talking to a LGP interpreter, they said they were not comfortable evaluating the language in this way. Even though they used to be a student of LGP, they had never evaluated a LGP test. We decided to not ask other interpreters to answer our questionnaire for the same reason, and consequently, our target became only LGP teachers. This is somewhat of a problem due to the low number of LGP teachers that are still active. According to a news report, only 87 active teachers were teaching LGP students in 2017 [78]. We were only able to get six volunteer teachers, through word-of-mouth and by asking the “Associação de Profissionais de leccionação de língua gestual” (Association of professional teachers of sign language) to share our questionnaire with their members. Despite the number of participants being low, the answers come from professionals at this task, and since the goal of the tests is to help refine the Avatar, we believe it is more

important than asking ordinary users if the sign is understandable or not. Furthermore, we asked the participants if LGP was their native language, and the six answered with yes.

6.2.2. Teachers Results

The data collected on the questionnaire gives the evaluation of four out of the five *cheremes* present in LGP, for 32 signs by six LGP teachers. One way to analyse this data is to group the answers of each teacher by dominant hand and non-dominant, and then compare them between each other.

We created collumn charts for each teacher and divided by hand, to show how many signs the teacher thought a *chereme* was good in. For example, on the left chart of Figure 45, we can see the evaluation of the first teacher towards the dominant hand, and he scored the hand configuration of 31 signs as correct, and one sign as wrong. On the left of the same figure, we can see the evaluation of the first teacher towards the non-dominant hand, and he scored the movement of all signs as wrong. The dominant hand charts show the evaluation in 32 signs, since all of them use the dominant hand, and the non-dominant hand charts evaluate 11 signs, which are the ones that use both hands. The results of the remaining five teachers can be seen in Figure 46, Figure 47, Figure 48, Figure 49 and Figure 50.

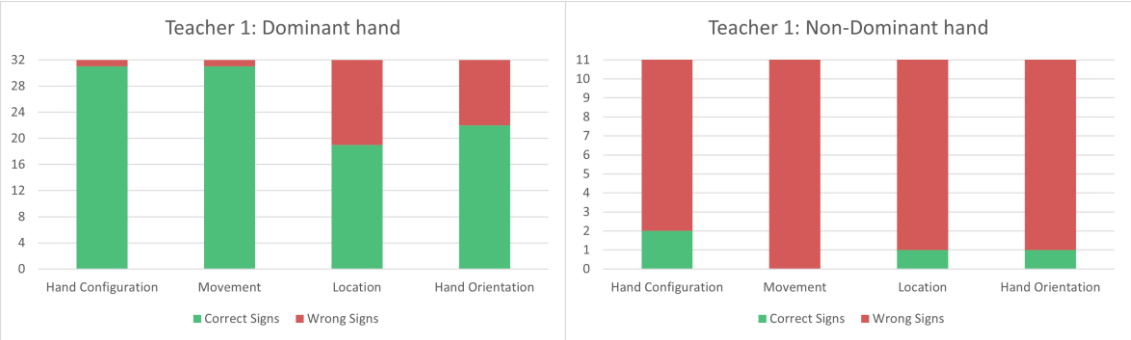

Figure 45 Teacher 1 Dominant (Left) and Non-Dominant (Right) Evaluations.

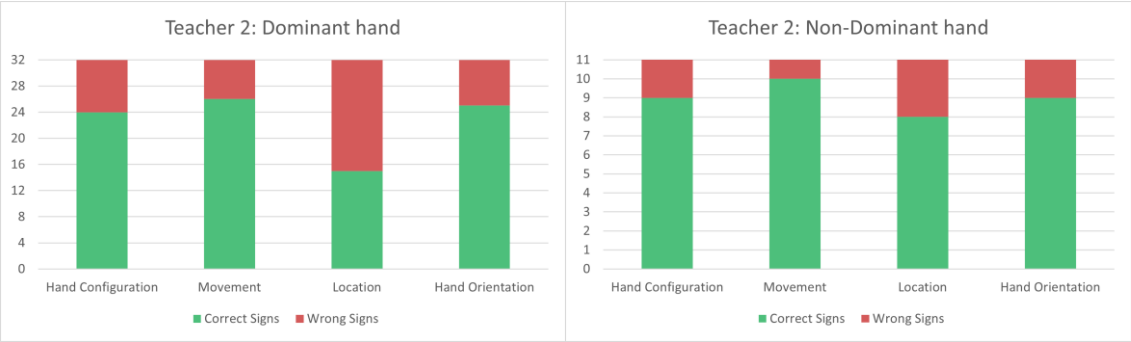

Figure 46 Teacher 2 Dominant (Left) and Non-Dominant (Right) Evaluations.

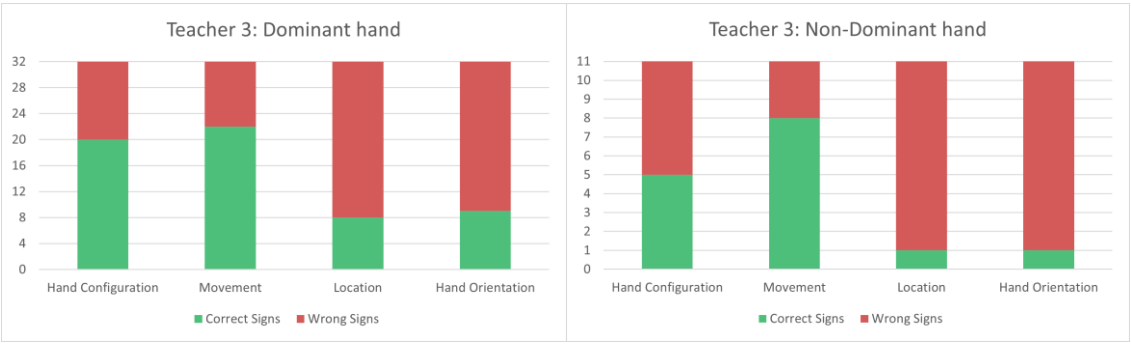

Figure 47 Teacher 3 Dominant (Left) and Non-Dominant (Right) Evaluations.

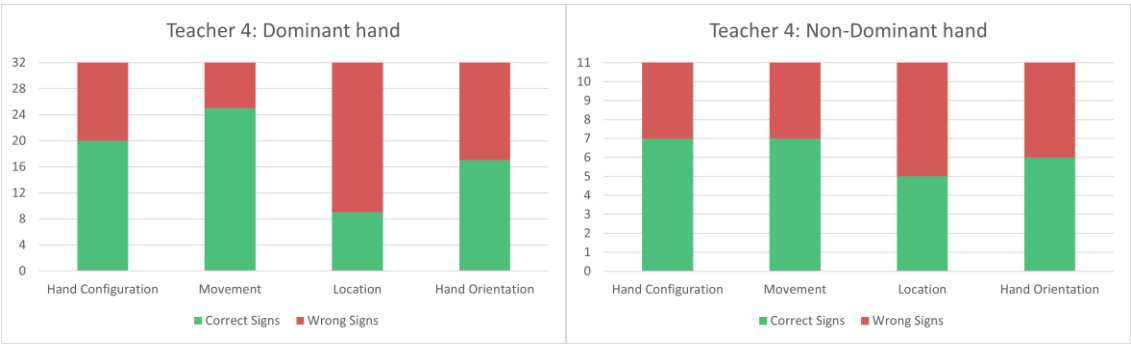

Figure 48 Teacher 4 Dominant (Left) and Non-Dominant (Right) Evaluations.

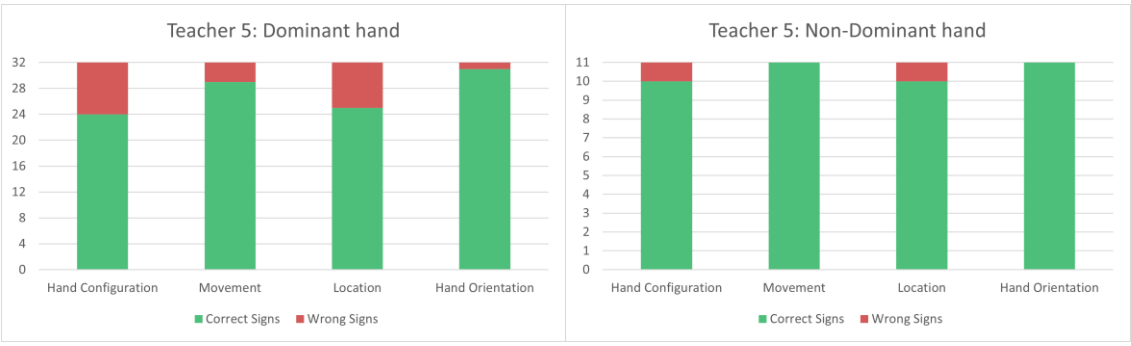

Figure 49 Teacher 5 Dominant (Left) and Non-Dominant (Right) Evaluations.

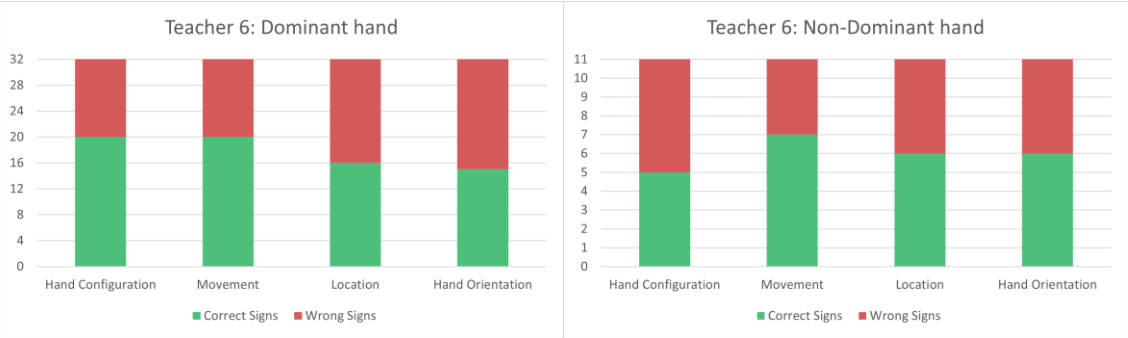

Figure 50 Teacher 6 Dominant (Left) and Non-Dominant (Right) Evaluations.

Through these charts we can examine that sign language is not as objectively evaluated as we originally thought, since the teachers disagree a lot. For example, Teacher 5 rated the *cheremes* of most signs as correct, notably the non-dominant hand movement and hand orientation as correct in all signs, while Teacher 1 believes the movement is wrong in all of them, and the orientation is correct in only one.

Furthermore, if we sum up all the correct *cheremes* that each teacher rated and divide them by the total number of *cheremes*, which is 172, we can get the score rated from 0 to 100%. Doing this, the teacher that gave the lowest overall score was Teacher 3 with only 74 out of the total 172 *cheremes* marked as correct, which is around 43%. Meanwhile the teacher that gave the highest overall score was Teacher 5, marking 151 correct *cheremes*, or 88%. With a disparity this high, we think it is better to sort the data in a different way.

6.2.3. Sign Results

To approach the results in a more focused way, we decided to attribute a score to each sign, and by doing this, we can determine which signs need the most changes, the ones that need minor refining, and the ones that are perfect. In order to calculate the score of each sign, we gathered the six evaluations of a sign, by the six teachers, and attributed one point for each correct *chereme*. The maximum score a single sign can have is 24, in the case of the four *cheremes* being marked as correct by the six evaluators. Similarly to the previous approach, we separated the dominant and non-dominant hand, since they are animated separately. The lowest score was one sign with 5 correct *cheremes* out of 24 scores, and only 13 signs were below 50% correct, 9 in the dominant hand and 4 in the non-dominant. The scores in numbers became a bit confusing, so they

were transformed into grades to be easier to understand. Grade A is the best, a sign with this grade means it was marked as perfect by all teachers. Grade B means a sign is good but needs refining, usually due to one or two *cheremes* being determined wrong by a few teachers. Grade C means a sign needs some changes, mostly because more than one *chereme* is wrong and most or all teachers agreed on it. Grade D is the worst, and it means the sign needs to be remade since at least two or three *cheremes* were poorly executed. The results of this grading system can be seen in Figure 51 and Figure 52. From this interpretation of the data, it is clear that most signs fall in grade C, 18 to be exact, meaning they need some improvements. There are still a lot of signs in grade D, 13 of them, which need to be refined urgently or be completely re-captured. On the brighter side, there are 9 signs that only need minor refining since they were graded B and 3 signs that do not need to be modified since they were perfect and scored grade A. The reason for the low grades is mostly due to the few changes that was done to the MoCap data. As previously mentioned, the goal of this prototype was to test if a valid system was possible using the tools available, so only a few adjustments were made to fix the bigger problems of magnetic interference and drifting, and individual refining of each animation was determined as future work.

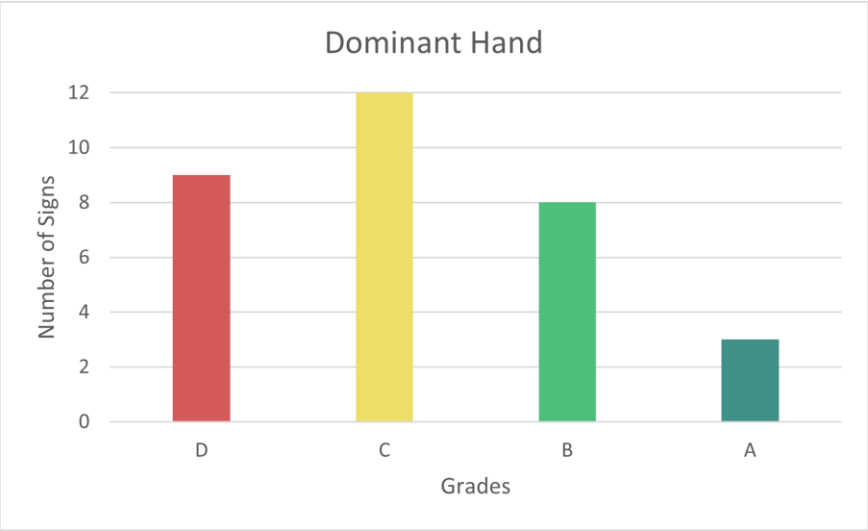

Figure 51 Grading Results of the Signs - Dominant Hand.

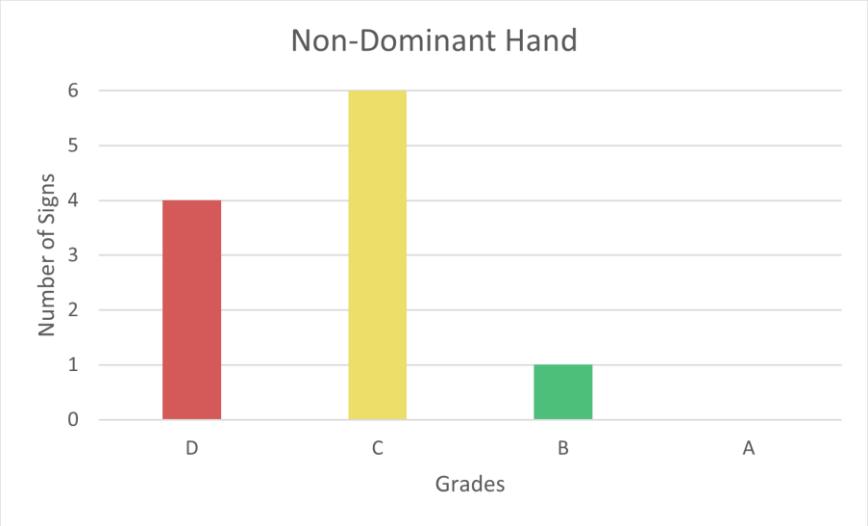

Figure 52 Grading Results of the Signs - Non-Dominant Hand.

6.2.4. Evaluating the MoCap data

To evaluate the biggest problems with our MoCap data, we decided to analyse the data in a final approach, which is to evaluate the *cheremes* independently of the signs. This way, it is possible to grasp the most problematic parts of the MoCap, so that in future sessions we can try to improve or solve them. The scoring system will be similar to the previous approaches, since we count the number of times each *chereme* was correct and demonstrate them all in a comprehensible chart, seen in Figure 53. From this chart, the location is clearly the one with the most problems, followed by hand orientation, then hand configuration and finally the movement. The easier to fix manually are most likely the location and hand orientation, since they both usually require one or two joints to be fixed. The movement is usually harder to fix since it requires multiple joints to be animated, and doing this manually tends to generate robotic animations, which defeats the purpose of using MoCap. In terms of hand configuration, it is definitely the hardest to animate manually and although the MoCap data proved to be effective in this aspect, it still needs improvement. The signs that had wrong hand configuration were mostly due to malposition of one or multiple fingers, which is the problem with having only six sensors on each glove. The only evident solution would be to purchase a better set of MoCap gloves, ideally with 11 or 16 sensors on each hand, two or three on each finger, to improve the quality of the data captured. This proved that MoCap produces very good results with minor refinements to the raw data, but a more focused post-edit is definitely a necessity.

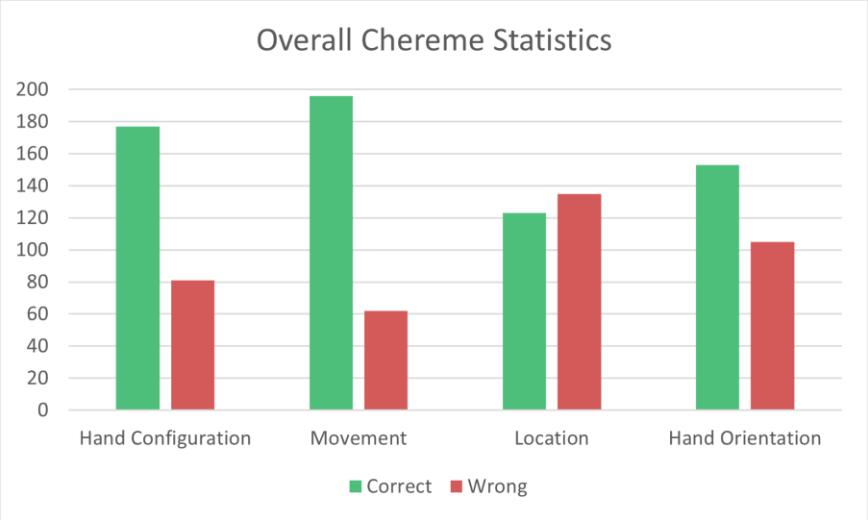

Figure 53 Overall Statistics of Each Chereme.

6.3. Discussion

Overall, the results of the tests showed that it is possible to have a mobile system that helps deaf people in their everyday lives, through the use of a virtual interpreter.

Testing the Video and Direct modules, we can conclude that the latter is the faster and most efficient way of producing animations for the user, which is why it was chosen for the final prototype. The difference in these solutions was evident in both fast and slow internet connections, and the performance was the same through the two devices that were tested. With the evolution of technology, internet connections will become even faster, meaning that in the future the Video solution might become viable. However, for this to be the case, the way of recording videos needs to be changed, since as of now, the Avatar animates in real time before rendering and sending the video to the user. This makes the waiting time become much longer. To solve this, Unity needs to develop a faster way to record the Avatar or it is necessary to discard Unity and change to a different engine, with a better recording system. On the other hand, the only noticeable problem with the current Direct solution using WebGL would be the render engine, since the current one is decent but there are better ones. Once again, with the evolution of technology, better render engines will eventually be available in mobile applications. This means that the Direct implementation, through WebGL or other framework, will still be a viable solution. For future prototypes, if a better render engine was really necessary and WebGL could not

support it, the livestream version could also be an option, although we were not able to develop it, as mentioned in the System Architecture chapter.

The analysis of the animation testing confirmed that MoCap is a good option for capturing animations, since it is faster than manual animation and produces more natural looking movements. Contrary to what was initially thought, the evaluation of SL is more subjective, since the six teachers that evaluated the animations gave very different scores. This made us reconsider how we should organise the data, which led us to a grading system of A to D, where A meant the sign was perfect and did not need to be changed, and D meant that the sign needed to be remade or heavily fixed in post-edit. Most signs were scored C, which means that a lot of corrections need to be made, and that the raw data from the MoCap needs a more in-dept editing before going to the database. However, it is very different to animate, fix and refine each *chereme*, since, for example, the hand configuration includes 15 joints, while the orientation of the hand usually only depends on the one wrist joint. To detect how difficult it would be to fix the MoCap data, we rearranged the results once again and divided the scores by *chereme*. Doing this, we noticed a small pattern where the *cheremes* that needed the most changes were the ones that are easier to animate, namely the hand orientation and location, while the ones that are harder to animate needed the least refining. This is why MoCap is so important and useful, the strengths of MoCap are usually the weaknesses of manual animation, and vice-versa. Using all this information, we can determine that albeit the grading system showed a lot of changes were required, those changes proved to be relatively easy to implement. Consequently, we believe that using a MoCap system and manually fixing the issues is faster than manually animating everything. Even though we did not perform formal testing to this comparison, in one session we were able to acquire 30 signs in less than 10 minutes, an efficiency that cannot be matched by manual animation. Even when taking into account the post-edit that the MoCap requires, between animating a sign completely and fixing it, the latter will always be faster, since the core of the animation is already animated.

7. Conclusions and Future Work

7.1. Conclusions

In recent years, the development of inclusive technologies has been getting more attention and importance, as it helps individuals of all ages and abilities with mobility, learning and communication. One area that has been getting more focus, yet still lacks improvement, is the communication between the deaf and hearing communities, since there are no ideal translators between spoken and sign languages. The general population falsely believes that all deaf people can read the language that is spoken in their country, as is the case in Portugal, where most people assume deaf people can read Portuguese. To a deaf person that speaks LGP, Portuguese is a foreign language, much like a spoken language from another country, e.g. English, or even a sign language from another country, e.g. British Sign Language. Some members of the deaf community dedicate themselves to learning Portuguese in order to communicate with hearing people through a written form, like texts or emails, but that is not the standard.

The work presented in this document aims to develop a translator from Portuguese to LGP, focusing mainly on the capturing, refining and displaying glosses in a 3D Avatar. Most solutions implemented in this area, regardless of what sign language they developed upon, discarded the facial expressions of each sign which for most, if not all, SLs, is a key feature for understanding certain signs. Contrarily, in the solution presented in this document, facial expressions were captured using a MoCap system, to ensure that each sign had the correct facial animation, and the module for animating the Avatar with facial expressions was developed, albeit disabled for the final prototype. This was due to licensing issues, which caused only 60 facial animations to be captured, out of the 612 total signs that have body animation. These body animations were also captured using a MoCap system instead of manual animation due to two main reasons. The first was to test whether the use of these systems made it possible to create animations faster while maintaining good quality, and the second was to try to create more natural and fluid animations, in contrast to other developed solutions

which had feedback saying the animations were too robotic. Furthermore, another goal of this application is to be developed for both web browsers and mobile devices, ensuring accessibility for the majority of people. A common issue observed while researching similar solutions is that they often involve projects aimed at testing the limits of sign language translation and display, discarding the distribution of the solution to the general population. Another step towards helping the user understand the sentence was the implementation of buttons to activate or deactivate the subtitles, change the speed of the animation and repeat the last animation received.

The results provide answers to the questions proposed in 1.2. Main Objectives. Firstly, they indicate that developing a mobile application capable of creating videocalls and featuring a 3D Avatar translating from Portuguese to LGP is feasible for today's smartphones, with only a short waiting time at the start of the videocall to initialize the Avatar component. After the initialization of the Avatar, the user can input sentences and the results are ready within a few seconds. Furthermore, since the Direct system was used, the addition of subtitles, as well as the options to change speed and repeat animations were available, if users struggle to understand a translation. Moreover, the results from the animation testing were organized in three different ways to get different conclusions. The first one proved that evaluation of sign language is more subjective than initially thought, since the six volunteer teachers who agreed to take part in the study here described, gave very different scores for the evaluated signs. This led to a scoring system that graded each sign based on the teachers' answers, ranging from A to D, with A representing the highest score and D representing the lowest. Some of the scores were unexpected so another way of analysing the responses was determined. By evaluating each *chereme* individually, the results proved that the MoCap helps in parts of the animation that are harder to animate manually and struggles in the easier aspects of manual animation. Thus, a combination of MoCap and manual refining is the optimal solution.

Considering all of this, the primary goals of this project have been achieved, and hopefully this prototype can be improved and used to help in the communication between spoken and sign languages, the same way that there are already applications to translate between two spoken languages. The steps

described in 1.3. Research Methodology were followed through the several sections that were presented in this document. The *Problem Identification and Motivation* being described in chapters 1.1. Context and 3. Literature Review, the *Definition of the Objectives for a Solution* being described in 1.2. Main Objectives, the *Design and Development* being described in 4. System Architecture and 5. System Implementation, the *Demonstration* being described in 5.3.3. Example of Using the Avatar and 6. Tests and Results, *Evaluation* being described in 6. Tests and Results, and, lastly, the *Communication* being achieved through the publication of two articles in an international conference, seen in 9.1. Published Papers in International Conferences.

7.2. Future Work

The developed solution has made significant strides in translating and displaying sign language in a 3D Avatar. However, recognizing the difficulty of developing the ideal translator, there exists a compelling opportunity for further enhancement and refinement of the work conducted thus far. Based on the development and evaluation made, the following list offers a broad range of issues that could be addressed as follow-up to this research.

- **Capture the full LGP dictionary:** Although we were able to capture over 600 signs, and the majority of the most common ones, this still pales in comparison to the full spectrum of the LGP dictionary. A definitive number of signs is yet to be determined, mainly due to the lack of published books on the language and the dispersion of the community. This leads to some words having different signs in certain parts of the country and some users of LGP knowing a lot more words than others, which is similar to any other language, except they cannot search for them in a dictionary like most spoken languages have. Despite this, an estimated 8000 to 11000 signs are common knowledge amongst the community, so over 8000 signs in the dictionary should be the goal for the future.
- **Capturing Facial Expressions:** With the Avatar, databases and connections ready, the final step to animate the Avatar with facial expressions is to capture the animations for all of the signs. This

includes the original 612 that we captured for the body, and the signs that will be captured in the future, which were mentioned in the previous bullet point. The capturing of facial animations got interrupted because the Rokoko license expired, but in the future the project will extend the license, so the capture of these animations can be made.

- **Editing All of the Captured Animations:** The post-editing is an essential part of MoCap, since the raw data usually is not as good as intended. Several issues can cause the movement to be slightly different than the user performed, for example, magnetic interference, drifting or even the system not having enough sensors to capture the right data, as is the case with the finger movement. For this first implementation, only the major cases of drifting and magnetic interference were corrected, but in the future all animations should be evaluated by LGP teachers and refined accordingly to meet the standards.
- **Further Testing:** As mentioned in the previous point, evaluation of all animations should be performed. This is not an easy task when taking into consideration the thousands of animations acquired and the few active LGP teachers in Portugal. However, to create the best possible system, this step is one of the priorities.
- **Enhancing the Loading Time of the Avatar:** The Direct approach, through WebGL, was proven to be the fastest solution, however, it still takes around 10 to 15 seconds to load the Avatar since it downloads the entire module every time the user starts a videocall. Researching for a faster solution would be ideal, either through implementing the Avatar in the application, and thus removing the downloading time, or by developing a module that is smaller and, consequently, faster to download.

8. References

[1] W. Stokoe, “Sign Language Structure,” *Annual Review of Anthropology,* vol. 9, pp. 365-390, 1980.

[2] R. J. Ruben, “Sign language: its history and contribution to the understanding of the biological nature of language,” *Acta Oto-Laryngologica,* vol. 125, no. 5, p. 464–467, 2005.

[3] Associação Portuguesa de Surdos., “Língua Gestual Portuguesa (LGP) - APSurdos,” [Online]. Available: https://apsurdos.org.pt/?page_id=3725.

[4] “Census 2021,” Instituto Nacional de Estatística, 2021. [Online]. Available: https://censos.ine.pt/. [Accessed 2021].

[5] Hevner, March, Park and Ram, “Design Science in Information Systems Research,” *MIS Quarterly,* vol. 28, no. 1, p. 75, 2004.

[6] K. Peffers, T. Tuunanen, M. A. Rothenberger and S. Chatterjee, “A Design Science Research Methodology for Information Systems Research,” *Journal of Management Information Systems,* vol. 24, no. 3, pp. 45-77, 2007.

[7] P. Carvalho, Breve História dos Surdos no Mundo e em Portugal, Lisboa: Surd´Universo, 2007.

[8] M. Kompara, M. Holbl, T. Welzer, P. Escudeiro and A. Veiga, Communication Challenges in Inclusive Education Faced by Deaf and Non-deaf People, 2022.

[9] S. d. G. Ornelas and Correia, A gramática em LGP: análise e materiais didáticos, 2023.

[10] I. Correia, “Morfologia Derivacional em Língua Gestual Portuguesa: Alguns Exemplos,” *Revista Científica da Escola Superior de Educação de Coimbra,* vol. 9, pp. 160-171, 2014.

[11] European Sign Language Centre, “Spread the Sign,” [Online]. Available: https://www.spreadthesign.com/. [Accessed October 2023].

[12] I. Mesquita and S. Silva, Guia Prático de Língua Gestual Portuguesa - Ouvir o Silêncio, Nova Educação, 2007.

[13] M. A. Amaral, A. Coutinho and M. R. D. Martins, Para Uma Gramática da Língua Gestual Portuguesa, Editorial Caminho, 1994.

[14] R. A. F. Rodrigues, Compreensão da Língua Gestual Portuguesa em Crianças Surdas. Proposta de um Instrumento de Avaliação, 2017.

[15] I. Almeida, L. Coheur and S. Candeias, “From European Portuguese to Portuguese Sign Language,” Stroudsburg, PA, USA, 2015.

[16] M. Gonçalves, L. Coheur, H. Nicolau and A. Mineiro, “PE2LGP: Translating European Portuguese into Portuguese Sign Language glosses,” 2021.

[17] Unity, “Unity Technologies,” [Online]. Available: https://unity.com.

[18] Autodesk, “Autodesk Maya,” [Online]. Available: https://www.autodesk.com/maya.

[19] M. Menolotto, D.-S. Komaris, S. Tedesco, B. O'Flynn and M. Walsh, “Motion Capture Technology in Industrial Applications: A Systematic Review,” 2020.

[20] S. Ciklacandir, S. Ozkan and Y. Isler, “A Comparison of the Performances of Video-Based and IMU Sensor-Based Motion Capture Systems on Joint Angles,” 2022.

[21] E. van der Kruk and M. M. Reijne, “Accuracy of human motion capture systems for sport applications; state-of-the-art review,” 2018.

[22] Vicon, “Vicon Vantage Camera Range,” [Online]. Available: https://docs.vicon.com/display/Vantage/Vicon+Vantage+camera+range. [Accessed October 2023].

[23] “Vicon Motion Capture System,” [Online]. Available: https://ps.is.mpg.de/pages/motion-capture.

[24] A. Stelzer, K. Pourvoyeur and A. Fischer, “Concept and Application of LPM—A Novel 3-D Local Position Measurement System,” 2004.

[25] Microsoft, “Microsoft Kinnect,” 2022. [Online]. Available: https://learn.microsoft.com/en-us/windows/apps/design/devices/kinect-for-windows.

[26] S. Dent, “How I turned my Xbox's Kinect into a wondrous motion-capture device,” [Online]. Available: https://www.engadget.com/2015-03-08-using-the-kinect-for-motion-capture.html. [Accessed August 2023].

[27] M. A. Brodie, A. Walmsley and W. Page, “Dynamic accuracy of inertial measurement units during simple pendulum motion,” 2008.

[28] A. Podoprosvetov, A. Alisejchik and I. Orlov, “Comparison of action recognition from video and IMUs,” 2021.

[29] Rokoko, “Rokoko Smartsuit Pro II,” [Online]. Available: https://www.rokoko.com/products/smartsuit-pro. [Accessed August 2023].

[30] Perception Neuron, “Perception Neuron Studio,” [Online]. Available: https://neuronmocap.com/pages/perception-neuron-studio-system. [Accessed August 2022].

[31] T. Weise, S. Bouaziz, H. Li and M. Pauly, “Realtime performance-based facial animation.,” 2011. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edb&AN=83471293&site=eds-live.

[32] H. Li, T. Weise and M. Pauly, “Example-Based Facial Rigging,” *ACM Transactions on Graphics (Proceedings SIGGRAPH 2010),* vol. 29, no. 3, 7 2010.

[33] Apple Inc, “ARKit Documentation - Blendshapes Location,” [Online]. Available: https://developer.apple.com/documentation/arkit/arfaceanchor/blendshapelocation. [Accessed May 2023].

[34] United Nations, “Sign languages unite us!,” [Online]. Available: https://www.un.org/en/observances/sign-languages-day. [Accessed October 2023].

[35] T. S. S and M. R. I, “A Extensive Survey on Sign Language Recognition Methods,” 2023.

[36] M. J. Page, J. E. McKenzie, P. M. Bossuyt, I. Boutron, T. C. Hoffmann, C. D. Mulrow, L. Shamseer, J. M. Tetzlaff, E. A. Akl, S. E. Brennan, R. Chou, J. Glanville, J. M. Grimshaw, A. Hróbjartsson, M. M. Lalu, T. Li, E. W. Loder, E. Mayo-Wilson, S. McDonald, L. A. McGuinness, L. A. Stewart, J. Thomas,

A. C. Tricco, V. A. Welch, P. Whiting and D. Moher, “The PRISMA 2020 statement: an updated guideline for reporting systematic reviews,” *BMJ,* pp. n71, DOI: 10.1136/bmj.n71, 3 2021.

[37] I. Almeida, "Exploring Challenges in Avatar-based Translation from European Portuguese to Portuguese Sign Language", Masther's thesis, Instituto Superior Técnico, Universidade de Lisboa, 2014.

[38] I. Almeida, L. Coheur and S. Candeias, “Coupling Natural Language Processing and Animation Synthesis in Portuguese Sign Language Translation,” Stroudsburg, PA, USA, 2015.

[39] Blender Foundation, “Blender,” [Online]. Available: https://www.blender.org.

[40] R. Santos, "PE2LGP: From Text to Sign Language (e vice-versa)", Masther's thesis, Instituto Superior Técnico, Universidade de Lisboa, 2016.

[41] P. Cabral, M. Gonçalves, H. Nicolau, L. Coheur and R. Santos, “PE2LGP Animator: A Tool To Animate A Portuguese Sign Language Avatar,” Marseille, France, 2020.

[42] P. Escudeiro, N. Escudeiro, R. Reis, P. Rodrigues, J. Lopes, M. Norberto, A. B. Baltasar, M. Barbosa and J. Bidarra, “Real Time Bidirectional Translator of Portuguese Sign Language,” 2015.

[43] P. Escudeiro, N. Escudeiro, R. Reis, J. Lopes, M. Norberto, A. B. Baltasar, M. Barbosa and J. Bidarra, “Virtual Sign - A Real Time Bidirectional Translator of Portuguese Sign Language,” 2015.

[44] T. Oliveira, N. Escudeiro, P. Escudeiro, E. Rocha and F. M. Barbosa, “The VirtualSign Channel for the Communication Between Deaf and Hearing Users,” *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje,* vol. 14, no. 4, pp. 188-195, DOI: 10.1109/RITA.2019.2952270, 11 2019.

[45] T. Oliveira, P. Escudeiro, N. Escudeiro, E. Rocha and F. M. Barbosa, “Automatic Sign Language Translation to Improve Communication,” 2019.

[46] P. Escudeiro, N. Escudeiro, M. Norberto and J. Lopes, “Virtualsign translator as a base for a serious game,” New York, NY, USA, 2015.

[47] M. Jemni and O. Elghoul, “A System to Make Signs Using Collaborative Approach,” Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 670-677, 10.1007/978-3-540-70540-6_96.

[48] N. Aouiti, “Towards an automatic translation from Arabic text to sign language,” 2013.

[49] N. Aouiti, M. Jemni and S. Semreen, “Arab gloss and implementation for Arabic Sign Language,” 2017.

[50] D. A. Alabbad, N. O. Alsaleh, N. A. Alaqeel, Y. A. Alshehri, N. A. Alzahrani and M. K. Alhobaishi, “A Robot-based Arabic Sign Language Translating System,” 2022.

[51] SoftBank Robotics, “Pepper Robot,” [Online]. Available: https://us.softbankrobotics.com/pepper.

[52] V. Lombardo, C. Battaglino, R. Damiano and F. Nunnari, “An Avatar-based Interface for the Italian Sign Language,” 2011.

[53] Sourceforge, “Enthusiasm,” [Online]. Available: https://enthusiasm.sourceforge.net/. [Accessed October 2023].

[54] S.-O. Caballero-Morales and F. Trujillo-Romero, “3D Modeling of the Mexican Sign Language for a Speech-to-Sign Language System,” *Computación y Sistemas,* vol. 17, no. 4, pp. 593-608, DOI: 10.13053/CyS-17-4-2013-011, 12 2013.

[55] DAZ Productions, “DAZ Studio 4 Pro,” [Online]. Available: https://www.daz3d.com/daz-studio-4-pro/. [Accessed October 2023].

[56] MathWorks, “Matlab,” [Online]. Available: https://www.mathworks.com/products/matlab.html. [Accessed October 2023].

[57] L. Vera, I. Coma, J. Campos, B. Martínez and M. Fernández, “Virtual Avatars Signing in Real Time for Deaf Students,” 2018.

[58] Optitrack, “Optitrack,” [Online]. Available: https://optitrack.com/. [Accessed October 2023].

[59] CyberGlove Systems, “CyberGlove II,” [Online]. Available: http://www.cyberglovesystems.com/cyberglove-ii. [Accessed October 2023].

[60] Cal3D, “Cal3D,” [Online]. Available: https://mp3butcher.github.io/Cal3D/.

[61] S. Krishna, A. R. Jindal, M. R, R. K and D. Jayagopi, “Virtual Indian Sign Language Interpreter,” New York, NY, USA, 2020.

[62] StopMotionPro, “Lip Sync Pro,” [Online]. Available: https://www.stopmotionpro.com/?page_id=45.

[63] P. Sonawane, K. Shah, P. Patel, S. Shah and J. Shah, “Speech To Indian Sign Language (ISL) Translation System,” 2021.

[64] P. Golimba and L. Stanciu, “Android Application to Support Communication Between Romanian Hearing and Deaf Peoples,” 2022.

[65] Google, “Android Studio,” [Online]. Available: https://developer.android.com/studio.

[66] Microsoft, “Hololens,” [Online]. Available: https://www.microsoft.com/pt-pt/hololens. [Accessed 2023 October].

[67] F.-C. Yang, C. Mousas and N. Adamo, “Holographic Sign Language Interpreters,” New York, NY, USA, 2022.

[68] StretchSense, “StretchSense,” [Online]. Available: https://stretchsense.com/. [Accessed October 2023].

[69] FaceWare Technologies, “FaceWare,” [Online]. Available: https://facewaretech.com/. [Accessed October 2023].

[70] M. Sanaullah, B. Ahmad, M. Kashif, T. Safdar, M. Hassan, M. H. Hasan and N. Aziz, “A real-time automatic translation of text to sign language,” 2022.

[71] T. Hanke, HamNoSys—Representing sign language data in language resources and language processing contexts, 2004.

[72] K. Kaur and P. Kumar, “HamNoSys to SiGML Conversion System for Sign Language Automation,” 2016.

[73] R. Elliott, J. Glauert, V. Jennings and R. Kennaway, “An overview of the SiGML notation and SiGMLSigning software system,” 2004.

[74] OpenVidu, “OpenVidu,” [Online]. Available: https://openvidu.io. [Accessed 2022].

[75] V. Alves, J. Ribeiro, P. Faria and L. Romero, “Neural Machine Translation Approach in Automatic Translations between Portuguese Language and Portuguese Sign Language Glosses,” 2022.

[76] V. A. G. Alves, L. Romero, P. M. Faria and J. Ribeiro, *Glossifica: tradutor automático de português para a Glosa da língua gestual portuguesa,* Viana do Castelo, 2023.

[77] Rokoko, “Sensor Fusion 2.0: More accuracy, less drift,” [Online]. Available: https://www.rokoko.com/insights/sensor-fusion-2-0-more-accuracy-less-drift.

[78] Lusa, “Professores de língua gestual portuguesa querem deixar de ser contratados como técnicos,” Publico, 2017.

[79] B. Ribeiro, D. Dias, P. M. Faria and L. Romero, “Capturing and Processing Sign Animations to a Portuguese Sign Language 3D Avatar,” 2023.

[80] B. Ribeiro, D. Dias, V. Alves, P. M. Faria and L. Romero, “A Translation System from European Portuguese to Portuguese Sign Language,” 2023.

9. Appendices

9.1. Published Papers in International Conferences

9.1.1. Capturing and Processing Sign Animations to a Portuguese Sign Language 3D Avatar

2023 30th International Conference on Systems, Signals and Image Processing (IWSSIP) | 979-8-3503-3729-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/IWSSIP58668.2023.10180233

Capturing and Processing Sign Animations to a Portuguese Sign Language 3D Avatar

Bruno Ribeiro[1], Duarte Dias[1], Pedro Miguel Faria[1], Luís Romero[1]

[1] ADIT-LAB, Instituto Politécnico de Viana do Castelo, Viana do Castelo, Portugal

(bfiliperibeiro,duartedias)@ipvc.pt (pfaria,romero)@estg.ipvc.pt

***Abstract*—Usually, the deaf community faces enormous difficulties in communicating with the hearing community. Unfortunately, there are few systems available to the public to assist in the communication between the deaf community and the hearing community. Some countries, including Portugal, are trying to develop systems to mitigate this problem. Thus, in this paper, it is proposed a system that captures signs, using motion capture systems and produces the needed animations, represented by a 3D Avatar, so that a deaf user may understand them. This first implementation consists of a component, receiving text in Glosses format, from the Portuguese Sign Language, getting the corresponding animations from a database, and animating a 3D Avatar with these animations. The work being carried out is part of a larger system, that includes a user interface and another component that translates Portuguese Spoken Language to Portuguese Sign Language, through a Neural Network. The system aims to assist deaf people in communicating in all scenarios of everyday life. The first results show great potential for the deaf community to be able to use the proposed system.**

***Index Terms*—Deaf, Assistive Technology, Portuguese Sign Language, Avatar, Animation, 3D Models, 3D, Glosses, Motion Capture, State Machine, LGP.**

I. Introduction

Portuguese Sign Language (LGP - *Língua Gestual Portuguesa*) is one of the three official national languages, which is predominately used by the deaf community to communicate with each other. According to the Portuguese Deaf Association [1], this population is thought to consist of around 30.000 people in Portugal, or about 110 thousand users of LGP nationwide when taking into account everyone in the surrounding community, including friends, teachers, technicians, etc. According to the results of the 2021 Census, this represents only around 1% of the national population. However, in the pockets of almost everyone, there may be a solution to mitigate these difficulties. In the same way that it is already perfectly normal to use smartphones for translations between Portuguese Spoken language and other languages, why not also use them for translation between spoken language and sign language?

This paper proposes a system that uses a 3D avatar to show deaf people understandable sign language through a mobile application. The system should receive a text that lists a sequence of signs, which is generated by a neural network responsible for the translation. This written representation of a sign is known as a gloss (the plural being glosses) and should be in all upper case letters. With the translated text, the system proceeds to get the correct animations from a database and animate them for the user to see them. These animations were captured with several motion-capture (MoCap) suits to speed up the process of capturing the animations for the gloss dictionary and to give a more realistic look to the signs.

This paper is organized into five sections: the first is the current introduction, followed by a section to present related work, in which some research projects are presented. Section three presents the global architecture of the developed system. Section four presents a solution for the acquisition of the animations required by the system. Section five will explore the way the Unity software processes animations and how the transitions between them work. Section six will discuss the results of some captured animations, validated by LGP speakers. Finally, conclusions and future work are presented, as well as some references used to do this work.

II. Related Works

Since there are so many sign languages, it is difficult to create a universal system and, consequently, most solutions implemented so far focus only on one sign language. When translating spoken language to sign language, is required to have a system that allows a deaf person to watch and understand a sequence of glosses, without reading the pre-translated sentence. Before doing this, a translation from a written sentence (in a spoken language) to glosses is necessary since the grammar and structure of spoken languages and sign languages are different. This translation is made through a system that first processes the original sentence, and then converts it into glosses that make sense in the targeted sign language's structure. These glosses define the order that the glosses should be shown so that a deaf person can watch and understand the original sentence without reading it.

In this context, the use of 3D avatars is the most common implementation nowadays, since the translation requires to have a "person" able to do a translated gesture. An Avatar is a model made in 3D software with an appearance similar to a human being, that has a Rigg, which is a skeleton that gives the model the ability to be animated, and a mesh, a "skin" of polygons that covers the skeleton, giving it a human appearance. Most solutions that translate spoken language into

The work reported in this paper was supported by FEDER, Program Compete2020, through the POCI-01-0247-FEDER-068605 project.

B. Ribeiro, D. Dias, P. M. Faria and L. Romero, 2023 30th International Conference on Systems, Signals and Image Processing (IWSSIP), Ohrid, North Macedonia, 2023, pp. 1-5, doi: 10.1109/IWSSIP58668.2023.10180233 [79].

9.1.2. A Translation System from European Portuguese to Portuguese Sign Language

A Translation System from European Portuguese to Portuguese Sign Language

Bruno Ribeiro[1], Duarte Dias[1], Vasco Alves[1], Pedro Miguel Faria[1], Luís Romero[1]
[1] ADIT-LAB, Instituto Politécnico de Viana do Castelo, Viana do Castelo, Portugal
{bfiliperibeiro,duartedias,vascoa}@ipvc.pt
{pfaria,romero}@estg.ipvc.pt

2023 30th International Conference on Systems, Signals and Image Processing (IWSSIP) | 979-8-3503-3729-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/IWSSIP58668.2023.10180292

***Abstract*—The deaf community is usually isolated from the rest of the hearing community, and this problem is found in many countries, including Portugal, where only about 1% of the population know Portuguese Sign Language. There are still no suitable applications to help them and, therefore, in this work, a system is proposed that translates European Portuguese into Portuguese Sign Language, showing the translation to an end user through a 3D Avatar. The system is divided into two components, the first being the textual translation system and the second the translations' animated representation, using a 3D Avatar. The former uses a hybrid solution of rule-based and neural approaches. The latter uses a motion capture system to represent the signs' gestures and creates a visual interface to animate the translation through an avatar. The implementation consists in a system receiving text in Portuguese, converting it into Portuguese Sign language, getting the corresponding animations from a database and animating a 3D Avatar with these animations. The system aims to assist deaf people communicating in all scenarios of the everyday life. The first results obtained are very promising for the added value of the system under development.**

***Index Terms*—LGP, Portuguese Sign Language, Neural Network, Virtual Avatar, Gloss Notation.**

I. Introduction

Regarding Sign Languages (SL), William Stokoe [1] explained that one of the first questions presented by those unacquainted with the subject is whether there is a universal SL. Although humans universally make use of facial expressions, arms and other parts of the body to signal, this is not the case when these signs are structured to create complex systems of words and sentences. These grammatical-lexical systems are unique, not only to specific nations, but to some bigger nations, which have multiple SLs and/or dialects. The Língua Gestual Portuguesa (LGP – Portuguese Sign Language) is one of the three official national languages of Portugal, being the one used by the deaf community. It is estimated that 30.000 people in Portugal are deaf, according to [2], and with the surrounding community, such as teachers, relatives, among others, it reaches around 110.000 people. This represents only 1% of the national population, according to the values of Censos 2021 (Portugal's Census) [3]. Looking at these numbers, it is easy to understand the difficulties deaf people go through when trying to communicate with the general population, since most people do not know anything about SL or how to use it. To overcome this difficulty, there is the possibility of using mobile devices that offer translation capabilities between both languages.

The work reported in this paper was supported by FEDER, Program Compete2020, through the POCI-01-0247-FEDER-068605 project.

This paper proposes a system that translates European Portuguese Language to LGP and uses a 3D avatar to show deaf people an understandable SL, through a mobile application. The system should receive a text or an audio from the user, and translate it to LGP, using a textual translation system. The textual representation of a sign is known as a gloss (the plural being glosses) and the system outputs a sequence of glosses. With the translated text, the system proceeds to get the correct animations from a database, sequencing them for the user to see. These animations may be captured with a motion-capture (MoCap) suit to speed up the process of capturing the animations to create a glosses dictionary.

II. Related Works

Since there are so many different SLs, it is challenging to develop a universal system. As a result, the majority of systems implemented up to this point have only addressed one SL. When converting spoken language to SL, it is necessary to establish a system that enables a deaf person to view and comprehend an animation, without reading the pre-translated sentence. Prior to performing this, a translation from a written sentence (in a spoken language) to glosses is required because spoken languages and SLs have different grammar and structural elements. The original sentence is analysed by a system that then transforms it into glosses, that make sense in the structure of the intended SL. The Gloss Notation consists in a text label for each sign in a SL [14]. The glosses are written in all uppercase letters and in the source language (e.g., LGP is written in European Portuguese). The HamNoSys system is the most widely used and is a pictograph notation system. Containing around 330 symbols that represent motions of a sign, this system works better with multiple SLs, since it does not depend on spoken languages. However, it is harder to learn by users. It is worth noting that none of the notation systems are accepted as standard written forms of a SL [15]. Several works related with translating spoken language into SL have been done. Some focus on translation into Glosses or other notation systems, others focus on the representation of SL via avatars or videos, while other focus in both parts. In Table I we can verify a summary of works, focused on those containing the translation and representation modules. On the translation column, there are two main approaches, the

B. Ribeiro, D. Dias, V. Alves, P. M. Faria and L. Romero, 2023 30th International Conference on Systems, Signals and Image Processing (IWSSIP), Ohrid, North Macedonia, 2023, pp. 1-5, doi: 10.1109/IWSSIP58668.2023.10180292 [80].

9.2. Code

9.2.1. Script to Get the Asset Bundles names

```
<?php

$user = "admin";
$pass = "pass";

try {
    $dbh = new PDO('mysql:host=localhost;dbname=anims', $user, $pass);
} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
    die();
}

$glossesUrl = $_GET["glosses"];
$glosses = explode('-', $glossesUrl);
$bundles = array();
$animnames = array();
$anims = array();

foreach($glosses as $gloss){
    $query = $dbh->prepare("SELECT assetbundle, animname, name FROM anims
WHERE name=:nome");
    $query->bindParam(":nome", $gloss);
    $query->execute();
    while($linha=$query->fetch(PDO::FETCH_ASSOC)){
        if(!in_array($linha["assetbundle"], $bundles)){
            $bundles[] = $linha["assetbundle"];
            }
        if(!in_array($linha["animname"], $animnames)){
            $animnames[] = $linha["animname"];
            }
        if(!in_array($linha["name"], $anims)){
            $anims[] = $linha["name"];
            }

        }
    }

echo json_encode([$bundles, $animnames, $anims]);

?>
```

9.3. Extract From the Questionnaire

Avaliação de LGP num Avatar

O projeto IVLinG pretende criar uma tradução bidirecional entre LP (Língua Portuguesa) e LGP (Língua Gestual Portuguesa). O módulo de LP para LGP foi desenvolvido pelo IPVC (Instituto Politécnico de Viana do Castelo) em que um tradutor converte texto em LP para uma sequência de glosses em LGP, e procede a usar um Avatar para demonstrar estes glosses. Neste formulário, pretende-se que pessoas com conhecimento de LGP avaliem o desempenho deste Avatar de duas formas.

A **primeira fase** foca-se na avaliação de gestos individuais através dos queremas da LGP. 32 gestos aleatórios do nosso sistema foram gravados para serem avaliados. A expressão facial ainda não foi adicionada nesta etapa do Avatar, por isso não é avaliado esse querema.

A **segunda fase** foca nas transições entre gestos numa frase. Dois sistemas de transições foram desenvolvidos, e o objetivo desta fase é ver a mesma frase em ambos os sistemas e avaliar qual deles tem transições mais realistas entre gestos.

1. Qual é a sua relação com LGP?

   Tick all that apply.

   - [ ] Professor de LGP
   - [ ] Intérprete
   - [ ] Apenas Utilizador
   - [ ] Other: _____

2. A LGP é a sua língua materna?

   Mark only one oval.

   - ◯ Sim
   - ◯ Não

Avaliação Queremas

Depois de ver um vídeo, selecione a checkbox dos queremas que estão corretos sobre esse gesto.

Gesto: Agora

http://youtube.com/watch?v=f8tLGrkUT5k

3. Agora: (selecione os queremas corretos)

Tick all that apply.

	Configuração da Mão	Movimento	Local da Articulação	Orientação da Mão
Mão dominante	☐	☐	☐	☐
Mão não dominante	☐	☐	☐	☐

Gesto: Comprimido

http://youtube.com/watch?v=rQ3jAGXHh8Y

4. Comprimido: (selecione os queremas corretos)

Tick all that apply.

- [ ] Configuração da Mão
- [ ] Movimento
- [ ] Local da Articulação
- [ ] Orientação da Mão

Gesto: Claro

http://youtube.com/watch?v=przMv7-3ZL8

5. Claro: (selecione os queremas corretos)

Tick all that apply.

	Configuração da Mão	Movimento	Local da Articulação	Orientação da Mão
Mão dominante	☐	☐	☐	☐
Mão não dominante	☐	☐	☐	☐